



AFP2web Version 3.x

User's Guide and Reference

Maas High Tech Software GmbH

Integrate the future today!



AFP2web is a registered trademark of Maas High Tech Software GmbH.

Trademarks or registered trademarks of other companies used in this publication: AFP (IBM Corporation); Helvetica, Times (Linotype Hell AG); ITC Zapf Dingbats (International Typeface Corporation); PostScript, Acrobat (Adobe Systems Incorporated); Windows (Microsoft Corporation). All other trademarks are properties of their respective owners.

Copyright © 2008 Maas High Tech Software GmbH

All rights reserved.

Maas High Tech Software GmbH,

Hornbergstr. 49, D-70794 Filderstadt, Germany

Telephone: +49 (0) 711 – 77917 – 0

Internet: <http://www.maas.de>

Table of Contents

Preface	7
About the AFP2web and Scripting Facility User's Guide	9
New Since AFP2web Version 2.x	13
Deprecated Features	17
Important Notices	19
Migrating from AFP2web Version 2.x to Version 3.x	21
 Chapter 1: Introduction to AFP2web	 23
1.1 Overview of the AFP2web Functionality	25
Supported Formats	26
AFP2web core components	27
Methods of Integrating AFP2web	28
1.2 AFP2web Application Scenarios	29
Archival of Documents and Index Update	29
On-the-fly Conversion for PDF On-Demand	30
AFP Viewer on Workstations	31
Document Distribution per Post (Print), E-Mail, or Fax	32
 Chapter 2: AFP2web User Guide	 33
2.1 Installing AFP2web	35
System Prerequisites	35
Installing AFP2web on Windows	37
Installing AFP2web on Unix	38
AFP2web Installation Results	38
Removing AFP2web	40
2.2 Configuring AFP2web	41
Setting Parameters in the INI File	42
Converting Sample Documents and Fine Tuning the Configuration	43
Providing AFP Resources	44
2.3 Using AFP2web	47

Quick Start	47
Starting/Stopping AFP2web From the Command Line	48
Controlling the Conversion Process	49
Selecting Documents for Conversion	50
Selecting Pages for Conversion	51
Processing AFP Indexes	52
Storing Output Documents in Sub Folders	54
Processing Included Objects (Included Images)	55
Notes on Barcode Support	57
Creating PDF Bookmarks for Index Elements	59
Adding Text as Watermark Behind the Document Text	60
Stopping Conversion When a Required Resource Is Missing	61
Using the Log File	62
 Chapter 3: Handling Fonts	 65
3.1 The Configuration Files and the Defaults Used for Font Processing	67
The INI File (afp2web.ini)	68
The Font Mapping File (mapping.def)	69
Code Page Files	70
Default Assumptions If Mappings or Font Resources are Missing	71
3.2 AFP2web Font Processing Modes	73
Use of the Original Fonts (Default)	73
Font Substitution and Referencing	77
Font Substitution and Embedding	84
 Chapter 4: AFP2web Scripting Facility User Guide	 87
4.1 Basic Concepts of the AFP2web Scripting Facility	89
Scripting Facility Applications	89
Scripting Facility Interaction With AFP2web	93
Scripting Facility Processing Features	98
4.2 Using AFP2web Scripting Facility	103
Activating the Scripting Facility	104
Setting the Perl Environment	106
Additional Usage Notes	108
 Chapter 5: AFP2web Reference	 109
5.1 Overview of INI Parameters and Command Line Options	111
5.2 INI File Parameters (afp2web.ini)	119
Parameters of the Section [Settings]	120

Parameters of the Section [FORMDEF]	139
Parameters of the Section [AFPColorTable]	140
User-defined Sections for Logical Locations	141
5.3 AFP2web Command Line Options	143
5.4 Parameters of the mapping.def	159
Function and Structure	159
Usage Notes	160
[CHARSET RENDERING] Section	161
[XFONT] Section	164
[CHARSET] Section	165
[FGID] Section	166
[FONTSUFFIX] Section	167
[PDFFONT] Section	168
[WINFONT] Section	169
[UNIXFONT] Section	170
[AFPFONT] Section	171
[CODEPG] Section	173
5.5 Code Page Files (CP Files)	175
5.6 Font Logging Output	177
Format of the Font Logging Message	177
Codes Used in the Font Logging Messages	178
Examples of Font Logging Messages	181
5.7 Identifying Fonts Using the IBM Naming Convention	185
The AFP Font Naming Convention	186
5.8 Scripting Facility Quick Reference	191
Scripting Facility Interface and Processing Levels	191
Script Routine and Methods	192
Methods and Where Used in a Script	199
5.9 Scripting Facility API Reference	207
Overview	207
afp2web.pm	208
a2w::Config (mhtIni [DEPRECATED])	214
a2w::Document (mhtDocument [DEPRECATED])	217
a2w::Font (mhtAFPFont [DEPRECATED])	225
a2w::Index (mhtAttributeElement, mhtIndexElement, mhtPagePageGroupIndex [DEPRECATED])	236
a2w::Kernel [NEW]	242
a2w::Line [NEW]	244
a2w::MediumMap [NEW]	251
a2w::NOP (mhtNOP [DEPRECATED])	254
a2w::Overlay (mhtResOverlay [DEPRECATED])	257

a2w::PSEG [NEW]	264
a2w::Page (mhtPtocaPage [DEPRECATED], mhtLPDPage [DEPRECATED]) ..	266
a2w::Text (mhtTextObject [DEPRECATED])	287
5.10 AFP2web Messages	295
General Information About Error Messages, Return Codes, and Error Files ...	295
AFP2web Error Messages	296
MHTIMG Library Messages	308
 Chapter A: Appendix	 313
A.1 Migrating Scripting Facility Modules to Version 3.x	315
A.2 Tutorial: Using the Scripting Facility Examples	319
Introduction	319
dumpIniParms.pm: Dumping INI Parameters	325
autosplit.pm: Splitting an AFP Spoolfile into Documents and Pages	329
dumpPageTextObjs.pm: Dumping Text and Font Information	339
sortPageTextObjs.pm: Sorting Text Objects in their Print Sequence	346
dumpMediumMaps.pm: Output List of Medium Maps to a Dump File	352
dumpNOPs.pm: Output the Contents of NOP Fields Into a Dump File	356
eyecatcher.pm: Find and Extract Text Objects	360
addBookmarks.pm: Adding Bookmarks to the PDF output document	370
addWatermark.pm: Adding Watermark Text to Pages	374
addAnnotations.pm: Adding a Hypertext Annotation	376
afp2xml.pm: Script Used to Produce XML Out of AFP	379
A.3 PDF/A Support in AFP2web	385
AFP2web's Commitment to the PDF/A Standard	385
What You Can Find in This Appendix	385
Important Notes	385
INI Parameters and Command Line Options	386
Fonts	387
Metadata in XMP Format	388
Features Not to Be Used for PDF/A	389
A.4 Frequently Asked Questions (FAQs)	391
E088: Scripting Facility Error (rc=-998): Unable to parse the file	391
 Index	 393

Preface

This preface offers

- a short introduction to the AFP2web User Guide
- a list of the features, which are new in AFP2web Version 3.x.
- notices to be aware of before using AFP2web
- a checklist for migrating from AFP2web 2.x to Version 3.x.

About the AFP2web and Scripting Facility

User's Guide

AFP2web is a utility for the intelligent recognition, separation, conversion, indexing, and distribution of AFP mass print data.

AFP2web converts AFP spool files to the standard formats PDF, TIFF, ASCII, and XML and PDF documents to AFP. With AFP2web, it is possible to web-enable AFP data and yet keep the benefits of using the secure and powerful IBM S/390 or Unix systems. AFP2web is used for web-enabling, archiving, indexing, document exchange in workflows, for producing high-quality true fidelity output of conversion results, and as a component in a variety of application scenarios.

The AFP2web Scripting Facility is an enhancement to AFP2web. It provides a scripting interface that is used to intelligently control document recognition, document splitting, index extraction, and much more.

Who Should Read this Document

This document is intended for all users of AFP2web. To use AFP2web, it is sufficient to set up a job environment and the configuration files of AFP2web. Users who need to customize AFP2web will find information on how to do this with the Scripting Facility.

What's in this Document

This document is organized into the following chapters:

- This preface with an overview of the new features of AFP2web Version 3.x, important notices, recommendations for migrating from Version 2.x to Version 3.x.
- Introduction to AFP2web, its functionality and possible application scenarios.
- AFP2web User's Guide, which includes information about installing, configuring AFP2web for your application, general information on how to run AFP2web, and how to use basic processing features.
- AFP2web font handling, which points out that AFP2web uses the original AFP font resources without the need for your intervention. However, AFP2web still offers processing modes, which are described here and can be used to substitute fonts with replacement fonts.
- The AFP2web Scripting Facility Guide, which serves as a high level introduction to using the Scripting Facility.

- The AFP2web Reference with detailed information about the syntax and meanings of the AFP2web configuration parameters (INI file, command line options, mapping.def), the AFP font naming convention used for missing font information, the Scripting Facility quick reference and API reference, and finally a list of messages and error codes.
- The Appendix offers additional information, such as a checklist for migrating Scripting Facility modules to Version 3.x, and a tutorial, which introduces the Scripting Facility examples.

Where to Find Additional Information

A source for more in-depth information on AFP is <http://www.printers.ibm.com>.

If you want to learn more about Perl, start with the homepage <http://www.cpan.org/>.

Whom to Contact

This manual provides the information you need to get started with AFP2web and the Scripting Facility. Maas High Tech Software GmbH gladly constructs a complete Scripting Facility module for your specific AFP print data format. You can then use this script as a starting point for further customizing. We also recommend using the Scripting Facility examples, which are delivered with AFP2web.

Maas High Tech Software GmbH offers professional services and consulting related to the development and deployment of AFP2web-based solutions.

For more information, please visit our Web site at

<http://afp2web.com>

or contact us via:

E-mail: afp2web@maas.de

Conventions

XML Data

In examples showing XML Documents we break and indent lines to make them more readable. Coding examples, listings, output to the system console are rendered in a fixed-width font.

Highlighting Convention for XML Data

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Index SYSTEM "Index.dtd" >
<Index Doctype="PDF" DocName="pdf/xml/INSURE_s3k.0.pdf"
PageCount="3" Size="16726">
  <PageGroupIndex PageGroup="00000001">
    <Data>
      <Name>Insured</Name>
      <Value>Geoffrey R Stephens</Value>
    </Data>
    <Data>
    </PageGroupIndex>
  </Index>
```


New Since AFP2web Version 2.x

AFP2web and Scripting Facility is a product bundle, which you use for any level of application complexity. AFP2web offers a core functionality and the Scripting Facility to further customize AFP2web to meet specific demands.

Strong in its font mapping capabilities, PDF document security features, and its powerful Scripting Facility, AFP2web Version 3.x has evolved from a conversion tool to a versatile document conversion and processing framework.

This section summarizes the product features, which are new.

PDF to AFP Conversion

AFP2web converts PDF documents to AFP format, which is compliant with the MOD:CA P or MOD:CA Interchange Set 2.

New INI parameters are introduced: PrintableLineWidth, CurveSamplingFactor, AFPComplianceLevel. The INI parameter PageRotation has been enhanced.

The mapping.def [CHARSET RENDERING] section has been extended for finer control of font rasterizing. The new section [AFPFONT] has been added to specify replacement fonts.

When font rasterizing is turned on for the PDF to AFP conversion, AFP2web uses the fonts, which it finds in the input files. AFP2web will also search for external referenced fonts.

Font Support Enhancements

AFP2web offers major improvements in its font handling capabilities:

- Font Rasterizing is a new font processing mode, which uses the original AFP fonts, thus producing output documents with 100% true fidelity.
- Font Referencing/Substitution and Font Embedding are font processing modes which replace AFP fonts with their corresponding substitution fonts. The supported font types are: Type 1, TrueType, and OpenType containing TrueType as a single subset.

AFP Input Features

AFP2web offers major improvements in processing AFP spool files:

- Support of colored graphics in form of IOCA FS 42 and 45, GOCA and Object Container (TIFF, JPEG,...).

- AFP2web is capable of handling barcodes: Code 128. AFP2web uses Character Set B of Code 128 to draw the barcodes because it includes lowercase letters, uppercase letters, numbers and punctuation characters.
- AFP2web also supports various formats of the bi-dimensional DataMatrix 2D barcode. For more information, please refer to [*"Notes on Barcode Support" on page 57.*](#)
- Support of image compression algorithms is enhanced: Uncompressed, IBM RL4, LZW, G3, G4, Old JPEG, JPEG, IBM MMR, IOCA (FS 10, 11, 42, 45), CCITTRLE, CCITTRLEW, IT8CTPAD, IT8LW, IT8MP, IT8BL, PIXARFILM, PIXARLOG, DEFLATE, ADOBE_DEFLATE, THUNDERSCAN, DCS, NEXT, SGILOG, SGILOG24. (Note: AFP2web does not support CMYK (32Bits/Pixel) JPEGs.)
- FTP access to remote resources. You can configure AFP2web to retrieve AFP resources from any FTP server.

PDF and PDF/A Output Features

AFP2web offers major improvements for output to PDF:

- Supports PDF Security Option 40/128 bit, which allows the creator to retain control of the document and associated rights.
- It is possible to define Adobe Acrobat Reader preferences for resulting PDF files.
- PDF documents remain full-text-searchable, even when the original AFP fonts are used.
- To reduce PDF file size, AFP2web has been optimized to use a best-match compression algorithm depending on the image type being either black/white or color.
- Easy document adaptation: changing and amending of text, indexes, annotations, bookmarks and many more.

AFP2web supports the new output format PDF/A, which conforms to the PDF/A-1b standard. The standard is defined in ISO 19005-1. Document management — Electronic document file format for long-term preservation — Part 1: Use of PDF 1.4 (PDF/A-1) Reference number ISO 19005-1:2005(E)

PDF/A-compliant documents are self contained PDF files, which are used for long-term archival. The standard is based on PDF Version 1.4 and imposes restrictions on the PDF functionality.

Features for Raster Formats

AFP2web offers major improvements for raster formats, such as TIFF:

- For raster format input, the support of the compression algorithms is enhanced: Uncompressed, LZW, G3, G4, Old JPEG, JPEG, CCITTRLE, CCITTRLEW, IT8CTPAD, IT8LW, IT8MP, IT8BL, PIXARFILM, PIXARLOG, DEFLATE, ADOBE_DEFLATE, THUNDERSCAN, DCS, NEXT, SGILOG, SGILOG24.
- For output to raster format: Easy document adaptation: changing and amending of text, indexes, and many more.

For output to TIFF, the following AFP2web-specific tags are added to insert metadata from the INI-file:

<i>TIFF Tag</i>	<i>Contents</i>
667	Keyword
668	Subject
669	Title

Note: When opening TIFF files generated by AFP2web, some TIFF viewers may issue a warning "Unknown field/tag" because of these custom tags. This type of warning can be considered insignificant.

New Features of the Scripting Facility

The Scripting Facility offers access to printable and non-printable page content before the conversion process. New functions make it possible for you to add, modify, or delete page content.

The Scripting Facility can output to memory streams, thus avoiding time consuming file read and write.

You can find more information on the Scripting Facility in this document. A set of sample Scripting Facility modules are provided, which you can use as a starting point for your own solutions.

Supported Operating Systems

The list of supported operating systems has increased. For more information, please refer to the list under "*Hardware and Operating System Requirements*" on page 35.

afp2web.ini and command line options

New INI parameters and command line options include:

- Section [AFPColorTable] to map each color of the AFP standard OCA color table to their corresponding RGB values. The corresponding command line option is available for specifying a list of colors, such as:
-clr:<colorname>,<r>,<g>,,<colorname>,...
- Autosplit is an INI-file parameter for automatically splitting AFP spool files into documents and pages. This parameter affects the processing of the Scripting Facility.
- CMYKTORGB to convert IOCA FS45 CMYK color image planes to RGB. The corresponding command line option is used to switch this function off: -nocmyktorgb.
- FilenamePattern to specify the pattern for the names of the output files.
- LoggingLevel with new logging levels: 8, SF, RES, OLY, PSEG, FNTR, CDP, MMAP, FDEF, IOB. The corresponding command line option is -ll.
- PDFSecurity to set rights for accessing resulting PDF documents.
- PDFUIOptions, PDFWinOptions to set preferences for displaying a document in the PDF viewer. The corresponding command line options are -pui and -pwin.
- SkipPage, SkipObjectSize are new parameters that specify pages to ignore in the AFP spool file.
- User-defined sections to specify the FTP location of remote AFP resources.

Additional Enhancements

Additional enhancements include:

- Image Handling Enhancement: Faster image handling, better image compression, smaller PDF files.
- Color images to gray scaled images conversion for B&W output
- AFP2web displays its version information in the following format: AFP2web Version <version number> [Built for <OS name> on <Date> at <Time>]

Deprecated Features

This section lists the product features, which AFP2web Version 3.x no longer supports.

Maas Font Metric Files

AFP2web uses the original fonts and does not require additional font formatting hints. Thus, AFP2web no longer supports the use of font metric files.

Important Notices

Please use the AFP2web and Scripting Facility with great care. Incorrect settings of AFP2web parameters can affect the conversion results. In the worst case, documents might become useless for archiving purposes.

We therefore recommend:

Test thoroughly!

You should thoroughly test the run time parameters, the resources, and the font settings against each type of AFP document. Especially when you process documents for archival.

Only you know your fonts!

AFP2web cannot automatically detect any customizations you have made to your resources, for example any additions to your code pages. In doubt, please contact Maas High Tech Software GmbH to resolve any problems.

Make backups of your documents!

AFP2web does not replicate data for backup. It is your responsibility to ensure that you have backup copies of the data you pass to AFP2web.

Backup your previous installation of AFP2web!

Save your previous customized versions of your configuration files before you install a new release of AFP2web. Please note that you must migrate your old customizations to the new requirements as described later on in this document.

We recommend preserving your old versions of the files:

- INI file
- afp2web.pm (or your custom scripts for the Scripting Facility)
- mapping.def
- CP files (*.cp)
- fonts for PDF(*.pfm and *.pfb files)

Tip: Rename the target folder of your previous A2W installation (Example: A2W_version_date). Create a new target folder for the new installation using the old folder name (Example: A2W). After installing a new release of AFP2web, you can migrate your customizations to the new configuration files.

Use the AFP2web Scripting Facility with care!

Using AFP2web without care can result in loss of data. The appearance of documents can be distorted after conversion or the identification of documents can be lost because of incorrect document splitting.

We therefore recommend:

- You can use different instances of the Perl script `afp2web.pm` to handle different types of spool files. To do this, you use the `-sp` option to specify the Scripting Facility module to use.
- Please be sure to test the results of your programming in `afp2web.pm`. Your scripts are responsible for page skipping, document splitting, and the processing of index data.

Migrating from AFP2web Version 2.x to Version 3.x

AFP2web Version 3.x has new enhancements, which require changes in the configuration files and in the scripts used for the Scripting Facility.

Important: Any customizations in your previous version of AFP2web might no longer be necessary or should be adapted to the new requirements described below.

Changes in the mapping.def

AFP2web has improved its font handling functionality. The required font definitions made in the previous version of AFP2web are now obsolete because AFP2web now uses the original AFP fonts. In most cases, the mapping.def entries required for AFP2web Version 2.x may be removed.

However, there might be cases, where substitution fonts need to be used (font substitution, font referencing). In this case, the configuration files require definitions in the mapping.def.

AFP2web Version 3.x has the following changes in the definition of the mapping.def:

- In the [PDFFont] section, font names are no longer prefixed.
- [CHARSET RENDERING] section is new and specifies when to override AFP2web's default font processing.
- The [FONTSUFFIX] section is used to specify suffixes to the names of your font files. Because there is no standard convention followed for font file names, this enhancement will give you the freedom of specifying one or more conventions.

Changes in the CP Files

AFP2web Version 3.x has extended the format of the code page files (CP files) for Unicode support. This extension is required for adding text search capability to the PDF files.

Defining and Supplying Font Resources

AFP2web no longer requires a substitution font to be used as a replacement for an AFP font. If the AFP font resource is available either in-line with the AFP spool file or as external AFP font resource, then AFP2web uses this font.

However, there might be reasons for replacing an AFP font and for referencing or embedding a substitution font. AFP2web still offers its basic font replacement functionality. The AFP font resource delivers the basic attributes for a font, which determine which substitution font file to use. In addition, you can specify naming conventions for the suffixes to add to the name of font files.

Note: *In the previous version of AFP2web, fonts for the conversion to TIFF had to be installed in the system. AFP2web Version 3.x now will install a required font at runtime, if necessary, and will de-install the font upon process completion.*

For more information, please refer to "Handling Fonts" on page 65.

Scripting Facility Modifications

You must modify your custom scripts for the Scripting Facility.

The Scripting Facility has changed the name of its packages and methods. For a detailed list, refer to "Migrating Scripting Facility Modules to Version 3.x" on page 315.

The Scripting Facility has greatly enhanced its functionality. For a quick overview of all available packages and methods, please refer to "Scripting Facility Quick Reference" on page 191.

You can find the detailed API reference in this user guide.

Chapter 1: Introduction to AFP2web

This chapter gives a quick overview of AFP2web:

- Input and output formats supported for document conversion
- AFP2web core components
- Components used to build AFP2web applications
- Typical AFP2web application scenarios

1.1 Overview of the AFP2web Functionality

In the past, the standard for mass print data was tied to proprietary AFP technology. Today, AFP2web opens the door to a more effective communication. AFP2web makes AFP documents available to any kind of open system by converting them to the de facto standard Internet data formats, such as PDF, JPEG, ASCII, TIFF, XML. AFP2web enables an intelligent recognition, separation, conversion, indexing and distribution of mass print data.

One product focus lies on producing XML output for easy integration with upcoming Internet technology-based print solutions and for powerful data mining from print documents. AFP2web has proved to be inspiring. New applications are evolving. For more information about AFP2web solutions, please refer to the Web site <http://afp2web.com>.

In this section, we describe:

- Supported Formats
- AFP2web core components
- Methods of Integrating AFP2web
- AFP2web application scenarios.

Supported Formats

There is a matching solution for every conversion requirement. In addition to the AFP2web basic software, many options are available. Almost any combination of input and output formats can be selected. The AFP2web format options include converting documents from AFP or TIFF to ASCII, JPEG, PDF, TIFF, XML and other common formats. The following figure shows the format options, which are available.

Input and Output Formats supported by AFP2web

Output Input	AFP	TIFF	PDF	PDF/A	JPEG	PNG	ASCII	XML
AFP	X	X	X	X	X	X	X	X
Raster 1	X	X	X	X	X	X		
PDF	X	X	X	2	X	X	X	X
LPD	X	X	X	X	X	X	X	X
ASCII	X	X	X	X	X	X	X	X
MMD	X	X	X	X	X	X	X	X

1 = TIFF, PCX, BMP, GIF, JPEG

2 = planned / geplant

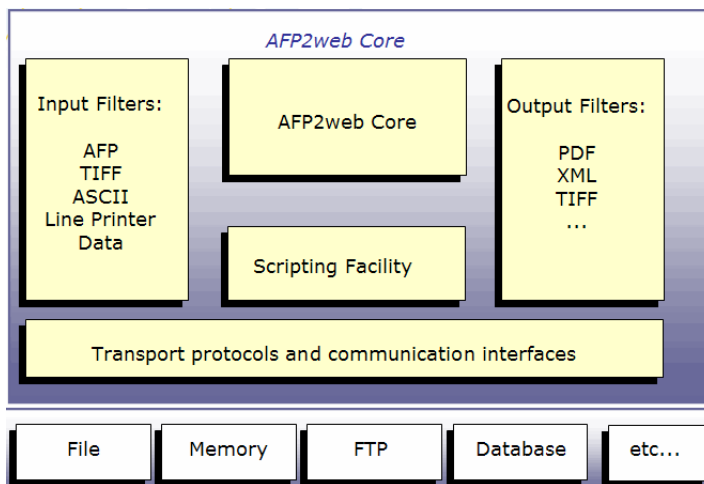
AFP2web core components

The future is open to AFP2web. The internal structure of this document conversion tool is highly modularized. The internal document model is mapped to internal adapter interfaces, which can accommodate new input and output formats, if required.

One of its major strengths is AFP2web's communication layer and its interface adapters. Daily requirements of many customers of AFP2web include the option of selecting any common channel for reading input and writing output: via File IO, in-memory buffer streams, remote FTP, proprietary databases, and any other channel, if required.

The following figure shows the internal module structure of AFP2web:

AFP2web Internal Module Structure

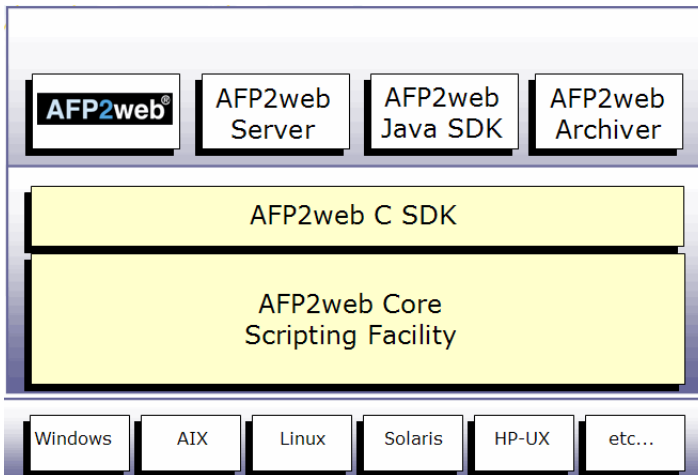


Methods of Integrating AFP2web

AFP2web offers a powerful base functionality, which is configured for the specific application. In its basic version, AFP2web is started as a console application from the command line to process AFP spool files. The AFP2web Scripting Facility is an enhancement, which can be used to formulate processing rules and to trigger external programs. Both AFP2web and the Scripting Facility enable intelligent and flexible document control.

The following figure shows how AFP2web components combine to build AFP2web products:

Components Used to Build AFP2web Products



The optional Software Development Kit (SDK) is available for integrating AFP2web into existing document management or content management solutions for on-demand document conversion. The AFP2web SDK provides the AFP2web Document Management System (DMS) Application Programming Interface (API). The AFP2web DMS API has functions for input/output via in-memory buffer streams (afp2web) and file IO (afp2web_f). In-memory buffer streams are typically used for on-demand conversion of single documents. File input and output are used to process mass print data in spool files.

The JNI interface ensures easy integration into J2EE-Server environments such as IBM WebSphere.

1.2 AFP2web Application Scenarios

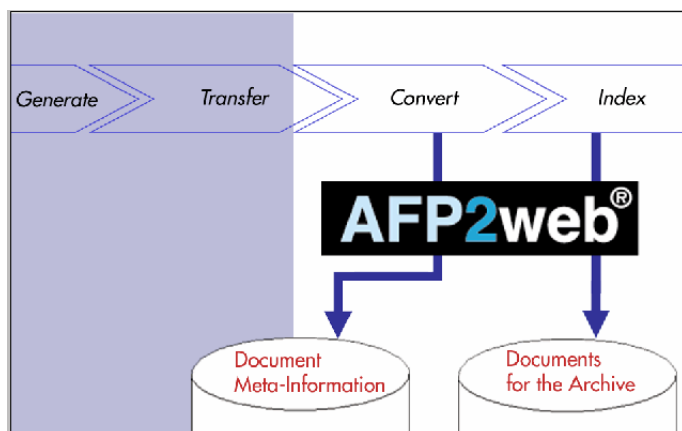
In the following, we describe typical application scenarios for AFP2web:

- Document archival and index update
- On-the-fly conversion for PDF on demand
- AFP viewer for workstations
- Document distribution per post (print), e-mail, or fax

Archival of Documents and Index Update

This is the scenario for an automated conversion, indexing and archival of documents. The process runs in batch mode and converts mass print data. The conversion process must run smoothly, efficiently, reliably, and must produce high-quality conversion results.

Scenario: Document Archival and Index Update



The document archival process is a workflow, which interfaces with AFP2web to convert the documents for archival. The steps of this process include:

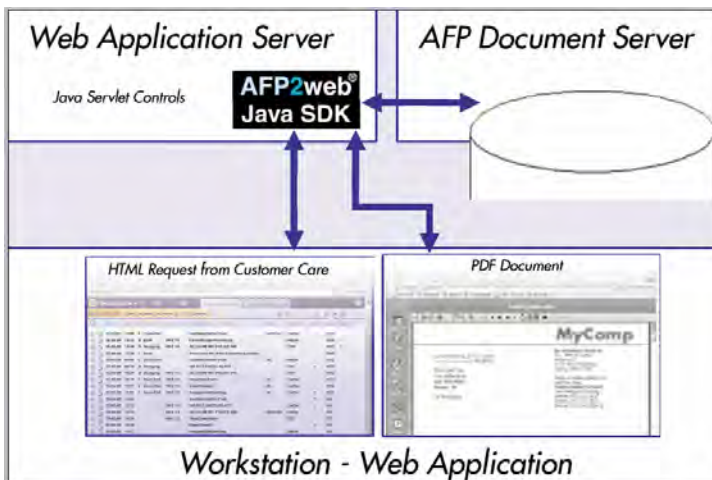
1. AFP data generation
2. AFP data transfer (via FTP)
3. AFP2web document conversion (PDF, one of the raster formats, or XML)
4. Document archival on the document server, delivering a technical document ID
5. Update of the index database with document meta information (delivered by AFP2web) and the technical document ID

AFP2web not only delivers excellent conversion quality. AFP2web can also be used as a central data extraction utility for standard AFP indexes. Moreover, the AFP2web Scripting Facility can be used to fine-tune the data extraction for any type of information in the document. The Scripting Facility can be used as a link for follow-on processing, for example in document-oriented work flows.

On-the-fly Conversion for PDF On-Demand

Display individual AFP documents on request in a front end Web browser. The desired format is PDF delivering a true 1:1 rendition of the original AFP document.

Scenario: On-the-Fly Conversion for PDF On-Demand

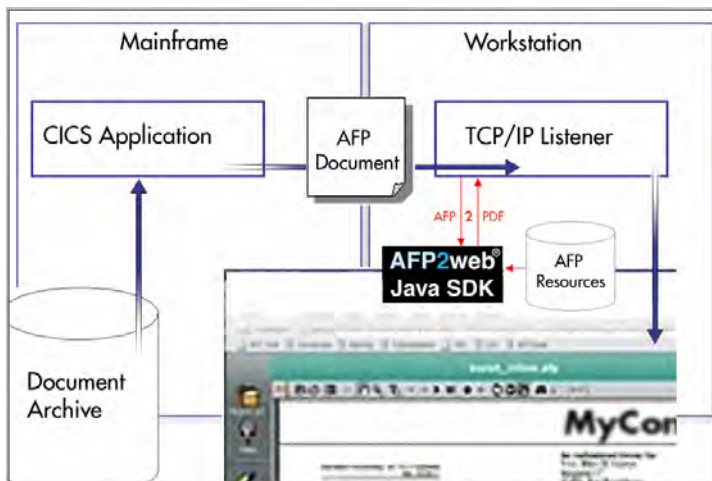


AFP2web is used, for example, as a component in a three-tier application, typically in an Internet application. Controlled by a Java servlet, AFP2web reads the AFP document from the input buffer and writes the PDF document to the output buffer. AFP2web avoids file input and output, works with in-memory data streams, and is thus optimized for quick processing. The efficiency and the high quality conversion make it possible to view and process documents quickly, online, and in distant branch offices.

AFP Viewer on Workstations

A document in proprietary AFP format, which is retrieved from the archive, needs to be delivered on demand and viewed as PDF online in a Web browser.

Scenario: AFP Viewer on Workstations



Using the AFP2web Java Source Development Kit, AFP2web is implemented as a standalone server, satisfying on-demand conversion requests. Combined with any PDF or TIFF viewer, AFP2web is the ideal AFP viewer for displaying AFP documents. AFP2web offers efficiency and high quality, on-the-fly conversion results.

Document Distribution per Post (Print), E-Mail, or Fax

Convert AFP documents to PDF or TIFF. Dispatch documents per post, e-mail, or fax.

No matter whether the output format be PDF or TIFF, produce the format for your purposes and:

- use an e-mail or fax server to quickly deliver AFP data.
- combine AFP2web with a print server, and be able to print AFP in color on non-AFP printers.

Chapter 2: AFP2web User Guide

This chapter describes how to:

- Install AFP2web
- Configure the system for AFP2web

You will also find the information you need to work with AFP2web, including:

- A quick start in using the demo files.
- How to start and stop AFP2web processing
- How to use specific features of AFP2web
- How to read the AFP2web error messages and log files to resolve any problems.

2.1 Installing AFP2web

System Prerequisites

Hardware and Operating System Requirements

AFP2web can run on one of these operating systems:

- Windows NT/2000/2003 with an x86 processor
- Linux with x86 processor
- AIX 5.1 32-bit RS/6000 PowerPC
- Sun Solaris 7 (SunOS 5.7) 32-bit on an Ultra-250 Sparc
- HP-UX 11.i 64-bit C3600a PA-RISC
- zLinux SuSE SLES7

The minimum amount of random memory should be available:

- At least 512 MB RAM.

Additional System Requirements

For conversion to one of the raster formats (TIFF, JPEG) on a UNIX system, X-Windows is required. If no XServer is running, you can use Xvfb or VNC.

For the following list of operating systems (processors), AFP2web has the Perl engine included. If you need to use your own Perl environment, please install Perl 5.8.7.

The Perl engines included in AFP2web are:

- Windows (Intel): Perl 5.8.7
- Aix 5.1 (RISC): Perl 5.8.7
- Solaris 2.7 (SPARC): Perl 5.8.7
- Suse Linux 9.1 (Intel): Perl 5.8.7
- HP UX 64: Perl 5.8.7

AFP2web is compiled on HP-UX using the GNU-Compiler (V3.4.2) and not the HP-Compiler (aCC).

Hard Disk Capacity

Plan for enough storage to hold your input and output files. To estimate the amount of storage required, the following figures might help:

- A resulting TIFF file is approximately 10 times larger than the respective document in the AFP data stream. This applies when using "G4" as the compression option. The factor is about 40 when using the option "packed". The factor is 100 when using no compression.
- A resulting PDF file is about 1 to 2 times larger than the respective document in the AFP data stream. This value depends on the structure of the AFP document.
- Use the **PDFDocLimits** parameter of the INI file to specify or increase the limits of memory resources. Using PDFDocLimits might have an impact on performance. When used with **DOC_COLD** (Scripting Facility) memory allocation will happen for each resulting PDF document. PDFDocLimits should therefore be set to the lowest possible value to increase performance. When used with **-DOC** option, memory allocation will only happen once, thus there is no side effect on performance.
- For each font embedded in the PDF file, the size of the PDF file will increase by an average of about 30 KB to 50 KB.
- If you use the option **-sod** for included objects, AFP2web writes these objects to disk to speed up processing and to highly optimize the object data for conversion. In this case, plan for extra hard disk capacity.
- Log files produced by AFP2web can be very large. AFP2web will also suffer a degrade in performance when producing log files during the conversion. Log files are normally used only during development and integration not in a production environment.

Fonts for the Conversion

If you specify this in the mapping.def as a font processing mode or if AFP2web cannot find the AFP font, AFP2web substitutes AFP fonts by those fonts specified in the configuration files. The requirements are:

For a raster format (TIFF, JPEG, PNG, AFP): The required fonts should be installed in the Windows, respectively Unix system.

Note: *In the previous version of AFP2web, fonts for the conversion to TIFF had to be installed in the system. AFP2web Version 3.x now will install a required font at runtime, if necessary, and will de-install the font upon process completion.*

For PDF: AFP2web either references or embeds the fonts in the PDF file.

Installing AFP2web on Windows

Preparation: It is recommended that you save the configuration files of your previous AFP2web installation. You might have customizations, which you want to redefine in the new configuration files. If you followed our installation recommendations you already have a backup copy of the following files:

- INI file
- afp2web.pm (or your custom scripts for the Scripting Facility)
- mapping.def
- code page (*.cp) files

To install AFP2web on Windows NT/2000/2003:

1. Start the setup routine `setup.exe`

The setup routine is a self extracting file, which stores all AFP2web installation files in the selected target directory.

2. Select the target directory for installing all files.

Click the button [...] to select a directory

- Select the option **Confirm overwrites** if you want to prevent overwriting existing files.
- Click OK to start the installation process.

The dialog **Unpacking files** shows the progress indicator:

The setup routine unpacks all files to the target directory and creates default subdirectories in a few seconds. You can stop installation by clicking **Cancel**.

3. If the dialog **File exists, overwrite? ... appears, confirm with:**

Yes if you want to overwrite an existing file

No to prevent overwriting this file

Cancel if you want to stop installing at this point.

Installing AFP2web on Unix

To install AFP2web on UNIX you unpack the file afp2web.tgz.

1. On Solaris, use two commands:
 - `gzip -d afp2web.tgz`
 - `tar xvf afp2web.tar`
2. Else use one command: `tar xvzf afp2web.tgz`

AFP2web Installation Results

The following files are installed for AFP2web

<i>Folder</i>	<i>Files</i>	<i>Description</i>
(target folder)	afp2web.exe (Windows) afp2web (Unix)	The AFP2web program
	afp2web.ini	Configuration file with general settings for AFP2web
	demo.bat (Windows) demo (Unix)	Batch file as example of calling the program
	history.txt	Change history
	license.txt	Licensing information
	quickstart.txt	Quick start instructions on using demo.bat (Windows), or demo (for UNIX)
	readme.txt	Last-minute information
	afp2web.pm	AFP2web Scripting Facility script template
	*.pm files	Scripting Facility examples
	Perl*.dll	Perl DLL for your operating system (if offered by AFP2web)
a2w	Perl modules (*.pm)	PERL packages of the AFP2web Scripting Facility
afpcp	codepages (*.cp)	Code page mapping files
	mapping.def	Configuration file, used to map AFP-specific fonts to substitution fonts

<i>Folder</i>	<i>Files</i>	<i>Description</i>
	gcid2unicode.def	Default Unicode mapping file.
afpcp\mfm		(deprecated)
doc	afp2web_de.pdf afp2web_en.pdf	AFP2web User's Guide (en=English, de=German)
log		Target folder for log files
pdf		Target folder for converted documents (Note: DTDs are included for the afp2xml.pm example)
extfont	*.pfb and *.pfm files	Folder for additional Adobe Type 1 and TrueType fonts used for the conversion to TIFF and PDF
samples		Examples
samples\resource		Resources for the examples
XML	Writer.pm	The Perl module required to run the example afp2xml.pm

Removing AFP2web

To remove AFP2web from your system, delete all files and directories created by the installation routine of AFP2web.

2.2 Configuring AFP2web

Overview

Configuring AFP2web involves setting parameters and providing resources:

- [*"Setting Parameters in the INI File" on page 42*](#)
- [*"Converting Sample Documents and Fine Tuning the Configuration" on page 43*](#)
- [*"Providing AFP Resources" on page 44*](#)

Setting Parameters in the INI File

The configuration file (INI file) defines global parameters of AFP2web, such as conversion options, output paths, paths to AFP resources, and other processing options. For a complete description, please refer to *"INI File Parameters (afp2web.ini)" on page 119*.

By default, AFP2web searches for an INI file with the name **afp2web.ini** in the current working directory. You can access a particular INI file with one of the following command line options:

-if	Path and filename of the INI file
-ip	Path to afp2web.ini

You can override most of the parameters by entering the appropriate command line option when starting AFP2web. Command line options are described in detail under *"AFP2web Command Line Options" on page 143*.

Converting Sample Documents and Fine Tuning the Configuration

Run-testing AFP2web by converting sample documents is the first step in the process of fine tuning the AFP2web configuration. The conversion results and the messages in the LOG file indicate what remains to be done.

In the following description, we give a few hints on what to take care of.

Download AFP Spool File (Host Documents) in Binary Format

Please take care to download any AFP spool file from the host in binary format. (A common mistake is to download AFP documents and resources from the host to the PC and, when doing this, to perform an ASCII conversion.)

Convert the AFP Spool File and Check the LOG File

Run the `afp2web.exe` program to convert your sample documents in the AFP spool file.

To quickly find out how to start AFP2web, you can follow the instructions and use the demo batch command file as described under [*"Quick Start" on page 47*](#).

AFP2web creates the following files as output:

- The converted spool file, documents, or a text file with error messages.
- A log file, if logging is activated.

When you start AFP2web, activate the logging option. Logging will slow down AFP2web, but is necessary at this stage. For example, to investigate font processing, use the INI parameter **LoggingLevel=FONT** (or the command line option **-lf**).

Check the converted document and any system messages to identify problems. For example, a missing resource or font might be the reason for a non-satisfactory result.

Examples include:

- A resource is missing. In this case, please check if the resource is located at the position specified in the INI file (or as specified in the command line options when calling AFP2web).
- If font substitution/referencing is used, the log file might indicate that a substitution font is missing. Check to see if the font is correctly specified in the `mapping.def`. In addition, this font must be either installed (for output to a raster format, such as TIFF) or must be found in the specified folders (for PDF).

For more information, please refer to ["Using the Log File" on page 62](#) or to ["Font Logging Output" on page 177](#).

Providing AFP Resources

AFP resources (such as Fonts, Formdefs, Overlays, Page Segments) can be defined in-line in the AFP data stream. If not, then AFP2web searches for an external resource in the folders you have specified.

Note: The INI parameter **CodePage** (or command line option **-dcp**) defines the default code page to use for translating resource names, NOPs and indexes of the AFP spool to ASCII.

Normally, no special configuration is required for AFP2web's font handling. If the AFP fonts are available either in-line or as external resources, AFP2web will take the actual fonts and either embed them (in the PDF output) or rasterize them (for output to a raster format such as TIFF). Thus, the actual font is used and the conversion results reproduce the original documents in true fidelity.

For more details about using fonts, associated code pages, and the configuration files, please refer to ["Handling Fonts" on page 65](#).

Path to External AFP Resources

There are different possibilities of providing external AFP resources. For example, you enter the command line option **-rf** when calling the program to use a particular resource file, or you use the settings of the INI file to specify paths for specific resource types.

For all parameters except for **ResPath (-rp)** and the option **-rf**, you can specify a comma-separated list of paths. The parameters, which you can use, are:

<i>INI file</i>	<i>Command line option</i>	<i>Type of AFP resource</i>
ResPath	-rp	All resource types
	-rf	All resource types in a particular resource file
AFPFontPath	-ap	AFP font resources
OverlayPath	-ovp	AFP Overlay resources
PageSegmentPath	-psp	Page Segments
FormDefPath	-fdp	Formdefs

The Standard Search Path Used to Find a Resource

AFP2web searches for external AFP resources in the following order:

1. The folder specified for the resource type (default is samples/resource)
2. Folder specified in **ResPath** (default is samples/resource)
3. Folder of the input files
4. In the default folder "samples/resource".

The Search Path for Non-AFP Resources

When AFP2web searches for non-AFP resources, for example, for TIFF/JPEG files requested in IOBs, AFP2web looks in the following paths in this order:

1. Resource path (given by **ResPath** INI attribute)
2. Input file path
3. Current path

Optional Feature: File Extensions for AFP Resources

It is possible to use the INI file to specify file extensions for the following resource types:

FormdefExt	File extension for files with Formdefs
OverlayExt	File extension for Overlays
PageSegExt	File extension for Page Segments
CharSetExt	File extension for Character Sets
CodePageExt	File extension for Code Pages
CodedFontExt	File extension for Coded Fonts

The Processing Flow of AFP2web When Searching for a Resource

AFP2web doesn't need an extension to the resource name. AFP2web searches for resources as follows:

1. AFP2web follows the INI file setting (default or the user-specified extension) to search for the resource. AFP2web goes through the standard search path to find the resource.
2. If the resource cannot be found in step 1 and the option **Strict** is on, AFP2web will stop the process.
3. If the resource is not found in step 1 and the option **Strict** is off, AFP2web will search for the required resource again in the standard search path. AFP2web takes the first resource found with any type of extension.

Important: AFP2web searches for resources, with file name beginnings that match the name of the resource. The first file found is taken. This might not be the resource you want.

Example: C1H20000 is required. Per default A2W will look for "C1H20000*". All the following resources will be valid: C1H20000, C1H20000.chs, C1H20000.240, C1H20000_ABC.def. If more than one resource matches the search pattern, A2W will take the first one found.

If the INI file specifies "chs" as the valid file type extension, then AFP2web will find the resource in step 1: "C1H20000.chs".

If the INI file has "*" or no settings, then the default applies and AFP2web will find the resource in step 3: "C1H20000".

Specifying FORMDEFs

As mentioned at the beginning of this section, AFP resources (such as Formdefs, Overlays, Page Segments) can be defined in-line in the AFP data stream. If not, then AFP2web searches for an external resource in the folders you have specified. For example, you can use the command line option **-fd** or **-rf** to reference a resource file containing a Formdef.

A Formdef is a special AFP resource that contains medium maps and/or copy groups, but which is not referenced in the AFP data stream. The **[FORMDEF]** section of the INI file can be used to map Medium Maps to Formdef resource names. This feature is useful, for example, if the command line option **-fd** is not used. The Formdef itself must be available as external resource.

2.3 Using AFP2web

Quick Start

To convert AFP documents with AFP2web:

1. Make sure that you download your AFP documents and resources in binary format from the host.
2. Copy your AFP spool file to the directory **samples/**.
3. Copy your AFP resources to the directory **samples/resource/**.
4. Edit **demo.bat** (Windows), respectively **./demo** (Unix) and specify the name of your AFP spool file
5. Run **demo.bat** (Windows), respectively **./demo** (Unix)
6. Check the output in the directory **pdf/**.
7. Repeat steps 1 to 6 for each and every AFP spool file.

Note: *If you encounter any problem, please contact afp2web@maas.de.*

Starting/Stopping AFP2web From the Command Line

Starting AFP2web

You start the program afp2web as a console application with the following command:

On Windows: afp2web.exe [-options ...] inputfile

On UNIX: ./afp2web [-options ...] inputfile

Wild cards are allowed for input file, for example: `"*.afp"` for all files ending with `"afp"`.

For a detailed description of the command line options you can enter, please refer to *"AFP2web Command Line Options"* on page 143.

Stopping AFP2web

To stop AFP2web press **CTRL+C** on the system console or stop the task in the Windows Task Manager.

On Unix, **kill** the process.

Controlling the Conversion Process

In this section, we list the parameters of the INI file, which you use to control the formats and the types of output documents to create.

The parameter **InputFormat** specifies the format of the input documents to convert. This parameter can have one of the following values:

AFP	AFP Document
TIFFIN	TIFF document

The parameter **OutputFormat** specifies the target format for the conversion. This can be one of the following:

ASCII	ASCII format with adjustments to line/column positioning
JPEG	JPEG format
PDF	PDF format
PNG	Portable Network Graphic
TIFF	TIFF format

The parameter **FileCreationMode** controls splitting or merging documents, creating index data or activating the Scripting Facility. One of the following options are possible:

ALL	One output file for each input file.
DOC	Multiple output files: One file per AFP document.
DOC_MERGE	Merge all files into one output file. For converting to a raster format, such as TIFF, to PDF or for converting AFP.
DOC_INDEX	Create one index file and one output document per indexing information unit.
DOC_COLD	Split documents and process document elements using the AFP2web Scripting Facility.
PAGE	One file per page in an AFP document.

Selecting Documents for Conversion

Processing AFP spool files normally involves converting individual AFP documents and processing AFP index data. AFP document boundaries are detected through document index elements. Or, when using the Scripting Facility, according to the rules defined in a script. To let AFP2web process standard index elements, specify **DOC_INDEX** (See also ["Processing AFP Indexes" on page 52](#)). To process index data and document splitting with the Scripting Facility, specify **DOC_COLD** (See also ["Basic Concepts of the AFP2web Scripting Facility" on page 89](#)).

You can instruct AFP2web to either start or end processing the spool file at a particular AFP document. AFP2web will parse the complete spool file to find the specified document. This option is used to handle special cases, such as emergency restarts or for debugging.

Tip: Using the option **-p** creates processing log which we call a „protocol“. This protocol is a useful reference for selecting documents for the conversion. In the protocol, you will find a list of the documents processed and their document IDs. The protocol is written to the folder for output files (INI file parameter **OutputFilePath**, or the **-op** command line option). Filename of the protocol is: „protocol“ + timestamp + „.txt“.

Example of a Protocol Showing Document IDs

```
Running AFP2Web Version 3.0rc1 [Built for Windows NT/2000/2003 on
Dec 18 2005 at 13:48:50] at 21:34:06 on 21 December 2005 using :
-q -PDF -c -p -DOC_INDEX:XML -op: pdf\xml samples\insure.afp
```

```
Processing Spool File: samples/insure.afp
[21:34:09] Processing Document Id : 1, pdf/xml/insure_s1mc.1.pdf
: Ok
[21:34:11] Processing Document Id : 2, pdf/xml/insure_s1mc.2.pdf
: Ok
[21:34:11] Processing Document Id : 3, pdf/xml/insure_s1mc.3.pdf
: Ok
[21:34:12] Processing Document Id : 4, pdf/xml/insure_s1mc.4.pdf
: Ok
[21:34:12] Processing Document Id : 5, pdf/xml/insure_s1mc.5.pdf
: Ok
```

```
AFP2Web Version 3.0rc1 [Built for Windows NT/2000/2003 on Dec 18
2005 at 13:48:50] ended at 21:34:13 on 21 December 2005, Elapsed
Time 00:00:07
```

A case for using this option: When AFP2web stops processing due to a failure (for example memory or file space exhausted) you can refer to the protocol to find out where AFP2web stopped.

After solving the problem you do not have to process the complete spool file again. Instead you use the parameters **StartingDocument** (or the option **-sd**) and **EndingDocument** (or the option **-ed**) to select a range of documents for processing. For example, to

start converting at particular point in the spool file, you start AFP2web with the option **-sd:docid**, where docid is the running document number which you can find as document ID in the “protocol”.

Parameters for processing spool files:

<i>INI file</i>	<i>Command line option</i>
FileCreationMode=DOC_INDEX or FileCreationMode=DOC_COLD	-DOC_INDEX or -DOC_COLD
Protocol=on	-p
StartingDocument	-sd
EndingDocument	-ed

Selecting Pages for Conversion

You can select a range of pages within the AFP documents to be processed. The parameters to use:

<i>INI file</i>	<i>Command line option</i>
StartingPage	-pp:[fp][-tp] (Process from/to pages)
EndingPage	

Processing AFP Indexes

In the following, we describe the standard processing of AFP index data with AFP2web. If you want to customize AFP2web standard processing, please refer to the description of the Scripting Facility.

To invoke the AFP2web standard index processing, you use the option **FileCreationMode=DOC_INDEX**.

By default, AFP2web reads standard AFP index elements and writes the index data either to the path for output documents as specified by the parameter **OutputFilePath** or to the path specified by **IndexPath**.

If the AFP index is not within the AFP data stream, but instead delivered as an external file, you use the command line option **-xf** to specify the external file.

With the parameter **IndexFormat**, you create index files formatted either as XML or as comma-separated values (CSV).

The following figure shows an example of an index file in XML format:

Standard XML Format for Index Data

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Index SYSTEM "Index.dtd" >
<Index Doctype="PDF" DocName="pdf/xml/INSURE_s3k.0.pdf"
PageCount="3" Size="16726">
  <PageGroupIndex PageGroup="00000001">
    <Data>
      <Name>Insured</Name>
      <Value>Geoffrey R Stephens</Value>
    </Data>
    <Data>
      <Name>Policy</Name>
      <Value>324-1443255-11</Value>
    </Data>
  </PageGroupIndex>
  <PageIndex Page="00000001">
    <Data>
      <Name>Contents</Name>
      <Value>Disability Income Policy</Value>
    </Data>
  </PageIndex>
</Index>
```

(additional lines not shown here...)

```
<PageIndex Page="00000003">
  <Data>
    <Name>Contents</Name>
    <Value>Definitions</Value>
  </Data>
</PageIndex>
</Index>
```

If you select CSV as the format for index files, you can use the **IndexRecord** parameter to add meta information to each index entry.

For example, IndexRecord=PAGE_GROUP_NAME, PAGE_NAME,PAGE_COUNT,FILE_NAME,FILE_TYPE,INDEX,FILE_SIZE will add these fields as meta information to each index entry in this order.

The following figure shows an example of index file in CSV format:

Standard CSV Format for Index Data

```
PAGE_GROUP_NAME, PAGE_NAME, PAGE_COUNT, FILE_NAME, FILE_TYPE, INDEX, FILE_SIZE
00000001, , 3, pdf/csv/INSURE_s73.0.pdf, PDF, Insured=Geoffrey R Stephens, 16726
00000001, , 3, pdf/csv/INSURE_s73.0.pdf, PDF, Policy=324-1443255-11, 16726
, 00000001, 3, pdf/csv/INSURE_s73.0.pdf, PDF, Contents=Disability Income Policy, 16726
, 00000002, 3, pdf/csv/INSURE_s73.0.pdf, PDF, Contents=Policy Schedule, 16726
, 00000002, 3, pdf/csv/INSURE_s73.0.pdf, PDF, Contents=Policy Schedule, 16726
, 00000002, 3, pdf/csv/INSURE_s73.0.pdf, PDF, Contents=Table of Benefits, 16726
, 00000003, 3, pdf/csv/INSURE_s73.0.pdf, PDF, Contents=Definitions, 16726
```

The parameters used for creating index files:

<i>INI file</i>	<i>Command line option</i>
InputFormat=AFP	
OutputFormat=PDF	-PDF
FileCreationMode=DOC_INDEX	-DOC_INDEX
IndexPath	-xp -xf
IndexFormat=CSV XML	
IndexRecord=fieldlist	

Storing Output Documents in Sub Folders

The number of resulting output documents might be too large for your file system to handle efficiently when all output documents are stored into one folder (directory).

For this case, AFP2web provides the command line option **-dc** (resp. the INI parameter **DocumentCount**), which you can use together with the option **DOC_COLD** or **DOC_INDEX**. You use the command line option **-dc** to create new sub folders for your output documents and to limit the number of documents per sub folder.

Example: The command line option **-dc:500** together with **-doc_cold:xml** limits the number of output files to 1000 per sub folder: You receive per sub folder 500 documents plus 500 index files in XML format (This is the conversion results of 500 documents).

Important: AFP2web neither prevents overwriting documents, which already exist, nor does AFP2web check the number of documents, which already exist in the output folder.

We therefore recommend:

- Please use AFP2web with the option **-dc** only to process one single large input pool file.
- Use **-dc** only to create new sub folders for your output. Do not use **-dc** to produce output in existing folders.

Example of output for the option -dc:500 and -doc_index:

```

Folder    ... \pdf
<DIR>          00
<DIR>          01

Subfolder  ... \pdf\01
insure_s1eo.1.pdf
insure_s1eo.1.xml
insure_s1eo.2.pdf
insure_s1eo.2.xml
insure_s1eo.3.pdf
insure_s1eo.3.xml
...
insure_s1eo.500.pdf
insure_s1eo.500.xml

Subfolder  ... \pdf\01
insure_s1eo.501.pdf
insure_s1eo.501.xml
...
```

Processing Included Objects (Included Images)

Non-AFP data (mostly images such as logos and colored images) are either embedded in or referenced by AFP object containers, called Included Objects.

AFP2web strives to optimize conversion results and the output size of the object:

- Color objects are stored in JPEG format (JPEG is a lossy format). You use the INI parameter **JPEGQuality** (or the command line option **-jq**) to increase or decrease the compression.
- Black/white objects are stored TIFF G4 format.

The Formats Supported, How AFP2web Finds the Object Data

The object data can be either in-line or in external files.

To load and process an external object file, AFP2web must know its file name and file type. AFP2web derives this information from the AFP data stream.

The object types supported and the file extensions AFP2web expects are:

<i>Object Type (Supported Formats)</i>	<i>File Extension</i>
TIFF (G4, G3, LZW, Packbits, Uncompressed, JPG, CCITRLE, CCITRLEW, IT8CTPAD, IT8LW, IT8MP, IT8BL, PIXARFILM, PIXARLOG, DEFLATE, ADOBE_DEFLATE, THUNDERSCAN, DCS, NEXT, SGILOG, SGILOG24)	tif
JPEG	jpg
Windows Bitmap	bmp
PCX	pcx

For an object container referencing an external object, the file name is a combination of:

<Object Name in SFI>.<File Extension>

Example: If the AFP stream has an included Object MOC1 of type TIFF, AFP2web searches for the file name: MOC1.tif

After building the object container file name, AFP2web searches for external AFP resources in the following order:

1. Folder specified in **ResPath** (default is samples/resource)
2. Folder of the input files
3. In the current path.

Optimizing the Conversion

By default, AFP2web loads and processes the object data in memory. If the AFP spool file contains especially large images, which are only occasionally used, there is no reason for holding this object data in memory.

For this case, AFP2web offers the INI file parameter **SaveOCDataOnDisk** (respectively the command line option **-sod**). This option specifies that an included object (the graphic file) be temporarily stored on disk instead of being loaded and processed in memory.

The temporary files for object container data are stored in the path specified by **TempPath** (the default path is the temporary path defined in the system, the corresponding command line option is **-tp**).

AFP2web removes these temporary files when cleaning up.

Notes on Barcode Support

AFP2web supports the Code 128 barcode (Character Set B of Code 128).

AFP2web also supports the bi-dimensional DataMatrix 2D barcode. With respect to the symbol size:

1. All square sizes having one data region are supported. Square sizes having more than one data region are not supported.
2. All rectangular symbol sizes are supported irrespective of the number of data regions.

The following table gives more details about the support for the DataMatrix 2D barcode. The number in parantheses stands for the number of data regions for that size.

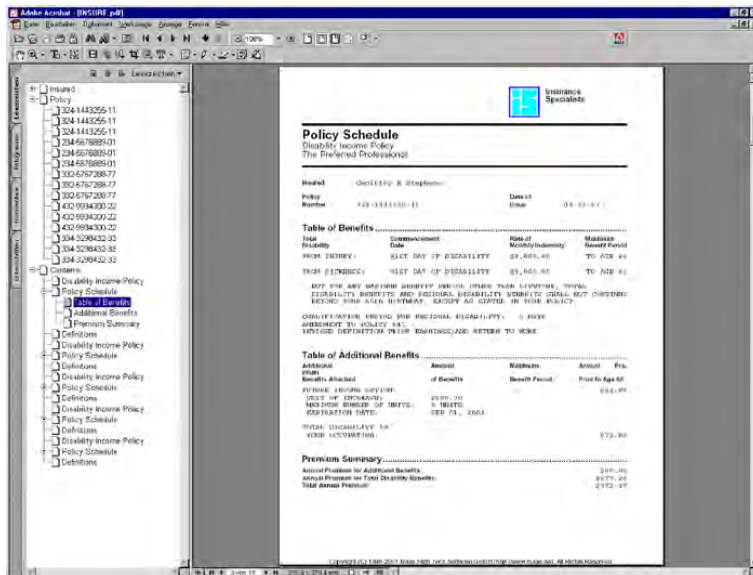
1a. Supported rectangular symbol sizes	8x18 (1)
	8x32 (2)
	12x26 (1)
	12x36 (2)
	16x36 (2)
	16x48 (2)
2a. Supported square sizes (These sizes have one data region.)	10x10
	12x12
	14x14
	16x16
	18x18
	20x20
	22x22
	24x24
	26x26

2b. Square sizes not supported (These sizes have more than one data region.)	32x32 (4)
	36x36 (4)
	40x40 (4)
	44x44 (4)
	48x48 (4)
	52x52 (4)
	64x64 (16)
	72x72 (16)
	80x80 (16)
	88x88 (16)
	96x96 (16)
	104x104 (16)
	120x120 (32)
	132x132 (32)
	144x144 (32)

Using **PDFBookmark** (or by using the command line option **-bm**), you create PDF bookmarks for index elements in the resulting PDF files.

The following figure shows an example of the bookmarks as they are displayed in the Acrobat Reader.

PDF Bookmarks for Index Elements



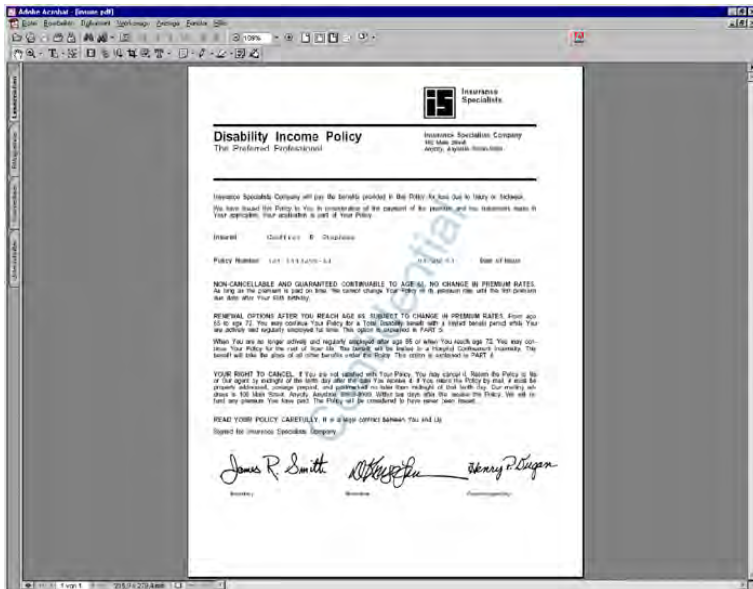
The parameters for creating PDF bookmarks:

<i>INI file</i>	<i>Command line option</i>
InputFormat=AFP	
OutputFormat=PDF	-PDF
PDFBookmark	-bm

Adding Text as Watermark Behind the Document Text

Using the INI file parameter **Watermark** or the command line option **-wm**, you can embed a text as watermark behind the document text, as shown in the following example:

Text Added as Watermark on a Document Page



Stopping Conversion When a Required Resource Is Missing

Using the command line option **-Strict**, AFP2web will stop processing if a required AFP resource cannot be found. The resources AFP2web checks for included Page Segments, Overlays, and Formdefs, AFP font resources and non-AFP resources.

This function is useful if you cannot allow such failures during conversion.

The parameter:

<i>INI file</i>	<i>Command line option</i>
Strict=on	-Strict

Using the Log File

Creating the Log File

Name of the log file: If logging is activated, AFP2web creates a log file for each AFP file. The name of log file is the filename and extension of input file with the additional extension ".log" added to it. For example, the input file INSURE.AFP will be converted to INSURE.PDF with the respective log file INSURE.AFP.log.

The log file path: Use the parameter **LogPath** in the INI file to set the target folder for log files. You override the INI file if you use the command line option **-lp** when calling the program **afp2web**.

Using the log file for font information: We recommend using the log file to investigate font information. You do this either: Globally in the INI file with the parameter **Logging-Font=on**, or once when calling the program using the command line option **-lf**.

Additional options: You can add more information to the log file, which is helpful for the AFP expert and AFP2web support incidents. To add more information to the log file, use:

- The **afp2web.ini** parameter **Logging=on**, or **LoggingLevel**, or **ExceptionLoggingLevel**
- The command line option **-l**, or **-ll**.

Setting the ExceptionLoggingLevel

By default, AFP2web will issue exception messages with no further details. You can add more information to exception messages with the parameter **ExceptionLoggingLevel**.

Structure of the Log File

The LOG file has two parts:

Parsing section	<p>showing the AFP parameters found while parsing the AFP data stream. Here you might find messages indicating whether resource files were found or not. The parsing section ends with an end-of-file message.</p> <p>This part is intended for, and useful for the AFP expert and AFP2web Support.</p>
Builder section	<p>which traces font substitution. If you want to use the log file, we recommend that you use the builder section to identify problems with font substitution.</p>

The following table indicates which information is displayed in the log file:

<i>Parsing-Section Record Structure</i>	
Column 1 - 10:	File offset of the Structured Field Introducer (SFI) in Hex format. The SFI can be taken as the MO:DCA-command.
Column 14 - 21:	The SFI in Hex format.
After Column 23:	The parameters found in the SFI.

<i>Builder-Section Record Structure</i>	
PDF page number	Example: Writing Page 1
Font (Character Set, Code Page)	Example: ==>CS=C1H400B0, TF=HELVETICA LATIN1, W=B, S=12.00, RF, CP=T1Gl0361(MD:2065.cp)
Font used and the text with text position	Example: -->UsedFont=C1H400B0, TF=F1, S=12.00, T3 @ (8496,862)>Insurance< @ (8496,1120)>Specialists<

Chapter 3: Handling Fonts

In this chapter, we describe each font processing mode and show a few examples. But before we do this, we give a short introduction to the configuration files which are used to control font processing.

Note: *For the sake of simplicity, we say input fonts when we talk about fonts found in the input documents, and output fonts to refer to the fonts used by the output documents.*

Fonts are either embedded or available in line in the documents or are referenced as external fonts. Fonts, which are used, must be available to AFP2web.

3.1 The Configuration Files and the Defaults Used for Font Processing

The configuration files used for fonts are:

- the INI file (afp2web.ini)
- the font mapping file (mapping.def)
- code page files (CP files)

If AFP2web cannot find a specified font, then AFP2web will use a fallback font to produce conversion results.

The INI File (afp2web.ini)

The INI file is the configuration file for AFP2web processing. If a command line option is entered, it will replace the corresponding INI parameter.

You can find a detailed description of each parameter in the INI file in "*Overview of INI Parameters and Command Line Options*" on page 111 and "*INI File Parameters (afp2web.ini)*" on page 119.

The following table lists the INI parameters which are relevant to font processing.

<i>INI Parameter</i>	<i>Command line option</i>
Parameters to print a log with font information:	
LoggingLevel=FONT	-ll:FONT or -lf
LoggingLevel=ALL	-ll:ALL
LoggingLevel=FNTR	-ll:FNTR
Processing flag used to stop processing when resources are missing:	
Strict=on	-Strict
Parameters used to specify locations of AFP font resources and substitution fonts:	
AFPFontPath=<path>	-ap
Named sections for FTP locations	
ResPath=<path>	-rp
ExtFontPath= <path>	-fp
Parameters used to specify naming conventions for AFP resource members:	
CharSetExt=<file extension>	
CodedFontExt=<file extension>	
CodePageExt=<file extension>	
Character mappings for code pages	
CodePage= <defaultcodepage>	-dcp
CpPath=<path>	-cp
CodePageDefaultChar = " "	

The Font Mapping File (mapping.def)

The *mapping.def* is the font processing configuration file.

AFP2web offers a standard processing, which uses the original fonts whenever possible. It is therefore basically possible for you to run AFP2web without modifying the *mapping.def*. You normally use the *mapping.def* to fine-tune font processing for your type of documents.

You use the *mapping.def* to:

- define the font processing mode for each character set or font.
- specify substitution fonts for a target format such as AFP, PDF, a raster format such as TIFF on either Windows or Unix systems.
- map character sets to code page files.
- specify file name suffixes to help AFP2web find the font files. For example, if you use the font "ARIAL, Italic" and your font file is named *Arial-Italic.ttf*, then you can specify that italic fonts use font file names with the suffix "-Italic".
- describe AFP fonts (character sets, code pages, font global IDs). For example, character sets and encoding in the AFP data stream might be out-dated, or no longer visible in your current AFP resource pool. In this case, you can define AFP font attributes manually in the AFP-specific sections of the *mapping.def*.

For detailed information on the syntax and parameters of this configuration file, please refer to "*Parameters of the mapping.def*" on page 159.

Code Page Files

Code page files (CP files) define the font encoding to use. They map EBCDIC to ASCII and Unicode code points. ASCII is the encoding, which the AFP2web Scripting Facility uses to identify objects in the AFP data stream.

AFP2web delivers a set of predefined code page files, which are commonly used and which can be found in the directory of the mapping.def file.

For more details about code page files, please refer to *"Code Page Files (CP Files)" on page 175*.

Default Assumptions If Mappings or Font Resources are Missing

If neither the input files nor any of the above configuration files deliver the font resources and mapping information, AFP2web adheres to a default processing.

Please note that in this mode, AFP2web cannot guarantee true fidelity of the conversion results.

For example, font characteristics can be derived from the naming conventions for these resources, if these follow the IBM font naming convention. Thus, AFP2web will still be able to map AFP fonts to substitution fonts. The rules for this automatic font mapping is described in detail in *"Identifying Fonts Using the IBM Naming Convention"* on page 185.

3.2 AFP2web Font Processing Modes

AFP2web offers the the following font processing modes:

- Use of the original fonts (the default processing)
- Font substitution and referencing
- Font substitution and embedding

Use of the Original Fonts (Default)

Normally, no special configuration is required for AFP2web's font handling. If the font resources are available either in-line or as external resources, the default behavior of AFP2web is to take the original fonts to produce the conversion results.

The Default Conversion of AFP Documents

General Information

When converting AFP documents, AFP2web uses the original fonts:

- For output to PDF, AFP2web converts fonts. An AFP outline font is converted to a Type 1 font. An AFP raster font is converted to a Type 3 font. In this mode, AFP2web always embeds the fonts in PDF.
- For output to TIFF, AFP rasterizes fonts.

If the AFP fonts are not provided in line in the AFP spool, they must be available as external font resources.

Note: *AFP Outline Fonts of type "CID-Keyed" are not supported by AFP2web Version 3.x.*

Example: AFP2PDF and Charset Rendering=0 (Font Rasterizing)

The input format AFP is to be converted to PDF. The original font of the AFP Character Set C1H400B0 is to be directly converted and embedded in the PDF document.

The configuration file afpcp\mapping.def specifies that all fonts be rasterized. This is the default processing of AFP2web:

```
[CHARSET RENDERING]
*=0
```

The following command starts the conversion:

```
afp2web.exe -lf -lp:pdf -c -q samples\insure.afp
```

The conversion result is a PDF document. Our example embeds fonts and therefore exceeds 100 kB in file size.



The information in the LOG-file:

```
==>CS=C1H400B0, FGI D=2305, TF=HELVETICA LATIN1, W=B, S=12.00, RF,
CP=T1GI 0361
-->UsedFont=C1H400B0, TF=F1, W=B, S=12.00, RF, ACP=MD: 2065.cp,
ENC=ANSI
  @(8496, 862)>Insurance<
  @(8496, 1120)>Specialists<
```

LOG Text	Description
HELVETICA LATIN1	The name of the font according to the AFP definition is "HELVETICA LATIN1"

<i>LOG Text</i>	<i>Description</i>
C1H400B0	The input font in AFP, the name of the AFP Character Set. AFP2web uses the original AFP font for the font rasterizing.
F1	For the PDF document, a PC font is generated and embedded with the typeface name F1 . In the PDF document, it can be identified as a Type 3 font.
W=B	Weight is "Bold"
RF	Raster Font. The font has been embedded into the PDF document as a raster font.

The Default Conversion of PDF Documents

General Information

When converting PDF documents to AFP, the standard processing of AFP2web is:

- The fonts in PDF are rasterized and inserted as IOCA images in the target AFP document.
- A Type 3 font in the PDF file will be converted to an AFP raster font and will be delivered as a font resource in the conversion results.

Note: *Fonts having Identity-H or Identity-V encoding, Custom encoding or Built-In encoding are always rasterized*

Example: PDF2AFP With CharsetRendering=0,1 (Default)

We want to convert PDF to AFP and rely on the default processing.

Default processing is to rasterize PDF fonts and inserted these as images into the resulting AFP document. No AFP font resources are produced. This processing is equivalent to the settings in the mapping.def:

```
[CHARSET RENDERING]
*=0, 1
```

The following command in the demo.bat starts the conversion:

```
afp2web.exe -q -c -afp -lf -lp:pdf samples\insure.pdf
```

The following LOG-file information shows that an embedded font is rasterized:

```
===>TF=Courier, W=M, S=10.00, OF,
-->UsedFont=extfont/courier.ttf, TF=Courier, W=M, S=10.00, RI
@(4896,10706)>Geoffrey R Stephens<
```

LOG Text	Description
Courier	The TrueType font in the PDF file
UsedFont	Indicates that the font is unpacked to a temporary folder
RI	Font is inserted as Raster Image

Font Substitution and Referencing

In cases, where it is not preferable to use the original font, you can specify that a substitution font replace the original font. Substitution fonts might be available and produce satisfactory results. You use font referencing to reduce the file size of the output document.

Converting AFP Documents with Substitution Fonts

General Information

You use the mapping.def to specify that an AFP font be replaced by a PC font:

- The section **[CHARSET RENDERING]** specifies the processing mode for each AFP Character Set.
- For the conversion to PDF, the section **[PDFFONT]** maps AFP Character Sets to the substitution font.
- For the conversion to a raster format (such as TIFF, JPEG), the section for the system platform, either **[WINFONT]** or **[UNIXFONT]**, maps AFP Character Sets to substitution fonts.

If the AFP fonts are not defined in-line in the input spool file, you must make the external font resources available to AFP2web.

Additional entries in the mapping.def determine which substitution font to use:

1. The AFP data stream delivers the base name for the font.
2. The mapping.def maps the font name to the typeface name.
3. The mapping.def section **[FONTSUFFIX]** determines the suffixes to add for the font attributes (bold, italic).
4. The font attributes are either taken from the AFP font resource / definition or from the settings in mapping.def section **[FGID]**.

AFP2web supports the following formats for substitution fonts:

- TrueType,
- OpenType (ONLY as Single True Type Font with the file extension ".ttf"),
- Type 1

Example: AFP2PDF and CharsetRendering=1 (Font Substitution and Referencing)

The input format is AFP and shall be converted to PDF.

The configuration file afpcp\mapping.def specifies that all fonts be replaced and referenced. If AFP2web cannot find the PC font in the system, then AFP2web will use a default substitution font. It is very important that the PC fonts are available to AFP2web.

The font processing mode in the mapping.def (replace and reference):

```
[CHARSET RENDERING]
*=1
```

The following command starts the conversion:

```
afp2web.exe -lf -lp:pdf -c -q samples\insurance.afp
```

The result is a PDF document, which is significantly reduced in file size, because fonts are only referenced. Our example is just about 45 kB in file size.

The information in the LOG-file shows that HELVETICA LATIN1 has been mapped to the PC font Helvetica-Bold:

```
==>CS=C1H400B0, FGID=2305, TF=HELVETICA LATIN1, W=B, S=12.00, RF,
CP=T1GI0361
-->UsedFont=Helvetica-Bold, TF=Helvetica-Bold, W=B, S=12.00, SF,
ACP=MD: 2065.cp, ENC=ANSI
  @(8496,862)>Insurance<
  @(8496,1120)>Specialists<
```

<i>LOG Text</i>	<i>Description</i>
C1H400B0	AFP character set with the typeface name HELVETICA LATIN1
UsedFont	The font used is Helvetica-Bold
SF	Standard font

The following information in the LOG-file shows that AFP2web could not find a PC font file in the system. A default font must therefore be used. A warning is issued and the default font used is indicated by the code "D":

```
==>CS=C1D0GT10 TF=G0THIC Warning! External font file not found.
==>CS=C1D0GT10, FGID=40, TF=G0THIC, W=M, S=10.00, RF, CP=T1D0BASE
-->UsedFont=Helvetica, TF=Helvetica(D), W=M, S=10.00, SF,
ACP=MD: 2063.cp, ENC=ANSI
  @(2448,5353)>Geoffrey R Stephens<
```

<i>LOG Text</i>	<i>Description</i>
Warning!	For the character set C1DOGT10 (Typeface GOTHIC), an external font file could not be found.
UsedFont	The font used is Helvetica
D	Default font

When using substitution fonts, it is advisable to check the conversion results. The original text string “Geoffrey R Stephens”, for example, uses a mono-spaced font in AFP:

Geoffrey R Stephens

The conversion result in PDF uses a proportional font because the GOTHIC typeface has not been mapped in the mapping.def:

G e o f f r e y R S t e p h e n s

We see that a conversion result might not be acceptable in all cases and that adjustments might be necessary.

Example: AFP2PDF and CharsetRendering=1 (Exactly Specifying the Substitution Font)

The input format AFP is to be converted to PDF.

If AFP2web cannot find a PC font in the system, AFP2web then uses a default. This is what we want to prevent.

The following definition in the configuration file afpcp\mapping.def specifies that the AFP Character Set C1D0GT10 be replaced by a substitution font and that this font shall be referenced in PDF. For all the other fonts, AFP2web is to use the original fonts:

```
[CHARSET RENDERING]
C1D0GT10=1
*=0
```

In the appropriate section of the mapping.def, we also specify which substitution font to use for the target format PDF. The file name of the font is **lucon.ttf** for Lucida Console. Because there is no global naming convention for fonts, AFP2web uses file names to find a PC font. We therefore assume that the typeface name is part of the file name and we specify the base part of the file name in the mapping.def:

```
[PDFFONT]
;
; user defined
GOTH1 C=lucon
```

Our example specifies the font path as **ExtFontPath=c:\WINDOWS\Fonts** and uses the font file lucon.ttf.

The following command starts the conversion:

```
afp2web.exe -lf -lp: pdf -c -q samples\insure.afp
```

The information in the LOG-file shows that the font mapping has taken place. The AFP Character Set C1D0GT10 has been mapped to the typeface "Lucida Console":

```
==>CS=C1D0GT10, FGID=40, TF=GOTHIC, W=M, S=10.00, RF, CP=T1D0BASE
-->UsedFont=C:/WINDOWS/Fonts/lucon.ttf, TF=LucidaConsole, W=M,
S=10.00, TT, Ref, ACP=MD:2063.cp, ENC=ANSI
@(2448,5353)>Geoffrey R Stephens<
```

LOG Text	Description
TF=GOTHIC	The AFP font has the typeface name "GOTHIC"
UsedFont	The font file used as substitution font. Its typeface name "LucidaConsole" is taken from the font file "lucon.ttf"
Ref	Indicates that the font is referenced in the PDF file

The conversion result has improved . The text string “Geoffrey R Stephens” looks like this in the original AFP:

A screenshot of the text "Geoffrey R Stephens" rendered in a blue, bold, serif font. The letters are slightly irregular and have a classic, formal appearance typical of AFP fonts.

and is now mapped to a more appropriate mono-spaced font in the resulting PDF:

A screenshot of the text "Geoffrey R Stephens" rendered in a blue, mono-spaced font. The letters are more uniform in width and height compared to the original AFP font, giving it a more modern, digital appearance.

Converting PDF Documents with Substitution Fonts

General Information

You use the mapping.def to specify that a PC font be replaced by an AFP font:

- The section **[CHARSET RENDERING]** specifies the processing mode for each PC font typeface.
- For the conversion to AFP, the section **[AFPFONT]** maps PC fonts to AFP Character Sets and Code Pages.
- For the conversion to a raster format (such as TIFF, JPEG), the section for the system platform, either **[WINFONT]** or **[UNIXFONT]**, maps fonts to substitution fonts.

When converting from PDF to AFP, the following applies:

- Embedded fonts are used as is. Referenced fonts must be available as external fonts. AFP2web searches for external fonts, as described below.
- When searching for a regular font, AFP2web searches for the full name of the font. For example: If PDF font name is "Arial", then AFP2web will search for a PC font "Arial.*" which will match with the file name "Arial.pfb" or "Arial.ttf"
- If a font name has special character like ', ' then the font file name must also include this character. For example, if the the font name is "Arial,Bold" then font file name should be "Arial,Bold.ttf"
- To identify a font subset, AFP2web ignores the first 7 characters of the font name. For example: If PDF font name is "NHAHHE+Arial", then AFP2web will search for "Arial.*" which will match "Arial.pfb" or "Arial.ttf".

Example: PDF2AFP With CharSetRendering=1

The following example specifies that all fonts with typeface names beginning with "Courier" are to be replaced with their appropriate substitution fonts:

```
[CHARSET RENDERING]
Courier*=1
```

If the RenderingFlag 1 is set, then you must list the substitution fonts to use. If necessary, you add your specific entries to the section **[AFPFONT]**. The following shows the entry for the "Courier":

```
[AFPFONT]
Courier=CZ4200, T1001141
```

The AFP substitution fonts, which you use, must be available as font resources (either as

Coded Font or as AFP Character Set and Code Page). Font resources must be found in the predefined resource paths. You define the resource paths in the INI-file, either as the general path for resources (**ResPath**) or as a path to AFP font resources (**AFPFontPath**).

The following command in the demo.bat starts the conversion:

```
afp2web.exe -q -c -afp -lf -lp: pdf samples\insure.pdf
```

The following lines in the LOG-file indicate that the font substitution has taken place. Arial,Bold is mapped to the character set CZH400:

```
==>CS=CZ4200, TF=Courier, W=M, S=10.00, OF,  
-->CS=CZ4200, TF=Courier, W=M, S=10.00, OF, , ACP=PG-DEF: 500. cp,  
ENC=WinAnsiEncoding  
@(509,1115)>Geoffrey R Stephens<
```

<i>LOG Text</i>	<i>Description</i>
CZ4200	The character set used in the AFP document
OF	Outline Font

Font Substitution and Embedding

In cases, where it is not preferable to use the original font, you can specify that a substitution font replace the original font. You choose to embed the font to make the target document self-contained, for example for archival.

Converting AFP Documents to PDF with Substitution Fonts

For general information on specifying substitution fonts for AFP, please refer to ["Converting AFP Documents with Substitution Fonts" on page 77.](#)

Example: AFP2PDF and CharSetRendering=2 (Mapping AFP GOTHIC to Windows Gothic)

The input format is AFP and the output format is PDF. AFP2web automatically associates the AFP font with the typeface name GOTHIC to the Windows font with the file name "Gothic.ttf" .

The configuration file afpcp\mapping.def specifies for the character set that a substitution font is to be used and embedded in PDF. All other fonts are to be rasterized:

```
[CHARSET RENDERING]
C1D0GT10=2
*=0
```

We do not need to specify a mapping in the mapping.def, if we can expect AFP2web to find the font file in the system. Therefore, we do not add definitions to the following section:

```
[PDFFONT]
;
; user defined
```

AFP2web will find the font if the following is true:

- The parameters of the INI-file (**ExtFontPath**) specifies the path to font files.
- The font file adheres to the **naming convention** (typefacename=filename.*).

Note: If the font has a weight of “Bold” or a style of “Italic” then the filename must have an appropriate suffix. For example, when searching for a font “GOTHIC,Bold”, AFP2web builds the file name to look for using the typeface name (GOTHIC) concatenated with the first suffix defined in the [FONTSUFFIX] section of the mapping.def. If no matching font file name (in our case, GOTHICb.*) is found, AFP2web builds a new file name using the second suffix (GOTHIC-Bold.*) and so on.

In the mapping.def we define file name suffixes for bold and italic fonts as follows:

```
[FONTSUFFIX]  
Bo l dSuffi x=b, bd, -Bo l d  
I tal i cSuffi x=i, -I tal i c, -Ob l i que  
Bo l dI tal i cSuffi x=bi, -Bo l dI tal i c, -Bo l dOb l i que
```

Our example uses **ExtFontPath=c:\WINDOWS\Fonts** and the font file GOTHIC.ttf.

The following command starts the conversion:

```
afp2web.exe -lf -lp:pdf -c -q samples\insure.afp
```

The information in the LOG-file:

```
==>CS=C1D0GT10, FGID=40, TF=GOTHIC, W=M, S=10.00, RF, CP=T1D0BASE  
-->UsedFont=C:/WINDOWS/Fonts/GOTHIC.TTF, TF=CenturyGothic, W=M,  
S=10.00, TT, Emb, ACP=MD: 2063. cp, ENC=ANSI  
@(2448, 5353)>Geoffrey R Stephens<
```

LOG Text	Description
TF=GOTHIC	The AFP document character set C1D0GT10 has the typeface name GOTHIC.
UsedFont	The font is correctly mapped to the specified font file
Emb	Indicates that the font is embedded in the PDF file

The text string “Geoffrey R Stephens” in the original AFP is:

Geoffrey R Stephens

The conversion result in PDF is:

Geof f r e y R S t e p h e n s

A font substitution cannot guarantee true fidelity of the conversion results, as can be seen in this example. We also assume that a mono-space font would be more appropriate as a substitution font.

In this example, we see that it is better to explicitly specify the substitution font and not to rely on an automatic search for font files.

We can improve the conversion results, by specifying a mono-spaced substitution font:

```
[CHARSET RENDERING]
C1D0GT10=2
*=0
```

```
[PDFFONT]
;
; user defined
GOTHIC=COUR
```

The information in the LOG-file:

```
==>CS=C1D0GT10, FGID=40, TF=GOTHIC, W=M, S=10.00, RF, CP=T1D0BASE
-->UsedFont=c:/WINDOWS/Fonts/cour.ttf, TF=CourierNewPSMT, W=M,
S=10.00, TT, Emb, ACP=MD:2063.cp, ENC=ANSI
@(2448,5353)>Geoffrey R Stephens<
```

The text string in the PDF file:

Geoffrey R Stephens

Chapter 4: AFP2web Scripting Facility User Guide

This chapter explains how to use the AFP2web Scripting Facility to customize standard AFP2web processing. The information includes:

- Basic concepts and suggestions for what you can do,
- An introduction to the parsing events, packages, and methods

For a detailed information on syntax and commands, please refer to *"Scripting Facility API Reference"* on page 207 and *"Scripting Facility Quick Reference"* on page 191.

Script examples are delivered with the AFP2web product. You can use these examples as a starting point for your own custom scripts. For a description of these examples, please refer to the appendix.

4.1 Basic Concepts of the AFP2web Scripting Facility

Scripting Facility Applications

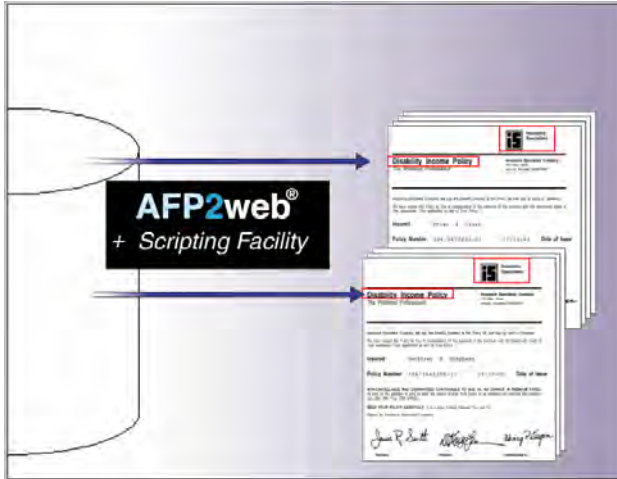
The Scripting Facility is the AFP2web interface used to customize AFP2web document parsing and conversion. When the Scripting Facility is activated, AFP2web passes control to a user-defined script at particular processing events. The Scripting Facility enables intelligent document processing. Examples of tasks performed by a script include:

- Splitting an AFP spool file into individual output documents and pages according to rules, which you define and which are not limited to any standard AFP structure. If your AFP data contains the standard AFP indexes as described elsewhere in this chapter, AFP2web splits AFP spool files automatically. If you are coping with different requirements, the Scripting Facility gives you all the flexibility you need to control this process.
- Extracting data from AFP document content to create detailed document indexes. Here again, AFP2web offers a standard method of extracting data from standard AFP indexes and delivers index data in basic formats. However, if the AFP format itself comes in application-specific variations or if you need more information, you use the Scripting Facility to extract index data from any object in the AFP data stream and to deliver index data in any required format.
- Adding, changing or removing document content, such as text, watermarks, bookmarks, background forms, images, indexes, annotations, and other elements. If you need to process, re-purpose, or enhance your documents, you use the Scripting Facility to do the job.
- Trigger any pre- or post-processing action for the current page or document. For example, you might want to forward document indexes to the archive system. Instead of having the archive polling for any incoming data, you can thus actively trigger an archive update.
- Acting as a plug-in mechanism for easy integration into existing applications. AFP2web works as a functional unit in document processing chain. Together with the Scripting Facility, AFP2web produces results for follow-on processes (converted documents, index data) in one single job step and thus helps streamline the workflow.

The possibilities of using the AFP2web Scripting Facility is basically unlimited. In the following sections, we give a few examples of typical application scenarios using AFP2web and the Scripting Facility.

Document Recognition and Separation

Define rules for splitting an AFP spool file into individual documents and document pages.



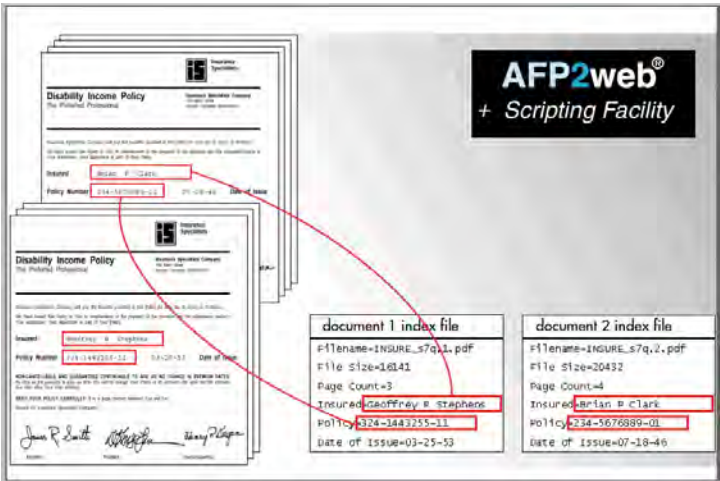
A script formulates rules for recognizing documents and pages. These rules can be based on printable and non-printable information in the AFP spool file, for example, printed text, the occurrence of overlays and other objects, and encoding values passed through AFP-specific NOP fields.

Document recognition and separation is run as a full automated process.

Example: An AFP data stream contains a spool file with insurance bills for many customers. To split this spool file for each customer, we can look for a particular text string (eye catcher) on that page. Finding this eye catcher, we mark this page as the beginning of a new document. In addition, we capture the name of the customer for our index.

Data Extraction and Document Indexing

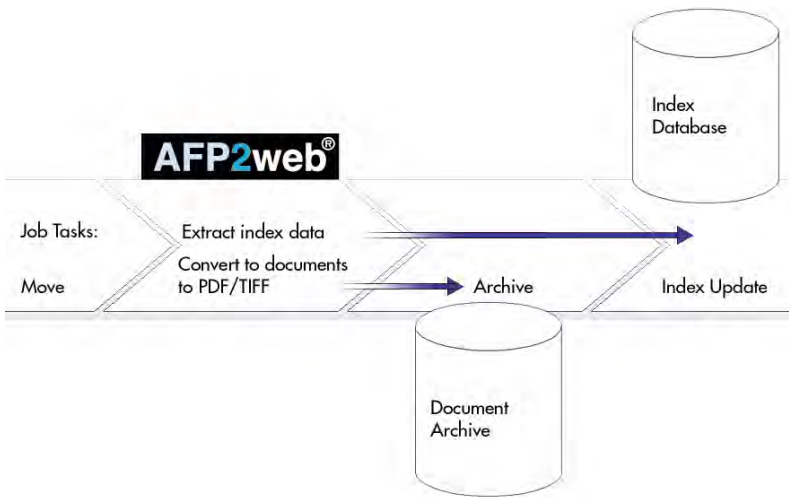
Extract visible and non-visible information from AFP documents. Deliver index data in any format, for example, CSV or XML.



Using Scripting Facility functions, a custom script is designed to access visible text and non-visible data anywhere within a document. An intelligent extraction of index data is possible. The Scripting Facility runs as a full automated process extracting index data from mass print documents. Index data can be delivered in any required format.

Document Conversion and Archival

The tasks to solve include converting incoming documents, extracting data, delivering converted documents to the archival system, obtaining the resulting document IDs, and finally delivering both document IDs and extracted data to the index database.

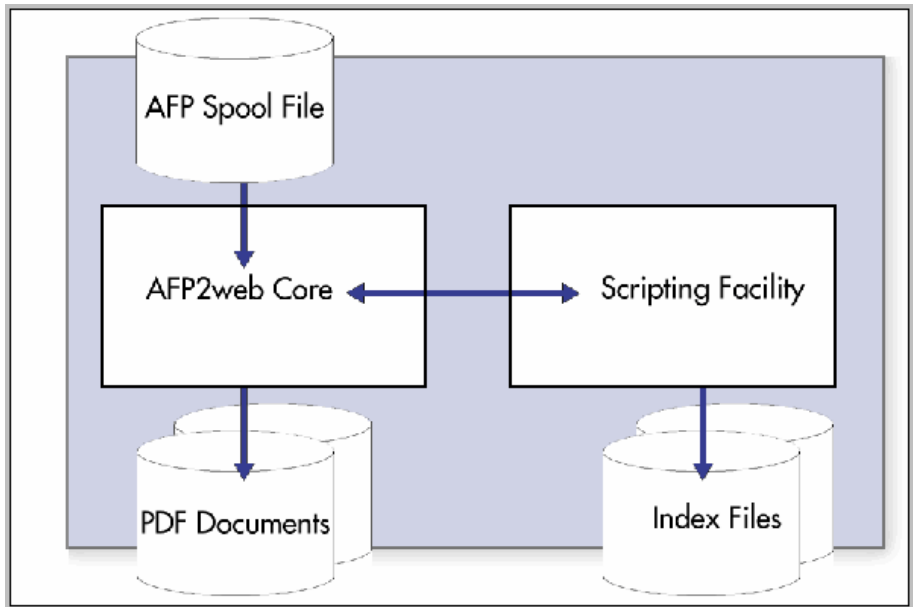


Thanks to the AFP2web core functionality, documents are quickly converted to PDF or TIFF for the archives. Using the AFP2web Scripting Facility, a custom script is designed to access visible text and non-visible data anywhere within a document. The custom script extracts index data of any required level of detail and delivers the index data in any required format.

After document conversion, the script finally intercepts a "finalizeDoc()" event to trigger the next process. This process forwards the converted document to the archives, obtains the document ID, and passes document ID and extracted index data to the index database.

Scripting Facility Interaction With AFP2web

AFP2web interacts with the AFP2web Scripting Facility, giving you the opportunity to intercept particular events, such as initializing, processing, and post-processing either the current document or page.



The AFP2web Scripting Facility is based on Perl. An embedded Perl engine (perl dll) is delivered with the AFP2web installation. To provide additional functionality or to add new modules, you must install Perl.

AFP2web also provides script examples showing how to solve basic tasks. These are described in the Appendix.

The Objects of the AFP2web Scripting Facility

To access all document objects required for your scripting, you include those Perl packages which define the methods for the objects you need.

The following table gives a short overview of the packages, which you can use to read or write object properties:

Scripting Facility Packages Used to Access Objects in AFP2web

<i>Package</i>	<i>Objects Which Are Accessed</i>
a2w::Config	AFP2web processing parameters
a2w::Document	The document
a2w::Font	Font settings
a2w::Index	Set attributes to be used as index data
a2w::Kernel	Exposes additional AFP2web functionality
a2w::Line	AFP line object
a2w::MediumMap	AFP medium map
a2w::NOP	Invisible AFP "No Operation Fields" used to carry data or processing instructions in the AFP data stream
a2w::Overlay	AFP overlay
a2w::Page	Document page
a2w::PSEG	AFP page segment
a2w::Text	Text

The Parsing Events of the AFP2web Scripting Facility

At specific parsing events, AFP2web activates a subroutine in your main script module (the default is `afp2web.pm`). The following chart gives you a quick overview of these parsing events and the correlating subroutines.

Parsing Events of the AFP2web Scripting Facility, Correlating Subroutines, Scopes

<i>AFP2web Kernel</i>	<i>Subroutine in <code>afp2web.pm</code></i>	<i>Scope</i>
Process begin	<code>initialize()</code> (Only once)	global
Document begin detected	<code>initializeDoc()</code>	global
The following subroutines are called in sequence as one logical step for a document page:		
Page begin detected	<code>initializePage()</code>	document
Page parsed	<code>afp2web()</code>	document
Page processed	<code>finalizePage()</code>	document
Document processed	<code>finalizeDoc()</code>	global
Process ending	<code>finalize</code> (Only once)	global

The Required Subroutines in a Script

Each subroutine takes particular objects as input parameters. In a subroutine, you save a reference to the object so that the methods of this object becomes available at a later event. In the following description, we indicate what can be done in a subroutine. For more details, please refer to the reference and to the script examples.

initialize()

This routine is performed only once at the beginning of the AFP2web process. It is used to access and configure run time parameters (INI file or command line options) and to read information about the spool file.

Usage examples:

- Reading the scripting arguments which are entered with the command line option `-sa`.
- Set the `AutoSplit` property to `TRUE` if you want to control document splitting using the standard AFP index.

initializeDoc()

This routine is performed at the beginning of each process preparing to write the output document. Called at the beginning of each output document process, user can initialize output document dependent processes.

initializePage()

This routine is performed at the beginning of each new page. From this point on, you can access the methods of the page.

afp2web()

The main entry point in the AFP2web Scripting Facility.

This routine is performed for each and every page. You use this routine to access any object of the page, for example:

- Retrieve references to all child objects
- Use the object's methods to access the object's properties.
- Formulate the parsing rules for identifying the first page of a new document, for skipping the page, for appending the page to the current document.
- Collect index data for the page.

finalizePage()

This routine is performed for each and every page. Called after processing each output page, user can do page finalizing processes

finalizeDoc()

This routine called after the document has been physically written. At this point, the document conversion is complete (PDF or a raster format, such as TIFF).

Normally, you use this routine to build, format and output the indexes for the document. You can also set the document name, for example with a time-stamp.

Because the output document is available, you can do any post-processing for the document like passing it to your archive.

finalize()

End of AFP2web process and after processing the input AFP spool file. You can use this routine for post processing actions.

Scripting Facility Processing Features

General Features

The Default Encoding

Please note that the Scripting Facility is ASCII-based. ASCII is the encoding used when parsing AFP resource names, AFP index elements, and is used as the base encoding for the Scripting Facility.

The mapping.def determines which code page file to use for an AFP code page. The code page file (CP file) lists the ASCII code points to use for each character.

The Coordinate System Used by the Scripting Facility

The top left is the origin of coordinate system used by the Scripting Facility.

The X value increases the direction to the right.

The Y value increases the direction downwards.

Global vs. Document Scope (Handling Fonts Within a Script)

Please note that a script operates in two different scopes:

Global scope which begins with the initialize() routine and ends with the finalize() routine

Document scope which starts with initializeDoc() and ends with finalizeDoc()

Fonts, which are allocated in the initialize() function, will have a global scope.

Fonts, which are allocated in one of the functions initializeDoc(), initializePage(), afp2web(), or finalizePage(), will have document scope

Important: The following is an issue in Perl - When you declare a font variable, try to avoid using the "my" keyword if you want to access the variable within a global scope.

The Processing of Document Index Data

From the viewpoint of AFP2web, the index data found in an AFP file is divided into two categories:

- Standard AFP indexes
- Non-standard indexes.

Standard AFP Indexes

We assume that you generally understand that elements in the AFP data stream are encoded as structured fields. For more information on AFP and MO:DCA, please refer to the appropriate documentation of IBM. In the following, we name the two elements, which are recognized as standard AFP indexes.

Standard AFP indexes are objects known as Index elements (IEL). Within the AFP data stream, an Index Element (IEL) is located within a document index – within the structured fields Begin Document Index (BDI) and End Document Index (EDI):.

The Tag Logical Element (TLE) provides the attribute name and attribute value, which can be used as an index entry for the page or page group. The TLE may contain an explicit reference to the page or document. A document or a page is generally referred to as an indexed object.

If the AFP spool file contains standard AFP indexes, the AFP2web Scripting Facility is not in all cases necessary to extract index data. AFP2web provides a standard method for processing standard AFP indexes. You activate this standard method if you set the INI-file parameter **FileCreationMode=DOC_INDEX**. Using this option, you create files with index information in one of the standard formats available.

When AFP standard indexes are available, the Scripting Facility is normally used to enhance these with additional data taken from the document, and to deliver index data in a custom format.

Non-Standard Indexes

Your application might require a custom solution for document indexing. Solutions, which are quite common, include:

- Passing document index information via NOP (no operation) structured fields.
- Capturing text (eye catchers) within the page content.
- Detecting specific overlays and capturing text marked by these overlays.

Using the AFP2web Scripting Facility, you can process these elements to collect index data from the document content. You can build new indexes or add information to existing indexes. You can format and output the results according to your own standards.

For example, using the `addIndex()` function of the `a2w::Document` or `a2w::Page` object, you can insert additional indexes at both document and page level. When used together with the command line option `-bm`, AFP2web creates PDF bookmarks for these indexes.

Splitting an AFP Spool File Into Documents and Pages

If the data stream contains standard AFP indexes, AFP2web will be capable of splitting documents automatically.

Automatic Splitting with Autosplit

If you want AFP2web to handle document splitting automatically while running the AFP2web Scripting Facility, you must explicitly set the `autosplit` option to `TRUE`. This is done when initializing in the script for the process.

During automatic document splitting, AFP2web actively triggers the parsing events – the script you have programmed for these events are thus executed automatically.

A small amount of control is, however, still possible, that is, you can set flags to tell AFP2web to:

- Append the page to the current document, or
- Skip the current page.

Script-Controlled Splitting without Autosplit

There are cases, in which it is preferable to use a script to control splitting an AFP spool file into documents and pages. Examples include AFP data without the standard AFP indexes or application specific rules for splitting.

To handle document recognition in your scripts, you turn off the `autosplit` property. In your script, you access and investigate objects in the AFP data stream, such as:

- Text objects in the presentation data (eye catchers),
- Overlays and their text objects,
- NOP fields in the AFP data.

These characteristics are used to identify the type of page being parsed. The script must explicitly set a flag to tell AFP2web that the current page

- Marks the beginning of a new document.

In addition, flags are also set to tell AFP2web to:

- Append the page to the current document, or
- Skip the page.

With autosplit turned off, you must set these flags if you want to AFP2web to trigger parsing events and thus execute the script subroutines.

Exception Handling

In a custom script procedure, you can set any exception by returning a negative value (negative return code, message text). The syntax is:

```
return (rc, "errortext").
```

where rc is a negative value.

When AFP2web encounters a negative return value, processing will stop and the exception is issued as error message.

Example:

```
return ( -123, "afp2web(): missing..." );
```

AFP2web returns a message to the console as follows:

```
E088: Scripting Facility Error (rc=-123): afp2web(): missing...
```


4.2 Using AFP2web Scripting Facility

AFP2web delivers several scripting examples, which you can use as a starting point for your own script procedures.

afp2web.pm is the master template to use. It has all required subroutines implemented and placeholders for basic functions.

Activating the Scripting Facility

You configure AFP2web to run with the Scripting Facility by setting the appropriate parameters in the AFP2web configuration file (INI-file) or you enter the appropriate command line options when you start AFP2web.

The following table summarizes parameters and options for the Scripting Facility:

<i>INI Parameter</i>	<i>Command Line Option</i>	<i>Description</i>
FileCreationMode=DOC_COLD	<code>-doc_cold</code>	<p>Using this option, you activate the AFP2web Scripting Facility.</p> <p>-doc_cold AFP2web will not write any index files. This has to be done by the custom scripting module.</p> <p>You can extend the command line option for AFP standard index elements - or if you used the addIndex method of the Scripting Facility to add standard index elements. In this case, AFP2web will handle formatting and output of index data for you. Use one of the following extensions:</p> <p>-doc_cold:csv AFP2web writes index files in CSV format</p> <p>-doc_cold:xml AFP2web writes index files in XML format</p> <p>Note: If you do not extend doc_cold option, your custom Perl routine is responsible for formatting and output of index data.</p>

<i>INI Parameter</i>	<i>Command Line Option</i>	<i>Description</i>
ScriptProcedure= <scriptFile>	-sp (scriptFile)	<p>The file name of the scripting routine to use. If not specified, AFP2web Scripting Facility will load "afp2web.pm" by default.</p> <p>Example of using the command line option: -sp:c:\afp2web\mysp.pm</p> <p>Note: If you call AFP2web.exe from a directory other than the AFP2web installation base, you must specify the absolute path to the main scripting module using the option -sp.</p>
ScriptArgument=parm1,parm2, ...	-sa:parm1,parm2, ...	<p>Using this option, you can pass any number of arguments to the AFP2web Scripting Facility.</p> <p>In the scripting routine, you use the method getScriptArgs() of the a2w::Config object to access these arguments as one single string and you parse this string according to your own rules (delimiter, keywords, etc.).</p>
Autosplit		<p>AFP2web will look for standard AFP index elements (TLEs) to recognize document boundaries. If Autosplit is set to TRUE the \$NEWDOC flag will have no effect.</p>
SkipPage, SkipObjectSize		<p>These INI parameters specify pages to ignore in the AFP spool file. The criteria you can set is a maximum byte size of a page's data stream. All pages which are smaller or equal to this size will be ignored. You can use these parameters to skip blank pages or pages with a standard content (for example: separator pages).</p>

Setting the Perl Environment

The AFP2web Scripting Facility has a Perl engine embedded (and delivered as Perl DLL). If you call AFP2web from the root directory of the AFP2web installation and if you do not use additional external Perl modules, you don't have to set up the Perl environment.

Case 1: Different working directory

If you do not use external modules, but you call AFP2web from another path than the root directory of the AFP2web installation, then you must do the following:

- Tell AFP2web where to find the AFP2web Perl packages (a2w*.pm): You can set the PERLLIB environment variable. Instead, the simplest method is to modify the internal Perl array @INC as described below under [*"Case 2: External modules" on page 106*](#).
- Tell the AFP2web Perl engine where to find the main module (afp2web.pm). This is done by passing the fully qualified name of the main module by using the -sp option.

Example:

```
-sp: c:\AFP2web\MyModule.pm
```

Case 2: External modules

If you use external modules, you must tell the AFP2web Perl engine where to find these external modules. You can set the PERLLIB environment. However, the simplest method is to modify the Perl internal array @INC.

Important: Modifying @INC must come as the very first statement in the main script module, which is loaded by AFP2web.

The following example shows how to add runtime directories to the Perl search path. For this example, we assume:

- Your Scripting Facility module (afp2web.pm) requires standard Perl modules (XML::Writer) and your Perl installation is located in C:\Perl\.
- Your Scripting Facility module is not in AFP2web root installation directory, but instead in C:\temp\xxx
- AFP2web's Scripting Facility Package (a2w:*) is not located in the AFP2web root installation directory, but instead in c:\temp.

You add all directories to the Perl search path with the following statement, which comes as the first statement of the main script module loaded by AFP2web:

```
BEGIN {
    unshift(@INC, ('C:\Perl\lib', 'C:\Perl\site\lib', 'C:\temp'));
```

```
}
```

All that is left to be done is to inform AFP2web, which Scripting Facility module to load. You do this

- Either with the INI file parameter `ScriptProcedure=C:\temp\xxx\afp2web.pm`
- Or by using the command line option `-sp:C:\temp\xxx\afp2web.pm`.

Additional Usage Notes

Using the INI Parameter MemoryOutputStream

When running AFP2web with the INI Parameter MemoryOutputStream set to ON, the AFP2web Kernel writes the output document to a buffer in memory instead of to file.

The purpose of this INI parameter is to make it possible for a scripting routine to access the output document using the appropriate methods of the a2w::Document package (getOutputBuffer, getOutputBufferLength).

A possible application of this is to post process document content (for example by archiving the document buffer, modifying document content, etc.).

The appropriate event for accessing a buffered document in a script would be the finalizeDoc() sub routine.

An example of the coding statements used to access an output document in the memory buffer:

```
#---- Get PDF buffer length
$PDFBuffLenTmp = $a2wDocument->getOutputBufferLength();

#---- Get PDF buffer
$PDFBuffTmp = $a2wDocument->getOutputBuffer();

#---- From here on you can write the PDF buffer to a PIPE or
a database or a file or ...

#---- Sample to write to a file
my $OPFileNameTmp = "pdf/sf_" . $a2wDocument->getOutput-
Filename();
open( OPFILE, ">$OPFileNameTmp" ) || die "Error! Unable to open
$0 : $!\n";
binmode( OPFILE );
syswrite( OPFILE, $PDFBuffTmp, $PDFBuffLenTmp );
close( OPFILE );
```

Chapter 5: AFP2web Reference

This chapter begins with a reference of the parameters controlling AFP2web core functionality:

- INI parameters (afp2web.ini)
- command line options
- mapping.def parameters
- IBM specific font naming conventions

Included is also the Scripting Facility Reference:

- Cross-reference overview of methods and where used in a script
- Scripting Facility API Reference

This chapter also includes:

- A list of error messages and codes.

5.1 Overview of INI Parameters and Command Line Options

On the following pages, the INI file parameters and their corresponding command line options are listed side by side as a quick reference. The entries are grouped by category.

Overview of INI Parameters and Command Line Options

<i>Licensing information</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
Licensee		Licensee
SerialNr		Serial number

<i>Input / output formats</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
InputFormat	-<inputformat>	Format of input documents
OutputFormat	-<outputformat>	Format to convert to

<i>Read from stdin, write to stdout</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
	-std	Both options: -si and -so
	-si	Read from stdin
	-so	Write to stdout
MemoryOutputStream		Write output document to a buffer in memory

<i>Type of output (document splitting/merging and creation of index files)</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
FileCreationMode	-DOC / ALL / -DOC_INDEX / -DOC_COLD / -DOC_MERGE / -PAGE	Documents to create
PageOutput	-po	Create one file per page of the output document (only for TIFF as output format)

<i>Processing index data</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
IndexFormat	-DOC_INDEX: CSV XML -DOC_COLD: XML CSV	Format of index data
IndexRecord		Additional fields for index data in CSV format
PDFBookmark	-bm	PDF bookmarks for index entries

<i>Generating file names and output folders</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
FilenamePattern	-fnpt	Pattern for output file names
GenerateUniqueFile	-u	Generate unique file names
DocumentCount	-dc	Create sub folders for output and limit the number of output documents per sub folder.

<i>Specifying folders (directories)</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
	-ip	Path to afp2web.ini

<i>Specifying folders (directories)</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
	-if	Path and file name of INI file
LogPath	-lp	Folder for LOG output
ResPath	-rp	Standard folder for AFP resources
CpPath	-cp	Folder with code pages
OutputFilePath	-op	Folder to write output documents
ExtFontPath	-fp	Font path to additional Type1 and TrueType fonts
IndexPath	-xp	Folder for index files.
OverlayPath	-ovp	Folder for AFP Overlay Resources
PageSegmentPath	-psp	Folder for AFP Page Segment resources
FormDefPath	-fdp	Folder for AFP FORMDEF Resources
AFPFontPath	-ap	Folder for AFP Font resources
TempPath	-tp	Temporary path is used to cache AFP2web temporary files. Default is the temporary path defined in the system. This temporary path must exist and AFP2web must have read/write permissions to this directory.

<i>Code page defaults</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
CodePageDefaultChar		Replacement character

<i>Code page defaults</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
CodePage	-dcp	Default code page to use when translating strings and indexes of the AFP spool file to ASCII

<i>Locating AFP resources</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
	-rf	AFP resource file
	-xf	The input file with the AFP index to be processed for the AFP document
	-fd	AFP FORMDEF
FormdefExt		File extension for AFP FORMDEFs
OverlayExt		File extension for AFP Overlays
PageSegExt		File extension for AFP Page Segments
CharSetExt		File extension for AFP Character Sets
CodePageExt		File extension for AFP code pages
CodedFontExt		File extension for AFP coded fonts

<i>The section [FORMDEF]</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
<formdefs and the related medium maps>		Lists AFP FORMDEF and for each FORMDEF the medium maps, which apply

<i>LOG files, messages, statistics</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
Quietmode	-q	Suppress or display processing information on the console
Logging	-l	LOG output
LoggingFont	-lf	LOG output: font infos
LoggingLevel	-ll	Restrict LOG output to information categories.
ExceptionLoggingLevel	-ell	Print error messages in a selected level of detail
Statistic	-s	Write a statistic report (stat*.txt)

<i>Spool file processing</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
Strict	-Strict	Stop processing if AFP resource cannot be found.
Protocol	-p	Write „protocol“ with Doc IDs
StartingDocument	-sd	Doc ID, start converting at this position in spool file.
EndingDocument	-ed	Doc ID, stop converting after this document in spool file.
StartingPage	-pp:[fp][-tp]	Page range to convert (Starting page)
EndingPage	-pp:[fp][-tp]	Page range to convert (Ending page)
SkipPage		Ignore standard page
SkipObjectSize		Size of standard page

<i>Additional program flags</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
	-h or - ?	Help info about AFP2web (list of command line options)
	-v	Display version of AFP2web
	-vall	Display versions of all AFP2web components
LaunchPreview	-pv	Display converted document in viewer

<i>PDF document properties, security settings, viewer preferences, conversion parameters</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
Creator		Author
Title		Title
Subject		Subject
Keywords		Keywords
PDFSecurity	-ps	PDF Security
PDFUIOptions	-pui	Viewer settings
PDFWinOptions	-pwn	Document display settings
PDFDocLimits		Memory allocation for PDF
Color	-c	Color

<i>TIFF (raster format) output</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
FormatType	-TIFF: compressionformat or: -compressionformat	TIFF format
Resolution	-pr	Resolution
Color	-c	Color

<i>ASCII output</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
MaxCols	-mc	Maximum columns per row
MaxRows	-mr	Maximum rows per page
	-iASCII	Display optimal values for the options -mc and -mr

<i>AFP output</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
AFPComplianceLevel		AFP Compliance level
CurveSamplingFactor	-csf	Curve sampling factor
PageRotation	-r	Page orientation
PrintableLineWidth	-plw	Minimal line width for printing

<i>Optimize included objects (graphics)</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
SaveOCDataOnDisk	-sod	Save Object Container Data on disk
TempPath	-tp	Temporary path is used to cache AFP2web temporary files. Default is the temporary path defined in the system. Temporary Path should exist.
JPEGQuality	-jq:n	Jpeg Quality where n can be 1 to 100

<i>Color, IOCA Images to PDF</i>		
<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
CMYKTORGB	-nocmyktorgb	Convert CMYK to RGB

Color, IOCA Images to PDF

<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
Section [AFPColorTable]	-clr	RGB values for standard AFP OCA colors

Document adjustments

<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
PageOrientation	-r	Page rotation
Watermark	-wm	Watermark

AFP2web Scripting Facility

<i>Parameter of INI file</i>	<i>Command line option</i>	<i>Function</i>
FileCreationMode=DOC_COLD	-doc_cold	Activates the AFP2web Scripting Facility. Also used to specify output format of index data.
ScriptProcedure	-sp	Name of the script file.
ScriptArgument	-sa	Arguments to pass to the script
ScriptUnitBase		Base unit of measurement
Autosplit		Split documents automatically using standard AFP Indexes (valid only for -doc_cold)

5.2 INI File Parameters (afp2web.ini)

The following description lists the parameters of the INI file in alphabetical order. For an overview of parameters grouped by category, please refer to *"Overview of INI Parameters and Command Line Options"* on page 111.

Function and Structure

afp2web.ini is the configuration file used to set global parameters of AFP2web. This file is divided into the sections:

- [Settings] For conversion parameters
- [FORMDEF] To map FORMDEFs to their Medium Maps.
- [AFPColorTable] To map standard AFP OCA colors to RGB values
- [user-defined section] To define FTP locations of remote AFP resources

Usage Notes

When you make changes to afp2web.ini, please note that:

- A semicolon (";") at the beginning of a line marks a comment line.
- A path without a drive will be interpreted as relative to the current directory.
- The parameters are not case-sensitive.
- An entry within the INI file must not exceed 1024 (1 Kb) characters.

Parameters of the Section [Settings]

AFPComplianceLevel=<modcap>

Output AFP which conforms with one of the following MO:DCA Interchange Sets:

modcais2	MO:DCA Interchange Set 2
modcap	MO:DCA P (Standard)

Example: AFPComplianceLevel=modcap

Default: modcap

AFPFontPath=<path>

Path to AFP font resources.

It is also possible to specify a comma-separated list of paths.

Example: samples/resource,c:\afp\fonts

Default: samples/resource

Autosplit=<on/off>

Automatic Document splitting based on standard AFP Indexes. Only for FileCreationMode=DOC_COLD.

If Autosplit=on, then the document begin/end events of the Scripting Facility are automatically triggered. This means, passing a return code of 2 from afp2web() will have no effect.

Default: off

CharSetExt=<file extension>

File extension for AFP Character Sets.

Default: *

Note: Normally AFP2web looks for <resource name>*, so it is normally not necessary to use this parameter. But if -Strict is set and a file extension is used, then AFP2web looks for <resource name>.<file extension>

CMYKTORGB=<on/off>

Only for IOCA FS45 color images output to PDF.

on	Combine IOCA FS45 CMYK color image planes as RGB.
off	Pass through IOCA FS45 CMYK color image planes.

Note: CMYK for PDF results in large PDF files and is therefore only recommended for printing and not for online viewing.

Default: CMYKTORGB=on

CodedFontExt=<file extension>

File extension for AFP Coded Fonts

Default: *

Note: Normally a2w looks for <resource name>*, so it is normally not necessary to use this parameter. But if -Strict is set and a file extension is used, then AFP2web looks for <resource name>.<file extension>

CodePage=<defaultcodepage>

Defines the default code page to use for translating resource names, NOPs and indexes of the AFP spool to ASCII. (Examples include: index names and values, SFI names such as Doc Name, Page Name, Resource Name, NOPs.)

The mapping.def maps this AFP code page to the code page file (CP file). The default is also set in the mapping.def.

Default: T1GIG500

CodePageDefaultChar=" "

Code page default character.

This is the character that will be used to display a non-mapped character.

Default: " "

CodePageExt=<file extension>

File extension for AFP code pages

Default: *

Note: Normally a2w looks for <resource name>*, so it is normally not necessary to use this parameter. But if -Strict is set and a file extension is used, then AFP2web looks for <resource name>.<file extension>

Color=<on/off>

Applies to TIFF:UNCOMPRESSED, TIFF:JPEG, to PDF or to JPEG as output formats. Leave colors as is or convert these to black/white (text and image):

off	Black/white (Default)
on	Preserve colors

CpPath=<path>

Path to code pages and mapping.def

Default: afpcp

Creator=<doc-info>

PDF document property: Author

Default: See default value in the INI file

CurveSamplingFactor=<n>

Factor used to smooth curves. A value from 1 to 65535. Modify this factor to get smooth curves on AFP output. The lower the factor the smoother the curves. Applies only when converting to AFP.

Example: CurveSamplingFactor=64

Default: 64

DocumentCount=<n>

Create sub folders for output and limit the number of documents per sub folder.

This parameter is used together with FileCreationMode=DOC_INDEX

Example: DocumentCount=500 together with FileCreationMode=DOC_INDEX limits the number of output documents to 1000 per sub folder: You receive per sub folder 500 documents plus 500 index files.

EndingDocument=<n>

Last document to convert in a spool file. For EndingDocument, enter the document ID as determined in the “protocol”.

This option is only possible if DOC_INDEX or DOC_COLD is selected

Default: 0

EndingPage=<endpage>

Page Range to convert (ending page)

Default: -1 (last page)

ExceptionLoggingLevel=<n>

Print error messages in a selected level of detail.

You specify one of the following:

1	Log exception message alone (default)
2	Log file name and line number with exception message

ExtFontPath=<path>

Font path to additional Type1 and TrueType fonts.

Default: extfont

FileCreationMode=<option>

Output files to produce:

ALL	One output file for all AFP documents (Default)
DOC	One output file per AFP document
DOC_INDEX	Create output documents with index files.
DOC_COLD	Activate the AFP2web Scripting Facility.
DOC_MERGE	Merge all files into one output file. For converting TIFF to PDF or for converting AFP.
PAGE	One file per page in an AFP document.

FilenamePattern=<pattern>,Keyword1,Keyword2...

Filename pattern (without extension) used to define any pattern for filenames by using constant text and/or placeholders for reserved keywords.

The syntax is:

`<pattern>,Keyword1,Keyword2...`

`<pattern>` specifies a formatting pattern in the syntax of a C printf command.

When used as pattern for the TIMESTAMP value (see below), you use the syntax of a C strftime command. The keywords you specify are used to create the filename according to the pattern. It is important to ensure that data types of the keyword match with the formatting pattern.

Possible as keywords:

DOCID	Document ID (data type: Integer)
PAGEID	Page ID (data type: Integer)
PID	Process ID (data type: String)
TIMESTAMP[:<pattern>]	Current time stamp (data type: String)
SPOOLNAME	Spool file name (data type: String)

Defaults:

The default TIMESTAMP format is "%y%m%d%H%M%S"

Which default is used as FilenamePattern depends on your settings for FileCreationMode and GenerateUniqueFile, as indicated in the following table:

<i>Condition</i>	<i>Default FilenamePattern</i>
FileCreationMode is either FILE_MERGE, DOC_INDEX, DOC_COLD, DOC, or PAGE	"%s_%s.%d",SPOOLNAME,PID,DOCID
GenerateUniqueFile=on	"%s_%s.%d",SPOOLNAME,PID,DOCID
FileCreationMode=ALL and GenerateUniqueFile=off	"%s",SPOOLNAME
PageOutput=on (or when using the command line option -po)	"%s_%s.%ld.%ld",SPOOLNAME,PID,DOCID,PAGEID

FormatType=<tiffoutputformat>

Applies only to TIFF: Compression option for the resulting TIFF file. Specify a value out of:

G3	ITU Group III
G4	ITU Group IV (Default)
JPG	TIFF with JPEG compression
LZW	TIFF with LZW compression
PACK	TIFF, packed
UNCOMPRESSED	TIFF without compression

FormdefExt=<file extension>

File extension for FORMDEF resources.

Default: *

Note: Normally a2w looks for <resource name>*, so it is normally not necessary to use this parameter. But if -Strict is set and a file extension is used, then AFP2web looks for <resource name>.<file extension>

FormDefPath=<path>

Path to FORMDEF resources.

It is also possible to specify a comma-separated list of paths.

Example: samples/resource,c:\afp\formdef

Default: samples/resource

GenerateUniqueFile=<on/off>

This parameter tells AFP2web to generate unique file names for output files.

This parameter is maintained for backward compatibility. For this version of AFP2web it is no longer required and we even recommend not to use it.

on	Generate unique file names
off	Do not generate (Default).

IndexFormat=<CSV/XML>

Format of the index file to create. Only if FileCreationMode=DOC_INDEX is specified.

CSV	Index file in CSV format
XML	Index file in XML format (Default)

IndexPath=<path>

Path to output index files. If this path is not specified, AFP2web will write the index files to the path specified in OutputFilePath.

Default: pdf

IndexRecord=<fieldlist>

Only for IndexFormat=CSV.

A list of keywords for meta information to be added to each index entry. Given in the order, in which each meta field must occur. Each meta field specified must have a delimiter (for example ",").

The following keywords are possible for <fieldlist>:

PAGE_GROUP_NAME	AFP Group name
PAGE_NAME	AFP Page name
PAGE_COUNT	Page count
FILE_NAME	Name of output file
FILE_TYPE	ASCII/JPEG/PDF/TIFF
FILE_SIZE	Size of output files
INDEX	Index as: <IndexName>=<indexValue>

Default: IndexRecord=

PAGE_GROUP_NAME,PAGE_NAME,PAGE_COUNT,FILE_NAME,FILE_TYPE,INDEX

InputFormat=<input format>

Input format. One of the following is possible:

AFP	AFP document (standard)
TIFFIN	TIFF document

JPEGQuality=<n>

Compression quality for colored graphics.

This value is used while saving object container to disk

Quality can be any value between 1 and 100:

1	Best compression but lowest quality
100	Lowest compression but best quality
50	is a recommended value (Default).

Example: JPEGQuality=50

JPEGQuality is used whenever the image is placed as JPEG in PDF output, for example:

For B/W output, color images in the spool file are converted to gray scale JPEG images.

For color output, when "CMYKTORGB" is set to "on", IOCA FS45 CMYK JPEG planes are merged as RGB JPEG and place in PDF output.

Keywords=<doc-info>

Keywords to be stored in the PDF document

Default: See default value in the INI file

LaunchPreview=<option>

Preview option. Launches Acrobat Reader (for PDF) or the appropriate ASCII, TIFF or JPEG viewer to display the resulting output file.

on	Display resulting output file.
off	Do not display (Default).

Licensee=<name>

The name of the licensee as determined by Maas High Tech Software GmbH. Please set the value within quotes. Example: Licensee="Maas High Tech Software GmbH"

Both SerialNr and Licensee will activate the input/output formats and the platform, which have been purchased as AFP2web options.

Default: See default value in the INI file

Logging=<on/off>

Log file (Part 1: Parsing-section, Part 2: Builder-section with font information):

on	Create complete log file. Same as LoggingLevel=1,2,3,4,FONT
off	Do not create (Default).

Note: If LoggingLevel=0 follows, it will switch off Logging.

LoggingFont=<on/off>

Log file (Part 2: Builder-section with font information):

on	Create log file, part 2 Same as LoggingLevel=FONT
off	Do not create (Default).

Note: If LoggingLevel=0 follows, it will switch off LoggingFont.

LoggingLevel=<n>

Syntax:

LoggingLevel=1,2,3,4,5,6,8,SF,[RES | (ResType)+]

Logging level. For <n>, specify one or more values out of:

0	No log (Default)
1	Structured Fields (SFI)
2	Data information of SFI
3	Repeating Groups
4	Patterns (like image)
5	FOCA information
6	Mapping.def entries added to the end of the log file. These entries can be used as template to modify the mapping.def
8	FTP Logging
ALL	All information including FOCA

FONT	Builder part, font information (same as LoggingFont=on)
Resource Logging Levels:	
RES	all resource types
	or any of the following:
CDP	Code Pages
FDEF	Formdefs
FNTR	Fonts (detailed list of all fonts used for current spool file)
IOB	Object containers
MMAP	Medium Maps
OLY	Overlays
PSEG	Page Segments
Image Logging Levels	
INFO	Log image processing informational messages
WARN	Log image processing warning and informational messages
DEBUG	Log warning, informational and debug messages during image processing

Example: LoggingLevel=1,INFO

Note: If LoggingFont follows LoggingLevel=0, LoggingFont will take effect

LogPath=<path>

Path to write the log file.

Default: log

MaxCols=<max columns>

MaxRows=<max rows>

Maximum number of columns and rows. For ASCII only. Limits the page size.

Default: If either no value or zero is specified, AFP2web calculates appropriate values.

MemoryOutputStream=<on / off>

Instructs A2W Kernel to write the output document to a buffer in memory instead of to file. Only valid when used together with the file creation mode "DOC_COLD". For usage notes, please refer to the Scripting Facility guide and the API reference.

Default: Off (write to file).

OutputFilePath=<path>

Path to write output files.

Default: pdf

OutputFormat=<output format>

Output format:

AFP	AFP format
ASCII	ASCII format with adjustments to line/column positioning.
JPEG	JPEG format
PDF	PDF format (Default)
PNG	Portable Network Graphics
TIFF	TIFF format

OutputSize=<width>,<height>

Size of the resulting output document specified as width and height in pixels. For example, to produce thumbnail images for document pages.

Default: No default - the document original size is used.

OverlayExt=<file extension>

File extension for Overlay resources.

Default: *

Note: Normally a2w looks for <resource name>*, so it is normally not necessary to use this parameter. But if -Strict is set and a file extension is used, then AFP2web looks for <resource name>.<file extension>

OverlayPath=<path>

Path to Overlay resources.

It is also possible to specify a comma-separated list of paths.

Example: samples/resource,c:\afp\overlays

Default: samples/resource

PageOutput=<on/off>

Create one output file for each page of an output document. Applies only to TIFF as output format.

Default: off

PageRotation=<rotation>

Page rotation clockwise in degrees or the page orientation for all output pages:

0
90
180
270
portrait
landscape

Default: off

PageSegExt=<file extension>

File extension for Page Segment resources.

Default: *

Note: Normally a2w looks for <resource name>*, so it is normally not necessary to use this parameter. But if -Strict is set and a file extension is used, then AFP2web looks for <resource name>.<file extension>

PageSegmentPath=<path>

Path to Page Segment resources.

It is also possible to specify a comma-separated list of paths.

Example: samples/resource,c:\afp\pagesegments

Default: samples/resource

PDFBookmark=<option>

Inserts bookmarks in PDF documents for index elements.

on	Insert bookmarks
off	Do not insert (Default)

PDFDocLimits=<limit>

Applies only to PDF: This parameter is used as a factor for allocating memory (fonts, images, other PDF objects).

Increase this limit if you encounter the error message: "Too many images in this PDF: nnn. Increase limits by mpdf_open()."

The formula for the amount of memory allocated:

```
Memory allocated(in bytes)
= 18200 * PDFDocLimits + 6892
Memory allocated(in Kbytes)
= (18200 * PDFDocLimits + 6892) / 1024
```

The following table gives a few examples of settings and the resulting allocation of memory:

<i>PDFDocLimits</i>	<i>Memory allocated (Kbytes)</i>
20	362
200 (default)	3561
500	8893
1000	17780

PDFSecurity=[<Owner Password>][,<User Password>][,<Key Length>][,<Permission Flags>]]]

PDF Security options

The parameters are fixed-positioned and have the following meanings:

Owner Password	Optional Owner Password. Used to change security options
User Password	Optional, User password, required when opening the document.
Key Length	Optional, Optional. Encryption Level Key Length. Possible is a multiple of 8 between 40 and 128 40 => Normal Encryption, 128 => Better Encryption Standard is 40.
Permission Flags	Optional. One or more flags separated by " " specified below. For Key Length 40, Flags 5,6,7,8 are not allowed

The following values are allowed as Permission Flags:

0	None, everything is disabled (Default)
1	Allow Low resolution printing
2	Allow Changing the document
3	Allow Content copying or extraction
4	Allow Authoring comments and form fields
5	Allow Form field fill-in or signing
6	Allow Content accessibility
7	Allow Document assembly
8	Allow High Quality Printing

Examples:

1. Neither Owner nor User Password (this is the default)

PDFSecurity=

2. Owner Password is MHT, Encryption Key Length is 128 bits, no Permission

PDFSecurity=MHT,,128,0

PDFUIOptions=[<HideMenubar>][,<HideToolbar>][,<Hid

eWindowUI>]

PDF viewer preferences for the "User Interface"

For each parameter, specify either "on" or "off" (without the quotes):

By default, all parameters are off.

HideMenubar	on = Hide menu bar.
HideToolbar	on = Hide tool bar.
HideWindowUI	on = Hide window controls.

Note: An empty value is allowed and default values are assumed for empty values. For example, *PDFUIOptions=,on,on* is equivalent to *PDFUIOptions=off,on,on*.

Examples:

To hide only the menu bar, use *PDFUIOptions=on*.

To hide menu bar and window controls, use *PDFUIOptions=on,,on*.

PDFWinOptions=[FitWindow][,<CenterWindow>][,<FullScreenWindow>][,<DisplayDocTitle>]

PDF viewer preferences "Window" options

For each parameter, specify either "on" or "off" (without the quotes):

By default, all parameters are off.

FitWindow	on = Resize the document's window to fit the size of the first displayed page.
CenterWindow	on = Position the document's window in the center of the screen.
FullScreenWindow	on = Display the document in full-screen mode, with no menu bar, window controls, or any other window visible.
DisplayDocTitle	on = Display the document description property Title in the window's title bar. off = Display PDF filename in the window's title bar.

Note: An empty value is allowed and the default is assumed for empty values. For example, *PDFWinOptions=,on,on,on* is equivalent to *PDFWinOptions=off,on,on,on*.

Examples:

To resize the window, use *PDFWinOptions=on*.

To center window and display in full screen mode, use PDFWinOptions=,on,on.

To display the document title, use PDFWinOptions=,,,on.

PrintableLineWidth=<n>

Minimum line width that has to be used on output to ensure that lines are printable. Specified in a unit of pixels. Applies when converting to AFP.

Example: PrintableLineWidth=1

Default: 1

Protocol=<on/off>

The "protocol" is a processing log listing documents converted from spool file. Documents are identified by document ID. This option is only possible if DOC_INDEX or DOC_COLD is selected.

on	Write „protocol“
off	Do not write (Default).

Quietmode=<on/off>

Suppress messages to system console:

on	No messages
off	Show message (Default)

Resolution=<dpi resolution>

Applies only to bitmap formats: Resolution in dpi. We recommend a value out of:

200	
240	Default
300	

ResPath=<path>

Path to AFP resources.

Default: samples/resource

SaveOCDataOnDisk=<on/off>

Save Object Container Data on disk.

Flag to cache the object container data temporarily in to file path given by "TempPath". This option is used to save memory and to optimize the output file size.

Default: off

ScriptArgument=<arguments>

Used for AFP2web Scripting Facility

(only if FileCreationMode=DOC_COLD).

Arguments to pass to the scripting procedure.

Default: null (no arguments)

ScriptProcedure=<scriptfile>

Used for AFP2web Scripting Facility

(only if FileCreationMode=DOC_COLD).

The scripting procedure file name.

Default: afp2web.pm

ScriptUnitBase=<baseunit>

Base unit of measurement for the Scripting Facility used to specify/receive position and size parameters. (only if FileCreationMode=DOC_COLD)

Possible one out of:

pixel	Pixels at page resolution (Default)
mm	Millimeters

SerialNr=<seriennr>

The serial number as determined by Maas High Tech Software GmbH.

Both SerialNr and Licensee will activate the input/output formats and the platform, which have been purchased as AFP2web options.

SkipPage=<on/off>

Skip empty pages (for example, a scanned empty page). This flag tells AFP2web to ignore a page if it has no content or if it contains one image, which is less or equal to the size specified in "SkipObjectSize".

Default: off

SkipObjectSize=<n>

The maximum byte size of an image resulting from a scanned empty page, which can be ignored. Even though a page (for example the back side) can be empty, the scanned image still might contain a few pixels. This parameter specifies the maximum size of this type of image.

Default: 2K (=2048 Bytes)

StartingDocument=<n>

First document to convert in a spool file. For StartingDocument, enter the document ID as determined in the "protocol".

This option is only possible if DOC_INDEX or DOC_COLD is selected

Default: 0

StartingPage=<startpage>

Page Range to convert (Starting page)

Default: 1

Statistic=<on/off>

Write a report with performance statistics.

on	Write report (Default).
off	No report

Strict=<on/off>

Stop processing if a required AFP resource is not found.

on	Stop processing if resource not found
-----------	---------------------------------------

off	Continue processing (Default)
------------	-------------------------------

Subject=<doc-info>

PDF document property: Subject

Default: See default value in the INI file

TempPath=<path>

Temporary path is used to cache AFP2web temporary files. The temporary path must exist.

AFP2web must have read/write access to the directory specified in this path. Otherwise, AFP2web will fail when trying to create and delete temporary files.

Default: the temporary path defined in the system.

Title=<doc-info>

PDF document property: Title

Default: See default value in the INI file

Watermark=<text, typeface,height,rotation,redmask,greenmask,bluemask>

Displays a text as a watermark behind the document text. The parameters that can be specified are:

text	text to display
typeface	Font typeface
height	Font size in tenths of points. Specify 90 for 9 points.
rotation	Rotation. A value from 0 to 360.
redmask	RGB Red mask value from 0 to 255
greenmask	RGB Green mask value from 0 to 255
bluemask	RGB Blue mask value from 0 to 255

Example: Watermark=Confidential,Helvetica,600,60,189,219,231

Default: no watermark

Parameters of the Section [FORMDEF]

<formdef>= <medium map-list>

This section is required if the AFP spool file requires a Formdef. The Formdef can be specified using the command line option `-fd` or `-rf`.

The [FORMDEF] section of the INI file can be used to map Medium Maps to Formdef Resource names. This feature is useful, for example, if the command line option `-fd` is not used. The Formdef itself must be available as external resource.

You use this section to list the medium maps, which are contained in a Formdef. A medium map must be unique to the Formdef. AFP2web will load the first Formdef it can find for a given medium map.

Example:

```
F1hqsd=HOCH, HOCHD, QUER, QUERD, HFACH1
```

Parameters of the Section *[AFPColorTable]*

The section *[AFPColorTable]* is used to map each color of the AFP standard OCA color table to their corresponding RGB values. You use this section to override the default values (For more information on the default values for each color, please refer to the documentation of IBM, document number SC31-6802-06, page 473/501).

RGB values should be in the range 0 to 255. Values outside this range are treated as zero.

Syntax:

```
Colorname=<Red>, <Green>, <Blue>
```

Example:

```
Blue=0, 0, 255
```

Color names and their default RGB values are:

Black	0,0,0
Blue	0,0,255
Brown	144,48,0
Cyan	0,255,255
Darkblue	0,0,170
Darkcyan	0,146,170
Darkgreen	0,146,0
Default	0,0,0
Gray	131,131,131
Green	0,255,0
Magenta	255,0,255
Medium	255,255,255
Mustard	196,160,32
Orange	255,128,0
Purple	170,0,170
Red	255,0,0
White	255,255,255
Yellow	255,255,0

User-defined Sections for Logical Locations

Any arbitrarily named section can be added to the INI file to define the location of remote AFP resources. The following parameters are possible:

[mysectionname]	Use the "[" Brackets to define your FTP Connection. Any name can be used (for example, [FONT300]). To tell AFP2web to use that FTP connection for a particular AFP font resource, specify the FTP connection name prefixed with an @ in your INI parameters. Example: AFPFontPath=@FONT300
Type=FTP	Communication protocol
HostName=<hostname or IP address>	Host name or IP address
UserName=<username>	FTP user name
Password=<password>	Password
StartDir=<starting directory>	Starting directory Examples: StartDir='SYS1.FONT300' ==> For a mainframe PO data set (Single quotes have to be used) StartDir=c:/afp/font3000 ==> For a Windows FTP server StartDir=/opt/afp/font3000 ==> For a Unix FTP server

You define one section for each location, for example:

```
[MYFONT300]
Type=FTP
HostName=192.168.1.1
UserName=afp2web
Password=*****
StartDir='SYS1.FONT300'
```


5.3 AFP2web Command Line Options

The following description lists the command line options in alphabetic order. For an overview of options grouped by function, please refer to *"Overview of INI Parameters and Command Line Options"* on page 111 .

Usage Notes

When you call afp2web, begin each option with a minus sign or hyphen ("-").

If you do not specify a command line option, AFP2web will look for the setting in the INI file or assume a default option. The sequence is:

- Command line option entered when calling AFP2web
- Parameter in the INI file
- Default value (as described for the INI file)

For options with multiple arguments, it is recommended to enclose the arguments in quotes. This ensures that AFP2web will not terminate the argument string at the first blank character.

Example:

`-wm:"Confidential Text,Helvetica,600,60,189,219,231"`

-<filecreationmode>

Output files to produce. For <filecreationmode> use one of the following:

ALL	One output file for all AFP documents.
DOC	One output file per AFP document
DOC_INDEX	Create output documents with index files. (See the description below.)
DOC_COLD	Used for AFP2web Scripting Facility. Create index files for the Scripting Facility AutoSplit function. (See the description below.)
DOC_MERGE	Merge all files into one output file. For converting TIFF to PDF or for converting AFP.
PAGE	One file per page in an AFP document.

DOC_INDEX with additional extension specifies the format of the index file to create. Possible is one of:

-DOC_INDEX:CSV	Index file in CSV format
-DOC_INDEX:XML	Index file in XML format

When used together with the AutoSplit function of the AFP2web Scripting Facility, DOC_COLD specifies processing and, in addition, the format of the index file to create:

-DOC_COLD	AFP2web will not write any index files. This has to be done by the custom scripting module.
-DOC_COLD:CSV	AFP2web writes index file in CSV format.
-DOC_COLD:XML	AFP2web writes index file in XML format.

-<input format>

Input format. By default, the standard input format is AFP. This command line option can be used to specify a different format:

TIFFIN	TIFF document
---------------	---------------

-<output format>

Output format:

AFP	AFP format
ASCII	ASCII format with adjustments to line/column positioning.
JPEG	JPEG format
PDF	PDF format
PNG	Portable Network Graphics
TIFF	TIFF format
TIF	TIFF format

Adding an extension to the option -TIFF specifies the TIFF compression to use. Possible is one of:

-TIFF:G3

-TIFF:G4
-TIFF:JPG
-TIFF:LZW
-TIFF:PACK
-TIFF:UNCOMPRESSED

-<tiffoutputformat>

The TIFF format can be entered as a separate option next to -TIFF. One of the following: :

-G3
-G4
-JPG
-LZW
-PACK
-UNCOMPRESSED

-ap:<path>

Path to AFP font resources.

It is also possible to specify a comma-separated list of paths.

Example: -ap:samples/resource,c:\afp\fonts

-bm

Inserts bookmarks in PDF documents for index elements.

-C

Leave colors as is.

Applies to JPEG, PDF, PNG, TIFF:JPEG , and TIFF:UNCOMPRESSED as output formats.

-clr:<colorname>,<r>,<g>,

RGB Value of the given color. You use this option to override default values (For more information on the default values for each color, please refer to the documentation of IBM, document number SC31-6802-06, page 473/501).

r, g and b should be integer value within the range of 0 to 255. Values outside this range are treated as zero.

Color names and their default RGB values are

Black	0,0,0
Blue	0,0,255
Brown	144,48,0
Cyan	0,255,255
Darkblue	0,0,170
Darkcyan	0,146,170
Darkgreen	0,146,0
Default	0,0,0
Gray	131,131,131
Green	0,255,0
Magenta	255,0,255
Medium	255,255,255
Mustard	196,160,32
Orange	255,128,0
Purple	170,0,170
Red	255,0,0
White	255,255,255
Yellow	255,255,0

-cp:<path>

Path to code pages and mapping.def

`-csf:<n>`

`-csf:<n>`

Factor used to smooth curves. A value from 1 to 65535. Modify this factor to get smooth curves on AFP output. The lower the factor the smoother the curves. Applies only when converting to AFP.

Example: CurveSamplingFactor=64

Default: 64

`-dc:<n>`

Create sub folders for output and limit the number of documents per sub folder.

This option is used together with

`-DOC_INDEX` or with

`-DOC_COLD:XML` or `DOC_COLD:CSV`.

Example:

The command line option `-dc:500` together with `-doc_cold:xml` limits the number of output documents to 1000 per sub folder: You receive per sub folder 500 documents plus 500 index files in XML format.

`-dcp:<codepage>`

Defines the default code page to use for translating resource names, NOPs and indexes of the AFP spool to ASCII. (Examples include: index names and values, SFI names such as Doc Name, Page Name, Resource Name, NOPs.)

The mapping.def maps this AFP code page to the code page file (CP file). The default is also set in the mapping.def.

`-ed:<docid>`

Last document to convert in a spool file. Enter the document ID as determined in the "protocol".

This option is only possible if `DOC_INDEX` (or `DOC_COLD`) is selected

`-ell:<n>`

Print error messages in a selected level of detail.

You specify one of the following:

1	Log exception message alone (default)
2	Log file name and line number with exception message

-fd:<formdef>

Standard FORMDEF to be used for the AFP document.

-fdp:<path>

Path to FORMDEF resources.

It is also possible to specify a comma-separated list of paths.

Example: samples/resource,c:\afp\formdef

-fnpt:<pattern>,Keyword1,Keyword2...

Corresponds with the INI parameter FilenamePattern and is) used to define any pattern for filenames by using constant text and/or placeholders for reserved keywords.

The syntax is:

-fnpt:<pattern>,Keyword1,Keyword2...

<pattern> specifies a formatting pattern in the syntax of a C printf command.

When used as pattern for the TIMESTAMP value (see below), you use the syntax of a C strftime command. The keywords you specify are used to create the filename according to the pattern. It is important to ensure that data types of the keyword match with the formatting pattern.

Possible as keywords:

DOCID	Document ID (data type: Integer)
PAGEID	Page ID (data type: Integer)
PID	Process ID (data type: String)
TIMESTAMP[:<pattern>]	Current time stamp (data type: String)
SPOOLNAME	Spool file name (data type: String)

Defaults:

The default TIMESTAMP format is "%y%m%d%H%M%S"

-fp:<path>

Which default is used as the filename pattern depends on your settings for the INI parameters FileCreationMode and GenerateUniqueFile (or their corresponding command line options), as indicated in the following table:

<i>Condition</i>	<i>Default Filename Pattern</i>
FileCreationMode is either FILE_MERGE, DOC_INDEX, DOC_COLD, DOC, or PAGE	"%s_%s.%d",SPOOLNAME,PID,DOCID
GenerateUniqueFile=on	"%s_%s.%d",SPOOLNAME,PID,DOCID
FileCreationMode=ALL and GenerateUniqueFile=off	"%s",SPOOLNAME
PageOutput=on (or when using the command line option -po)	"%s_%s.%ld.%ld",SPOOLNAME,PID,DOCID,PAGEID

-fp:<path>

Path to the Type 1 and TrueType fonts

-h or -?

Display help information (Command line options)

-iASCII

Enter this option to scan files and to determine the best values to take for the command line options -mc and -mr. Only for ASCII as output format.

-if:<infile>

Path and file name of INI file

-ip:<path>

Path to afp2web.ini

-jq:n

Jpeg Quality where n can be 1 to 100. 1 highest conversion, lowest quality, 100 is lowest conversion, highest quality. Recommended: 50

-l

Log file (Part 1: Parsing-section, Part 2: Builder-section with font information)

Note: If -ll:O follows, it will switch off -l. To avoid this, use -l as the last option or avoid -ll:O.

-lf

Log file (Part 2: Builder-section with font information)

Note: If -ll:O follows, it will switch off -lf. To avoid this, use -lf as the last option or avoid -ll:O.

-ll:<nn>

Logging level. For <n>, specify a value out of:

0	No log (Default)
1	Structured Fields (SFI)
2	Data information of SFI
3	Repeating Groups
4	Patterns (like image)
5	FOCA information
6	Mapping.def entries added to the end of the log file. These entries can be used as template to modify the mapping.def
8	FTP Logging
ALL	All information including FOCA
FONT	Builder part, font information (same as LoggingFont=on)
Resource Logging Levels:	
RES	all resource types

`-lp:<path>`

	or any of the following:
CDP	Code Pages
FDEF	Formdefs
FNTR	Fonts (detailed list of all fonts used for current spool file)
IOB	Object containers
MMAP	Medium Maps
OLY	Overlays
PSEG	Page Segments
Image Logging Levels	
INFO	Log image processing informational messages
WARN	Log image processing warning and informational messages
DEBUG	Log warning, informational and debug messages during image processing

Note: If `-lf` is entered after `-ll:0`, then `-lf` will take effect.

`-lp:<path>`

Path to write the log file.

`-mc:<max columns>`

`-mr:<max rows>`

Maximum number of columns and rows. For ASCII only. Limits the page size.

`-nocmyktorgb`

By default, the INI file parameter CMYKTORGB is on - that is, CMYK color planes are converted to RGB for PDF. Using this command line option it can be turned off.

Note: We recommend using this option only for print documents. CMYK for PDF results in large PDF files. You should apply the standard conversion to RGB if you need documents for online viewing.

-op:<path>

Path to write output files.

-os:<width>,<height>

Size of the resulting output document specified as width and height in pixels. For example, to produce thumbnail images for document pages.

Default: No default - the document original size is used.

-ovp:<path>

Path to Overlay resources.

It is also possible to specify a comma-separated list of paths.

Example: samples/resource,c:\afp\overlays

-p

The “protocol” is a processing log listing documents converted from spool file. Documents are identified by document ID. This option is only possible if DOC_INDEX (or DOC_COLD) is selected.

-plw:<n>

Minimum line width that has to be used on output to ensure that lines are printable. Specified in a unit of pixels. Applies when converting to AFP.

Example: PrintableLineWidth=1

Default: 1

-po

Create one output file for each page of an output document. Applies only to TIFF as output format.

-pp:[fp][-tp]

(Process from/to pages)

-pr:<resolution>

Applies only to TIFF: Resolution in dpi. We recommend one of the following values:

- pr:200
- pr:240
- pr:300

-ps:[<Owner Password>],[<User Password>][,Key Length[,Permission Flags]]

PDF Security options

The parameters are fixed-positioned and have the following meanings:

Owner Password	Optional Owner Password. Used to change security options
User Password	Optional, User password, required when opening the document.
Key Length	Optional, Optional. Encryption Level Key Length. Possible is a multiple of 8 between 40 and 128 40 => Normal Encryption, 128 => Better Encryption Standard is 40.
Permission Flags	Optional. One or more flags separated by " " specified below. For Key Length 40, Flags 5,6,7,8 are not allowed

The following values are allowed as Permission Flags:

0	None, everything is disabled (Default)
1	Allow Low resolution printing
2	Allow Changing the document

3	Allow Content copying or extraction
4	Allow Authoring comments and form fields
5	Allow Form field fill-in or signing
6	Allow Content accessibility
7	Allow Document assembly
8	Allow High Quality Printing

-psp:<path>

Path to Page Segment resources.

It is also possible to specify a comma-separated list of paths.

Example: samples/resource,c:\afp\pagesegments

-

pui:[<HideMenubar>][,<HideToolbar>][,<HideWindowUI>]

PDF viewer preferences for the "User Interface"

For each parameter, specify either "on" or "off" (without the quotes):

HideMenubar on = Hide menu bar (The default is off).

HideToolbar on = Hide tool bar (The default is off).

HideWindowUlon = Hide window controls (The default is off).

Note: An empty value is allowed and default value is assumed for empty values. For example, -pui:,,on,on is equivalent to -pui:off,on,on.

-pv

Preview option. Launches Adobe Reader (for PDF) or the appropriate ASCII, TIFF or JPEG viewer to display the resulting output file.

-

pwn:[FitWindow][,<CenterWindow>][,<FullScreen

Window>] [,<DisplayDocTitle>]

PDF viewer preferences "Window" options
For each parameter, specify either "on" or "off" (without the quotes):

FitWindow	on = Resize the document's window to fit the size of the first displayed page. (The default is off.)
CenterWindow	on = Position the document's window in the center of the screen. (The default is off.)
FullScreenWindow	on = Display the document in full-screen mode, with no menu bar, window controls, or any other window visible. (The default is off.)
DisplayDocTitle	on = Display the document description property Title in the window's title bar. off = Display PDF filename in the window's title bar. The default is off.

***Note:** An empty value is allowed and the default is assumed for empty values. For example, -pwin:.,on,on,on is equivalent to -pwin:off,on,on,on.*

-q

Suppress messages to system console

-r:<rotation>

Page rotation clockwise, in degrees:

0
90
180
270
portrait
landscape

-rf:<resourcefile>

File with AFP resources.

-rp:<path>

Path to AFP resources.

-S

Write a statistic report (stat*.txt)

-sa

Used for AFP2web Scripting Facility: Arguments to pass to the scripting procedure.

-sd:<docid>

First document to convert in a spool file. Enter the document ID as determined in the "protocol".

This option is only possible if DOC_INDEX (or DOC_COLD) is selected

-si

Read from stdin (only for AFP)

-so

Write to stdout (only for AFP)

-sod

Save Object Container Data on disk.

-sp

Flag to cache the object container data temporarily to the file path given by INI file parameter "TempPath". This option is used to save memory and to optimize the output file size.

-sp

Used for AFP2web Scripting Facility: The scripting procedure file name

-std

Both options `-si` and `-so`

-Strict

Stop processing if AFP resource cannot be found.

-tp:<path>

Path for temporary files.

AFP2web must have read/write access to the directory specified in this path. Otherwise, AFP2web will fail when trying to create and delete temporary files.

-u

Generate unique file names. This parameter is maintained for backward compatibility. For this version of AFP2web it is no longer required and we even recommend not to use it.

-V

Display version of AFP2web

-vall

Display versions of all AFP2web components

-wm:<text, typeface,height,rotation,redmask,greenmask, bluemask>

**-wm:<text,
typeface,height,rotation,redmask,greenmask,
bluemask>**

Displays a text as a watermark behind the document text. The parameters that can be specified are:

text	text to display
typeface	Font typeface
height	Font size in tenths of points. Specify 90 for 9 points.
rotation	Rotation. A value from 0 to 360.
redmask	RGB Red mask value from 0 to 255
greenmask	RGB Green mask value from 0 to 255
bluemask	RGB Blue mask value from 0 to 255

Example:

`-wm: "Confidential , Helvetica, 600, 60, 189, 219, 231"`

Default: no watermark

-xf:<afpindex>

Input file with the AFP index.

-xp:<path>

Folder for index files.

5.4 Parameters of the mapping.def

Function and Structure

The file mapping.def is a text file located in the code pages folder afpcp/ created during installation. This file is used to control font mapping.

Before we describe the details, here is the overview of the sections and their general purpose:

Specifies the mode for processing fonts:

[CHARSET RENDERING]	Specifies the font processing mode for each AFP Character Set (Font Rasterizing, Substitution/Referencing, or Embedding).
----------------------------	---

The sections, which define defaults for AFP fonts. Entries are only required if AFP2web cannot derive the information from the AFP input:

[XFONT]	AFP Character Set, AFP Code Page
[CHARSET]	AFP Character Set, Font Global ID
[FGID]	Font Global ID, AFP font name

In the following sections, each AFP font name is mapped to the respective substitution font. You can specify a list of fonts. AFP2web uses these as alternatives if it cannot find a particular font on the system:

[PDFFONT]	Font to be used for conversion to PDF.
[WINFONT]	Fonts to be used for conversion to raster output (TIFF/JPEG/...). These fonts must be installed in the Windows system

[UNIXFONT]	Fonts to be used for conversion to raster output (TIFF/JPEG/...). These fonts must be installed in the Unix system
[FONTSUFFIX]	Naming convention for font files.

Specifies the font to use for output to AFP:

[AFPFONT]	Maps PC fonts to AFP character sets and codepages
------------------	---

The following section maps codes pages and character sets:

[CODEPG]	Maps AFP code pages to code page files
-----------------	--

Usage Notes

You customize this file by adding your entries to the appropriate sections under ";user-defined". Doing this overrides the entries under ";standard". If you define rules, the first rule which applies will be executed, all succeeding are ignored.

Please note the following when you change the mapping.def:

- A comment line begins with a semicolon (";") as its first character.
- Each section has entries under "user defined" and "standard". An entry under "user defined" will override an existing entry under "standard".

Important: Do not change any entries under „standard“!

- If you make the AFP font resources available to AFP2web, entries in the sections [XFONT], [CHARSET], [FGID] are no longer necessary.
- Any entry in the mapping.def must not exceed 1024 (1 Kb) characters.

[CHARSET RENDERING] Section

Syntax, Example

Syntax:

charsetname=RenderingFlag, option
Example: COH000A0=2

Function

Used to specify processing flags for AFP character sets.

Parameters

Parameter	Description
charsetname	AFP Character Set. If required, the AFP Character Set is defined in the section [CHARSET] of the mapping.def. <i>Note: Wild card characters are allowed while specifying character set names in [CHARSET RENDERING] section of mapping.def (Please see the usage notes below).</i>
RenderingFlag	One of the following values: 0 Font Rasterizing (Default) Font processing is based on the original fonts. This is the default font processing mode. 1 Font Referencing/Substituting (Use Substitution Font) <ul style="list-style-type: none">• PDF: Reference the 14 standard fonts for PDF or the supported external font• TIFF: Force writing text using the substitution font 2 Font Embedding (Use Substitution Font) <ul style="list-style-type: none">• PDF: Embed the substitution font into the PDF file• TIFF: Not applicable

<i>Parameter</i>	<i>Description</i>
option	<p>Additional option for the conversion from PDF to AFP. Only valid for RenderingFlag=0.</p> <p>Note: Options annotated as “for future use” are planned for a later version of AFP2web.</p> <p>A value out of:</p> <ul style="list-style-type: none"> 0 Use original font (default). AFP2web converts PDF fonts as they are provided from the PDF input. Type 3 fonts are converted to AFP raster fonts for future use: Type 1 and TrueType fonts are converted to AFP outline fonts 1 Inline Image: AFP2web converts and rasterizes PDF Type 1 and TrueType fonts and creates IOCA images. 2 for future use: AFP raster font: AFP2web converts Type 1 and TrueType fonts to AFP raster fonts. 3 for future use: AFP outline font: AFP2web converts Type 1 and TrueType fonts to AFP outline fonts.

Usage

When using the rendering flag 0 for Font Rasterizing (this is the default and uses the original font), the result when converting to PDF will be an embedded font subset.

We recommend using a more recent version of the Adobe Acrobat Reader (at least Version 7) to view PDF files.

Using Wildcards for Character Sets

The following wild card characters are allowed while specifying character set names in the [CHARSET RENDERING] section of mapping.def

<i>Wild card character</i>	<i>Meaning</i>
*	Match 1 or more sequence of characters
?	Match exactly one character

Note: Please use wildcards carefully. AFP2web processes the entries in the section [CHARSET RENDERING] sequentially, top-down. The first match will take effect.

Example:

```
[CHARSET RENDERING]
; Syntax: <Charset Name>=<Rendering Flag>
; Rendering Flag=>Values
; 0=>Rasterize
; 1=>Reference / Substitute
; 2=>Embed
;
C?H400B0=1
C?H400*=2
```

Result:

- Case 1: The character set "C0H400B0" matches both patterns "C?H400B0=1", i.e. "C<Any character>H400B0" and "C?H400*=2", i.e. "C<Any character>H400<Any string>". AFP2web will take the first pattern with the rendering flag 1.
- Case 2: The character set "C4H40090" matches the second pattern "C?H400*=2", i.e. "C<Any character>H400<Any string>". AFP2web will take the second pattern with the rendering flag 2.

[XFONT] Section

Syntax, Example

Syntax:

`xfont = character set, codepage`

Example: X0GB02=C0D0GB10,T1V10273

Function

Specifies the AFP character set and the AFP code page for a coded font.

Parameters

<i>Parameter</i>	<i>Description</i>
xfont	Coded font (See the notes below.)
character set	AFP Character Set (See the notes below.) This AFP Character Set must be defined in the section [CHARSET] of the mapping.def.
codepage	AFP code page This AFP code page must be defined in the section [CODEPG] of the mapping.def.

Notes on xfont, character set

The second character in the name of a coded font or character set indicates either raster font (either 0 or 1) or outline font (only Z). For example, a raster font is indicated if the name of a coded font begins with X0... or X1..., resp. the name of a character set begins with C0... or C1...

The following rules apply:

- If the second character is ?, AFP2web looks only for a raster font - not for an outline font.
- If the second character is either 0 or 1 (raster font), AFP2web will look for the defined value. If not found AFP2web will also look for the alternative value 0 or 1.
- The second character must be Z, if you want AFP2web to look for an outline font.

Usage

You need to add entries to this section if you use your own coded fonts or if you modify existing coded fonts.

Please put your entries under „user-defined“. Do not change any entries under „standard“.

The result in PDF will be an embedded font subset.

We recommend using an up-to-date version of the Adobe Acrobat Reader (Version 7) to view PDF files.

[CHARSET] Section

Syntax, Example

Syntax:

`charset = fgid, height, width, strikeover, underline`

Example: `COD0GB10=39,120,144,0,0`

Function

Assigns AFP character set to a global font identifier.

Sets additional font attributes (height, width)

Parameters

Parameter	Description
charset	AFP Character Set
fgid	Font Global ID Must be defined in the section [FGID]
height	Average height in tenths of a point A value within the range 1 to 999 Example: Enter 90 to specify 9 points.
width	Average width in tenths of a point. A value within the range 1 to 999 Example: Enter 90 for a width of 9 points. (This parameter is reserved for future use.)

<i>Parameter</i>	<i>Description</i>
strikeover	Draws a horizontal line through the middle of the character 0 no 1 strikeover
underline	Underlines the character. 0 no 1 underline

Usage

You need to add entries to this section if you use your own AFP character sets or if you modify existing AFP character sets.

Please put your entries under „user-defined“. Do not change any entries under „standard“.

[FGID] Section

Syntax, Example

Syntax:

```
fgid = logicalname, style, weight, italic
Example: 39 = GOTHIC ,MODERN,BOLD,0
```

Function

Assigns a font global identifier to a logical font name. Specifies additional font attributes (weight, italic).

Parameters

<i>Parameter</i>	<i>Description</i>
fgid	Font Global ID
logicalname	AFP logical font name. Must be defined in the section [PDF-FONT], resp. [WINFONT] or [UNIXFONT].

<i>Parameter</i>	<i>Description</i>
style	Style: SWISS Proportional, without Serifs ROMAN Proportional, with Serifs SCRIPT mono-spaced, decorative ‘handwriting’ MODERN mono-spaced, with/without Serifs DISPLAY decorative Only used for TIFF under Windows
weight	Weight: BOLD bold MED normal LIGHT thin For the current release of AFP2web, only BOLD or MED are valid.
italic	0 straight, no italic 1 italic (oblique)

Usage

Please put your entries under „user-defined“. Do not change any entries under „standard“.

[FONTSUFFIX] Section

Syntax, Example

Syntax:

`<Style>=<suffix1>[, <suffix2> . . , <suffixN>]`
Example: BoldItalicSuffix=bi,b,-BoldItalic ,-BoldOblique

Function

Used to specify suffixes added to the name of font files.

Because file naming conventions vary for fonts, AFP2web will assume a default file naming convention. Suffixes are normally added to the file names of substitution fonts and are normally taken from additional font attributes, such as bold, italic. You can use this section to define the suffixes which are typically used on your system.

Note: Suffixes are case-sensitive under Unix.

Parameters

<i>Parameter</i>	<i>Description</i>
Style	One of the following values: BoldSuffix ItalicSuffix BoldItalicSuffix

Usage

For example, for the font name: Verdana with the font attributes: BoldItalic, we specify BoldItalicSuffix=b,-BoldItalic. Based on this definition, AFP2web first looks for Verdanab.* and then for Verdana-BoldItalic.* in the specified font directories (pdffont\ as default). If there are more than one file (such as Verdana-BoldItalic.pfm, Verdana-BoldItalic.ttf), the first file found is used.

[PDFFONT] Section

Syntax, Example

Syntax:

Logical name = Font, Alias_1, Alias_2, . . . , Alias_n

Example: GOTHIC=Gothic,GothicText,CourierNew

Function

This section controls the font substitution for the conversion to PDF. The section [WINFONT] or [UNIXFONT] controls the font substitution for the conversion to TIFF.

Assigns fonts and alternative fonts to an AFP logical font name.

AFP2web searches through this list from left to right and takes the first font it can find in the system.

Parameters

<i>Parameter</i>	<i>Description</i>
logicalname	AFP logical font name References a value in the section [FGID]
Font	Font name (PostScript Type 1 font).
Alias_1, Alias_2,..., Alias_n	Additional fonts to look for.

Usage

Please put your entries under „user-defined“. Do not change any entries under „standard“.

Specific entries are required to reference the 14 standard fonts for PDF. Other Postscript Type 1 fonts can be either embedded or referenced. The fonts must be provided and they must have specific file names.

[WINFONT] Section

Syntax, Example

Syntax:

```
Logicalname = Font, Alias_1, Alias_2, . . . , Alias_n
```

Example: GOTHIC=Gothic,Gothic Text,Courier New

Function

This section controls the font substitution for the conversion to TIFF under Windows. The section [PDFFONT] controls the font substitution for the conversion to PDF.

Assigns fonts and alternative fonts to an AFP logical font name.

AFP2web searches through this list from left to right and takes the first font it can find in the Windows system.

Parameters

<i>Parameter</i>	<i>Description</i>
logicalname	AFP logical font name References a value in the section [FGID]
Font	Name of a font family. Take the name as displayed in the Windows control panel „fonts“.
Alias_1, Alias_2,..., Alias_n	Additional fonts to be used as alternatives.

Usage

Please put your entries under „user-defined“. Do not change any entries under „standard“.

[UNIXFONT] Section

Syntax, Example

Syntax:

`logical name = Font, Alias_1, Alias_2, . . . , Alias_n`

Example: `GOTHIC=gothic,courier`

Function

This section controls the font substitution for the conversion to TIFF under Unix. The section [PDFFONT] controls the font substitution for the conversion to PDF.

Assigns fonts and alternative fonts to an AFP logical font name.

AFP2web searches through this list from left to right and takes the first font it can find in the Unix system.

Parameters

<i>Parameter</i>	<i>Description</i>
logicalname	AFP logical font name References a value in the section [FGID]
Font	Name of a font family.
Alias_1, Alias_2,..., Alias_n	Additional fonts to be used as alternatives.

Usage

Please put your entries under „user-defined“. Do not change any entries under „standard“.

[AFPFONT] Section

Syntax, Example

Syntax:

```
typeface[-weight][style]=  
AFPOutlineCodedFont|AFPOutlineCharacterSet, AFPCodePage
```

Example:

```
CourierNew=CZ4200, T1001141  
CourierNew-Italic=CZ4300, T1001141  
CourierNew-Oblique=CZ4300, T1001141  
CourierNew-Bold=CZ4400, T1001141  
CourierNew-BoldItalic=CZ4500, T1001141
```

Function

This section is used for converting to AFP.
Specifies for each typeface the substitution font to use for AFP.

Parameters

<i>Parameter</i>	<i>Description</i>
typeface	Name of the typeface
weight	Either: Bold or Light
style	Either: Oblique or Italic
AFPOutlineCodedFont	Name of the AFP Outline Coded Font
AFPOutlineCharacterSet	Name des AFP Outline Character Set
AFPCodePage	Name des AFP Code Page

Usage

Please put your entries under „user-defined“. Do not change any entries under „standard“.

[CODEPG] Section

Syntax, Example

Syntax:

```
codepage = cpgid, wincp  
Example: T1V10273=273,ANSI
```

Function

Assigns an AFP code page to a Windows ANSI code page.

Parameters

Parameter	Description
codepage	AFP code page
cpgid	Name of the code page (specify the file name without the file extension ".cp") For ASCII-encoded AFP data enter 819 to specify the standard map file 819.cp
wincp	One of the Windows character sets: ANSI Standard character set SYMBOL NULL Built-in encoding

Usage

If you create your own AFP code pages or if you modify existing AFP code pages you must:

- Create the respective code page (<codepage ID>.cp) and place this file into the code page directory (afpcp/) used by AFP2web. Please consult Maas High Tech Software GmbH for more information.
- Add a new entry in the section [CODEPG] of the MAPPING.DEF.

Please put your entries under „user-defined“. Do not change any entries under „standard“.

5.5 Code Page Files (CP Files)

A code page file (CP file) maps each Graphic Character Global Identifier (GCGID) to an EBCDIC, ASCII and Unicode code point. These mappings are used for the following cases:

- Unicode is used when converting original fonts from AFP to PDF. Encoding makes PDF files available for a full-text search.
- ASCII is the encoding used by the Scripting Facility when parsing AFP resource names, AFP index elements, and is used as the base encoding for the Scripting Facility.
- When substituting fonts, codes are mapped properly to preserve the textual data. For example, AFP fonts have EBCDIC codes, PC fonts normally have ASCII codes. The CP files map EBCDIC to ASCII code points.
- Your AFP font resources might not use the standard Graphic Character Global Identifiers (GCGIDs). Thus, the CP files can also be used to define the logical AFP names for the characters in your AFP Character Set and their character codes.

You add a new custom CP file only if you need to define custom code pages or if your AFP data does not use the standard IBM names to identify characters in your AFP character sets (using the Graphic Character Global Identifiers GCGID).

The CP files have an additional column, which you can use to map a GCGID to the name of a particular glyph in your font. The glyph name is the one defined in the font file.

To clarify the character codes of your fonts, please consult your font vendor.

The format of CP file (which uses the file extension *.cp):

```
<GCGID> <EBCDICValue> <UNICODEValue> <ASCIIValue>[<GlyphName>]
Where
```

- GCGID is the name used to uniquely identify the character.
- The EBCDIC value and the ASCII value are given as two digit hexadecimal values.
- The Unicode value is given as 4 digit hexadecimal value.
- (Optional) The glyph name is the one defined in the font file.

Example:

```
LA020000 C1 0041 41
LB020000 C2 0042 42
LC020000 C3 0043 43
; ohungaruml aut
L0250000 CF 0151 F5 ohungaruml aut
```

If Unicode values are missing in a code page file (CP file), AFP2web takes the default from `gcid2unicode.def`. This file is located in the code pages folder `afpcp`, which is created during installation. `gcid2unicode.def` maps IBM standard GCGIDs to Unicode. For a definition of GCGIDs, please refer to the Technical Reference for IBM Expanded Core Fonts (S544-5228-01).

5.6 Font Logging Output

You use the INI-parameter **LoggingLevel=FONT** or **LoggingFont=on** or the command line option **-lf** to produce a detailed list of all fonts used for the current spool file.

Format of the Font Logging Message

Input Fonts

Information about the input font starts with the character sequence "==" , as shown in the following example:

```
*==>[CF=<Coded Font Name>,[ CS=<Char Set Name>,[ FGID=<FGID>,[
TF=<Type Face Name>,[ W=M|L|B|I ], S=<Si ze>,[ SF|OT|TT|T1|RF|OF|,
CP=<CodePageName> | CPGID=<Code Page Global ID>],[
ENC=<Encoding>]*
```

Output Fonts

Information about the output font starts with the character sequence "--" , as shown in the following example:

```
*-->[CF=<Coded Font Name>,[ CS=<Char Set Name>,[ FGID=<FGID>,[
TF=<Type Face Name>,[ W=M|L|B|I ], S=<Si ze>,[ SF|OT|TT|T1|RF|OF|RI [,
CP=<CodePageName> | CPGID=<Code Page Global ID>],[ ACP=<CP
Source>: <CP File>],[ ENC=<Encoding>]*
```

Text to Which the Font Applies

The text to which the font applies is enclosed in angled brackets: > text < , as shown in the following example:

```
>some text <
>some text <
```

Codes Used in the Font Logging Messages

The codes used in the font logging messages and their meanings:

<i>Code</i>	<i>Description</i>
ACP	ASCII Code Page. "ACP" says the CP file that is used for AFP code page and how the CP file name is obtained. It also reports the encoding used.
CF	AFP Coded Font name. Valid only for AFP as either input or output font. For AFP as input font: Written only if input font type is "Coded Font".
CP	AFP Code Page. Valid only for AFP. For AFP as input: Written only if input font type is "Coded Font" or "Character Set", not in case of "FGID". For AFP as output: Written only if output font type is "Coded Font" or "Character Set".
CP File	Name of ASCII code page (*.cp) file
CP Source	CP Source says where from CP file is obtained. The following codes are possible: MD CP File obtained from code page name specific Mapping.def entry CP-DSC CP File obtained from "Code Page Global ID" specified either in "Code Page Descriptor" SFI or in "Map Coded Font" SFI NC CPFile obtained from Naming Convention (last four letters of code page name, left most zeros will be truncated) MD-DEF CPFile obtained from "DEFAULT" Mapping.def entry INI CPFile obtained from INI File default "CodePage" Parameter PG-DEF Program Default, AFP2web predefined EBCDIC-TO-ASCII translation array is used for mapping
CPGID	AFP Code Page Global ID. Written only if input font type is "FGID". Valid only for AFP input

<i>Code</i>	<i>Description</i>
CS	Character Set. AFP Character Set name. Valid only for AFP input or output. For AFP as input font: Written only if input font type is "Coded Font" or "Character Set" not in case of "FGID".
Emb	Font is embedded, used along with OT, TT and T1
ENC	<p>Encoding used</p> <p>For PDF input:</p> <p>ANSI Windows ANSI encoding</p> <p>ROMAN Macintosh ROMAN encoding</p> <p>EXPERT Macintosh EXPERT encoding</p> <p>STANDARD Adobe STANDARD encoding</p> <p>SYMBOL Symbol encoding</p> <p>NULL Font's built-in encoding</p> <p>CUSTOM PDF creator defined encoding</p> <p>Identity-H Standard Adobe CMAP table</p> <p>For PDF and TIFF output:</p> <p>ANSI Windows ANSI encoding</p> <p>SYMBOL Symbol encoding</p> <p>NULL Font's default built-in encoding</p> <p>For AFP output:</p> <p>ASCII ASCII encoding</p> <p>EBCDIC EBCDIC encoding</p>
FGID	AFP Font Global ID. Written when input font type is "Coded Font" or "Character Set" or "FGID". Valid only for AFP input
OF	Font type is AFP Outline font
OT	Font type is Open Type
Ref	Font is referenced, used along with OT, TT and T1
RF	Font type is AFP Raster font

<i>Code</i>	<i>Description</i>
RI	Text Rasterized as Inline image
S	Font size (height) in points
SF	Font type is Standard or System (for the input font). For PDF output: Standard Font For TIFF output: System Font
T1	Font type is Type 1
TF	Font typeface name.
TT	Font type is True Type
UsedFont	For PDF and TIFF output: Name of used font For AFP output: Coded font (and or) Character Set name
W	Font weight and slant, it can have following possible values: M (Medium), L (Light), B (Bold), I (Italic). Examples: W=B means font weight is bold W=MI means font weight is medium and slant is italic

Note: whenever an AFP font resource is missing, an asterisk (*) will be added as a suffix to the resource name. Example: CS=C1H400B0(*), TF=...

Examples of Font Logging Messages

Log for Raster Fonts

PDF output example:

```
==>CF=X0A00550, CS=COA07580, FGID=2308, TF=SONORAN SANS SERIF,
W=M, S=8.00, RF, CP=T1GI0361
-->UsedFont=COA07580, TF=F1, W=M, S=8.00, RF, UsedCP=MD: 2065.cp,
Enc=ANSI
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
Note: "F1" is the font name obtained from the mpdf library, and font is referenced using
this name within the PDF file.
```

TIFF output example:

```
==>CF=X0A00550, CS=COA07580, FGID=2308, TF=SONORAN SANS SERIF,
W=BI, S=8.00, RF, CP=CP=NULL
-->UsedFont=COA07580, TF=SONORAN SANS SERIF, W=BI, S=8.00, RF,
UsedCP=MD: 2065.cp, Enc=ANSI
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

AFP output (Coded Font):

```
==>CF=X0A00550, CS=NULL, FGID=2308, TF=SONORAN SANS SERIF, W=M,
S=8.00, RF, CP=NULL
-->CF=X0A00550, CS=NULL, FGID=2308, TF= SONORAN SANS SERIF, W=M,
S=8.00, CF,
CP= NULL, ACP=1141.cp, Enc=EBCDIC
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

AFP output (Character Set):

```
==>CF=NULL, CS=COA07580, FGID=2308, TF=SONORAN SANS SERIF, W=M,
S=8.00, RF, CP=T1GI0361
-->CF=NULL, CS=COA07580, FGID=2308, TF= SONORAN SANS SERIF, W=M,
S=8.00, CS, CP=T1GI0361, ACP=1141.cp, Enc=EBCDIC
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

Log for Outline Fonts

PDF/TIFF output:

```
==>CF=X0A00050, CS=COA00050, FGID=2308, TF=SONORAN SANS SERIF,
W=B, S=8.00, OF, CP=T1VI0500
```

```
-->UsedFont=COA00050, TF=SONORAN SANS SERIF, W=B, S=8.00, OF,
UsedCP=NC: 500.cp, Enc=ANSI
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

AFP output (Coded Font):

```
==>CF=X0A00050, CS=NULL, FGID=2308, TF=SONORAN SANS SERIF, W=B,
S=8.00, OF, CP=NULL
-->CF=NULL, CS=COA07580, FGID=2308, TF= SONORAN SANS SERIF, W=M,
S=8.00, CP=NULL, ACP=1141.cp, Enc=EBCDIC
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

AFP Output (Character Set):

```
==>CF=X0A00550, CS=COA07580, FGID=2308, TF=SONORAN SANS SERIF,
W=M, S=8.00, OF, CP=T1GI0361
-->CF=NULL, CS=COA07580, FGID=2308, TF= SONORAN SANS SERIF, W=M,
S=8.00, CP=T1GI0361, ACP=1141.cp, Enc=EBCDIC
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

Log for Font Referencing

CASE 1: Referencing one of the 14 standard fonts for PDF output, or system fonts for TIFF output

PDF/TIFF output:

```
==>CF=X0H400B0, CS=C1H400B0, FGID=2308, TF=HELVETICA LATIN1, W=B,
S=8.00, RF, CP=T1VI0500
-->UsedFont=Helvetica-Bold, TF=Helvetica-Bold, W=B, S=8.00, SF,
UsedCP=NC: 500.cp, Enc=ANSI
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

CASE 2: Referencing external Type1/TrueType font files

PDF/TIFF output:

```
==>CF=X0H410DC, CS=C0H410DC, FGID=2308, TF=Optima, W=B, S=8.00,
RF, CP=T1VI0500
-->UsedFont=C:\winnt\Optima-Bold.pfm, TF=Optima-Bold, W=B,
S=8.00, T1, Ref, UsedCP=NC: 500.cp, Enc=ANSI
>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <
```

CASE 3: Referencing external AFP fonts

AFP output (Coded Font)

```
==>CF=X0H400B0, CS=C1H400B0, FGID=2308, TF=HELVETICA LATIN1, W=B,
S=8.00, RF, CP=T1VI0500
```

```
-->UsedFont= X0H400B0, TF= HELVETICA LATIN1, W=B, S=8.00, CF, Ref,  
Enc=EBCDIC  
>Vos paiements sont acceptés à la plupart des banques ou <  
>N° de compte <
```

AFP output (Character Set)

```
==>CF=X0H400B0, CS=C1H400B0, FGID=2308, TF=HELVETICA LATIN1, W=B,  
S=8.00, RF, CP=T1VI0500  
-->UsedFont= C1H400B0, TF= HELVETICA LATIN1, W=B, S=8.00, CS, Ref,  
CP= T1VI0500, Enc=EBCDIC  
>Vos paiements sont acceptés à la plupart des banques ou <  
>N° de compte <
```

Log for Font Embedding

PDF output:

```
==>CF=X0H410DC, CS= C0H410DC, FGID=2308, TF=Optima, W=B, S=8.00,  
RF, CP=T1VI0500  
-->UsedFont=C:\winnt\Optima-Bold.pfb, TF=Optima-Bold, W=B,  
S=8.00, T1, Emb, UsedCP=NC:500.cp, Enc=ANSI  
>Vos paiements sont acceptés à la plupart des banques ou <  
>N° de compte <
```

For TIFF output, font embedding is not possible, TIFF output is therefore always treated as “Font Referencing”.

AFP output is not handled.

Log for FGID Font

PDF/TIFF output:

```
==>FGID=2307, TF=Helvetica, W=BI, S=8.00, CPGID=500  
-->UsedFont=Helvetica-Bold-Italic, TF=Helvetica-Bold-Italic,  
W=BI, S=8.00, SF, UsedCP=CPGID:500.cp, Enc=ANSI  
>Vos paiements sont acceptés à la plupart des banques ou <  
>N° de compte <
```

AFP output is not handled.

Log for Text added through the Scripting Facility

PDF/TIFF output:

```
==>TF=Helvetica, W=BI, S=8.00  
-->UsedFont=Helvetica-Bold-Italic, TF=Helvetica-Bold-Italic,  
W=BI, S=8.00, SF, Enc=ANSI
```

>Vos paiements sont acceptés à la plupart des banques ou <
>N° de compte <

5.7 Identifying Fonts Using the IBM Naming Convention

Overview

If you do not provide the AFP font resources Character Set and Coded Font in-line, AFP2web will take the settings of the mapping.def to identify the substitution fonts to be used for the conversion.

If no information is found in the mapping.def, AFP2web will derive font characteristics from the naming conventions for these resources. Thus, AFP2web will still be able to map AFP fonts to substitution fonts.

In this section, we describe the rules for this automatic font mapping. Please note that we follow conventions, which are specific to IBM.

The AFP Font Naming Convention

The General Structure

An AFP Character Set or Coded Font has a name with up to 8 characters. Each character has the following information encoded:

A	F	R	S	T	C	P	X
----------	----------	----------	----------	----------	----------	----------	----------

	<i>Meaning</i>	<i>Used by AFP2web</i>
A	Component	Yes
F	Format or orientation	Ignored
R	Type family	Yes
S	Style	Yes
T	Weight	Yes
C	Complement.	Ignored
P	Point size	Yes
X	Used to determine the code page.	Ignored

For example, the name of AFP resource COT07560 has the following information encoded:

A=C	The resource is a Character Set.
F=0	(Ignored)
R=T	The type family is Sonoran Serif.
S=0	Style is Roman.
T=7	Weight is Bold.
C=5	(Ignored)
P=6	Point size 6.
X=0	(Ignored)

The encoding rules are summarized in the following tables.

A: Component

A	Component
C	Character Set
X	Coded Font
A	Component

R: Type family

For regular fonts R is one the following:

R	Type Family	AFP2web Default Type Family for PDF	AFP2web Default Type Family for UNIX TIFF	AFP2web Default Type Family for Windows TIFF
4	Courier	Courier	courier	Courier
5	Letter Gothic	Courier	courier	Courier
6	Gothic Text	Courier	courier	Courier
7	Prestige	Courier	courier	Courier
8	Boldface	Times-Roman	times	Times New Roman
9	OCR	Courier	courier	Courier
B	BookMaster	Courier	courier	Courier
H	Helvetica	Helvetica	helvetica	Helvetica
N	Times New Roman	Times-Roman	times	Times New Roman

For licensed fonts, R is one the following:

R	Type Family	AFP2web Default Type Family for PDF	AFP2web Default Type Family for UNIX TIFF	AFP2web Default Type Family for Windows TIFF
2	Proprinter Emulation	Helvetica	helvetica	Helvetica
A	Sonoran Sans Serif	Helvetica	helvetica	Helvetica

<i>R</i>	<i>Type Family</i>	<i>AFP2web Default Type Family for PDF</i>	<i>AFP2web Default Type Family for UNIX TIFF</i>	<i>AFP2web Default Type Family for Windows TIFF</i>
B	Bar Code	Helvetica	helvetica	Helvetica
C	Century School- book	NewCenturySc hbk	new century schoolbook	Century School- book
G	Monotype Garamond	Courier	courier	Courier
J	Sonoran Dis- play	Times-Roman	times	Times New Roman
M	Mathematic and Science	Symbol	symbol	Symbol
O	Optical Character Rec- ognition	Courier	courier	Courier
P	Pi Sans Serif	Helvetica	helvetica	Helvetica
Q	Pi Serif	Times-Roman	times	Times New Roman
S	ITC Souvenir	Helvetica	helvetica	Helvetica
T	Sonoran Serif	Times-Roman	times	Times New Roman
V	ITC Avant Garde Gothic	Helvetica	helvetica	Helvetica
Z	Sonoran Petite	Times-Roman	times	Times New Roman

S: Style

<i>S</i>	<i>Style</i>	<i>AFP2web Default Style</i>
0	Roman	Medium
1	Italic	Italic
2	Roman Medium	Medium
3	Italic Medium	Italic
4	Roman Bold	Bold
5	Italic Bold	Italic Bold
6	Roman Medium Reverse	Medium

T: Weight

For licensed fonts T is one the following (for regular fonts T is ignored):

<i>T</i>	<i>Weight</i>	<i>AFP2web Default Weight: TIFF output under Windows</i>	<i>AFP2web Default Weight: PDF and TIFF output under Unix</i>
1	Ultralight	Light	Medium
2	Extralight	Light	Medium
3	Light	Light	Medium
4	Semilight	Light	Medium
5	Medium	Medium	Medium
6	Semibold	Bold	Bold
7	Bold	Bold	Bold
8	Extrabold	Bold	Bold
9	Ultrabold	Bold	Bold

P: Point size

<i>P</i>	<i>Point Size (points)</i>	<i>P</i>	<i>Point Size (points)</i>
4	4	K	21
5	5	L	22

<i>P</i>	<i>Point Size (points)</i>	<i>P</i>	<i>Point Size (points)</i>
6	6	M	23
7	7	N	24
8	8	O	25
9	9	P	26
0	10	Q	27
A	11	R	28
B	12	S	29
C	13	T	30
D	14	U	31
E	15	V	32
F	16	W	33
G	17	X	34
H	18	Y	35
I	19	Z	36
J	20		

5.8 Scripting Facility Quick Reference

Scripting Facility Interface and Processing Levels

The Scripting Facility offers an interface used to intercept AFP2web processing at particular processing events. A custom script must have the following sections defined:

- sub afp2web()
- sub initialize()
- sub initializeDoc()
- sub initializePage()
- sub finalizePage()
- sub finalizeDoc()
- sub finalize()

In the following tables, a quick-reference shows which API method is available in which scripting subroutine.

Script Routine and Methods

<i>Script Routine</i>	<i>Package</i>	<i>Method</i>
initialize	a2w::Config	getAttribute, getIndexFilePath, getOutputFilePath, getScriptArgs, setAttribute, setAutoSplit
	a2w::Font	getEncoding, getHeight, getName, getWidth, isBold, isItalic, isLight, isStrikeover, isUnderline, new, setBold, setEncoding, setFamilyName, setHeight, setItalic, setLight, setName, setStrikeover, setUnderline, setWidth
	a2w::Kernel	getIndexFilename, getResourceFilename, getSpoolFilename
	a2w::Page	new
initializeDoc	a2w::Config	getAttribute, getIndexFilePath, getOutputFilePath, getScriptArgs
	a2w::Document	addBookmark, addIndex, addNOP, addPage, getFirstIndex, getID, getName, getNextIndex, getOutputFilename, getPID, getSimpleFilename, setName, setOutputFilename
	a2w::Font	getEncoding, getHeight, getName, getWidth, isBold, isItalic, isLight, isStrikeover, isUnderline, new, setBold, setEncoding, setFamilyName, setHeight, setItalic, setLight, setName, setStrikeover, setUnderline, setWidth
	a2w::Index	getEBDICName, getEBDICValue, getIndexedObjectName, getIndexedObjectType, getLevelNumber, getName, getNameLength, getSequenceNumber, getValue, getValueLength, new, remove, setLevelNumber, setName, setSequenceNumber, setValue
	a2w::Kernel	getIndexFilename, getResourceFilename, getSpoolFilename
	a2w::NOP	getEBDICValue, getLength, getValue, new, remove, setValue
	a2w::Page	new
initializePage	a2w::Config	getAttribute, getIndexFilePath, getOutputFilePath, getScriptArgs

<i>Script Routine</i>	<i>Package</i>	<i>Method</i>
	a2w::Document	addBookmark, addIndex, addNOP, addPage, getFirstIndex, getId, getName, getNextIndex, getOutputFilename, getPID, getSimpleFilename, setName, setOutputFilename
	a2w::Font	getEncoding, getHeight, getName, getWidth, isBold, isItalic, isLight, isStrikeover, isUnderline, new, setBold, setEncoding, setFamilyName, setHeight, setItalic, setLight, setName, setStrikeover, setUnderline, setWidth
	a2w::Index	getEBCDICName, getEBCDICValue, getIndexedObjectName, getIndexedObjectType, getLevelNumber, getName, getNameLength, getSequenceNumber, getValue, getValueLength, new, remove, setLevelNumber, setName, setSequenceNumber, setValue
	a2w::Kernel	getIndexFilename, getResourceFilename, getSpoolFilename
	a2w::Line	getColor, getLength, getWidth, getXPos, getYPos, isHorizontal, isNegative, isVertical, new, remove, setColor, setHorizontal, setLength, setNegative, setVertical, setWidth, setXPos, setYPos
	a2w::MediumMap	getDuplexControl, getFormdefName, getHeight, getName, getNupControl, getResolution, getWidth
	a2w::NOP	getEBCDICValue, getLength, getValue, new, remove, setValue
	a2w::Overlay	getFirstLine, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getIncludedXPosition, getIncludedYPosition, getName, getNextLine, getNextOverlay, getNextPageSegment, getNextText, getResolution, getWidth
	a2w::PSEG	getIncludedXPosition, getIncludedYPosition, getName

<i>Script Routine</i>	<i>Package</i>	<i>Method</i>
	a2w::Page	addAnnotation, addBookmark, addImage, addIndex, addLine, addNOP, addOverlay, addPSEG, addText, getAppliedMediumMap, getFirstIndex, getFirstLine, getFirstMediumOverlay, getFirstNOP, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getName, getNextIndex, getNextLine, getNextMediumOverlay, getNextNOP, getNextOverlay, getNextPageSegment, getNextText, getOutputFilename, getPageGroupName, getParsed, getResolution, getSimpleFilename, getText, getType, getWidth, isConstantBack, isOverlayFound, new, setBackgroundColor, setHeight, setOutputFilename, setResolution, setWidth
	a2w::Text	getAngle, getColor, getEBCDICText, getFont, getMappedFontLocalId, getText, getTextLen, getXPos, getYPos, new, remove, setAngle, setColor, setFont, setText, setTextLen, setXPos, setYPos
afp2web	a2w::Config	getAttribute, getIndexFilePath, getOutputFilePath, getScriptArgs
	a2w::Document	addBookmark, addIndex, addNOP, addPage, getFirstIndex, getId, getName, getNextIndex, getOutputFilename, getPID, getSimpleFilename, setName, setOutputFilename
	a2w::Font	getCharacterSetName, getCodedFontName, getCodePageName, getEncoding, getHeight, getName, getTypefaceName, getWidth, isBold, isItalic, isLight, isStrikeover, isUnderline, new, setBold, setEncoding, setFamilyName, setHeight, setItalic, setLight, setName, setStrikeover, setUnderline, setWidth
	a2w::Index	getEBCDICName, getEBCDICValue, getIndexedObjectName, getIndexedObjectType, getLevelNumber, getName, getNameLength, getSequenceNumber, getValue, getValueLength, new, remove, setLevelNumber, setName, setSequenceNumber, setValue
	a2w::Kernel	getIndexFilename, getResourceFilename, getSpoolFilename
	a2w::Line	getColor, getLength, getWidth, getXPos, getYPos, isHorizontal, isNegative, isVertical, new, remove, setColor, setHorizontal, setLength, setNegative, setVertical, setWidth, setXPos, setYPos

<i>Script Routine</i>	<i>Package</i>	<i>Method</i>
	a2w::MediumMap	getDuplexControl, getFormdefName, getHeight, getName, getNupControl, getResolution, getWidth
	a2w::NOP	getEBCDICValue, getLength, getValue, new, remove, setValue
	a2w::Overlay	getFirstLine, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getIncludedXPosition, getIncludedYPosition, getName, getNextLine, getNextOverlay, getNextPageSegment, getNextText, getResolution, getWidth
	a2w::PSEG	getIncludedXPosition, getIncludedYPosition, getName
	a2w::Page	addAnnotation, addBookmark, addImage, addIndex, addLine, addNOP, addOverlay, addPSEG, addText, getAppliedMediumMap, getFirstIndex, getFirstLine, getFirstMediumOverlay, getFirstNOP, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getName, getNextIndex, getNextLine, getNextMediumOverlay, getNextNOP, getNextOverlay, getNextPageSegment, getNextText, getOutputFilename, getPageGroupName, getParsed, getResolution, getSimpleFilename, getText, getType, getWidth, isConstantBack, isOverlayFound, new, setBackgroundColor, setHeight, setOutputFilename, setResolution, setWidth
	a2w::Text	getAngle, getColor, getEBCDICText, getFont, getMappedFontLocalId, getText, getTextLen, getXPos, getYPos, new, remove, setAngle, setColor, setFont, setText, setTextLen, setXPos, setYPos
finalizePage	a2w::Config	getAttribute, getIndexFilePath, getOutputFilePath, getScriptArgs
	a2w::Document	addBookmark, addIndex, addNOP, addPage, getFirstIndex, getId, getName, getNextIndex, getOutputFilename, getPID, getSimpleFilename, setName, setOutputFilename
	a2w::Font	getCharacterSetName, getCodedFontName, getCodePageName, getEncoding, getHeight, getName, getTypefaceName, getWidth, isBold, isItalic, isLight, isStrikeover, isUnderline, new, setBold, setEncoding, setFamilyName, setHeight, setItalic, setLight, setName, setStrikeover, setUnderline, setWidth

<i>Script Routine</i>	<i>Package</i>	<i>Method</i>
	a2w::Index	getEBDICName, getEBDICValue, getIndexedObjectName, getIndexedObjectType, getLevelNumber, getName, getNameLength, getSequenceNumber, getValue, getValueLength, new, remove, setLevelNumber, setName, setSequenceNumber, setValue
	a2w::Kernel	getIndexFilename, getResourceFilename, getSpoolFilename
	a2w::Line	getColor, getLength, getWidth, getXPos, getYPos, isHorizontal, isNegative, isVertical, new, remove, setColor, setHorizontal, setLength, setNegative, setVertical, setWidth, setXPos, setYPos
	a2w::MediumMap	getDuplexControl, getFormdefName, getHeight, getName, getNupControl, getResolution, getWidth
	a2w::NOP	getEBDICValue, getLength, getValue, new, remove, setValue
	a2w::Overlay	getFirstLine, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getIncludedXPosition, getIncludedYPosition, getName, getNextLine, getNextOverlay, getNextPageSegment, getNextText, getResolution, getWidth
	a2w::PSEG	getIncludedXPosition, getIncludedYPosition, getName
	a2w::Page	addAnnotation, addBookmark, addImage, addIndex, addLine, addNOP, addOverlay, addPSEG, addText, getAppliedMediumMap, getFirstIndex, getFirstLine, getFirstMediumOverlay, getFirstNOP, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getName, getNextIndex, getNextLine, getNextMediumOverlay, getNextNOP, getNextOverlay, getNextPageSegment, getNextText, getOutputFilename, getPageGroupName, getParsed, getResolution, getSimpleFilename, getText, getType, getWidth, isConstantBack, isOverlayFound, new, setBackgroundColor, setHeight, setOutputFilename, setResolution, setWidth
	a2w::Text	getAngle, getColor, getEBDICText, getFont, getMappedFontLocalId, getText, getTextLen, getXPos, getYPos, new, remove, setAngle, setColor, setFont, setText, setTextLen, setXPos, setYPos

<i>Script Routine</i>	<i>Package</i>	<i>Method</i>
finalizeDoc	a2w::Config	getAttribute, getIndexFilePath, getOutputFilePath, getScriptArgs
	a2w::Document	getFirstIndex, getFirstPage, getId, getName, getNextIndex, getNextPage, getOutputBuffer, getOutputBufferLength, getOutputFilename, getPageCount, getPID, getSimpleFilename, getSize
	a2w::Font	getCharacterSetName, getCodedFontName, getCodePageName, getEncoding, getHeight, getName, getTypefaceName, getWidth, isBold, isItalic, isLight, isStrikeover, isUnderline
	a2w::Index	getEBCDICName, getEBCDICValue, getIndexedObjectName, getIndexedObjectType, getLevelNumber, getName, getNameLength, getSequenceNumber, getValue, getValueLength
	a2w::Kernel	getIndexFilename, getResourceFilename, getSpoolFilename
	a2w::Line	getColor, getLength, getWidth, getXPos, getYPos, isHorizontal, isNegative, isVertical
	a2w::MediumMap	getDuplexControl, getFormdefName, getHeight, getName, getNupControl, getResolution, getWidth
	a2w::NOP	getEBCDICValue, getLength, getValue
	a2w::Overlay	getFirstLine, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getIncludedXPosition, getIncludedYPosition, getName, getNextLine, getNextOverlay, getNextPageSegment, getNextText, getResolution, getWidth
	a2w::PSEG	getIncludedXPosition, getIncludedYPosition, getName
	a2w::Page	getAppliedMediumMap, getFirstIndex, getFirstLine, getFirstMediumOverlay, getFirstNOP, getFirstOverlay, getFirstPageSegment, getFirstText, getHeight, getId, getName, getNextIndex, getNextLine, getNextMediumOverlay, getNextNOP, getNextOverlay, getNextPageSegment, getNextText, getOutputBuffer, getOutputBufferLength, getOutputFilename, getPageGroupName, getParseld, getResolution, getSimpleFilename, getSize, getText, getType, getWidth, isConstantBack, isOverlayFound

<i>Script Routine</i>	<i>Package</i>	<i>Method</i>
	a2w::Text	getAngle, getColor, getEBCDICText, getFont, getMappedFontLocalId, getText, getTextLen, getXPos, getYPos
finalize	a2w::Config	getAttribute, getIndexFilePath, getOutputFilePath, getScriptArgs
	a2w::Kernel	getIndexFilename, getResourceFilename, getSpoolFilename

Methods and Where Used in a Script

<i>a2w::Config Methods</i>	<i>Available in Script Routines</i>
getAttribute	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc, finalize
getIndexFilePath	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc, finalize
getOutputFilePath	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc, finalize
getScriptArgs	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc, finalize
setAttribute	initialize
setAutoSplit	initialize

<i>a2w::Document Methods</i>	<i>Available in Script Routines</i>
addBookmark	initializeDoc, initializePage, afp2web, finalizePage
addIndex	initializeDoc, initializePage, afp2web, finalizePage
addNOP	initializeDoc, initializePage, afp2web, finalizePage
addPage	initializeDoc, initializePage, afp2web, finalizePage
getFirstIndex	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getFirstPage	finalizeDoc
getId	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getName	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getNextIndex	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getNextPage	finalizeDoc
getOutputBuffer	finalizeDoc
getOutputBufferLength	finalizeDoc
getOutputFilename	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getPageCount	finalizeDoc

<i>a2w::Document Methods</i>	<i>Available in Script Routines</i>
getPID	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getSimpleFilename	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getSize	finalizeDoc
setName	initializeDoc, initializePage, afp2web, finalizePage
setOutputFilename	initializeDoc, initializePage, afp2web, finalizePage

<i>a2w::Font Methods</i>	<i>Available in Script Routines</i>
getCharacterSetName	afp2web, finalizePage, finalizeDoc
getCodedFontName	afp2web, finalizePage, finalizeDoc
getCodePageName	afp2web, finalizePage, finalizeDoc
getEncoding	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getHeight	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getName	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getTypefaceName	afp2web, finalizePage, finalizeDoc
getWidth	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
isBold	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
isItalic	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
isLight	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
isStrikeover	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
isUnderline	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
new	initialize, initializeDoc, initializePage, afp2web, finalizePage
setBold	initialize, initializeDoc, initializePage, afp2web, finalizePage
setEncoding	initialize, initializeDoc, initializePage, afp2web, finalizePage

<i>a2w::Font Methods</i>	<i>Available in Script Routines</i>
setFamilyName	initialize, initializeDoc, initializePage, afp2web, finalizePage
setHeight	initialize, initializeDoc, initializePage, afp2web, finalizePage
setItalic	initialize, initializeDoc, initializePage, afp2web, finalizePage
setLight	initialize, initializeDoc, initializePage, afp2web, finalizePage
setName	initialize, initializeDoc, initializePage, afp2web, finalizePage
setStrikeover	initialize, initializeDoc, initializePage, afp2web, finalizePage
setUnderline	initialize, initializeDoc, initializePage, afp2web, finalizePage
setWidth	initialize, initializeDoc, initializePage, afp2web, finalizePage

<i>a2w::Index Methods</i>	<i>Available in Script Routines</i>
getEBCDICName	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getEBCDICValue	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getIndexedObjectName	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getIndexedObjectType	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getLevelNumber	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getName	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getNameLength	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getSequenceNumber	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getValue	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getValueLength	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
new	initializeDoc, initializePage, afp2web, finalizePage

<i>a2w::Index Methods</i>	<i>Available in Script Routines</i>
remove	initializeDoc, initializePage, afp2web, finalizePage
setLevelNumber	initializeDoc, initializePage, afp2web, finalizePage
setName	initializeDoc, initializePage, afp2web, finalizePage
setSequenceNumber	initializeDoc, initializePage, afp2web, finalizePage
setValue	initializeDoc, initializePage, afp2web, finalizePage

<i>a2w::Kernel Methods</i>	<i>Available in Script Routines</i>
getIndexFilename	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc, finalize
getResourceFilename	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc, finalize
getSpoolFilename	initialize, initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc, finalize

<i>a2w::Line Methods</i>	<i>Available in Script Routines</i>
getColor	initializePage, afp2web, finalizePage, finalizeDoc
getLength	initializePage, afp2web, finalizePage, finalizeDoc
getWidth	initializePage, afp2web, finalizePage, finalizeDoc
getXPos	initializePage, afp2web, finalizePage, finalizeDoc
getYPos	initializePage, afp2web, finalizePage, finalizeDoc
isHorizontal	initializePage, afp2web, finalizePage, finalizeDoc
isNegative	initializePage, afp2web, finalizePage, finalizeDoc
isVertical	initializePage, afp2web, finalizePage, finalizeDoc
new	initializePage, afp2web, finalizePage
remove	initializePage, afp2web, finalizePage
setColor	initializePage, afp2web, finalizePage
setHorizontal	initializePage, afp2web, finalizePage
setLength	initializePage, afp2web, finalizePage
setNegative	initializePage, afp2web, finalizePage
setVertical	initializePage, afp2web, finalizePage
setWidth	initializePage, afp2web, finalizePage
setXPos	initializePage, afp2web, finalizePage
setYPos	initializePage, afp2web, finalizePage

<i>a2w::MediumMap Methods</i>	<i>Available in Script Routines</i>
getDuplexControl	initializePage, afp2web, finalizePage, finalizeDoc
getFormdefName	initializePage, afp2web, finalizePage, finalizeDoc
getHeight	initializePage, afp2web, finalizePage, finalizeDoc
getName	initializePage, afp2web, finalizePage, finalizeDoc
getNupControl	initializePage, afp2web, finalizePage, finalizeDoc
getResolution	initializePage, afp2web, finalizePage, finalizeDoc
getWidth	initializePage, afp2web, finalizePage, finalizeDoc

<i>a2w::NOP Methods</i>	<i>Available in Script Routines</i>
getEBCDICValue	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getLength	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
getValue	initializeDoc, initializePage, afp2web, finalizePage, finalizeDoc
new	initializeDoc, initializePage, afp2web, finalizePage
remove	initializeDoc, initializePage, afp2web, finalizePage
setValue	initializeDoc, initializePage, afp2web, finalizePage

<i>a2w::Overlay Methods</i>	<i>Available in Script Routines</i>
getFirstLine	initializePage, afp2web, finalizePage, finalizeDoc
getFirstOverlay	initializePage, afp2web, finalizePage, finalizeDoc
getFirstPageSegment	initializePage, afp2web, finalizePage, finalizeDoc
getFirstText	initializePage, afp2web, finalizePage, finalizeDoc
getHeight	initializePage, afp2web, finalizePage, finalizeDoc
getIncludedXPosition	initializePage, afp2web, finalizePage, finalizeDoc
getIncludedYPosition	initializePage, afp2web, finalizePage, finalizeDoc
getName	initializePage, afp2web, finalizePage, finalizeDoc
getNextLine	initializePage, afp2web, finalizePage, finalizeDoc
getNextOverlay	initializePage, afp2web, finalizePage, finalizeDoc
getNextPageSegment	initializePage, afp2web, finalizePage, finalizeDoc
getNextText	initializePage, afp2web, finalizePage, finalizeDoc
getResolution	initializePage, afp2web, finalizePage, finalizeDoc

<i>a2w::Overlay Methods</i>	<i>Available in Script Routines</i>
getWidth	initializePage, afp2web, finalizePage, finalizeDoc

<i>a2w::PSEG Methods</i>	<i>Available in Script Routines</i>
getIncludedXPosition	initializePage, afp2web, finalizePage, finalizeDoc
getIncludedYPosition	initializePage, afp2web, finalizePage, finalizeDoc
getName	initializePage, afp2web, finalizePage, finalizeDoc

<i>a2w::Page Methods</i>	<i>Available in Script Routines</i>
addAnnotation	initializePage, afp2web, finalizePage
addBookmark	initializePage, afp2web, finalizePage
addImage	initializePage, afp2web, finalizePage
addIndex	initializePage, afp2web, finalizePage
addLine	initializePage, afp2web, finalizePage
addNOP	initializePage, afp2web, finalizePage
addOverlay	initializePage, afp2web, finalizePage
addPSEG	initializePage, afp2web, finalizePage
addText	initializePage, afp2web, finalizePage
getAppliedMediumMap	initializePage, afp2web, finalizePage, finalizeDoc
getFirstIndex	initializePage, afp2web, finalizePage, finalizeDoc
getFirstLine	initializePage, afp2web, finalizePage, finalizeDoc
getFirstMediumOverlay	initializePage, afp2web, finalizePage, finalizeDoc
getFirstNOP	initializePage, afp2web, finalizePage, finalizeDoc
getFirstOverlay	initializePage, afp2web, finalizePage, finalizeDoc
getFirstPageSegment	initializePage, afp2web, finalizePage, finalizeDoc
getFirstText	initializePage, afp2web, finalizePage, finalizeDoc
getHeight	initializePage, afp2web, finalizePage, finalizeDoc
getId	finalizeDoc
getName	initializePage, afp2web, finalizePage, finalizeDoc
getNextIndex	initializePage, afp2web, finalizePage, finalizeDoc
getNextLine	initializePage, afp2web, finalizePage, finalizeDoc
getNextMediumOverlay	initializePage, afp2web, finalizePage, finalizeDoc
getNextNOP	initializePage, afp2web, finalizePage, finalizeDoc

<i>a2w::Page Methods</i>	<i>Available in Script Routines</i>
getNextOverlay	initializePage, afp2web, finalizePage, finalizeDoc
getNextPageSegment	initializePage, afp2web, finalizePage, finalizeDoc
getNextText	initializePage, afp2web, finalizePage, finalizeDoc
getOutputBuffer	finalizeDoc
getOutputBufferLength	finalizeDoc
getOutputFilename	initializePage, afp2web, finalizePage, finalizeDoc
getPageGroupName	initializePage, afp2web, finalizePage, finalizeDoc
getParseId	initializePage, afp2web, finalizePage, finalizeDoc
getResolution	initializePage, afp2web, finalizePage, finalizeDoc
getSimpleFilename	initializePage, afp2web, finalizePage, finalizeDoc
getSize	finalizeDoc
getText	initializePage, afp2web, finalizePage, finalizeDoc
getType	initializePage, afp2web, finalizePage, finalizeDoc
getWidth	initializePage, afp2web, finalizePage, finalizeDoc
isConstantBack	initializePage, afp2web, finalizePage, finalizeDoc
isOverlayFound	initializePage, afp2web, finalizePage, finalizeDoc
new	initialize, initializeDoc, initializePage, afp2web, finalizePage
setBackgroundColor	initializePage, afp2web, finalizePage
setHeight	initializePage, afp2web, finalizePage
setOutputFilename	initializePage, afp2web, finalizePage
setResolution	initializePage, afp2web, finalizePage
setWidth	initializePage, afp2web, finalizePage

<i>a2w::Text Methods</i>	<i>Available in Script Routines</i>
getAngle	initializePage, afp2web, finalizePage, finalizeDoc
getColor	initializePage, afp2web, finalizePage, finalizeDoc
getEBCDICText	initializePage, afp2web, finalizePage, finalizeDoc
getFont	initializePage, afp2web, finalizePage, finalizeDoc
getMappedFontLocalId	initializePage, afp2web, finalizePage, finalizeDoc
getText	initializePage, afp2web, finalizePage, finalizeDoc
getTextLen	initializePage, afp2web, finalizePage, finalizeDoc

<i>a2w::Text Methods</i>	<i>Available in Script Routines</i>
getXPos	initializePage, afp2web, finalizePage, finalizeDoc
getYPos	initializePage, afp2web, finalizePage, finalizeDoc
new	initializePage, afp2web, finalizePage
remove	initializePage, afp2web, finalizePage
setAngle	initializePage, afp2web, finalizePage
setColor	initializePage, afp2web, finalizePage
setFont	initializePage, afp2web, finalizePage
setText	initializePage, afp2web, finalizePage
setTextLen	initializePage, afp2web, finalizePage
setXPos	initializePage, afp2web, finalizePage
setYPos	initializePage, afp2web, finalizePage

5.9 Scripting Facility API Reference

Overview

Conventions used in this document

- “[NEW]” indicates a new module or interface.
- “[DEPRECATED]” indicates that a module or interface, which is no longer recommended. It will be removed in the future. The same functionality can be achieved using another new generic interface.
- “[MODIFIED]” indicates a change to an existing module or interface.
- “[EXTENDED]” indicates additional functionality.

afp2web.pm

The module afp2web.pm is the gateway script of the AFP2web Scripting Facility and this module will be loaded and executed by the AFP2web Kernel (using an embedded PERL interpreter). The description below lists the interfaces exposed by the module afp2web.pm and which can be used to communicate with the AFP2web Kernel.

afp2web

Parameter: None

Return value Integer
data type:

Remarks: Called while processing (after binding resources) each output page. At this stage, it is possible to analyse page content and, by setting the appropriate return code, to instruct the AFP2web Kernel to change its flow.

One of the following return values are possible:

Return value	Description
<0	A negative value indicates an error to AFP2web. When returning an error code to AFP2web, it is also possible to pass an error message for a more descriptive error report.
0	Append page to current document
1	Skip page
2	First page / New document

Example:

```
sub afp2web(){
$APPEND = 0;# append page to Current Document
$SKIP = 1;# skip page
$NEWDOC = 2;# new document
$svRetTmp = $APPEND; # default: append page
#---- Do something
...
#---- Return value
if ( $svProcessOk ){
return $svRetTmp;#---- Success
} else {
return (-1, "afp2web failed" );#---- Error
}
}
```

finalize

Parameter: None

Return value Integer
data type:

Remarks: Called after processing each spool. This routine is used for spool finalizing.

The return value must be zero or greater than zero. A negative value indicates an error to AFP2web. When returning an error code to AFP2web, it is also possible to pass an error message for a more descriptive error report.

Example:

```
sub finalize (){
#---- Do something
...
#---- Return value
if ( $svProcessOk ){
return 0;#---- Success
} else {
return (-1, "afp2web failed" );#---- Error
}
}
```

finalizeDoc

Parameter: None

**Return value
data type:** Integer

Remarks: Called after processing each output document. This routine is used for document finalizing.

The return value must be zero or greater than zero. A negative value indicates an error to AFP2web. When returning an error code to AFP2web, it is also possible to pass an error message for a more descriptive error report.

Example:

```
sub finalizeDoc(){
#---- Do something
...
#---- Return value
if ( $svProcessOk ){
return 0; #---- Success
} else {
return (-1, "afp2web failed" ); #---- Error
}
}
```

finalizePage

Parameter: None

**Return value
data type:** Integer

Remarks: Called after processing each output page. This routine is used for page finalizing.

The return value must be zero or greater than zero. A negative value indicates an error to AFP2web. When returning an error code to AFP2web, it is also possible to pass an error message for a more descriptive error report.

Example:

```
sub finalizePage(){
#---- Do something
...
#---- Return value
if ( $svProcessOk ){
return 0;#---- Success
} else {
return (-1, "afp2web failed" );#---- Error
}
}
```

initialize

Parameter: Configuration instance of type a2w::Config
Kernel instance of type a2w::Kernel

Return value Integer
data type:

Remarks: Called at the beginning of each input spool. Using a configuration instance, it is possible to read and modify configuration parameters (which were entered in INI file or as command line options). Using an instance of the kernel object, it is possible to read information about the input spool.

The return value must be zero or greater than zero. A negative value indicates an error to AFP2web. When returning an error code to AFP2web, it is also possible to pass an error message for a more descriptive error report.

Example:

```
sub initialize(){
#---- Get parameters of initialize ( a2w::Config,
a2w::Kernel )
( $a2wConfigPar, $a2wKernelPar ) = @_ ;
#---- Do something
...
#---- Return value
if ( $svProcessOk ){
return 0;#---- Success
} else {
return (-1, "Initialization failed" );#---- Error
}
}
```

initializeDoc

Parameter: Document instance of type a2w::Document

Return value Integer
data type:

Remarks: Called at the beginning of each output document process, the user can initialize output document dependent processes.

The return value must be zero or greater than zero. A negative value indicates an error to AFP2web. When returning an error code to AFP2web, it is also possible to pass an error message for a more descriptive error report.

Example:

```
sub initializeDoc(){
#---- Get parameters of initializeDoc ( a2w::Document )
( $a2wDocumentPar ) = @_;
#---- Do something
...
#---- Return value
if ( $svProcessOk ){
return 0;#---- Success
} else {
return (-1, "InitializeDoc failed" );#---- Error
}
}
```

initializePage

Parameter: Page instance of type a2w::Page

Return value Integer
data type:

Remarks: Called at the beginning of each output page process, user can initialize output page dependent processes.

The return value must be zero or greater than zero. A negative value indicates an error to AFP2web. When returning an error code to AFP2web, it is also possible to pass an error message for a more descriptive error report.

Example:

```
sub initializePage(){
#---- Get parameters of initializePage ( a2w::Page )
( $a2wPagePar ) = @_;
#---- Do something
...
#---- Return value
if ( $svProcessOk ){
return 0;#---- Success
} else {
return (-1, "InitializePage failed" );#---- Error
}
}
```

a2w::Config (mhtIni [DEPRECATED])

The a2w::Config object wraps the AFP2web Kernel object mhtIni and exposes the following interfaces to the AFP2web Scripting Facility.

getAttribute [NEW]

Parameter: Attribute name of type string

**Return value
data type:** String

Remarks: Gets the current value of the INI attribute.

Example:

```
$svIndexPathTmp = $a2wConfig->getAttribute(
    "IndexPath" );
$svOutputFilePathTmp = $a2wConfig->getAttribute(
    "OutputFilePath" );
$svScriptArgsTmp = $a2wConfig->getAttribute(
    "ScriptArgument" );
```

getIndexPath [DEPRECATED]

Parameter: None

**Return value
data type:** String

Remarks: Gets the path specified for the index file.
Note: This method is deprecated. Please use the method "getAttribute".

Example:

```
$svIndexPathTmp = $a2wConfig->getIndexPath();
```

getOutputFilePath [DEPRECATED]

Parameter: None

**Return value
data type:** String

Remarks: Gets the path specified for the output file.
Note: This method is deprecated. Please use the method "getAttribute".

Example: `$svOutputFilePathTmp = $a2wConfig->getOutput-
FilePath();`

getScriptArgs [DEPRECATED]

Parameter: None

**Return value
data type:** String

Remarks: Gets the arguments passed to the script.
Note: This method is deprecated. Please use the method "getAttribute".

Example: `$svScriptArgsTmp = $a2wConfig->getScriptArgs();`

setAttribute [NEW]

Parameter: Attribute name of type string
Attribute value of type string

**Return value
data type:** Void

Remarks: Sets the INI attribute (the processing parameter, which is originally set in the configuration file called the INI file).

Example: `$a2wConfig->setAttribute("Quietmode", "on");`

setAutoSplit [DEPRECATED]

Parameter: Autosplit of type boolean

**Return value
data type:** Void

Remarks: Setting Autosplit to true means that document splitting will be done based on the standard AFP index elements. In this case, a return value of 2 of the `afp2web()` function of the AFP2web Scripting Facility module is ignored.

Setting Autosplit to false means that document splitting will be defined by the `afp2web()` function.

Note: This method is deprecated. Please use the method "setAttribute" with the key "AutoSplit" to set AutoSplit to "on" or "off".

Example: `$a2wConfig->setAutoSplit(1);`

a2w::Document (mhtDocument [DEPRECATED])

a2w::Document object wraps the AFP2web Kernal object “mhtDocument” and exposes the following interfaces to the Scripting Facility.

addBookmark [NEW]

Parameter: Bookmark name of type string
 Bookmark value of type string

Return value Void
data type:

Remarks: Adds a bookmark to the document.

Example: \$a2wDocument->addBookmark("Software", "AFP2web");

addIndex [DEPRECATED]

Parameter: Index name of type string
 Index value of type string

Return value Void
data type:

Remarks: Adds an index element to the document.
 This method is deprecated. Please use the method “addIndex” of this object.

Example: \$a2wDocument->addIndex("Producer", "Maas");

addIndex [NEW]

Parameter: Index instance of type a2w::Index.

Return value Void
data type:

Remarks: Adds an index element to the document.

Example:

```
#---- Create new index
$a2wIndexTmp = new a2w::Index();
#---- Set index info
$a2wIndexTmp->setName( "Producer" );
$a2wIndexTmp->setValue( "Maas" );
#---- Add index to document
$a2wDocument->addIndex( $a2wIndexTmp );
```

addNOP [NEW]

Parameter: NOP instance of type a2w::NOP

Return value Void
data type:

Remarks: Adds an NOP element to the document. The NOP element is inserted before first BPG of the document.

Example:

```
#---- Create new NOP
$a2wNOPTmp = new a2w::NOP();
#---- Set NOP value
$a2wNOPTmp->setValue( "Created by AFP2web" );
#---- Add NOP to document
$a2wDocument->addNOP( $a2wNOPTmp );
```

addPage [NEW]

Parameter: Page instance of type a2w::Page
Page identifier of the output document of type unsigned integer

Return value Void
data type:

Remarks: Adds a page to the document.

Example:

```
#---- Create new Page
$a2wPageTmp = new a2w::Page();
#---- Add some content
$a2wPageTmp->addText( $a2wTextTmp );
#---- Add Page to document
$a2wDocument->addPage( $a2wPageTmp, 2 );
```

getFirstIndex

Parameter: None

Return value Index instance of type a2w::Index
data type:

Remarks: Gets the first index at the document level.

Example:

```
$a2wIndexTmp = $a2wDocument->getFirstIndex();
while ( $a2wIndexTmp != 0 ){
#---- Access index info
...
#---- Get next index
$a2wIndexTmp = $a2wDocument->getNextIndex();
}
```

getFirstPage

Parameter: None

Return value Page instance of type a2w::Page
data type:

Remarks: Gets the first page of the document.

Example:

```
$a2wPageTmp = $a2wDocument->getFirstPage();
while ($a2wPageTmp != 0 ){
#---- Access page info
...
#---- Get next page
$a2wPageTmp = $a2wDocument->getNextPage();
}
```

getId

Parameter: None

**Return value
data type:** String

Remarks: Gets the document ID.

Example: \$svDocIdTmp = \$a2wDocument->getId();

getName

Parameter: None

**Return value
data type:** String

Remarks: Gets the document name.

Example: \$svDocNameTmp = \$a2wDocument->getName();

getNextIndex

Parameter: None

Return value Index instance of type *a2w::Index*
data type:

Remarks: Gets the next index at the document level. This method is normally used in an iteration loop after first using *getFirstIndex*.

Example:

```
$a2wIndexTmp = $a2wDocument->getFirstIndex();  
while ($a2wIndexTmp != 0 ){  
#---- Access index info  
...  
#---- Get next index  
$a2wIndexTmp = $a2wDocument->getNextIndex();  
}
```

getNextPage

Parameter: None

Return value Page instance of type *a2w::Page*
data type:

Remarks: Gets the next page of the document. This method is normally used in an iteration loop after first using *getFirstPage*.

Example:

```
$a2wPageTmp = $a2wDocument->getFirstPage();  
while ($a2wPageTmp != 0 ){  
#---- Access page info  
...  
#---- Get next page  
$a2wPageTmp = $a2wDocument->getNextPage();  
}
```

getOutputBuffer

Parameter: None

Return value Buffer of type *byte**
data type:

Remarks: Gets the document's output buffer.

Example: `$svDocBufferTmp = $a2wDocument->getOutputBuffer();`

getOutputBufferLength

Parameter: None

**Return value
data type:** Integer

Remarks: Gets the length of the document's output buffer.

Example: `$svDocBufferLenTmp = $a2wDocument->getOutputBufferLength();`

getOutputFilename

Parameter: None

**Return value
data type:** String

Remarks: Gets the name of the output file.

Example: `$svOutputFilenameTmp = $a2wDocument->getOutputFilename();`

getPID

Parameter: None

Return value String
data type:

Remarks: Gets the process ID.

Example: \$svProcessIdTmp = \$a2wDocument->getPID();

getPageCount

Parameter: None

Return value Integer
data type:

Remarks: Gets the document's page count.

Example: \$svPageCountTmp = \$a2wDocument->getPageCount();

getSimpleFilename

Parameter: None

Return value String
data type:

Remarks: Gets the simple file name of the output file.

Example: \$svSimpleFilenameTmp = \$a2wDocument->getSimple-
Filename();

getSize

Parameter: None

**Return value
data type:** Long

Remarks: Gets the size of the output document.

Example: `$svDocSizeTmp = $a2wDocument->getSize();`

setName

Parameter: Document name of type string

**Return value
data type:** Void

Remarks: Sets the name of the document.

Example: `$a2wDocument->setName("MaasSample");`

setOutputFileName

Parameter: Output filename of type string

**Return value
data type:** Void

Remarks: Sets the name of the output file for this document.

Example: `$a2wDocument->setOutputFileName("MaasSample.pdf");`

a2w::Font (*mhtAFPFont* [DEPRECATED])

The *a2w::Font* object wraps the AFP2web Kernel object *mhtAFPFont* and exposes the following interfaces to the AFP2web Scripting Facility.

getCharacterSetName [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Gets the name of the character set.

Example: `$svCharSetNameTmp = $a2wFont->getCharacterSetName();`

getCodedFontName [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Gets the coded font name.

Example: `$svCodedFontNameTmp = $a2wFont->getCodedFontName();`

getCodepageName [NEW]

Parameter: None

Return value String
data type:

Remarks: Gets the name of the mapped code page.

Example: `$svCodePageNameTmp = $a2wFont->getCodePageName();`

getEncoding [NEW]

Parameter: None

Return value String
data type:

Remarks: Returns the font encoding.

Example: `$svFontEncodingTmp = $a2wFont->getEncoding();`

getHeight [NEW]

Parameter: None

Return value Float
data type:

Remarks: Returns the font's height.

Example: `$svFontHeightTmp = $a2wFont->getHeight();`

getName [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Returns the font name.

Example: `$a2wNameTmp = $a2wFont->getName();`

getTypefaceName [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Returns the AFP font type face name.

Example: `$svTypefaceNameTmp = $a2wFont->getTypefaceName();`

getWidth [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the font's width.

Example: `$svFontWidthTmp = $a2wFont->getWidth();`

isBold [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns whether the font's weight is bold or not.

Example: `$svWeightTmp = $a2wFont->isBold();`

isItalic [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns whether the font's slant is italic or not.

Example: `$svSlantTmp = $a2wFont->isItalic();`

isLight [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns whether the font's weight is light or not.

Example: `$svWeightTmp = $a2wFont->isLight();`

isStrikeover [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns whether font is strike over or not.

Example: `$svStrikeOverTmp = $a2wFont->isStrikeover();`

isUnderline [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns whether font is underlined or not

Example: `$svUnderLineTmp = $a2wFont->isUnderLine();`

new

Parameter: None

**Return value
data type:** Font instance of type *a2w::Font*

Remarks: Allocates and returns a new font instance.

Example: `$a2wFont = new a2w::Font();`

setBold

Parameter: Font weight of type boolean

**Return value
data type:** Void

Remarks: Sets the font's weight to bold.

Example: `$a2wFont->setBold(1);`

setEncoding [NEW]

Parameter: Encoding of type string. Possible values are either ANSI, SYMBOL or NULL.

**Return value
data type:** Void

Remarks: Sets the encoding of the font.

Example: `$a2wFont->setEncoding("ANSI");`

setFamilyName

Parameter: Family name of type string

**Return value
data type:** Void

Remarks: Sets the family name of the font.

Example: `$a2wFont->setFamilyName("Latin1");`

setHeight

Parameter: Font height of type float

Return value Void
data type:

Remarks: Sets the height of the font. The value must be in points.
NOTE: On floating height value, only one digit is considered in the precision part. For example, the value "10.768" will be treated as "10.7".

Example: `$a2wFont->setHeight(100);`

setItalic

Parameter: Font slant of type boolean

Return value Void
data type:

Remarks: Sets the font's slant to italic

Example: `$a2wFont->setItalic(1);`

setLight

Parameter: Font weight of type boolean

Return value Void
data type:

Remarks: Sets the font's weight to light.

Example: `$a2wFont->setLight(1);`

setName

Parameter: Font name of type string

Return value Void
data type:

Remarks for Sets the name of a font.
PDF Output: Font name should be one of the standard Type1 fonts as given below.

Times-Roman
Times-Bold
Times-Italic
Times-BoldItalic
Helvetica
Helvetica-Bold
Helvetica-Oblique
Helvetica-BoldOblique
Courier
Courier-Bold
Courier-Oblique
Courier-BoldOblique
Symbol
ZapfDingbats

The default font name is "Helvetica".

Note: Adobe has its own definition for above set of fonts, so they can be used with PDF.

Example for `$a2wFont->setName("Time-Roman");`
PDF Output:

Remarks For TIFF Output: Font name is the typeface name of font. To find out the typeface name of a font follow given steps below appropriate to OS. Default font name is "Helvetica" for both Unix & Windows.

Example for TIFF Output: `$a2wFont->setName("Verdana");`

How to Determine Font Names on Windows:

- Open "Control Panel->Fonts"
- Double click on the required font
- The font window will be displayed showing font information
- For TrueType fonts, use the name displayed for "Typeface name" as parameter to the Scripting Facility method "Font::setName()"
- For Type1 fonts, use the name given on top of the font window as parameter to Scripting Facility method "Font::setName()".

How to Determine Font Names on a Unix Operating System:

- Type "xlsfonts" at the command prompt
prompt\$>xlsfonts
- If issuing this command on the terminal window, then make sure that the "DISPLAY" environment variable is set properly with display ID.
- The "xlsfonts" command will display the names of all available in XLFD (X Logical Font Descriptor) format (as given below):

prompt\$>xlsfonts

-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso10646-1

-adobe-courier-medium-o-normal--10-100-75-75-m-60-iso10646-1

-adobe-helvetica-bold-o-normal--10-100-75-75-p-60-iso10646-1

-adobe-helvetica-medium-o-normal--10-100-75-75-p-57-iso10646-1

- In XLFD format, font attributes are separated by '-' character
- Use the second attribute that passes the typeface name of the font as parameter to setName().

Example of a Font Name on Unix: -adobe-courier-bold-o-normal--10-100-75-75-m-60-iso10646-1
 Note: The text "courier" is the font typeface name.

```
$a2wFont->setName( "courier" );
```

setStrikeover

Parameter: Font strikeover of type boolean

Return value data type: Void

Remarks: Sets the font to strike over.

Example: \$a2wFont->setStrikeover(1);

setUnderline

Parameter: Font underline of type boolean

Return value data type: Void

Remarks: Sets the font to underlined.

Example: \$a2wFont->setUnderline(1);

setWidth

Parameter: Font width of type integer

Return value Void
data type:

Remarks: Sets the width of the font.

Example: `$a2wFont->setWidth(40);`

a2w::Index (mhtAttributeElement, mhtIndexElement, mhtPagePageGroupIndex [DEPRECATED])

The a2w::Index object wraps the AFP2web Kernel object mhtAttributeElement and exposes the following interfaces to the AFP2web Scripting Facility.

getEBCDICName [NEW]

Parameter: None

Return value data type: String

Remarks: Gets the the EBCDIC name of the index.

Example: `$svEBCDICNameTmp = $a2wIndex->getEBCDICName();`

getEBCDICValue [NEW]

Parameter: None

Return value data type: String

Remarks: Gets the index value in EBCDIC.

Example: `$svEBCDICValueTmp = $a2wIndex->getEBCDICValue();`

getIndexedObjectName

Parameter: None

**Return value
data type:** String

Remarks: Gets the name of the indexed object. The indexed object can be an indexed page or page group.

Example: `$svIndexedObjectNameTmp = $a2wIndex->getIndexedObjectName();`

getIndexedObjectType

Parameter: None

**Return value
data type:** Integer

Remarks: Gets the type of the indexed object (page or page group).

Example: `$svIndexedObjectTypeTmp = $a2wIndex->getIndexedObjectTypeName();`

getLevelNumber

Parameter: None

**Return value
data type:** Integer

Remarks: Gets the level number (used to position hierarchically).

Example: `$svLevelNrTmp = $a2wIndex->getLevelNumber();`

getName [MODIFIED]

Parameter: None

**Return value
data type:** String

Remarks: Gets the index name.
Note: This method was previously named "getAttributeName".

Example: `$svNameTmp = $a2wIndex->getName();`

getNameLength [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Gets the the length of the index name. This length is used to interpret the name which is retrieved as EBCDIC string.

Example: `$svNameLengthTmp = $a2wIndex->getNameLength();`

getSequenceNumber

Parameter: None

**Return value
data type:** Integer

Remarks: Gets the sequence number (used to identify identical indexes).

Example: `$svSeqNrTmp = $a2wIndex->getSequenceNumber();`

getValue [MODIFIED]

Parameter: None

**Return value
data type:** String

Remarks: Gets the index value
Note: This method was previously named "getAttributeValue".

Example: `$svValueTmp = $a2wIndex->getValue();`

getValueLength [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Get the length of the index value in EBCDIC.

Example: `$svValueLengthTmp = $a2wIndex->getValueLength();`

new [NEW]

Parameter: None

**Return value
data type:** Index instance of type `a2w::Index`

Remarks: Allocates and returns the index instance.

Example: `$a2wIndexTmp = new a2w::Index();`

remove [NEW]

Parameter: None

**Return value
data type:** None

Remarks: Removes the index from the presentation data.

Example: `$a2wIndex->remove();`

setLevelNumber

Parameter: Level number of type integer

**Return value
data type:** Void

Remarks: Sets the level number (used to position hierarchically).

Example: `$a2wIndex->setLevelNumber(1);`

setName [MODIFIED]

Parameter: Index name of type string

**Return value
data type:** Void

Remarks: Sets the index name.
 Note: This method was previously called "setAttributeName".

Example: `$a2wIndex->setName("Producer");`

setSequenceNumber

Parameter: Sequence number of type integer

Return value Void
data type:

Remarks: Sets the sequence number of the index (used to identify identical indexes).

Example: `$a2wIndex->setSequenceNumber(10);`

setValue [MODIFIED]

Parameter: Index value of type string.

Return value Void
data type:

Remarks: Sets the index value.
Note: This method was previously called as “setAttributeValue”.

Example: `$a2wIndex->setValue("Maas");`

a2w::Kernel [NEW]

The a2w::Kernel object wraps the AFP2web Kernel object mhtAFP2web and exposes the following interfaces to the AFP2web Scripting Facility.

getIndexFilename [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Returns name of the index file used for the current spool.

Example: `$svIndexFilenameTmp = $a2wKernel ->getIndexFilename();`

getResourceFilename [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Returns the name of the resource file used for the current spool.

Example: `$svResFilenameTmp = $a2wKernel ->getResourceFilename();`

getSpoolFilename [NEW]

Parameter: None

Return value data type:	String
Remarks:	Returns the name of the spool file, which is currently being processed.
Example:	<code>\$svSpoolFilenameTmp = \$a2wKernel ->getSpoolFilename();</code>

a2w::Line [NEW]

The “a2w::Line” object wraps the AFP2web Kernel object mhtLineObject and exposes the following interfaces to the AFP2web Scripting Facility.

getColor [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the color of the line.

Example: `$svColorTmp = $a2wLine->getColor();`

getLength [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the line length.

Example: `$svLengthTmp = $a2wLine->getLength();`

getWidth [NEW]

Parameter: None

Return value Integer
data type:

Remarks: Returns the line width.

Example: `$svWidthTmp = $a2wLine->getWidth();`

getXPos [NEW]

Parameter: None

Return value Integer
data type:

Remarks: Returns the X position of the line.

Example: `$svXPosTmp = $a2wLine->getXPos();`

getYPos [NEW]

Parameter: None

Return value Integer
data type:

Remarks: Returns the Y position of the line.

Example: `$svYPosTmp = $a2wLine->getYPos();`

isHorizontal [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns whether the line is horizontal or not.

Example: `$svHorizontalTmp = $a2wLine->isHorizontal();`

isNegative [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns the line is set to negative or not.

Example: `$svNegativeTmp = $a2wLine->isNegative();`

isVertical [NEW]

Parameter: None

**Return value
data type:** Boolean

Remarks: Returns whether the line is vertical or not.

Example: `$svVerticalTmp = $a2wLine->isVertical();`

new [NEW]

Parameter: None

Return value data type: Instance of a line object of type a2w::Line

Remarks: Returns a new allocated line instance.

Example: `$a2wLine = new a2w::Line();`

remove [NEW]

Parameter: None

Return value data type: Void

Remarks: Removes the line from its container, but positioning is still done in the coordinate system.

Example: `$a2wLine->remove();`

setColor [NEW]

Parameter: Color of type integer

Return value data type: Void

Remarks: Sets the color of the line.

Example: `$a2wLine->setColor(0x000000FF);`

setHorizontal [NEW]

Parameter: Horizontal flag of type boolean

**Return value
data type:** Void

Remarks: Sets the horizontal attribute of the line.

Example: `$a2wLine->setHorizontal (1);`

setLength [NEW]

Parameter: Line length of type integer

**Return value
data type:** Void

Remarks: Sets the line length.

Example: `$a2wLine->setLength(150);`

setNegative [NEW]

Parameter: Negative flag of type boolean

**Return value
data type:** Void

Remarks: Sets the negative attribute of the line.

Example: `$a2wLine->setNegative(1);`

setVertical [NEW]

Parameter: Vertical flag of type boolean

**Return value
data type:** Void

Remarks: Sets the vertical attribute of the line.

Example: `$a2wLine->setVertical (1);`

setWidth [NEW]

Parameter: Line width of type integer

**Return value
data type:** Void

Remarks: Sets the line's width.

Example: `$a2wLine->setWidth(2);`

setXPos [NEW]

Parameter: X position of type integer

**Return value
data type:** Void

Remarks: Set the X position of the line.

Example: `$a2wLine->setXPos(100);`

setYPos [NEW]

Parameter: Y position of type integer

Return value Void
data type:

Remarks: Sets the Y position of the line.

Example: `$a2wLine->setYPos(200);`

a2w::MediumMap [NEW]

The a2w::MediumMap object wraps the AFP2web Kernel object mhtMediumMap and exposes the following interfaces to the AFP2web Scripting Facility.,

getDuplexControl [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the duplex control setting, like simple, double or triple.

Example: `$svDupCtrlTmp = $a2wMediumMap->getDuplexControl();`

getFormdefName [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the name of the Formdef.

Example: `$svFormdefNameTmp = $a2wMediumMap->getFormdefName();`

getHeight [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the height of the medium map.

Example: `$svHeightTmp = $a2wMediumMap->getHeight();`

getName [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Returns the name of medium map.

Example: `$svNameTmp = $a2wMediumMap->getName();`

getNupControl [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the N-up control setting and the return value must be in between 1 to 4.

Example: `$svNupCtrlTmp = $a2wMediumMap->getNupControl();`

getResolution [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the resolution of medium

Example: `$svResolutionTmp = $a2wMediumMap->getResolution();`

getWidth [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the width of the medium map.

Example: `$svWidthTmp = $a2wMediumMap->getWidth();`

a2w::NOP (mhtNOP [DEPRECATED])

The a2w::NOP object wraps the AFP2web Kernel object mhtNOP and exposes the following interfaces to the AFP2web Scripting Facility.

getEBCDICValue

Parameter: None

**Return value
data type:** String

Remarks: Gets the NOP's EBCDIC value

Example: `$svEBCDICValueTmp = $a2wNOP->getEBCDICValue();`

getLength

Parameter: None

**Return value
data type:** Integer

Remarks: Gets the NOP's length

Example: `$svLengthTmp = $a2wNOP->getLength();`

getValue

Parameter: None

**Return value
data type:** String

Remarks: Gets the NOP's value.

Example: `$svValueTmp = $a2wNOP->getValue();`

new [NEW]

Parameter: None

**Return value
data type:** NOP instance of type a2w::NOP

Remarks: Returns newly allocated instance of a2w::NOP

Example: `$a2wNOP = new a2w::NOP();`

remove [NEW]

Parameter: None

**Return value
data type:** Void

Remarks: Removes NOP from list of page/document NOPs

Example: `$a2wNOP->remove();`

setValue

Parameter: NOP value of type string

Return value Void
data type:

Remarks: Set nop value

Example: `$a2wNOP->setValue("Created by AFP2web");`

a2w::Overlay (mhtResOverlay [DEPRECATED])

The a2w::Overlay object wraps the AFP2web Kernel object mhtRes-Overlay and exposes the following interfaces to the AFP2web Scripting Facility.

getFirstLine [NEW]

Parameter: None

**Return value
data type:** Line instance of type a2w::Line

Remarks: Returns the first line object of overlay; The return type is a pointer to "mhtLineObject".

Example:

```
$a2wLineTmp = $a2wOverlay->getFirstLine();  
while ( $a2wLineTmp != 0 ){  
#---- Access line info  
...  
#---- Get next line  
$a2wLineTmp = $a2wOverlay->getNextLine();  
}
```

getFirstOverlay [NEW]

Parameter: None

**Return value
data type:** Overlay instance of type a2w::Overlay

Remarks: Returns the first included overlay resource; The return type is a pointer to "mhtResOverlay".

Example:

```

$a2wOverlayTmp = $a2wOverlay->getFirstOverlay();
while $a2wOverlayTmp != 0 ){
#---- Access overlay info
...
#---- Get next overlay
$a2wOverlayTmp = $a2wOverlay->getNextOverlay();
}

```

getFirstPageSegment [NEW]

Parameter: None

Return value data type: Page segment instance of type a2w::PSEG

Remarks: Returns the first included page segment resource; The return type is a pointer to "mhtResPSEG".

Example:

```

$a2wPSEGTmp = $a2wOverlay->getFirstPageSegment();
while $a2wPSEGTmp != 0 ){
#---- Access page segment info
...
#---- Get next page segment
$a2wPSEGTmp = $a2wOverlay->getNextPageSegment();
}

```

getFirstText [MODIFIED]

Parameter: None

Return value data type: Text instance of type a2w::Text

Remarks: Get the first text object of overlay.
Note: This method was previously called "getFirstTextObject".

Example:

```
$a2wTextTmp = $a2wOverlay->getFirstText();  
while ($a2wTextTmp != 0 ){  
#---- Access text info  
...  
#---- Get next text  
$a2wTextTmp = $a2wOverlay->getNextText();  
}
```

getHeight [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the height of the overlay.

Example: \$svHeightTmp = \$a2wOverlay->getHeight();

getIncludedXPosition [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the X position of this overlay included in the page.

Example: \$svInclXPosTmp = \$a2wOverlay->getIncludedXPosition();

getIncludedYPosition [NEW]

Parameter: None

Return value data type: Integer

Remarks: Returns the Y position of this overlay included in the page.

Example: `$svInclYPosTmp = $a2wOverlay->getIncludedYPosition();`

getName

Parameter: None

Return value data type: String

Remarks: Returns the name of the overlay.

Example: `$svNameTmp = $a2wOverlay->getName();`

getNextLine [NEW]

Parameter: None

Return value data type: Line instance of type a2w::Line

Remarks: Returns the next line object of the overlay; The return type is a pointer to "mhtLineObject".

Example:

```
$a2wLineTmp = $a2wOverlay->getFirstLine();
while $a2wLineTmp != 0 ){
#---- Access line info
...
#---- Get next line
$a2wLineTmp = $a2wOverlay->getNextLine();
}
```

getNextOverlay [NEW]

Parameter: None

Return value Overlay instance of type *a2w::Overlay*
data type:

Remarks: Returns the next included overlay resource (used to traverse the list of included overlays); The return type is a pointer to "mhtResOverlay".

Example:

```
$a2wOlyTmp = $a2wOverlay->getFirstOverlay();  
while $a2wOlyTmp != 0 ){  
#---- Access overlay info  
...  
#---- Get next overlay  
$a2wOlyTmp = $a2wOverlay->getNextOverlay();  
}
```

getNextPageSegment [NEW]

Parameter: None

Return value Page segment instance of type *a2w::PSEG*
data type:

Remarks: Returns the next included page segment resource (used to traverse the list of included overlays); The return type is a pointer to "mhtResPSEG".

Example:

```
$a2wPSEGTmp = $a2wOverlay->getFirstPageSegment();  
while $a2wPSEGTmp != 0 ){  
#---- Access page segment info  
...  
#---- Get next page segment  
$a2wPSEGTmp = $a2wOverlay->getNextPageSegment();  
}
```

getNextText [MODIFIED]

Parameter: None

**Return value
data type:** Text instance of type a2w::Text

Remarks: Get the next text object of overlay (used to iterate the list of overlay text objects after calling "getFirstText").
Note: This method was previously called "getNextTextObject".

Example:

```
$a2wTextTmp = $a2wOverlay->getFirstText();
while ($a2wTextTmp != 0 ){
#---- Access text info
...
#---- Get next text
$a2wTextTmp = $a2wOverlay->getNextText();
}
```

getResolution [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the resolution of the overlay.

Example: \$svResTmp = \$a2wOverlay->getResolution();

getWidth [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns the width of the overlay

Example: `$svWidthTmp = $a2wOverlay->getWidth();`

a2w::PSEG [NEW]

The a2w::PSEG object wraps the AFP2web Kernel object mhtResPSEG and exposes the following interfaces to the AFP2web Scripting Facility.

getIncludedXPosition [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns included X position of this page segment in page

Example: `$svInclXPosTmp = $a2wPSEG->getIncludedXPosition();`

getIncludedYPosition [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Returns included Y position of this page segment in page

Example: `$svInclYPosTmp = $a2wPSEG->getIncludedYPosition();`

getName [NEW]

Parameter: None

Return value String
data type:

Remarks: Returns page segment name

Example: \$svNameTmp = \$a2wPSEG->getName();

a2w::Page (mhtPtocaPage [DEPRECATED], mhtLPDPage [DEPRECATED])

The a2w::Page object wraps the AFP2web Kernel objects mhtPtocaPage, mhtTIFFPage, mhtLPDPage and exposes following interfaces to the AFP2web Scripting Facility.

addAnnotation

Parameter:

- X position of type integer
- Y position of type integer
- Width of type integer
- Height of type integer
- URL text of type string
- Border width of type integer
- Color of type integer
- Flags of type integer

Return value Void
data type:

Remarks: Add annotation to page, useful when PDF output is generated else ignored

Example: `$a2wPage->addAnnotation(0, 0, 100, 10, "MAAS", 2, 0xFF0000, 0);`

addBookmark [NEW]

Parameter:

- Bookmark name of type string
- Bookmark value of type string

a2w::Page (*mhtmlPage* [DEPRECATED], *mhtmlPage* [DEPRECATED])

Return value Void
data type:

Remarks: Add bookmark to page

Example: \$a2wPage->addBookmark("Software", "AFP2web");

addImage [NEW]

Parameter: Image name of type string
 X include position of type integer
 Y include position of type integer
 Presentation Width of type integer
 Presentation Height of type integer
 Rotation of type integer

Return value Void
data type:

Remarks: Add raster image to page

Example: \$a2wPage->addImage("BGF0RM", 0, 100, 1000, 1500, 0);

addIndex [DEPRECATED]

Parameter: Index name of type string
 Index value of type string

Return value Void
data type:

Remarks: Add index to page, deprecated by new "addIndex" interface explained later in this chapter 14.35

Example: `$a2wPage->addIndex("Producer", "Maas");`

addIndex [NEW]

Parameter: Index instance of type `a2w::Index`

Return value Void
data type:

Remarks: Add index to page

Example:

```
#---- Create new index
$a2wIndexTmp = new a2w::Index();
#---- Set index info
$a2wIndexTmp->setName( "Producer" );
$a2wIndexTmp->setValue( "Maas" );
#---- Add index to page
$a2wPage->addIndex( $a2wIndexTmp );
```

addLine [NEW]

Parameter: Line instance of type `a2w::Line`

Return value Void
data type:

Remarks: Add line to page

Example:

```
#---- Create new line
$a2wLineTmp = new a2w::Line();
#---- Set line info
$a2wLineTmp->setLength( 100 );
...
#---- Add line to page
$a2wPage->addLine( $a2wLineTmp );
```

addNOP [NEW]

Parameter: NOP instance of type `a2w::NOP`
 Before “BPG” flag of type Boolean (True means before BPG, default is false)

Return value Void
data type:

Remarks: Add NOP to page

Example:

```
#---- Create new NOP
$a2wNOPTmp = new a2w::NOP();
#---- Set NOP value
$a2wNOPTmp->setValue( "Created by AFP2web" );
#---- Add NOP to page
$a2wPage->addNOP( $a2wNOPTmp );
```

addObject [DEPRECATED]

Parameter: Graphic object instance of type `a2w::Text`

Return value Void
data type:

Remarks: Adds a text object to page, deprecated by the new interface “addText”

Example: `$a2wPage->addObject($a2wTextTmp);`

addOverlay [NEW]

Parameter: Overlay name of type string
 X include position of overlay of type integer
 Y include position of overlay of type integer

Return value Void
data type:

Remarks: Add overlay to page

Example: `$a2wPage->addOverlay("01MEDF01", 0, 100);`

addPSEG [NEW]

Parameter: Page segment name of type string
X include position of type integer
Y include position of type integer
Presentation Width of type integer
Presentation Height of type integer
Rotation of type integer

Return value Void
data type:

Remarks: Add page segment to page

Example: `$a2wPage->addPSEG("S1MAAS", 0, 100, 1000, 1500, 0);`

addText [NEW]

Parameter: Text instance of type a2w::Text

Return value Void
data type:

Remarks: Add text to page

a2w::Page (mhtPlocaPage [DEPRECATED], mhtLPDPPage [DEPRECATED])

Example:

```
#---- Create new text
$a2wTextTmp = new a2w::Text();
#---- Set text info
$a2wTextTmp ->setText( "Maas Test" );
...
#---- Add text to page
$a2wPage->addText( $a2wTextTmp );
```

getAppliedMediumMap [NEW]

Parameter: None

Return value String
data type:

Remarks: Returns medium map applied on given page. Return type is pointer to "mhtMediumMap"

Example: `$a2wMediumMap = $a2wPage->getAppliedMediumMap();`

getFirstIndex

Parameter: None

Return value Index instance of type a2w::Index
data type:

Remarks: Get first index of page

Example:

```
$a2wIndexTmp = $a2wPage->getFirstIndex();
while ( $a2wIndexTmp != 0 ){
#---- Access index info
...
#---- Get next index
$a2wIndexTmp = $a2wPage->getNextIndex();
}
```

getFirstLine [NEW]

Parameter: None

Return value data type: Line instance of type a2w::Line

Remarks: Returns first line object of page. Return type is pointer to "mhtmlLineObject"

Example:

```
$a2wLineTmp = $a2wPage->getFirstLine();
while ( $a2wLineTmp != 0 ){
#---- Access line info
...
#---- Get next line
$a2wLineTmp = $a2wPage->getNextLine();
}
```

getFirstMediumOverlay [NEW]

Parameter: None

Return value data type: Overlay instance of type a2w::Overlay

Remarks: Returns first medium applied overlay. Return type is pointer to "mhtmlRes-Overlay"

Example:

```
$a2wOverlayTmp = $a2wPage->getFirstMediumOverlay();
while ( $a2wOverlayTmp != 0 ){
#---- Access overlay info
...
#---- Get next overlay
$a2wOverlayTmp = $a2wPage->getNextMediumOverlay();
}
```

getFirstNOP

Parameter: None

a2w::Page (mhlPlocaPage [DEPRECATED], mhlLPDPage [DEPRECATED])

Return value NOP instance of type a2w::NOP
data type:

Remarks: Get first nop of page

Example:

```

$a2wNOPTmp = $a2wPage->getFirstNOP();
while ( $a2wNOPTmp != 0 ){
#---- Access NOP info
...
#---- Get next NOP
$a2wNOPTmp = $a2wPage->getNextNOP();
}

```

getFirstOverlay

Parameter: None

Return value Overlay instance of type a2w::Overlay
data type:

Remarks: Get first page included overlay

Example:

```

$a2wOverlayTmp = $a2wPage->getFirstOverlay();
while ( $a2wOverlayTmp != 0 ){
#---- Access overlay info
...
#---- Get next overlay
$a2wOverlayTmp = $a2wPage->getNextOverlay();
}

```

getFirstPageSegment [NEW]

Parameter: None

Return value Page segment instance of type a2w::PSEG
data type:

Remarks: Returns first page included page segment. Return type is pointer to "mhtmlPSEG"

Example:

```

$a2wPSEGTmp = $a2wPage->getFirstPageSegment();
while ( $a2wPSEGTmp != 0 ){
#---- Access page segment info
...
#---- Get next page segment
$a2wPSEGTmp = $a2wPage->getNextPageSegment();
}

```

getFirstText [MODIFIED]

Parameter: None

Return value data type: Text instance of type a2w::Text

Remarks: Get first text object of page, previously called as "getFirstTextObject"

Example:

```

$a2wTextTmp = $a2wPage->getFirstText();
while ( $a2wTextTmp != 0 ){
#---- Access Text info
...
#---- Get next text
$a2wTextTmp = $a2wPage->getNextText();
}

```

getHeight

Parameter: None

Return value data type: Integer

Remarks: Get page height

Example: \$svHeightTmp = \$a2wPage->getHeight();

getId

Parameter: None

**Return value
data type:** Integer

Remarks: Get page id

Example: `$svIdTmp = $a2wPage->getId();`

getName [NEW]

Parameter: None

**Return value
data type:** String

Remarks: Returns page name

Example: `$svNameTmp = $a2wPage->getName();`

getNextIndex

Parameter: None

**Return value
data type:** Index instance of type `a2w::Index`

Remarks: Get next index of page (used to iterate list of page indexes after calling "getFirstIndex")

Example:

```

$a2wIndexTmp = $a2wPage->getFirstIndex();
while ( $a2wIndexTmp != 0 ){
#---- Access index info
...
#---- Get next index
$a2wIndexTmp = $a2wPage->getNextIndex();
}

```

getNextLine [NEW]

Parameter: None

Return value Line instance of type a2w::Line
data type:

Remarks: Returns next line object of page (used to traverse the list of page lines).
 Return type is pointer to “mhtLineObject”

Example:

```

$a2wLineTmp = $a2wPage->getFirstLine();
while ( $a2wLineTmp != 0 ){
#---- Access line info
...
#---- Get next line
$a2wLineTmp = $a2wPage->getNextLine();
}

```

getNextMediumOverlay [NEW]

Parameter: None

Return value Overlay instance of type a2w::Overlay
data type:

Remarks: Returns next medium applied overlay (used to traverse the list of medium applied overlays). Return type is pointer to “mhtResOverlay”

a2w::Page (*mhtmlPage* [DEPRECATED], *mhtmlPage* [DEPRECATED])

Example:

```
$a2wOverlayTmp = $a2wPage->getFirstMediumOverlay();
while ( $a2wOverlayTmp != 0 ){
#---- Access overlay info
...
#---- Get next overlay
$a2wOverlayTmp = $a2wPage->getNextMediumOverlay();
}
```

getNextNOP

Parameter: None

Return value NOP instance of type *a2w::NOP*
data type:

Remarks: Get next NOP of page (used to iterate list of page nops after calling "getFirstNOP")

Example:

```
$a2wNOPTmp = $a2wPage->getFirstNOP();
while ( $a2wNOPTmp != 0 ){
#---- Access NOP info
...
#---- Get next NOP
$a2wNOPTmp = $a2wPage->getNextNOP();
}
```

getNextOverlay

Parameter: None

Return value Overlay instance of type *a2w::Overlay*
data type:

Remarks: Get next page included overlay (used to iterate list of page indexes after calling "getFirstOverlay")

Example:

```

$a2wOverlayTmp = $a2wPage->getFirstOverlay();
while ( $a2wOverlayTmp != 0 ){
#---- Access overlay info
...
#---- Get next overlay
$a2wOverlayTmp = $a2wPage->getNextOverlay();
}

```

getNextPageSegment [NEW]

Parameter: None

Return value Page segment instance of type a2w::PSEG
data type:

Remarks: Returns next page included page segment (used to traverse the list of page included page segments). Return type is pointer to "mhtmlResPSEG"

Example:

```

$a2wPSEGTmp = $a2wPage->getFirstPageSegment();
while ( $a2wPSEGTmp != 0 ){
#---- Access page segment info
...
#---- Get next page segment
$a2wPSEGTmp = $a2wPage->getNextPageSegment();
}

```

getNextText [MODIFIED]

Parameter: None

Return value Text instance of type a2w::Text
data type:

Remarks: Get next text object of page (used to iterate list of page text object after calling "getFirstText"), previously called as "getNextTextObject"

a2w::Page (*mhtmlPage* [DEPRECATED], *mhtmlPage* [DEPRECATED])

Example:

```
$a2wTextTmp = $a2wPage->getFirstText();
while ( $a2wTextTmp != 0 ){
#---- Access Text info
...
#---- Get next text
$a2wTextTmp = $a2wPage->getNextText();
}
```

getOutputBuffer.[NEW]

Parameter: None

Return value Buffer of type byte*
data type:

Remarks: Gets the page output buffer.

Example: \$svPageBufferTmp = \$a2wPage->getOutputBuffer();

getOutputBufferLength.[NEW]

Parameter: None

Return value Integer
data type:

Remarks: Gets the length of the page output buffer.

Example: \$svPageBufferLenTmp = \$a2wPage->getOutputBufferLength();

getOutputFilename.[NEW]

Parameter: None

Return value String
data type:

Remarks: Gets the name of the output file of this page.

Example: \$svOutputFilenameTmp = \$a2wPage->getOutputFilename();

getPageGroupName [NEW]

Parameter: None

Return value String
data type:

Remarks: Returns name of page group, which contain this page

Example: \$svPageGroupNameTmp = \$a2wPage->getPageGroupName();

getParseId

Parameter: None

Return value Integer
data type:

Remarks: Get page parse id (sequence id of page as in input spool)

Example: \$svParseIdTmp = \$a2wPage->getParseId();

getResolution

Parameter: None

**Return value
data type:** Integer

Remarks: Get resolution

Example: `$svResTmp = $a2wPage->getResolution();`

getSimpleFilename.[NEW]

Parameter: None

**Return value
data type:** String

Remarks: Gets the simple file name of the output file of this page.

Example: `$svSimpleFilenameTmp = $a2wPage->getSimpleFilename();`

getSize.[NEW]

Parameter: None

**Return value
data type:** Long

Remarks: Gets the size of the output file of this page.

Example: `$svOutputSizeTmp = $a2wPage->getSize();`

getType

Parameter: None

**Return value
data type:** Integer

Remarks: Get page type (AFP or LPD)

Example: `$svTypeTmp = $a2wPage->getType();`

getWidth

Parameter: None

**Return value
data type:** Integer

Remarks: Get page width

Example: `$svWidthTmp = $a2wPage->getWidth();`

includeObject [DEPRECATED]

Parameter: Object type of type byte
Object name of type string
X position of type integer
Y position of type integer
Rotation of type integer
Object sub type of type string

**Return value
data type:** Void

a2w::Page (*mhtmlPage* [DEPRECATED], *mhtmlPage* [DEPRECATED])

Remarks: Include an object (an overlay, image etc). Useful to apply some background forms to the page, deprecated by new interfaces like "addOverlay", "addPageSegment" and "addImage"

Example: `$a2wPage->includeObject(0x92, "BGFORM", 0, 0, 0, "JPEG");`

isConstantBack

Parameter: None

Return value data type: Boolean

Remarks: Returns true if page is medium applied constant back

Example: `$svConstantBackTmp = $a2wPage->isConstantBack();`

isOverlayFound

Parameter: Overlay name of type string

Return value data type: Boolean

Remarks: Check whether given overlay is included in page or not

Example: `$svOverlayFoundTmp = $a2wPage->isOverlayFound("01MEDF01");`

new [NEW]

Parameter: Page type of type integer and value is interpreted as given in the table:

Page Type Value	Description
1	AFP Page
2	LPD Page
3	TIFF Page

Return value Page instance of type a2w::Page
data type:

Remarks: Creates new page instance and returns the same

Example: `$a2wPageTmp = new a2w::Page(1);`

setBackgroundColor

Parameter: Page color of type integer

Return value Void
data type:

Remarks: Set page background color

Example: `$a2wPage->setBackgroundColor(0x00FF0000);`

setHeight

Parameter: Page height of type integer

a2w::Page (*mhtmlPage* [DEPRECATED], *mhtmlPage* [DEPRECATED])

Return value Void
data type:

Remarks: Set page height

Example: `$a2wPage->setHeight(584);`

setOutputFileName.[NEW]

Parameter: Output filename of type string

Return value Void
data type:

Remarks: Sets the name of the output file for this page.

Example: `$a2wPage->setOutputFilename ("MaasSample.tif");`

setResolution

Parameter: Page resolution of type integer

Return value Void
data type:

Remarks: Set page resolution

Example: `$a2wPage->setResolution(240);`

setWidth

Parameter: Page width of type integer

Return value Void
data type:

Remarks: Set page width

Example: `$a2wPage->setWidth(840);`

a2w::Text (mhtTextObject [DEPRECATED])

“a2w::Text” module wraps “mhtTextObject” kernel object and expose following interfaces to scripting facility

getAngle

<i>Parameter:</i>	None
<i>Return value data type:</i>	Integer
<i>Remarks:</i>	Get angle (rotation)
<i>Example:</i>	<code>\$svAngleTmp = \$a2wText->getAngle();</code>

getColor [NEW]

<i>Parameter:</i>	None
<i>Return value data type:</i>	Integer
<i>Remarks:</i>	Get text color of 4 bytes length (as given in below format)



Example: `$svColorTmp = $a2wText->getColor();`

getEBCDICText

Parameter: None

**Return value
data type:** String

Remarks: Get EBCDIC value of text object

Example: `$svEBCDICTextTmp = $a2wText->getEBCDICText();`

getFont [NEW]

Parameter: None

**Return value
data type:** Font instance of type a2w::Font

Remarks: Returns font used to render this text, return type is pointer to "mhtAFPFont"

Example: `$a2wFontTmp = $a2wText->getFont();`

getMappedFontLocalId [NEW]

Parameter: None

**Return value
data type:** Integer

Remarks: Get text-applied font's local map id

Example: `$svFontLocalIdTmp = $a2wText->getMappedFontLocalId();`

getText

Parameter: None

**Return value
data type:** String

Remarks: Get value of text object

Example: `$svTextTmp = $a2wText->getText();`

getTextLen

Parameter: None

**Return value
data type:** Integer

Remarks: Get value length of text object

Example: `$svTextLenTmp = $a2wText->getTextLen();`

getXPos

Parameter: None

Return value Integer
data type:

Remarks: Get X position of text object

Example: `$svXPosTmp = $a2wText->getXPos();`

getYPos

Parameter: None

Return value Integer
data type:

Remarks: Get Y position of text object

Example: `$svYPosTmp = $a2wText->getYPos();`

new

Parameter: None

Return value Text instance of type `a2w::Text`
data type:

Remarks: Allocate new text instance

Example: `$a2wText = new a2w::Text();`

remove [NEW]

Parameter: None

**Return value
data type:** Void

Remarks: Removes text from presentation, but positioning text object on output coordinate system is done still

Example: `$a2wText->remove();`

setAngle

Parameter: Angle of type integer

**Return value
data type:** Void

Remarks: Set angle (rotation) of text object

Example: `$a2wText->setAngle(90);`

setColor

Parameter: Color of type integer

**Return value
data type:** Void

Remarks: Set color of text object

Example: `$a2wText->setColor(0x00FF0000);`

setFont

Parameter: Font instance of type `a2w::Font`

Return value Void
data type:

Remarks: Set font of text object

Example:

```
#---- Create font
$a2wFontTmp = new a2w::Font();
#---- Set font details
$a2wFont->setName( "Verdana" );
...
#---- Set above created font as font in text
$a2wText->setFont( $a2wFontTmp );
```

setText

Parameter: Text of type string

Return value Void
data type:

Remarks: Set value of text object

Example: `$a2wText->setText("Maas Test");`

setTextLen

Parameter: Text length of type integer

**Return value
data type:** Void

Remarks: Set value length of text object

Example: `$a2wText->setTextLen(9);`

setXPos

Parameter: X position of type integer

**Return value
data type:** Void

Remarks: Set X position of text object

Example: `$a2wText->setXPos(100);`

setYPos

Parameter: Y position of type integer

**Return value
data type:** Void

Remarks: Set Y position of text object

Example: `$a2wText->setYPos(200);`

5.10 AFP2web Messages

General Information About Error Messages, Return Codes, and Error Files

afp2web.exe delivers a return code on completion.

AFP2web also writes error messages to a separate text file. You will find this file in the log directory. The file name begins with "error_" and is completed with the processing timestamp. For example: error_20000413_101127.txt

AFP2web passes the return code when issuing messages:

- The return code 0 indicates that no errors occurred. The prefix "I" in the message ID indicates an informational message.
- A negative return code indicates a processing error. The value corresponds with the numerical part of the Message ID in the error message. For example, the return code -23 indicates that the error E023 occurred. The prefix "E" in the message ID indicates an error message, the prefix "W" indicates a warning message.

Note: If you do not find the error messages self explanatory, please contact Maas High Tech Software GmbH to resolve any issue.

AFP2web Error Messages

<i>Return Code</i>	<i>Message Description</i>
0	I000: Process completed Process successfully terminated.
	I001: End of spool: <SpoolFileName> Spool successfully processed.
0	I022: User requested no. of pages parsed. Ignoring other pages Occurs when the command option -pp is used. No solution required.
0	I083: User requested no. of documents parsed. Ignoring other documents Occurs when the command option -ed is used. No solution required.
Warning Messages (Messages that do not abort the process)	
	W016: Missing mandatory value : <Value>. MO:DCA-specific warning Check the AFP document. If the problem cannot be solved, please contact Maas High Tech Software GmbH.
	W060: Invalid SFI. <StreamLocationAndName> Offset=<HexOffset> MO:DCA specific warning Check the AFP document.
	W093: Insufficient Font Pattern data for GCGID <GCGIDName> for resource <ResName>. Available Bytes: <DataLength> Required Bytes: <Req.DataLength>. FOCA-Specific warning Check the Specified Font Resource document.
	W154: Invalid Index Path (<InfoText>) Path for writing index files is invalid. Please ensure that the index path specified exists.

<i>Return Code</i>	<i>Message Description</i>
Error Messages	
-1	E001: Unknown error. Program failure. Please contact Maas High Tech Software GmbH.
-2	E002: Unknown exception occurred, leaving... Program failure. Please contact Maas High Tech Software GmbH.
-3	E003: AFP2web is not licensed for <OS name Input format Output format Scripting Facility>. Please contact afp2web@maas.de AFP2web is not licensed for one of the following options: Requested operating system, input format, output format, or Scripting Facility. Please contact Maas High Tech Software GmbH.
-4	E004: <FileType> File <FileName> not Found. AFP2web does not find the resource or index file. Please make sure that external resource or index file exists in the specified path.
-5	E005: Missing End Page Group SFI. File Name: <FileName> MO:DCA specific error. Check the AFP document.
-6	E006: Missing End Document SFI. File Name: <FileName> MO:DCA specific error. Check the AFP document.
-7	E007: Illegal reserved value. (at <HexOffset>) MO:DCA specific error. Check the AFP document.
-8	E008: AFP font representation object is null. FOCA specific error. Check the AFP font resources
-9	E009: Missing Mandatory Triplet. (at <HexOffset>) MO:DCA specific error. Check the AFP document.
-10	E010: Valid spool beginning not found. File Name: <FileName> MO:DCA specific error. Check the AFP document.

<i>Return Code</i>	<i>Message Description</i>
-11	E011: Character metrics missing for <Degree> degree rotation. CharsetName: <CharsetName> FOCA specific error. Check the FOCA AFP document.
-12	E012: AFP2web is not built for<FormatType> <"Input"/"Output" Format>. Please contact Maas High Tech Software GmbH at afp2web@maas.de The combination of key values in the INI file are not valid. Please contact Maas High Tech Software GmbH.
-13	E013: <MHT Image Library Error> Error occurred when image is processed by MHT Image Library Check MHT Image Library Error Messages.
-14	E014: Missing End Page SFI. File Name: <FileName> MO:DCA specific error. Check the AFP document.
-15	E015: Missing End Overlay SFI. File Name: <FileName> MO:DCA specific error. Check the AFP document.
-17	E017: Message id <MsgID> define into ErrorCode enum structure, but Error message missing within achErrorMsg[]. Check code. Program failure. Please contact Maas High Tech Software GmbH.
-18	E018: <CompressionName> compressed <ImageType> images is not supported. AFP2web does not support specified compression for the image type Please contact Maas High Tech Software GmbH.
-19	< Bits/Pixel> Bits/Pixel <ImageType> image is not supported AFP2web does not support specified image type having specified Bits/Pixel . Please contact Maas High Tech Software GmbH.
-20	E020: Unable to find any substitutable external font for [CF=<CodedFont name>][CS=<Charset name>][TF=<Typeface name>] AFP2web cannot find the external font to be used as substitution font Please check if font file that corresponds to a specified Typeface name exists in external font path given using ini attribute "extfont" or command line argument "-fp"
-21	E021: Java Exception occurred while reading stream (<Stream Name>). Msg:<InfoText> Java input stream could not be read by AFP2web for the reason given in <InfoText> Please check if source of input stream is readable

<i>Return Code</i>	<i>Message Description</i>
-23	E023: Input file (<InfoText>) not found. Input file cannot be found. Please check your processing options for afp2web.exe or those in the afp2web.ini.
-24	E024: Invalid Output Format: <Output format> Invalid output file format requested for conversion Please check whether valid output format is specified while invoking AFP2web
-25	E025: Insufficient Memory - Unable to create display Device context. The resolution selected might be too high for the conversion to TIFF. Increase the size of your computer's RAM.
-26	E026: Insufficient Memory - Unable to create Memory Device context. The resolution selected might be too high for the conversion to TIFF. Increase the size of your computer's RAM.
-27	E027: Insufficient memory while allocating Space for Bitmap data. The resolution selected might be too high for the conversion to TIFF. Increase the size of your computer's RAM.
-28	E028: Insufficient memory - Unable to create bitmap for a page (Error-Code: <Last Error Code of Win32 API>) The resolution selected might be too high for the conversion to TIFF/JPG/PNG. Increase the memory size of your computer or reduce the output resolution (-pr option).
-29	E029: Could not install outline font to system. Characterset: <Charset Name> CodePage:<CP Name> Reason:<Reason> A2W is not able to install the type 1 font created from AFP outline font to the system Check if user has rights to install external fonts to the system.
-30	E030: Unable to read a TIFF header of <Filename> (Bad TIFF). This message may occur when trying to convert a TIFF document to PDF. The TIFF document cannot be read. Check the TIFF file. If the problem cannot be solved, please contact Maas High Tech Software GmbH.
-31	E031: Could not install external font to system. Font File: <FileName>, Reason: <Reason> A2W is not able to install type1/truetype font to the system Check if user has rights to install external fonts to the system.
-32	E032: Input file (<InfoText>) is not a TIFF file. The TIFF file must have the file extension ".tif" or ".tiff". Add the file extension to the TIFF file.

<i>Return Code</i>	<i>Message Description</i>
-33	E033: Unhandled/Unknown compression for image (<ImageName>, <ImageType>) AFP2web does not support specified compression Please contact Maas High Tech Software GmbH.
-34	E034: Color input image <<ImageType>, <ImageName>, <Compression>> is not supported for <OutputFormat> output AFP2web does not support color input images of specified compression for the specified output format type. Please contact Maas High Tech Software GmbH.
-35	E035: Java Input Stream is null for (<StreamName>) NULL Input stream is passed to AFP2web! Please check if valid input stream is passed to JAVA SDK
-36	E036: Java <MethodName> Method Handle is null for (<StreamName>) AFP2web could not get specified Method handle from Java Runtime environment Please check installed Java version is same as or higher than the specified Java version in A2W user guide.
-37	E037: Input stream does not support random access. Setting offset failed for stream(<StreamName>). If input stream contains some invalid AFP data, AFP2web tries to find valid AFP data within the stream. In such case, it sets the offset directly in the input stream. This exception is thrown when the input stream passed to AFP2web does not support setting an offset within the stream (random access) Please check if AFP input stream contains valid data and that it supports random access
-38	E038: Unable to create bitmap for rasterization of AFP font <Charset Name>. (ErrorCode: <Errorcode>, Msg: <ErrorMessage>) AFP2web is not able to create bitmap representation of AFP Font Please check error message returned by A2W and take needed action. Else Please contact Maas High Tech Software GmbH.
-39	E039: Conversion of <FontType> font to <FontType> font not yet supported AFP2web does not support font conversion of specified font types. Please consult Maas High Tech Software GmbH.
-40	E040: Conversion of <FontType> font to <FontType> font failed. Font-Name: <FontName> Reason: <Reason> Check if Input font is valid Please consult Maas High Tech Software GmbH.

Return Code	Message Description
-41	E041: Unique System Id generation failed. (ErrorCode: <ErrorCode>, Msg:<Reason>) AFP2web could not get information required to generate "Unique System Id" Please consult Maas High Tech Software GmbH.
-42	E042: Invalid Operating System specified on Licensee("<Licensee>") in <PathToInifile>\<IniFileName>. Please contact Maas High Tech Software GmbH at afp2web@maas.de. The Licensee text in afp2web.ini file is invalid. Ensure Licensee text is entered properly in afp2web.ini
-43	E043: Invalid A2W Edition Id: <EditionID>. Please contact Maas High Tech Software GmbH at afp2web@maas.de. The Licensee text in afp2web.ini file is invalid. Please consult Maas High Tech Software GmbH.
-52	E052: Error reading Repeated Triplets (at <HexOffset>) MO:DCA specific error. Check the AFP document.
-53	E053: NULL Mapcodedfont object in MCFTriplets() Memory allocation error. Increase the memory limit in the parameter DocLimit.
-54	E054: Illegal Resource Type in MCF Resource Local Identifier Triplet (at <HexOffset>) MO:DCA specific error. Check the AFP document.
-55	E055: Illegal Local id value in MCF Resource Local Identifier Triplet (at <HexOffset>) MO:DCA specific error. Check the AFP document.
-56	E056: Error Illegal Page Overlay Type - <OverlayTypeld> (at <HexOffset>) MO:DCA specific error. Check the AFP document.
-57	E057: Not Able to Get PDF Doc Handle. Exiting... Memory allocation error. Increase the memory limit in the parameter DocLimit.
-58	E058: Resource (<ResName>) not found Path to resource file is incorrect. Check the INI file and command options. Ensure that the correct path is used for resource files.

<i>Return Code</i>	<i>Message Description</i>
-59	E059: Null Data Buffer System-internal failure. Please contact Maas High Tech Software GmbH.
-61	E061: Input Stream Open Failed System-internal failure. Please contact Maas High Tech Software GmbH.
-62	E062: SFI missing. <StreamLocationAndName> Offset=<HexOffset> MO:DCA specific error. Check the AFP document.
-63	E063: SFI data is null System-internal failure. Please contact Maas High Tech Software GmbH.
-65	E065: [CHARSET] Section: Missing or Invalid Font Height (<InfoText>) in "mapping.def" Entries in the mapping.def have invalid values. Check the mapping.def.
-67	E067: Error Invalid Interchange Set Triplet Id <TripletId> (at <HexOffset>) MO:DCA specific error. Check the AFP document.
-68	E068: File read error Error occurred when trying to read one of the following files: INI file, mapping.def file, ASCII codepage file (*.cp). Ensure that the required files exist.
-69	E069: Error while writing <FileName> file Error when trying to write specified file. Ensure that AFP2web has write access to the path of the specified file.
-70	E070: Unable to move a file pointer to a specified location. File Name: <FileName> File operation error Please contact Maas High Tech Software GmbH.
-72	E072: File (<InfoText>) is empty. Error when reading the specified file. The file is empty. Please check the specified file.
-73	E073: Unable to read data from the stream: File <FileName>, ErrorMessage: <InfoText> AFP2web could not read the specified file. Ensure that AFP2web has read access to the specified file.

Return Code	Message Description
-74	<p>E074: SFI does not begin with 5A. <StreamLocationAndName> Offset=<HexOffset> We distinguish 2 kinds of AFP files. The one having an 0x5A Hex value preceding each and every Structured Field Identifier (SFI). And the one not having it. This message occurs when at least the very first SFI has an 0x5A Hex value preceding it but some others in the same file not. Please check the AFP data.</p>
-75	<p>E075: Unable to write data into the stream: File <FileName>, ErrorMsg: <InfoText> AFP2web could not write the specified file. Ensure that AFP2web has write access to the specified file.</p>
-76	<p>E076: Not able to find a valid SFI within the 4 Kb of data. <StreamLocationAndName> Offset=<HexOffset> AFP2web could not find any valid Structured Field Introducer (SFI) within the 4 Kb of data following offset <HexOffset>. Process has been aborted. Please check the AFP document or contact Maas High Tech Software GmbH.</p>
-77	<p>E077: Missing/Invalid Licensee(<Licensee>) or Serial Number(<Serial-Number>) in <PathToInifile>\<IniFileName>. Please contact afp2web@maas.de. The combination of key values in the INI file are not valid. Please ensure that you specify the correct values for Licensee and Serial number in the INI file. If you cannot solve the problem, please contact Maas High Tech Software GmbH.</p>
-78	<p>E078: Unable to load a font (<InfoText>) Font cannot be loaded. Occurs when converting to TIFF on Unix. Please check that the font is installed on the system. Else consult Maas High Tech Software GmbH.</p>
-79	<p>E079: Unable to create a pixmap. Occurs when converting to TIFF on Unix. Please consult Maas High Tech Software GmbH.</p>
-80	<p>E080: Image stretching error Occurs when converting to TIFF on Unix. Please consult Maas High Tech Software GmbH.</p>
-81	<p>E081: Out of memory error. Unable to write intermediate file <FileName> Occurs when converting to TIFF on Unix. Please consult Maas High Tech Software GmbH.</p>
-82	<p>E082: Error while writing intermediate file <FileName> Occurs when converting to TIFF on Unix. Please contact Maas High Tech Software GmbH.</p>

<i>Return Code</i>	<i>Message Description</i>
-84	E084: Required Resource (<ResName>) of Type (<ResType>) not Found System-internal failure. Please contact Maas High Tech Software GmbH.
-85	E085: X Window Error (<info Text>) Occurs when the X Window API fails or AFP2web is unable to access X Window. Please contact Maas High Tech Software GmbH.
-86	E086: PDF Library Error: <InfoText> Maas PDF Library error encountered Please contact Maas High Tech Software GmbH.
-87	E087: TIFF Library Error <InfoText> LIBTIFF Library error encountered. Please contact Maas High Tech Software GmbH.
-88	E088: Scripting Facility Error (rc=<errorcode>): <infotext> Occurs when AFP2web Scripting Facility interface fails to process the script. Check the script file. Contact Maas High Tech Software GmbH if the problem cannot be resolved.
-90	E090: Image Library Error: <InfoText> Image Library error encountered while decompressing IOCA and IOB images. Please contact Maas High Tech Software GmbH with error and log files.
-91	E091: Output buffer of IOCA/IOB object (<Object name>) is null. Image Library error encountered while decompressing IOCA and IOB images. Please contact Maas High Tech Software GmbH.
-92	E092: Unsupported image type <ImageType>. Only TIFF, JPEG Image types is supported for AFP Container Objects. Unsupported image type occurred in AFP Container Object. Please contact Maas High Tech Software GmbH.
-94	E094: AFP font representation object for Characerset <CSName> is null. FOCA specific error. Check the AFP font resources
-95	E095: Used code point information is missing. CharacterSet:<CSName> CodePage:<CPName> Internal failure Please contact Maas High Tech Software GmbH.

Return Code	Message Description
-96	E096: Error while creating type3 output font representation object for CharacterSet <CSName> and CodePage <CPName>. FOCA specific error. Check the AFP font resources
-97	E097: Type3 character bitmap array is null for CharacterSet <CSName> and CodePage <CPName>. FOCA specific error. Check the AFP font resources
-98	E098: Unable to open file <FileName>. Reason: <InfoText> AFP2web could not open the specified file Please check the for reason given in the exception. If you could not solve the problem, Please contact Maas High Tech Software GmbH.
-99	E099: Invalid AFP file. System-internal failure. If you cannot solve the problem, please contact Maas High Tech Software GmbH.
-100	E100: Memory allocation error : <infotext> Insufficient memory. Increase the size of your computer's RAM.
-101	E101: Invalid INI File Path (<InfoText>) Path to INI file is not correct. Check the command option specifying the path to afp2web.ini.
-102	E102: Error while opening Statistics file <FileName> Path for statistics file is not correct. Check the settings in the INI file and command options. Ensure that the path for writing the statistics file exists.
-103	E103: Invalid Output path (<InfoText>). Path for output files incorrect. Check the INI file and command line parameters. Ensure that the correct path is used for the output files (PDF or TIFF).
-104	E104: INI File (<InfoText>) not found. INI file -cannot be found. Check the specified path to the afp2web.ini file and that the INI file exists.
-105	E105: [<SectionName>] section not found in the INI File (<FileName>) A required section is missing in the afp2web.ini file. Check the INI file.
-106	E106: Type3 Encoding stream is null for CharacterSet <CSName> and CodePage <CPName>. FOCA specific error. Check the AFP font resources

<i>Return Code</i>	<i>Message Description</i>
-107	E107: Missing or Invalid Spool File Type parameter Invalid Spool type specified Check spool type parameter specified in INI file or as command line option
-108	E108: Missing Input File parameter Missing input file. Specify at least one input file to be processed.
-112	E112: <[SectionName]> Section: Missing or Invalid Value <LineText> in mapping.def. Values missing or incorrect in specified section in mapping.def Please check the specified section in mapping.def
-114	E114: Error while setting font. CharsetName: <CharsetName> Reason: <InfoText> AFP2web could not set font to output. Please check the for reason given in the exception. If you could not solve the problem, Please contact Maas High Tech Software GmbH.
-115	E115: Product version expired. Please contact afp2web@maas.de Product version expired. Please check the for reason given in the exception. If you could not solve the problem, Please contact Maas High Tech Software GmbH.
-118	E118: Output font representation is null for CharacterSet=<CSName>, CodePage=<CPName>, OutputFontType=<FontType> Corrupted or invalid external font(s) Please check the external font files in "extfont" path. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-119	E119: Invalid Type1 PFM File <FileName> Corrupted or invalid PFM file Please check the specified PFM file. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-120	E120: Error while parsing True Type font file <FileName>. Reason: <InfoText>. Corrupted or invalid True Type font file Please check the specified TrueType file. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-155	E155: Error while opening the Index File (<FileName>), leaving... AFP2web could not open the index file for writing. Please ensure that the path exists and that AFP2web has write access to the path of the index file. If you cannot solve the problem, please contact Maas High Tech Software GmbH.

Return Code	Message Description
-158	E158: File Pointer is null Internal failure Please contact Maas High Tech Software GmbH.
-167	E167: Conversion from <InputFileType> to <OutputFileType> is not supported Unsupported file conversion. No solution required.
-190	E190: Internet Connection failed. Reason:<InfoText> AFP2web could not create a Internet session. Please check the reason specified in the exception. If you could not solve the problem, Please contact Maas High Tech Software GmbH.
-191	E191: FTP Connection failed. Reason:<InfoText> AFP2web could not establish a FTP connection. Please check the reason specified in the exception. If you could not solve the problem, Please contact Maas High Tech Software GmbH.
-192	E192: Unable to open FTP file <FileName>. Reason: <InfoText>. AFP2web could not open a file over FTP Connection. Please check the reason specified in the exception. If you could not solve the problem, Please contact Maas High Tech Software GmbH.
-193	E193: Unable to read FTP file <FileName>. Reason: <InfoText>. AFP2web could not read a file over FTP Connection. Please check the reason specified in the exception. If you could not solve the problem, Please contact Maas High Tech Software GmbH.
-198	E198: Null FTP Connection. Internal failure Please contact Maas High Tech Software GmbH.
-199	E199: Reading bytes beyond the bounds. Occurs when premature EOF is encountered in the input spool Check if Input spool readable or check if it has been deleted by any other process
-200	E200: Null Input Buffer It occurs When null input buffer is passed to A2WSDK. Please make sure that input buffer passed to A2WSDK contains valid data.
-207	E207: Output file name is null Output file name not given in the command line options Please ensure that output file name is given in command line options
-231	E231: libXPDF Error (rc=<ErrorCode>): <ErrorMessage> at <API Name> Occurs while parsing PDF files that contains invalid entries Check if PDF file is valid and viewable in Acrobat Reader

MHTIMG Library Messages

<i>Return Code</i>	<i>Message Description</i>
-1	MIL001: Unknown error. Please contact Maas High Tech Software GmbH.
-2	MIL002: Message id <MsgID> define into ErrorCode enum structure, but Error message missing within achImgLibErrorMsg[]. Check code. Program failure. Please contact Maas High Tech Software GmbH.
-5	MIL005: Image header information is null Program failure. Please contact Maas High Tech Software GmbH.
-6	MIL006: DIB buffer is null Program failure. Please contact Maas High Tech Software GmbH.
-7	MIL007: Error while reading file <FileName>. Rc:<ReturnCode> Msg:<Reason> MHTIMG Library could not read the specified file Ensure that MHTIMG Library has read access to the specified file. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-8	MIL008: File name parameter is null or empty File name passed to MHTIMG Library is null Ensure that right file name is passed to the MHTIMG Library
-9	MIL009: Unsupported image type <ImageType> MHTIMG Library does not support the specified image type Please contact Maas High Tech Software GmbH.
-10	MIL010: Decompression of <BPP> Bits/Pixel <Comp.Type> <ImageType> images is not supported Unsupported decompression requested Please contact Maas High Tech Software GmbH.
-12	MIL012: Source image buffer is null or buffer length parameter is zero Invalid parameters passed to MHTIMG Library. Ensure that proper image buffer is passed to MHTIMG Library. If you could not solve the problem, please contact Maas High Tech Software GmbH.

<i>Return Code</i>	<i>Message Description</i>
-13	MIL013: Invalid compression parameter id <ParamId> Invalid compression parameter passed to MHTIMG Library Ensure valid compression parameters are passed to MHTIMG Library
-14	MIL014: IOCA Library handle is null Program failure. Please contact Maas High Tech Software GmbH.
-15	MIL015: IOCA Library Error: Rc:<ReturnCode> Msg:<InfoText> Error occurred in IOCALib Library. Please check the input image file/stream for message given in the exception. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-16	MIL016: Decompression of <SPP> Samples/Pixel <ImageType> images is not supported Unsupported decompression requested Please contact Maas High Tech Software GmbH.
-17	MIL017: CxImage Library Error: <InfoText> Error occurred in CxImage Library. Please check the input image file/stream for the message given in the exception. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-18	MIL018: Error while opening file <FileName>. Rc:<ReturnCode> Msg:<Reason> MHTIMG Library could not open the specified file. Ensure that the specified file exist.
-19	MIL019: Error while creating TIFF handle for <Filename "Memory Stream"> Program failure. Please contact Maas High Tech Software GmbH.
-20	MIL020: TIFF Library Error: <InfoText> Error occurred in TIFF Library. Please check the input image file/stream for the message given in the exception. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-21	MIL021: Memory allocation error Insufficient Memory Increase the size of your computer's RAM.
-22	MIL022: Compression <Compression> is not supported for <OutputFormat> Unsupported compression requested for a specified output format Please contact Maas High Tech Software GmbH.

<i>Return Code</i>	<i>Message Description</i>
-23	MIL023: <ImageType> Image encoder is null Program failure. Please contact Maas High Tech Software GmbH.
-24	MIL024: <ImageType> Image decoder is null Program failure. Please contact Maas High Tech Software GmbH.
-25	MIL025: JPEG compression failed: <InfoText> Program failure. Please check the message given in the exception.If you could not solve the problem, please contact Maas High Tech Software GmbH.
-26	MIL026: <MethodName> not implemented for <ClassName> Program failure. Please contact Maas High Tech Software GmbH.
-27	MIL027: JPEG Library handle is null Program failure. Please contact Maas High Tech Software GmbH.
-28	MIL028: Compression parameter array is null Program failure. Please contact Maas High Tech Software GmbH.
-29	MIL029: Cannot convert IOCA <Colorspace> image planes to <Color-space> color space Program failure. Please contact Maas High Tech Software GmbH.
-30	MIL030: CMKY image buffer passed to CMYK Converter is null Program failure. Please contact Maas High Tech Software GmbH.
-31	MIL031: IOCA RGB image plane buffer is null Occurs while parsing AFP IOCA sample that contains invalid data Check if AFP IOCA contains valid data
-32	MIL032: Unsupported file format <FileType> MHTIMG Library does not support the specified file type Please contact Maas High Tech Software GmbH.
-33	MIL033: Grayscale conversion of<Bits/Pixel> Bits/Pixel images is not supported MHTIMG Library does not support Grayscale conversion of images having specified bits/pixel Please contact Maas High Tech Software GmbH.

<i>Return Code</i>	<i>Message Description</i>
-34	MIL034: Black/White conversion of <Bits/pixel> Bits/Pixel images is not supported MHTIMG Library does not support B?W conversion of images having specified bits/pixel Please contact Maas High Tech Software GmbH.
-37	MIL037: IOCA logical resolution unit not supported MHTIMG Library does not support IOCA Logical resolution unit Please contact Maas High Tech Software GmbH.
-38	MIL038: Decompression of <Comp.Type> <ImageType> images is not supported Unsupported decompression requested Please contact Maas High Tech Software GmbH.
-39	MIL039: Error while writing file <FileName> MHTIMG Library could not write the specified file. Ensure that MHTIMG Library has write access to the specified path. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-40	MIL040: Incorrect call to buffer access methods Program failure. Please contact Maas High Tech Software GmbH.
-42	MIL042: Unable to prepare header for OJPEG TIFF data Invalid/Corrupted TIFF file/TIFF Image buffer Please check the tiff header of tiff file. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-43	MIL043: Error while extracting TIFF image data from strip <StripNumber> Invalid/Corrupted TIFF file/TIFF Image buffer Please check the TIFF File. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-44	MIL044: JPEG Library Error: <InfoText> Error occurred in JPEG Library Please check the input image file/stream for the message given in the exception. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-45	MIL045: Images with Bits/Pixel greater than one can not converted to <ImageType> Unsupported conversion requested Please contact Maas High Tech Software GmbH.
-46	MIL046: TIFF Library handle is null Program failure. Please contact Maas High Tech Software GmbH.

<i>Return Code</i>	<i>Message Description</i>
-47	MIL047: Error while writing <SFIType> SFI Program failure. Please contact Maas High Tech Software GmbH.
-48	MIL048: Image data is null Program failure. Please contact Maas High Tech Software GmbH.
-50	MIL050: <ImageType> compression is not supported MHTIMG Library does not support specified compression Please contact Maas High Tech Software GmbH.
-51	MIL051: Error while decompressing IOCA <Colorspace> planes of image data Invalid IOCA Image Data Please check if IOCA has valid Image data. If you could not solve the problem, please contact Maas High Tech Software GmbH.
-52	MIL052: Decompression of <BPP> Bits/Pixel <ImageType> images is not supported. Unsupported decompression requested Please contact Maas High Tech Software GmbH.
-53	MIL053: Decompression of <ImageType> images having <PhotoMetric-Name> Photometric Interpretation is not supported Unsupported decompression requested Please contact Maas High Tech Software GmbH.
-55	MIL055: Insufficient data read from <StreamName>. Expected-Bytes=<ScanlineSize> ReadBytes=<BytesReadFromStream> Image Data stream has lesser number of bytes available than needed for a complete scanline Check if input image file/stream has valid data.
-57	MIL057: TIFF Images with Planar format(separate planes of data) are not supported TIFF Images in Planar format (separate planes of data) are not supported Please contact Maas High Tech Software GmbH.

Chapter A: Appendix

This appendix includes:

- Tips for migrating the Scripting Facility modules to AFP2web Version 3.x
- Guide to the Scripting Facility examples
- Frequently asked questions

A.1 Migrating Scripting Facility Modules to Version 3.x

The following table shows the differences between scripting modules from the previous version 2.1 to those of Version 3.x. This can be used as a checklist for migration.

<i>Version 2.1</i>	<i>Version 3.x</i>
use a2w::mhtIni;	use a2w::Config;
use a2w::mhtDocument;	use a2w::Document;
use a2w::mhtPtocaPage;	use a2w::Page;
use a2w::mhtTextObject;	use a2w::Text;
use a2w::mhtNOP;	use a2w::NOP;
use a2w::mhtResOverlay;	use a2w::Overlay;
use a2w::mhtAttributeElement;	use a2w::Index;
use a2w::mhtAFPFont;	use a2w::Font;
use a2w::mhtPagePageGroupIndex;	use a2w::Index;
use a2w::mhtIndexElement;	use a2w::Index;
sub initialize(mhtIni)	sub initialize(a2w::Config, a2w::Kernel)
\$a2wIniPar->setAutosplit(\$TRUE);	\$a2wConfigPar->setAttribute("Autosplit", "on");
\$svIndexPathTmp = \$a2wConfig->get- IndexPath();	\$svIndexPathTmp = \$a2wConfigPar- >getAttribute("IndexPath");
\$svOutputFilePathTmp = \$a2wConfig-> getOutputFilePath();	\$svOutputFilePathTmp = \$a2wConfigPar- >getAttribute("OutputFilePath");
\$svScriptArgsTmp = \$a2wConfig->get- ScriptArgs();	\$svScriptArgsTmp = \$a2wConfigPar- >getAttribute("ScriptArgument");
\$a2wPageTmp = \$a2wDocument->get- FirstPageObject();	\$a2wPageTmp = \$a2wDocumentPar- >getFirstPage();
\$a2wPageTmp = \$a2wDocument->get- NextPageObject();	\$a2wPageTmp = \$a2wDocumentPar- >getNextPage();

<i>Version 2.1</i>	<i>Version 3.x</i>
<code>\$a2wPage->getFirstTextObject();</code>	<code>\$a2wPagePar->getFirstText();</code>
<code>\$a2wPage->getNextTextObject();</code>	<code>\$a2wPagePar->getNextText();</code>
<code>\$a2wPageGroupIndexTmp = \$a2wDocument->getFirstIndexObject();</code>	<code>\$a2wIndexTmp = \$a2wDocumentPar-> getFirstIndex();</code>
<code>\$a2wPageGroupIndexTmp = \$a2wDocument->getNextIndexObject();</code>	<code>\$a2wIndexTmp = \$a2wDocumentPar-> getNextIndex();</code>
<code>\$a2wIndexTmp = \$a2wPageGroupIndexTmp->getFirstAt- tributeElement();</code>	directly access to index thru: <code>\$a2wIndexTmp = \$a2wDocumentPar-> getFirstIndex();</code> or <code>\$a2wIndexTmp = \$a2wPagePar->getFirstIndex();</code>
<code>\$a2wIndexTmp = \$a2wPageGroupIndexTmp->getNextAt- tributeElement();</code>	directly access to index thru: <code>\$a2wIndexTmp = \$a2wDocumentPar-> getNextIndex();</code> or <code>\$a2wIndexTmp = \$a2wPagePar->getNextIndex();</code>
<code>\$a2wIndex->setAttributeName("Insured");</code>	<code>\$a2wIndex->setName("Insured");</code>
<code>\$a2wIndex->setAttributeValue("John Doe");</code>	<code>\$a2wIndex->setValue("John Doe")</code>
<code>\$svName = \$a2wIndex->getAttributeN- ame();</code>	<code>\$svName = \$a2wIndex->getName();</code>
<code>\$svValue = \$a2wIndex->getAt- tributeValue();</code>	<code>\$svValue = \$a2wIndex->getValue()</code>
<code>\$sIndexNameTmp = \$a2wIndexTmp-> getAttributeName();</code>	<code>\$sIndexNameTmp = \$a2wIndexTmp->get- Name();</code>
<code>\$sIndexValueTmp = \$a2wIndexTmp-> getAttributeValue();</code>	<code>\$sIndexValueTmp = \$a2wIndexTmp-> getValue();</code>
<code>\$a2wOverlay->getFirstTextObject();</code>	<code>\$a2wOverlay->getFirstText();</code>
<code>\$a2wOverlay->getNextTextObject();</code>	<code>\$a2wOverlay->getNextText();</code>

<i>Version 2.1</i>	<i>Version 3.x</i>
<code>\$a2wDocument->addIndex("Insured", "John Doe");</code>	<code>\$a2wIndexTmp = new a2w::Index();</code>
	<code>\$a2wIndexTmp->setName("Insured");</code>
	<code>\$a2wIndexTmp->setValue("John Doe");</code>
	<code>\$a2wDocumentPar->addIndex(\$a2wIndexTmp);</code>
<code>\$a2wPage->addIndex("Insured", "John Doe");</code>	<code>\$a2wIndexTmp = new a2w::Index();</code>
	<code>\$a2wIndexTmp->setName("Insured");</code>
	<code>\$a2wIndexTmp->setValue("John Doe");</code>
	<code>\$a2wPagePar->addIndex(\$a2wIndexTmp);</code>
<code>\$a2wPage->includeObject(0x92, "BGFORM", \$XPos, \$YPos, \$Rotation, "JPEG");</code>	<code>\$a2wPagePar->addImage("BGFORM", \$XPos, \$YPos, \$svWidth, \$svHeight, \$Rotation);</code>
<code>\$a2wPage->includeObject(0xDF, "O1BGFORM", \$XPos, \$YPos, \$Rotation);</code>	<code>\$a2wPagePar->addOverlay("O1BGFORM", \$XPos, \$YPos);</code>
<code>\$a2wPage->includeObject(0x5F, "S1BGFORM", \$XPos, \$YPos, \$Rotation);</code>	<code>\$a2wPagePar->addPSEG("S1BGFORM", \$XPos, \$YPos);</code>
<code>\$a2wPage->addObject(\$a2wTextTmp);</code>	<code>\$a2wPagePar->addText(\$a2wTextTmp);</code>
<code>\$a2wTextTmp->setColor(0x00BBGGRR);</code>	<code>\$a2wTextTmp->setColor(0x00RRGGBB);</code>

A.2 Tutorial: Using the Scripting Facility Examples

This appendix serves as a quick-start guide to using the Scripting Facility examples, which are delivered with AFP2web Version 3. Each Scripting Facility example performs a small, basic task. In-line comments make each example easy to understand.

Introduction

Environment For Running the Scripting Facility Examples

AFP2web Version 3.x is quickly installed. For the general operating system installation steps and requirements, please refer to the AFP2web user guide.

The Scripting Facility examples are written in Perl, the files with the file extension *.pm, which are located in the root folder of the AFP2web installation.

Note: It is not necessary to install PERL for the tutorial examples. AFP2web delivers a Perl DLL, which will be used if it is found in the current working directory or if it is found in the environment variable PERLLIB. If you plan to use your own Perl environment, please refer to the system prerequisites in the installation chapter of the AFP2web User Guide.

AFP2web derives its run time parameters from the afp2web.ini file (INI file). Any command line option entered will override the corresponding parameter in the INI file. By default, output is written to the subfolder /pdf and in case of logging messages to the subfolder /log. For a detailed documentation of parameters and command line options, please refer to the AFP2web user guide.

Running a Scripting Facility Example

Selecting Commands in the demo batch command file

The in-line header of each Perl module shows how to start AFP2web and to run the script. For example, the following listing shows how to start AFP2web with the Scripting Facility example `sortPageTextObj.s.pm`:

Listing

Demo batch file (Starting AFP2web with a Script)

```
On Windows:
afp2web.exe -q -c -doc_cold -sp: sortPageTextObj.s.pm samples\insure.afp

On Unix:
./afp2web -q -c -doc_cold -sp: sortPageTextObj.s.pm samples/insure.afp
```

The command line options and their meanings:

Parameters:	Description
-q	Run AFP2web in quiet mode.
-doc_cold	Activate the AFP2web Scripting Facility
-sp	The name of the Scripting Facility file to use, if you do not want to use the default <code>afp2web.pm</code>
-sa	Arguments to pass to the script, if your script requires any parameters.

For more information on INI parameters and command line options, please refer to the AFP2web user guide. For example, you might want to use the command line option `-ll` to create a processing log.

The following lines show how you can trap and display the return code. If you want to use it, just insert these lines before the exit command.

Listing Demo.bat (Trapping an Error Level)
<pre>IF %ERRORLEVEL% EQU 0 (@ECHO AFP2web: Done %ERRORLEVEL%) ELSE (@ECHO AFP2web: Error %ERRORLEVEL%)</pre>

Running a script with demo.bat

Step	Description
1	Edit the batch command file demo.bat, resp. demo. Enter the appropriate command to start AFP2web for the script example.
	To start AFP2web, just start the demo batch command file. Under Windows, just double-click this file in the Windows Explorer. The system console is opened showing messages issued by AFP2web. If you turned AFP2web's logging on, then AFP2web will log information into a log file in the /log sub folder by default.
2	View the results.

Overview of the Script Examples

The following table gives a short description of each script example:

<i>Script Example</i>	<i>Description</i>
dumpIniParms.pm	dumpIniParms.pm writes the current settings of all INI parameters to an output file. (If a command line option was entered, this option replaces the corresponding INI file parameter.)
autosplit.pm	autosplit.pm shows how to activate the AutoSplit functionality. Autosplit tells AFP2web to split AFP spool file automatically into individual documents and pages. This is, however, only possible, if the AFP spool file contains Page Groups and Tag Logical Elements (TLEs). The script also shows how to collect and output the index data found in the TLEs.
dumpPageTextObjs.pm	dumpPageTextObjs.pm shows how to access all text objects of a page and how to write their text values to an output file (the dump file). Whenever a font change occurs, this information is written to the dump file just before the text affected by this change.
sortPageTextObjs.pm	AFP print data does not necessarily group and present text objects in the order in which they are printed. This makes it difficult to read AFP data dumped into text files. sortPageTextObjs.pm shows how to access all text objects of a page. It sorts all text objects according to their X/Y coordinate positions and then writes their text values to an output file (the dump file) in this order.
dumpMediumMaps.pm	dumpMediumMaps.pm shows how to list all medium maps applied to a document page.
dumpNOPs.pm	Using NOP fields is a common method of passing non-visible document information. Various applications use their conventions for handling NOP fields - for example for indexing, for archiving, for processing control. You might want to view the existing NOP information in your AFP spool files. dumpNOPs.pm shows how to produce a list of the values passed as NOP fields per page.
eyecatcher.pm	eyecatcher.pm shows how to find and extract predefined text strings in the content of a page. Each text string has a predefined area for its X/Y coordinate position. This Scripting Facility example uses each text string either as a signal to identify a new document and to retrieve index data from the page content.
addBookmarks.pm	addBookmarks.pm generates bookmarks in the PDF document.
addWatermark.pm	This example shows how to add a background image to a page as watermark.

<i>Script Example</i>	<i>Description</i>
addAnnotations.pm	addAnnotations.pm creates an hyperlink annotation in the PDF file.
afp2xml.pm	This Scripting Facility reads AFP document content and writes output to a text file in XML format. It requires PERL to be installed.

The Basic Structure of a Script

When you formulate a Scripting Facility module, please use the template afp2web.pm. It shows where to implement basic functions (such as including packages, reading start parameters, logging, indexing and document page splitting).

A script must have the following subroutines:

- sub afp2web()
- sub initialize()
- sub initializeDoc()
- sub initializePage()
- sub finalizePage()
- sub finalizeDoc()
- sub finalize()

The purpose of each routine is to trap processing events of AFP2web.

<i>AFP2web Kernel</i>		<i>Subroutine in afp2web.pm</i>
Process begin		initialize() (Only once)
Document begin detected		initializeDoc()
The following subroutines are called in sequence as one logical step for a document page:		
Page begin detected		initializePage()
Page parsed		afp2web()
Page processed		finalizePage()
Document processed		finalizeDoc()

<i>AFP2web Kernel</i>		<i>Subroutine in <code>afp2web.pm</code></i>
Process ending		finalize (Only once)

Each subroutine takes particular objects as input parameters. In a subroutine, you save a reference to the object so that the methods of this object becomes available at later event. For detailed information, please refer to the AFP2web and Scripting Facility user guide and reference.

Each script requires packages to be included, which provide the Scripting Facility functionality. For example:

<i>Listing dumpIniParams.pm - Packages to Include</i>	
<pre>use a2w: : Conf i g; use a2w: : Kernel ;</pre>	

Additional packages provided add functionality to the script module.

<i>Important:</i>	
	In this chapter, we focus on the particular subroutines. For details about the coding, please refer to the module itself. There are plenty of in-line comments which make the module self explanatory.

dumpIniParams.pm: Dumping INI Parameters

The Script Facility example `dumpIniParams.pm` demonstrates how to access the AFP2web processing parameters. These parameters have the values, which are set in the configuration file (`afp2web.ini` file) or which have been overridden by a command line option when calling AFP2web.

Understanding dumpIniParams.pm

This script requires that following packages be included:

<i>Listing</i> <i>dumpIniParams.pm - Packages to Include</i>
<pre>use a2w: : Conf i g; use a2w: : Kernel ;</pre>

The subroutine `initialize()` is called at the beginning of the AFP2web process. At this point, we have access to the global parameters. We use the Perl variable `@_` to save the arguments, which are passed to this subroutine.

<i>Listing</i> <i>dumpIniParams.pm - sub initialize() - arguments</i>
<pre>#---- Get Parameter of initialize(Par: a2w: : Conf i g, a2w: : Kernel) (\$a2wConf i gPar, \$a2wKernel Par) = @_;</pre>

The next statement turns the module logging on if the corresponding INI parameter or command line option has been set.

Listing
dumpIniParams.pm - sub initialize() - turning logging on if requested

```
#---- Set/Reset Logging
$bLog = $FALSE;
if (index( lc($a2wConfigPar->getAttribute("LoggingLevel")), "sf") >= 0 ){
    $bLog = $TRUE;
}
```

In this context, we can use the functions of `a2w::Config` and `a2w::Kernel`. For example, to access any processing parameter, use the `getAttribute` method of `a2w::Config` with the name of the INI parameter as argument. The following lines show how to use these functions:

Listing
dumpIniParams.pm - sub initialize() -

```
my $svsScriptProcTmp = $a2wConfigPar->getAttribute("ScriptProcedure");
my $svsScriptArgsTmp = $a2wConfigPar->getScriptArgs();
$svIndexFilePath = $a2wConfigPar->getIndexFilePath();
$svOutputFilePath = $a2wConfigPar->getOutputFilePath();
$svSpoolFilename = $a2wKernelPar->getSpoolFilename();
```

The following lines prepare to write output to a separate dump file. The name of this file must have been passed as argument using the command line option `-sa`. If this command line option is not entered, the script returns an error code to AFP2web and AFP2web will stop processing .

Listing
dumpIniParams.pm - sub initialize() - defining the file to dump output

```
#---- Open Dump file
my ($svSpoolFilenamePathTmp, $svDumpFilenameTmp) = ($svSpoolFilename =~
/^(?\. *[:\\\/])?(.*)/s);
$svDumpFilenameTmp = $svOutputFilePath . $svDumpFilenameTmp . ".txt";
open( fDumpFile, ">$svDumpFilenameTmp" );
print "Running $svScriptProcTmp: Dumping to $svDumpFilenameTmp...\n";
```

In the following code, we build an array containing all the AFP2web processing parameters. The for loop iterates through that array to get each parameter and then writes its value to the output file.

Listing
dumpIniParms.pm - sub initialize() - writing output to the dump file

```
my $ptrTmp = 0;
my @iniParmList = ();

@iniParmList[$ptrTmp++] = "License";
@iniParmList[$ptrTmp++] = "Serial Nr";
@iniParmList[$ptrTmp++] = "Title";
@iniParmList[$ptrTmp++] = "Subject";
@iniParmList[$ptrTmp++] = "Keywords";
# ----- additional items not listed here...
$iniParmCountTmp = @iniParmList;

print fDumpFile ("===== Ini Parameter List =====\n");
for ( $ptrTmp = 0; $ptrTmp < $iniParmCountTmp; $ptrTmp++ ){
print fDumpFile ( "\t" . @iniParmList[$ptrTmp] . "=>" . $a2wConfigPar-
>getAttribute(@iniParmList[$ptrTmp]) . "<=\n" );

}
print fDumpFile ("===== \n");
close(fDumpFile);
return 0;
```

There are no output documents resulting from this script. Because we return \$SKIP in the afp2web() subroutine, all pages are skipped:

Listing
dumpIniParms.pm - afp2web()

```
sub afp2web(){

if ( $bLog == $TRUE ){
print "afp2web(): PageId " . $a2wPagePar->getParseId() . "\n";
}

$APPEND= 0; # append page to Current Document
$SKIP= 1; # skip page
$NEWDOC= 2; # new document

$svRetTmp = $SKIP; # skip page

return $svRetTmp;
}
```

Running the Script Example

You use the following command in demo.bat to start AFP2web with the Scripting Facility:

Listing Demo.bat

```
@ECHO OFF
:On Windows: afp2web.exe -q -c -doc_col d -sp: dumpIniParms.pm sam-
ples\insure.afp

:On Unix: ./afp2web -q -c -doc_col d -sp: dumpIniParms.pm samples/i nsure.afp
```

Results of dumpIniParms.pm

You will find the output file written by dumpIniParms.pm in the output sub folder (the default is the file dumpIniParms.txt in the sub folder /pdf):

Listing dumpIniParms.pm - the contents of the dump file

```
===== Ini Parameter List =====
License=>Maas High Tech Software GmbH MMHHTT SF<=
SerialNr=>7CDEEOC4-542D980C<=
Title=>AFP2Web Version 3.0 [Built for Windows NT/2000/2003 on Aug 5 2005
at 12: 07: 32]<=
Subject=>AFP and TIFF Conversions (afp2web.com)<=
Keywords=>AFP LPD PDF TIFF<=
Logging=>off<=
LoggingFont=>off<=
LoggingLevel=>0<=
ExceptionLoggingLevel=>1<=
LogPath=>./log/<=
ResPath=>./samples/resource/<=
CpPath=>./afpcp/<=
CodePage=>T1V10273<=
OutputFilePath=>./pdf/<=
IndexPath=>./pdf/<=

..... additional lines not displayed here.....
```

autosplit.pm: Splitting an AFP Spoolfile into Documents and Pages

AFP documents are normally merged in spool files to be directed to a printer. One AFP spool file can contain any number of documents and pages. Non-visible information in this spool file can be, for example, the index data defined by Tagged Logical Elements (TLEs) of AFP. TLEs are logically related to documents and document pages to which they apply and are thus a means for AFP2web to recognize document and page boundaries.

If your type of AFP spool file has TLEs included, you can let AFP2web handle document and page splitting automatically and still use a script to perform other tasks. All you have to do is set the AutoSplit property to ON and then AFP2web will automatically split an AFP spool file into documents and pages.

This Perl script `autosplit.pm` shows how to use the AutoSplit functionality of AFP2web.

Understanding autosplit.pm

In this example, we focus on the subroutines:

<i>Subroutine</i>	<i>Description</i>
<code>sub afp2web()</code>	Main entry for AFP2web. The return code in this routine tells AFP2web to begin a new document with this page or not, to skip the current page, or to stop AFP2web processing.
<code>sub initialize()</code>	Accesses processing parameters, activates Autosplit, and initializes variables.
<code>sub initializeDoc()</code>	Saving document object, restart page count for this new document
<code>initializePage()</code>	Saving the current page object
<code>addDocumentIndexes()</code>	Collecting index data
<code>addPageIndexes()</code>	Collecting index data
<code>sub finalizeDoc()</code>	Output index data

The routine `initialize()` is called once at the beginning of the process. This routine is used to access processing parameters. For our example, we activate the `AutoSplit` functionality and initialize variables.

Listing
autosplit.pm - initialize() - Activating the autosplit functionality and initializing variables

```
...

#---- Reset Page Id
$pageId = 0;

#---- Reset/Create Current Index List
@indexList = ();

#---- Set AutoSplit to true
$a2wConfigPar->setAttribute( "AutoSplit", "on" );

...
```

AFP2web actively triggers the routine `initializeDoc()` when it starts processing a new document. Here too, we reset the page count to zero. Most important: we save a reference to the current document in the variable `$a2wDocumentPar`. Using this reference, we will later access the document objects. Because we can access the document index data at this point, we call the subroutine `addDocumentIndexes()` to collect the index data.

Listing
autosplit.pm - initializeDoc() - Saving document object, restart page count for this new document

```
sub initializeDoc(){
#---- Get Parameter of initializeDoc( Par: a2w::Document )
($a2wDocumentPar) = @_;

if ( $bLog == $TRUE ){
print "initializeDoc(): DocId " . $a2wDocumentPar->getId() . "\n";
}

#---- Reset Page Id
$pageId = 0;

#---- Add Document Indexes to Index List
addDocumentIndexes();

return 0;
}
```

In the routine `initializePage()`, we save a reference to the current page in the variable `$a2wPagePar`. Using this reference, we will later access page index data.

Listing
autosplit.pm - initializePage() - Saving the current page object

```
sub initializePage(){  
  
  #---- Get Parameter of initializePage( Par: a2w::Page )  
  ($a2wPagePar) = @_;  
  
  if ( $bLog == $TRUE ){  
    print "initializePage()\n";  
  }  
  return 0;  
}
```

The routine `afp2web()` is the main entry point for AFP2web. This routine is called for each document page.

Because AFP2web handles document and page splitting, we don't have much programming logic for this part. For example, we don't have to implement rules for recognizing the start of a new document and we therefore don't need to set the return code to `$NEWDOC`.

For document splitting, all we do is set a default return value (append current page to document) and increment the page count for the current document.

Finally, for each page, we can access the index data of this page. This is done using the subroutine `addPageIndexes()`.

Listing
autosplit.pm - sub afp2web()

```
sub afp2web(){  
  
    if ( $bLog == $TRUE ){  
        print "afp2web(): PageId " . $a2wPagePar->getParseId() . "\n";  
    }  
  
    $APPEND= 0; # append page to Current Document  
    $SKIP= 1; # skip page  
    $NEWDOC= 2; # new document  
  
    #---- Set default return value  
    my $svRetTmp = $APPEND; # default: append page  
  
    #---- Increment Page Id  
    $PageId++;  
  
    #---- Add Page Indexes to Index List  
    addPageIndexes();  
  
    return $svRetTmp;  
}
```

The following lines show our custom routines, which use the functions of either the document and page object to collect index data.

Listing
autosplit.pm - addDocumentIndexes() -

```
sub addDocumentIndexes(){  
  
    if ( $bLog == $TRUE ){  
        print "addDocumentIndexes()\n";  
    }  
  
    #---- Define temp variables  
    my $IndexPtrTmp = @IndexList;  
  
    #---- Fetch first Document Index  
    my $a2wIndexTmp = $a2wDocumentPar->getFirstIndex();  
  
    #---- Add PageGroup Name  
    if ( $a2wIndexTmp != 0 ){  
  
        #---- Add PageGroup Name to Index List  
        @IndexList[$IndexPtrTmp++] = "PageGroup=" . $a2wIndexTmp->getIndexOb-  
            jectName();  
    }  
  
    #---- Loop thru Indexes  
    while ( $a2wIndexTmp != 0 ){  
  
        #---- Add Index Record to Index List  
        @IndexList[$IndexPtrTmp++] = $a2wIndexTmp->getName() . "=" .  
            $a2wIndexTmp->getValue();  
  
        #---- Fetch next Document Index  
        $a2wIndexTmp = $a2wDocumentPar->getNextIndex();  
    }  
}
```

Listing
autosplit.pm - addPageIndexes() -

```

sub addPageIndexes(){
  if ( $bLog == $TRUE ){
    print "addPageIndexes()\n";
  }

  #---- Define temp variables
  my $IndexPtrTmp = @IndexList;

  #---- Add Page Id to Index List
  @IndexList[$IndexPtrTmp++] = "Page=" . $a2wPagePar->getName();

  #---- Fetch first Page Index
  my $a2wIndexTmp = $a2wPagePar->getFirstIndex();

  #---- Loop thru Indexes
  while ( $a2wIndexTmp != 0 ){

    #---- Add Index Record to Index List
    @IndexList[$IndexPtrTmp++] = $a2wIndexTmp->getName() . "=" .
      $a2wIndexTmp->getValue();

    #---- Fetch next Page Index
    $a2wIndexTmp = $a2wPagePar->getNextIndex();
  }
}

```

At the end of each document, that is, in the routine `finalizeDoc()`, we can print the index data collected for this document. The following listing shows a few lines of this routine. For the complete coding, please refer to the script example.

Listing
autosplit.pm - sub finalizeDoc() - output index data

```
#---- Open Index file
my $svFileOpenSuccessTmp = open( fIndexFile, ">$IndexFileNameTmp" );

if ( $svFileOpenSuccessTmp ){

#---- Write Document Id, Type, Name, PageCount and Size
print fIndexFile ("DocId=". $a2wDocumentPar->getId() .
", DocType=". $svDocType .
", DocName=". $svOutputFilePath . $a2wDocumentPar->getOutputFilename() .
", PageCount=". $a2wDocumentPar->getPageCount() .
", Size=". $a2wDocumentPar->getSize() . "\n");

#---- Write Indexes
for(my $IndexPtrTmp = 0; $IndexPtrTmp < $docIndexCountTmp; $IndexPtrTmp++){
print fIndexFile ("@IndexList[$IndexPtrTmp]\n");
}
print fIndexFile ("\n");
close( fIndexFile );
}
```

Listing
autosplit.pm - sub finalizeDoc() - resetting the index collection for the next document

```
#---- Reset Current Index List
@IndexList = ();
```

Running the Script Example

You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

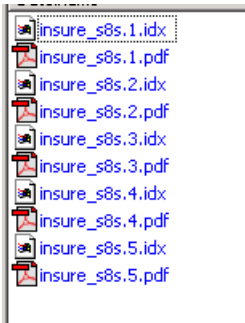
Listing
Demo Batch File

```
On Windows:
afp2web.exe -q -c -doc_col d -sp: autospl i t. pm samples\i nsure. afp

On Unix:
./afp2web -q -c -doc_col d -sp: autospl i t. pm samples/i nsure. afp
```

Results of autosplit.pm

This example produces PDF files for each document and a corresponding text file with index data:



A PDF document will have a start page, which begins like this:



The contents of the text file with index data will look like:

Listing
autosplit.pm - contents of a text file with index data

```
DocId=1, DocType=PDF, DocName=./pdf/insure_s8s.1.pdf, PageCount=3,  
Size=126321  
PageGroup=324-1443255-1100000001  
Insured=Geoffrey R Stephens  
Policy=324-1443255-11  
Page=00000001  
Contents=Disability Income Policy  
Page=00000002  
Contents=Policy Schedule  
Contents=Policy Schedule  
Contents=Table of Benefits  
Contents=Additional Benefits  
Contents=Premium Summary  
Page=00000003  
Contents=Definitions
```

For this example, we don't want to describe the structure of AFP elements in detail. If you want to investigate your AFP data, you might want to use the command option `-ll` to create an AFP2web log file.

The following screen shot shows part of the log file created when running the command option `-ll:ALL` for our example. We can find entries for index data, the first Index is located at the beginning of the document :

```

> samples/insure.afp
}3 A8 A7 BDI Begin Document Index
    Fully Qualified Name Triplet
    Fully Qualified Name used (FQNTtype): 0x01
    This GID replaces the first parameter in the structured field that contains
    Fully Qualified Name Triplet
    Fully Qualified Name used (FQNTtype): 0x83
    The triplet contains a GID reference to a Begin Document structured field
    Fully Qualified Name Triplet
    Fully Qualified Name used (FQNTtype): 0x0a
    The triplet contains a GID reference to a Begin Resource Group structured
}3 B2 A7 IEL Index Element
    Object Byte Extent Triplet
    Byte Extent of Indexed object: 0x2ca1
    Direct Byte Offset Triplet
    Offset of the indexed object in bytes: 0x94
    Object Structured Field Extent Triplet
    Extent, in structured fields, to the indexed object: 0x62
    Object Structured Field Offset Triplet
    Offset, in structured fields, to the indexed object: 0x02
    Medium Map Page Number Triplet
    Sequence Number of The Indexed Page: 0x01
    Fully Qualified Name Triplet
    Fully Qualified Name used (FQNTtype): 0x0d
    The triplet contains a GID reference to a Begin Named Page Group structure
}3 A0 90 TLE Tag Logical Element
    Sequence Number: 00000000
    Level Number: 00000000
    Fully Qualified Name Triplet
    Fully Qualified Name used (FQNTtype): 0x0b
    The triplet contains the GID of a document attribute: Insured
    Attribute Value Triplet
    Attribute Value: Geoffrey R Stephens
}3 A0 90 TLE Tag Logical Element
    Sequence Number: 00000000
    Level Number: 00000000
    Fully Qualified Name Triplet
    Fully Qualified Name used (FQNTtype): 0x0b
    The triplet contains the GID of a document attribute: Policy
    Attribute Value Triplet
    Attribute Value: 324-1443255-11
}2 B2 A7 TFI Index Element

```

dumpPageTextObjs.pm: Dumping Text and Font Information

The Perl script `dumpPageTextObjs.pm` shows how to access all text objects of a page and to write their text values to an output file (the dump file). Whenever a font information changes, this information will be written to the dump file immediately before the text affected by this change.

Understanding dumpPageTextObjs.pm

In this example, we focus on the subroutines:

<i>Subroutine</i>	<i>Description</i>
<code>sub initialize</code>	Initializes and creates the dump file.
<code>sub initializeDoc()</code>	Saves the current document object
<code>sub initializePage()</code>	Saves the current document page
<code>sub afp2web()</code>	Main entry for AFP2web. The return code in this routine tells AFP2web to begin a new document with this page or not, to skip the current page, or to stop AFP2web processing. In this routine, our example writes the text of the page to the dump file. Whenever a change in the font information occurs, this information is also written to the dump file.
<code>sub finalizeDoc()</code>	Writes index data to file for this document.

The subroutine `initialize()` includes commands to initialize and create the dump file:

Listing
dumpPageTextObjs.pm - initializeDoc()

```

sub initialize(){#-- Get Parameter of initialize( Par: a2w::Config,
a2w::Kernel )
( $a2wConfigPar, $a2wKernelPar ) = @_;

my $svScriptProcTmp = $a2wConfigPar->getAttribute("ScriptProcedure");
my $svScriptArgsTmp = $a2wConfigPar->getScriptArgs();
$svIndexPath = $a2wConfigPar->getIndexPath();
$svOutputFilePath = $a2wConfigPar->getOutputFilePath();
$svSpoolFilename = $a2wKernelPar->getSpoolFilename();

#---- Open Dump file
my ($svSpoolFilePathTmp, $svDumpFilenameTmp) = ($svSpoolFilename =~
/^(?:(?:[\\\/])?)(.*)$/s);
$svDumpFilenameTmp = $svOutputFilePath . $svDumpFilenameTmp . ".txt";
open( fDumpFile, ">$svDumpFilenameTmp" );
print "Running $svScriptProcTmp: Dumping to $svDumpFilenameTmp...\n";

return 0; }

```

The subroutine `initializeDoc()` is called when AFP2web signals the beginning of a new document. We use the Perl variable `@_` to save the document, which is passed as argument .

Listing
dumpPageTextObjs.pm - initializeDoc()

```

sub initializeDoc(){
#---- Get Parameter of initializeDoc( Par: a2w::Document )
($a2wDocumentPar) = @_;

return 0; }

```

It is equally important that we save the current page object so that we can later access its elements. This is done in the subroutine `initializePage()`:

Listing
dumpPageTextObjs.pm - initializePage()

```
sub initializePage(){#---- Get Parameter of initializePage( Par: a2w: Page
)
($a2wPagePar) = @_ ;
return 0; }
```

The heart of the logic is located in the routine `afp2web()`. The first action is to initial the return code value, which will control AFP2web processing and print the the current page ID (number of the document in the spool file) to the dump file.

Listing
dumpPageTextObjs.pm - afp2web() - initializing the return code

```
sub afp2web(){
if ( $bLog == $TRUE ){
print "afp2web(): PageId " . $a2wPagePar->getParseId() . "\n";
}

$APPEND= 0; # append page to Current Document
$SKIP= 1; # skip page
$NEWDOC= 2; # new document

$svRetTmp = $APPEND; # default: append page

#---- Get Page Id
my $svPageIdTmp = $a2wPagePar->getParseId();

#---- Dump
printf fDumpFile ("===== ");
printf fDumpFile ("Page %06d", $svPageIdTmp);
printf fDumpFile (" =====\n");

.... additional lines not shown here....

return $svRetTmp;
}
```

The routine `afp2web()` is called for each document page. We use this routine to loop thru all text objects that can be found on the page and to print the text found to the dump file.

Listing***dumpPageTextObjs.pm - afp2web() - writing all text to the dump file***

```
#---- Fetch first Text Object
my $a2wTextTmp = $a2wPagePar->getFirstText();

#---- Loop thru all the Texts
while ( $a2wTextTmp != 0 ){

.... additional lines... for dumping font information .(see next list-
ing)....

#---- Build String to be dumped with details
printf fDumpFile (" @(" . $a2wTextTmp->getXPos() . ", " . $a2wTextTmp-
>getYPos . ")>" . $a2wTextTmp->getText() . "<\n");

#---- Get the next Text Object
$a2wTextTmp = $a2wPagePar->getNextText();
} # end-while
```

While looping thru text objects we investigate the font information and whenever a change occurs, we write new font information to the dump file. We extend this loop with the following lines:

Listing

dumpPageTextObjs.pm - afp2web() - writing font information to the dump file

```
#---- Write font entry with font details
$svTmp = $a2wTextTmp->getMappedFontLocalId();

if ( $svTmp != $svFontLocalIdTmp ){ # Compare Font Local Id against Current one

#---- Save Font Local Id as Current
$svFontLocalIdTmp = $svTmp;

#---- Fetch Font Object
$a2wFontTmp = $a2wTextTmp->getFont();

#---- Build String to be dumped
$svFontEntryTmp = "===";

#---- Fetch Coded Font name
$svTmp = $a2wFontTmp->getCodedFontName();
if ( $svTmp ne "" ){
$svFontEntryTmp .= "CF=$svTmp, ";
}

#---- Fetch Character Set name
$svTmp = $a2wFontTmp->getCharacterSetName();
if ( $svTmp ne "" ){
$svFontEntryTmp .= "CS=$svTmp, ";
}

.... additional font attributes ....

#---- Dump Font Entry to File
printf fDumpFile (" $svFontEntryTmp\n");
}
```

Running the Script Example


You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

Listing
Demo Batch File

```
On Windows:
afp2web.exe -q -c -doc_col d -sp: dumpPageTextObjs. pm samples\i nsure. afp#0n
Unix:
./afp2web -q -c -doc_col d -sp: dumpPageTextObjs. pm samples/i nsure. afp#
```

Results of dumpPageTextObjs.pm

The resulting PDF document will have a start page, which begins like this:



Insurance Specialists

Disability Income Policy

The Preferred Professional

Insurance Specialists Company
100 Main Street
Anycity, Anystate 99999-9999

Insurance Specialists Company will pay the benefits provided in this Policy for loss due to Injury or Sickness.
We have issued this Policy to You in consideration of the payment of the premium and the statements made in Your application. Your application is part of Your Policy.

Insured *Geoffrey R Stephens*

Policy Number *324-1443255-11* *03-25-53* **Date of Issue**

You will find the dump file in the output sub folder (the default is the sub folder /pdf). The following example shows a few lines of the content you will find in the dump file:

Listing
dumpPageTextObjs.pm - contents of the dump file (extract)

```
===== Page 000001 =====  
==>CS=C1H400B0, TF=HELVETICA LATIN1, W=B, S=12.00, CP=T1GI 0361  
@(8496, 862)>Insurance<  
@(8496, 1120)>Specialists<  
==>CS=C1H400J0, TF=HELVETICA LATIN1, W=B, S=20.00, CP=T1GI 0361  
@(792, 2308)>Disability Income Policy<  
==>CS=C1H200D0, TF=HELVETICA LATIN1, W=R, S=14.00, CP=T1GI 0361  
@(792, 2638)>The Preferred Professional<  
==>CS=C1H40000, TF=HELVETICA LATIN1, W=B, S=10.00, CP=T1GI 0361  
@(7330, 2209)>Insurance Specialists Company<  
==>CS=C1H20090, TF=HELVETICA LATIN1, W=R, S=9.00, CP=T1GI 0361  
@(7330, 2418)>100 Main Street<  
@(7330, 2627)>Anycity, Anystate 99999-9999<  
==>CS=C1H20000, TF=HELVETICA LATIN1, W=R, S=10.00, CP=T1GI 0361  
@(792, 4104)>Insurance Specialists Company will pay the benefits provided  
in this Policy for loss due to Injury or Sickness.<  
@(792, 4471)>We have issued this Policy to You in consideration of the pay-  
ment of the premium and the statements made in<  
@(792, 4681)>Your application. Your application is part of Your Policy.<  
==>CS=C1H40000, TF=HELVETICA LATIN1, W=B, S=10.00, CP=T1GI 0361  
@(792, 5353)>Insured<
```

sortPageTextObjs.pm: Sorting Text Objects in their Print Sequence

Raw AFP print data does not necessarily group and present text objects in the order in which they are printed. This makes it difficult to read AFP data dumped into text files.

sortPageTextObjs.pm shows how to access all text objects of a page. It sorts all text objects according to their X/Y coordinate positions and then writes their text values to an output file (the dump file) in this order.

Understanding sortPageTextObjs.pm

In this example, we focus on the subroutines:

<i>Subroutine</i>	<i>Description</i>
sub afp2web()	Main entry for AFP2web. The return code in this routine tells AFP2web to begin a new document with this page or not, to skip the current page, or to stop AFP2web processing. In this routine, our example collect the text of the page and then calls the subroutine buildLines() to sort and write the text to the dump file.
sub buildLines()	Sorts the text by y-coordinate (line number) and then by x-coordinate (position within the line)
sub complex_arrays()	Specifies which keys to use for the sort algorithm.

The routine afp2web() is the main entry point for AFP2web. This routine is called for each document page.

The first action is to initialize the return code value, which will control AFP2web processing. In addition, the current page ID (number of the document in the spool file) is written to the dump file. We do not show this here.

In the following, we show how afp2web() collects all text objects of the current page and then calls custom subroutine buildLines to sort the text:

Listing

sortPageTextObjs.pm - afp2web() - Gathering text objects and processing the results

```
#---- Add the Page Text Objects to an unsorted Object List
@unsortedObjList = ();
my $PtrTmp = @unsortedObjList;
#---- Define temp variables
my $svTextTmp = "";
my $svTextXPosTmp = 0;
my $svTextYPosTmp = 0;

#---- Fetch first Text Object
my $a2wTextTmp = $a2wPagePar->getFirstText();

#---- Loop thru all the Text Objects
while ( $a2wTextTmp != 0 ){
    $svTextTmp = $a2wTextTmp->getText();
    $svTextXPosTmp = $a2wTextTmp->getXPos();
    $svTextYPosTmp = $a2wTextTmp->getYPos();
    if ( $bLog == $TRUE ){
        print " @(" . $svTextXPosTmp . "," . $svTextYPosTmp . ">" . $svTextTmp .
        "<\n";
    }
    #---- Add Object to unsorted Object List
    @unsortedObjList[$PtrTmp++] = ({XPOS => $svTextXPosTmp,
    YPOS => $svTextYPosTmp,
    TEXT => $svTextTmp
    });
    #---- Get the next Text Object
    $a2wTextTmp = $a2wPagePar->getNextText();
} # end-while

#---- Build/Dump Lines
buildLines();
```

The subroutine buildLines() first sorts the text and then prints the text in the sort order to the dump file. The Y-coordinate determines the line number. Whenever a new line occurs, the new line number is printed to the dump file.

Listing
sortPageTextObjs.pm - buildLines()

```
sub buildLines(){

#---- Define temp variables
my $LineTmp = "";
my $LineCountTmp = 0;
my $currYPosTmp = 0;
my @sortedObjList = ();

#---- Sort Array: first Key is YPOS, second Key is XPOS
@sortedObjList = sort complex_arrays @unsortedObjList;

#---- Build Lines based on sorted Array
foreach (@sortedObjList){

$YPosTmp = $_->{YPOS};
$TextTmp = $_->{TEXT};

if ( $currYPosTmp != $YPosTmp && $currYPosTmp != 0 ){

#---- Add line
printf fDumpFile ("%06d:", $LineCountTmp);
print fDumpFile (" $LineTmp\n");

$LineCountTmp++;
$LineTmp = $TextTmp;

}
else{ # append Text to current line
$LineTmp = $LineTmp . $TextTmp;
}
$currYPosTmp = $YPosTmp;
}
}
```

The subroutine `complex_arrays()` delivers the first set of sort keys required for the sort command. In this case, the sort should be done first for the y-coordinates (the line number of the text objects) and then for the x-coordinates (the position of the text within the line).

Listing
sortPageTextObjs.pm - complex_arrays()

```
#-----  
# Sorting Algorithm  
#  
#1st Key is YPOS, 2nd Key is XPOS  
#-----  
sub complex_arrays{  
  $a->{YPOS} <=> $b->{YPOS} ||  
  $a->{XPOS} <=> $b->{XPOS};  
}
```

Running the Script Example

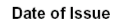
You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

Listing
Demo Batch File

```
On Windows:  
afp2web.exe -q -c -doc_col d -sp: sortPageTextObj s. pm samples\i nsure. afp  
  
On Unix:  
. /afp2web -q -c -doc_col d -sp: sortPageTextObj s. pm samples/i nsure. afp
```

Results of sortPageTextObjs.pm

The resulting PDF document will have a start page, which begins like this:



Copyright © 2008 Maas High Tech Software GmbH

Listing
sortPageTextObjs.pm - contents of the dump file (extract)

```
===== Page 000001
=====
000000: Demo Version
000001: Specialists
000002: Insurance Specialists Company
000003: Disability Income Policy
000004: 100 Main Street
000005: Anycity, Anystate 99999-9999
000006: Demo Version
000007: Insurance Specialists Company will pay the benefits provided in
this Policy for loss due to Injury or Sickness.
000008: We have issued this Policy to You in consideration of the payment
of the premium and the statements made in
000009: Your application. Your application is part of Your Policy.
000010: Demo VersionGeoffrey R Stephens
000011: Policy Number324-1443255-1103-25-53Demo Version
000012: Demo Version
000013: As long as the premium is paid on time, We cannot change Your Pol-
icy or its premium rate until the first premium
000014: due date after Your 65th birthday.
000015: RENEWAL OPTIONS AFTER YOU REACH AGE 65. SUBJECT TO CHANGE IN PRE-
MIUM RATES. Demo Version
000016: 65 to age 72, You may continue Your Policy for a Total Disability
benefit with a limited benefit period while You
000017: are actively and regularly employed full time. This option is
explained in PART 5.

..... additional lines not shown here....
```

dumpMediumMaps.pm: Output List of Medium Maps to a Dump File

This Perl script `dumpMediumMaps.pm` shows how to create a dump file with a list of all medium maps applied to a document page.

Understanding dumpMediumMaps.pm

In this example, we focus on the subroutines:

<i>Subroutine</i>	<i>Description</i>
<code>sub afp2web()</code>	Main entry for AFP2web. The return code in this routine tells AFP2web to begin a new document with this page or not, to skip the current page, or to stop AFP2web processing. We show how <code>afp2web()</code> accesses and prints information about the current medium map applied to the page.

The routine `afp2web()` is the main entry point for AFP2web. That is, it will be invoked at each parsing event, which can be trapped by the Scripting Facility. In our case, this routine will be called at the beginning of each document page.

The commands used to print information on the current medium map:

Listing

dumpMediumMaps.pm - afp2web() -

```
#---- Get Page Id
my $svPageIdTmp = $a2wPagePar->getParseId();

#---- Dump
printf fDumpFile ("===== ");
printf fDumpFile ("Page %06d", $svPageIdTmp);
printf fDumpFile (" =====\n");

#---- Fetch applied medium map
my $a2wMediumMapTmp = $a2wPagePar->getAppliedMediumMap();

if ( $a2wMediumMapTmp != 0 ){

#---- Dump Formdef Name
printf fDumpFile ("Formdef: " . $a2wMediumMapTmp->getFormdefName() .
"\n");

#---- Dump Medium Map Name
printf fDumpFile ("Medium Map: " . $a2wMediumMapTmp->getName() . ", ");

#---- Dump Medium Map Width, Height, Resolution, Duplex Control and N-Up
Control
printf fDumpFile ("Width=". $a2wMediumMapTmp->getWidth() .
", Height=". $a2wMediumMapTmp->getHeight() .
", Resolution=". $a2wMediumMapTmp->getResolution() .
", Duplex Control=" . $a2wMediumMapTmp->getDuplexControl() .
", N-Up Control=" . $a2wMediumMapTmp->getNupControl() .
"\n");
}
```

Starting the Script Example

You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

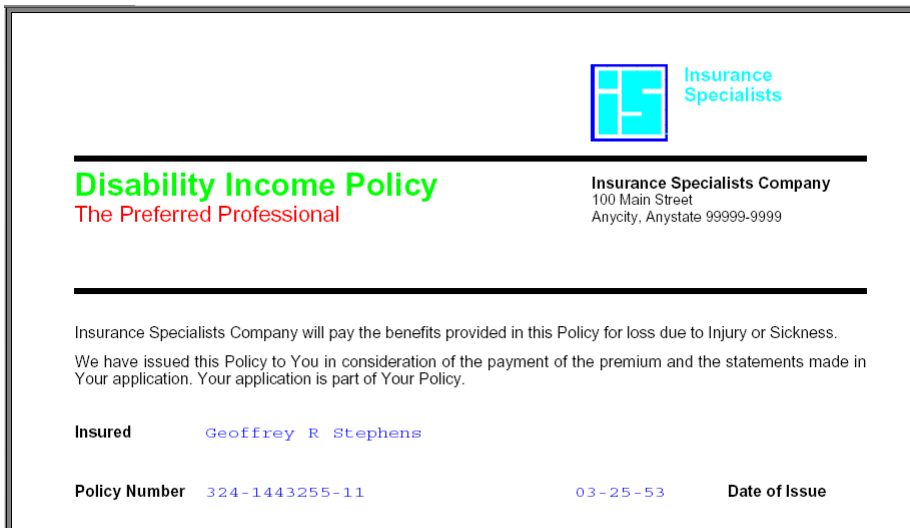
Listing
Demo Batch File

```
On Windows:
afp2web.exe -q -c -doc_col d -sp: dumpMedi umMaps. pm samples\i nsure. afp

On Unix:
./afp2web -q -c -doc_col d -sp: dumpMedi umMaps. pm sampl es/i nsure. afp
```

Results of dumpMediumMaps.pm

The resulting PDF document will have a start page, which begins like this:



You will find the dump file in the output sub folder (the default is the sub folder /pdf). In our example, the same medium map is applied to all pages. The first lines of the dump file are:

Listing
dumpMediumMaps.pm - contents of the dump file (extract)

```
===== Page 000001
=====
Formdef: F1DEMO
Medium Map: OGL, Width=0, Height=0, Resolution=240, Duplex Control=1, N-Up
Control=1
===== Page 000002
=====
Formdef: F1DEMO
Medium Map: OGL, Width=0, Height=0, Resolution=240, Duplex Control=1, N-Up
Control=1
===== Page 000003
=====
Formdef: F1DEMO
Medium Map: OGL, Width=0, Height=0, Resolution=240, Duplex Control=1, N-Up
Control=1

.... additional lines not shown here....
```

dumpNOPs.pm: Output the Contents of NOP Fields Into a Dump File

NOP fields is a common method of passing non-visible document information. Various applications use their conventions for handling NOP fields - for example for printer controls, for indexing, for processing control.

You might want to view the existing NOP-information in your AFP spool files.

This Perl script `dumpPageTextObjs.pm` shows how to produce a list of the values passed as NOP fields per page.

Understanding dumpNOPs.pm

In this example, we focus on the subroutines:

<i>Subroutine</i>	<i>Description</i>
<code>sub afp2web()</code>	<p>Main entry for AFP2web.</p> <p>The return code in this routine tells AFP2web to begin a new document with this page or not, to skip the current page, or to stop AFP2web processing.</p> <p>What we show is how <code>afp2web()</code> processes all NOP objects of the current page.</p>

The routine `afp2web()` is the main entry point for AFP2web. That is, it will be invoked at each parsing event, which can be trapped by the Scripting Facility. In our case, this routine will be called at the beginning of each document page.

We show how `afp2web()` collects all NOP objects of the current page and then prints the NOP values to dump file:

Listing
dumpNOPs.pm - afp2web() - ... printing NOP values to the dump file

```
#---- Get Page Id
my $svPageIdTmp = $a2wPagePar->getParseId();

#---- Dump
printf fDumpFile ("===== ");
printf fDumpFile ("Page %06d", $svPageIdTmp);
printf fDumpFile (" =====\n");

#---- Fetch 1st NOP
my $a2wNOPTmp = $a2wPagePar->getFirstNOP();

if ( $a2wNOPTmp != 0 ){

my $svNOPDataTmp = $a2wNOPTmp->getValue();

print fDumpFile ("NOP=$svNOPDataTmp\n");

#---- Fetch next NOP
$a2wNOPTmp = $a2wPagePar->getNextNOP();

#---- Loop for each and every NOP
while ( $a2wNOPTmp != 0 ){

$svNOPDataTmp = $a2wNOPTmp->getValue();

#---- Write NOP to dump file
print fDumpFile ("NOP=$svNOPDataTmp\n");

#---- Fetch next NOP
$a2wNOPTmp = $a2wPagePar->getNextNOP();

}; # end-while
}
```

Starting the Script Example

You use the following command in the demo batch file to start AFP2web with the Scripting Facility:


Listing
Demo Batch File

```
On Windows:
afp2web.exe -q -c -doc_col d -sp: dumpNOPs. pm  sampl es\i nsure. afp

On Unix:
./afp2web -q -c -doc_col d -sp: dumpNOPs. pm  sampl es/i nsure. afp
```

Results of dumpNOPs.pm

The resulting PDF document will have a start page, which begins like this:



Insurance
Specialists

Disability Income Policy

The Preferred Professional

Insurance Specialists Company
100 Main Street
Anycity, Anystate 99999-9999

Insurance Specialists Company will pay the benefits provided in this Policy for loss due to Injury or Sickness.
We have issued this Policy to You in consideration of the payment of the premium and the statements made in
Your application. Your application is part of Your Policy.

Insured *Geoffrey R Stephens*

Policy Number *324-1443255-11* *03-25-53* **Date of Issue**

NON-CANCELLABLE AND GUARANTEED CONTINUABLE TO AGE 65. NO CHANGE IN PREMIUM RATES.

You will find the dump file in the output subfolder (the default is the sub folder /pdf). Each line is preceded by the document page number. However, we find only one NOP object:

Listing
dumpNOPs.pm - contents of the dump file (extract)

```
===== Page 000001 =====
NOP=SCRIPT/VS 4.0.0: DEVICE AFP4 CHARS XON2200E
===== Page 000002 =====
===== Page 000003 =====
===== Page 000004 =====
===== Page 000005 =====
===== Page 000006 =====
===== Page 000007 =====
===== Page 000008 =====
===== Page 000009 =====
===== Page 000010 =====
===== Page 000011 =====
===== Page 000012 =====
===== Page 000013 =====
===== Page 000014 =====
===== Page 000015 =====
```

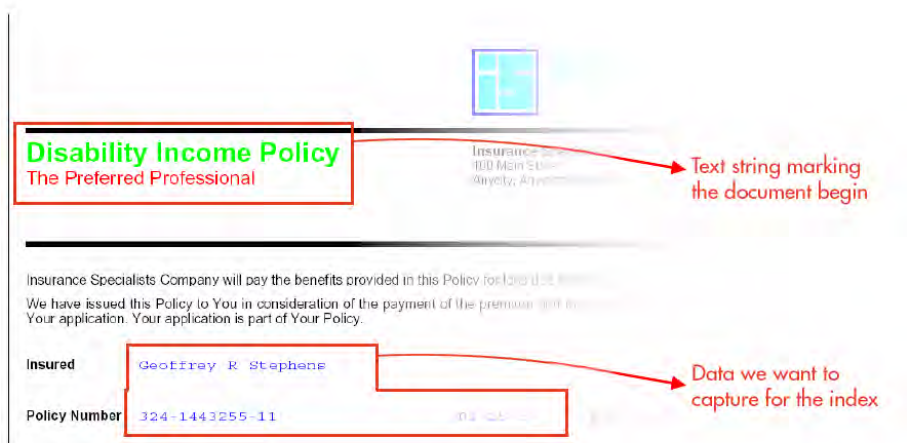
Using the command line option `-ll:ALL`, we create a log file. When looking into the log file created for this example, we see that there is actually only one NOP object :

The screenshot shows a PDF viewer window with a search dialog titled "Lines containing find string:". The search results list several lines, with the first one highlighted: "0x00000FE9 : D3 EE EE NOP No Operation". The PDF content below the search dialog shows a series of lines, including "Interchange Set Type: Pr", "Interchange set identifi", "Fully Qualified Name Tri", "Fully Qualified Name use", "This GID replaces the fi", "Fully Qualified Name Tri", "Fully Qualified Name use", "The triplet contains a G", "Fully Qualified Name Tri", "Fully Qualified Name used (FontType): 0x98", "The triplet contains a GID reference to a Begin Document Index structured field: SMLIN", "0x00000FE9 : D3 EE EE NOP No Operation", "E2 C3 D9 C9 D7 E3 61 E5 E2 40 F4 4B F0 4B F0 7A 40 C4 C5 E5 C9 C3 C5 40 C1 C6 D7 C1", "FO F0 C5", "SCRIPT/VS 4.0.0: DEVICE AFP4 CHARS XON2200E", and "0x0000101D : D3 A8 AD BNG Begin Named Page Group 00000001".

eyecatcher.pm: Find and Extract Text Objects

The Perl script `eyecatcher.pm` shows how to find and extract predefined text strings on a page. Each text string is located within a given y/x coordinate range.

- A particular text string indicates the first page in a document.
- Three text strings deliver the values for the document index, which shall be written to an output file.



Using eyecatcher.pm to find text coordinates

First, we want to determine the page position of these text elements. We run the AFP2web Scripting Facility with the following commands in the subroutine `afp2web()`:

Listing

eyecatcher.pm - Investigating the text positioning properties

```
sub afp2web(){
$APPEND = 0; # append page to Current Document
$SKIP = 1; # skip page
$NEWDOC = 2; # new document
#---- Set default return value
my $svRetTmp = $APPEND; # default: append page

#---- Fetch first Text Object
my $a2wTextTmp = $a2wPagePar->getFirstText();

#---- Define temp variables
my $svTextTmp = "";
my $svTextXPosTmp = 0;
my $svTextYPosTmp = 0;

#---- Loop thru all the Text Objects
while ( $a2wTextTmp != 0 ){

$svTextTmp = $a2wTextTmp->getText();
$svTextXPosTmp = $a2wTextTmp->getXPos();
$svTextYPosTmp = $a2wTextTmp->getYPos();
print $svTextXPosTmp . ", " . $svTextYPosTmp . ">" . $svTextTmp . "\n";

#---- Fetch next Text Object
$a2wTextTmp = $a2wPagePar->getNextText();
}

#---- Increment Page Id
$pageId++;

return $svRetTmp;
}
```

We use the following command in demo.bat to start AFP2web with the Scripting Facility:

Listing

Demo.bat (lines 1 to 4)

```
afp2web.exe -q -c -doc_cold -sp:eyecatcher.pm samples\insure.afp > dump-
file.txt
```

The script produces the following output, which helps us determine the x/y-coordinates of the text strings:

Listing
eyecatcher.pm -

```
8496, 862>Insurance
8496, 1120>Specialists
792, 2308>Disability Income Policy
792, 2638>The Preferred Professional
7330, 2209>Insurance Specialists Company
7330, 2418>100 Main Street
7330, 2627>Anycity, Anystate 99999-9999
792, 4104>Insurance Specialists Company will pay the benefits provided in
this Policy for loss due to Injury or Sickness.
792, 4471>We have issued this Policy to You in consideration of the payment
of the premium and the statements made in
792, 4681>Your application. Your application is part of Your Policy.
792, 5353>Insured
2448, 5353>Geoffrey R Stephens
792, 6097>Policy Number
2448, 6097>324-1443255-11
7114, 6097>03-25-53
....additional lines not shown here....
```

Using eyecatcher.pm to split documents and retrieve index data

In this example, we focus on the subroutines

<i>Subroutine</i>	<i>Description</i>
sub initializeDoc()	Saves the current document object. Resets page numbering
sub afp2web()	The return code in this routine tells AFP2web to begin a new document with this page or not, to skip the current page, or to stop AFP2web processing. In this routine, our example recognizes the start page of a new document. Calls the subroutine addPageEyeCatcherIndexes() for the first page of the document.

<i>Subroutine</i>	<i>Description</i>
sub finalizeDoc()	Writes index data to file for this document.
sub addPageEyeCatcherIndices()	A custom routine that extracts three text values from the current page and saves it as index data.

The subroutine initializeDoc() is called when AFP2web signals the beginning of a new document. We use the Perl variable @_ to save the document, which is passed as argument . And we reset the page counter to zero.

Listing
eyecatcher.pm - initializeDoc()

```
sub initializeDoc(){  
  
#---- Get Parameter of initializeDoc( Par: a2w::Document )  
($a2wDocumentPar) = @_  
  
return 0;  
}
```

It is equally important that we save the current page object so that we can later access its elements. This is done in the subroutine initializePage():

Listing
eyecatcher.pm - initializePage()

```
sub initializePage(){  
  
#---- Get Parameter of initializePage( Par: a2w::Page )  
($a2wPagePar) = @_  
  
}
```

The routine afp2web() is used to process each document page. This routine formulates the return code value to be passed to AFP2web:

Listing***eyecatcher.pm - afp2web() - initializing the return code***

```
# Main entry method
# Return values:
# < 0: error
# 0: append page to Current Document
# 1: skip page
# 2: first page / new document
#-----
sub afp2web(){

$APPEND= 0; # append page to Current Document
$SKIP= 1; # skip page
$NEWDOC= 2; # new document

$svRetTmp = $APPEND; # default: append page

... additional lines with processing logic not shown here...

return $svRetTmp;
}
```

The processing logic in the routine `afp2web()` handles splitting the spool file into documents. The rule is: The start page of a new document always has the text "Disability Income Policy" on a fixed position on that page. If this text value is found, we set the return code for `AFP2web` to initiate processing for the end of the previous document and the begin of a new document.

Listing

eyecatcher.pm afp2web() - Using an eyecatcher for document splitting

```
.....
#---- Fetch first Text Object
my $a2wTextTmp = $a2wPagePar->getFirstText();

#---- Loop thru all the Text Objects
while ( $a2wTextTmp != 0 ){

#---- If true ==> new doc
if ( $a2wTextTmp->getXPos() >= 782 &&
$a2wTextTmp->getXPos() <= 802 &&
$a2wTextTmp->getYPos() >= 2298 &&
$a2wTextTmp->getYPos() <= 2318 &&
$a2wTextTmp->getText() eq "Di sabil ity Income Policy" ){

#---- Reset Page Id
$PageId = 0;

$svRetTmp = $NEWDOC;
last; # leave while loop
}

#---- Fetch next Text Object
$a2wTextTmp = $a2wPagePar->getNextText();
}

#---- Increment Page Id
$PageId++;
```

When AFP2web closes a document, it will invoke the subroutine `finalizeDoc()`. Here, we fetch and output the index data from the document. This simple routine calls the subroutine `addFirstPageEyecatcherIndexes()`, which is responsible for fetching the index data. You have full control of the document content even during the event of finalizing the document. To get the start page of the document, just use the method `$a2wDocumentPar->getFirstPage()` and scan that page for your index data.

Listing***eyecatcher.pm - addFirstPageEyecatcherIndexes() called by finalizeDoc()***

```
sub addFirstPageEyecatcherIndexes(){

#---- Fetch the first Page of Document
my $a2wFirstPageTmp = $a2wDocumentPar->getFirstPage();

#---- If a Page exists...
if ( $a2wFirstPageTmp != 0 ){

#---- Define the list of Eyecatcher positions we should look for
my @arrEyeCatcherIndexListTmp = ( ( 2448, 5353 ), # Insured
@(2448, 5353)>Geoffrey R Stephens<
( 2448, 6097 ), # Policy Number @(2448, 6097)>324-1443255-11<
( 7114, 6097 ) ); # Date of Issue @(7114, 6097)>03-25-53<

#---- Define temp variables
my $arrECILLenTmp = @arrEyeCatcherIndexListTmp / 2;

.... additional lines not shown here...
```

Listing

eyecatcher.pm - addFirstPageEyecatcherIndexes() called by finalizeDoc()

```
#---- Get the Index list count
my $IndexPtrTmp = @IndexList;

#---- Fetch first Text Object
my $a2wTextTmp = $a2wFirstPageTmp->getFirstText();

#---- Loop thru all the Text Objects
while ( $a2wTextTmp != 0 ){

    $svTextTmp = $a2wTextTmp->getText();
    $svTextXPosTmp = $a2wTextTmp->getXPos();
    $svTextYPosTmp = $a2wTextTmp->getYPos();

    #---- Search all the defined Eyecatchers
    for (my $i = 0; $i < $arrECILLenTmp; $i++ ){
        $svArrPosTmp = $i * 2;
        $svIdxTextXTmp = $arrEyeCatcherIndexListTmp[ $svArrPosTmp ];
        $svIdxTextYTmp = $arrEyeCatcherIndexListTmp[ $svArrPosTmp + 1 ];

        #---- If true ==> we found one of the defined Eyecatchers
        if ( $svTextXPosTmp >= ( $svIdxTextXTmp - 10 ) &&
            $svTextXPosTmp <= ( $svIdxTextXTmp + 10 ) &&
            $svTextYPosTmp >= ( $svIdxTextYTmp - 10 ) &&
            $svTextYPosTmp <= ( $svIdxTextYTmp + 10 ) ){

            if ( $i == 0 ){
                $svIndexValueTmp = "Insured=" . $svTextTmp;
            } elsif ( $i == 1 ){
                $svIndexValueTmp = "Policy Number=" . $svTextTmp;
            } elsif ( $i == 2 ){
                $svIndexValueTmp = "Date of Issue=" . $svTextTmp;
            }

            #---- Add Eyecatcher to Index List
            @IndexList[$IndexPtrTmp++] = $svIndexValueTmp;

            if ( $bLog == $TRUE ){
                print "$svIndexValueTmp\n";
            }
        }

        #---- Fetch next Text Object
        $a2wTextTmp = $a2wFirstPageTmp->getNextText();
    }
}
```

Starting the Script Example

You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

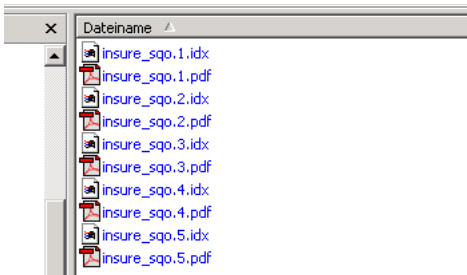
Listing **Demo Batch File**

```
On Windows:
afp2web.exe -q -c -doc_col d -sp: eyecatcher.pm samples\i nsure.afp

On Unix:
./afp2web -q -c -doc_col d -sp: eyecatcher.pm sampl es/i nsure.afp
```

Results of eyecatcher.pm

You will find the output file in the output sub folder (the default will be in the sub folder /pdf):



The contents of the index file

Listing **eyecatcher.pm - contents of an index file**

```
DocId=1, DocType=PDF, DocName=. /pdf/i nsure_s2qc. 1. pdf, PageCount=3,
Si ze=114669
Insured=Geoffrey R Stephens
Pol icy Number=324-1443255-11
Date of Issue=03-25-53
```



Insurance
Specialists

Insurance Specialists Company
100 Main Street
Anycity, Anystate 99999-9999

NON-CANCELLABLE AND GUARANTEED CONTINUABLE TO AGE 65. NO CHANGE IN PREMIUM RATES.

addBookmarks.pm: Adding Bookmarks to the PDF output document

addBookmarks.pm generates bookmarks in the PDF document.

Understanding addBookmarks.pm

In this example, we focus on the subroutines

<i>Subroutine</i>	<i>Description</i>
sub initializeDoc()	Saves the current document object. Resets page numbering Adds a bookmark for the document
sub afp2web()	Main entry for AFP2web. The return code in this routine tells AFP2web to begin a new document with this page or not, to skip the current page, or to stop AFP2web processing. Adds a bookmark for the page.
sub initialize()	This routine is also used to set the flags for AutoSplit and PDFBookmark. We do not show this routine in this document.

The subroutine initializeDoc() is called when AFP2web signals the beginning of a new document. We use the Perl variable @_ to save the document, which is passed as argument . And we reset the page counter to zero. The following lines show how we set the bookmarks for the document:

Listing

addBookmarks.pm - initializeDoc()

```
#---- Add document level bookmarks ----#
if ($a2wDocumentPar->getId() == 1){
#---- Add creator
$a2wDocumentPar->addBookmark( "Creator", "Maas Hi gh Tech Software GmbH" );

#---- Add creation date & time
my $svLocal TimeTmp = local time;
$a2wDocumentPar->addBookmark( "Created On", $svLocal TimeTmp );

#---- Add spool filename
$a2wDocumentPar->addBookmark( "Spool ", $svSpool Fi l e n a m e );
return 0;
}
```

It is equally important that we save the current page object so that we can later access its elements. This is done in the subroutine `initializePage()`:

Listing

addBookmarks.pm - initializePage()

```
sub initializePage(){
#---- Get Parameter of initializePage( Par: a2w::Page )
($a2wPagePar) = @_;
```

The routine `afp2web()` is used to process each document page. This routine formulates the return code value to be passed to AFP2web. Bookmarks are added to each page.

Listing***addBookmarks.pm - afp2web() - Adding Bookmarks***

```

#-----
# Main entry method
# Return values:
# < 0: error
# 0: append page to Current Document
# 1: skip page
# 2: first page / new document
#-----
sub afp2web(){

if ( $bLog == $TRUE ){
print "afp2web(): PageId " . $a2wPagePar->getParseId() . "\n";
}

$APPEND= 0; # append page to Current Document
$SKIP= 1; # skip page
$NEWDOC= 2; # new document

#---- Set default return value
my $svRetTmp = $APPEND; # default: append page

#---- Add page level bookmarks ----#
if ($a2wDocumentPar->getId() == 1){

#---- Add page number (in spool)
$a2wPagePar->addBookmark( "Page", $a2wPagePar->getParseId() );
}
return $svRetTmp;

}

```

Starting the Script Example

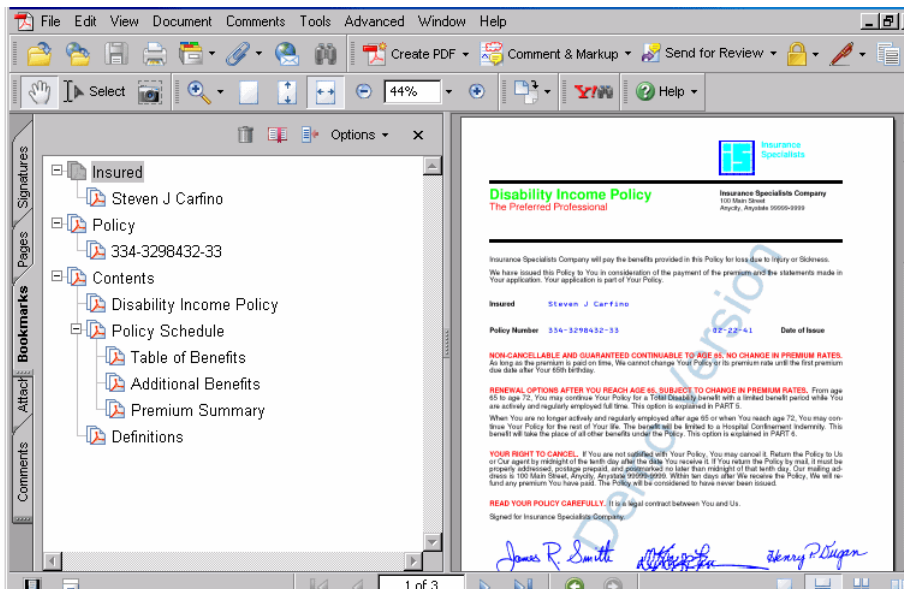
You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

Listing Demo Batch File

```
On Windows:  
afp2web.exe -q -c -doc_col d -sp: addBookmarks. pm samples\insure. afp  
  
On Unix:  
./afp2web -q -c -doc_col d -sp: addBookmarks. pm samples/i nsure. afp
```

Results of *addBookmarks.pm*

You will find the output PDF file in the output sub folder (the default will be in the sub folder /pdf). When opened in Acrobat Reader, you can find the bookmarks as follows:



addWatermark.pm: Adding Watermark Text to Pages

This example shows how to add text to a page as watermark.

Understanding addWatermark.pm

In this example, we focus on the subroutine:

<i>Subroutine</i>	<i>Description</i>
sub afp2web()	Main entry for AFP2web. We show how afp2web() adds text as watermark.

The following commands are used to add watermark text:

Listing *addWatermark.pm - sub afp2web()*

```
#---- Add Watermark ----#
#---- Create Helvetica 60 points Font (Watermark font)
$a2wFontTmp = new a2w::Font();
$a2wFontTmp->setName("Hel veti ca");
$a2wFontTmp->setHei ght(60);
$a2wFontTmp->setBol d($TRUE);

#---- Create Text Object ----#
my $a2wWatermarkTmp = new a2w::Text();
$a2wWatermarkTmp->setText("Confi dential ");
$a2wWatermarkTmp->setXPos(81); # X=81mm
$a2wWatermarkTmp->setYPos(204); # Y=204mm
$a2wWatermarkTmp->setAngle(60); # Rotation Angle = 60 degrees
$a2wWatermarkTmp->setColor(0xBDDBE7); # RRGGBB col or value (BD=>189,
DB=>219, E7=>231 is light blue)
$a2wWatermarkTmp->setFont($a2wFontTmp); # set font

#---- Add Text to Page
$a2wPagePar->addText( $a2wWatermarkTmp );
```

Starting the Script Example

You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

Listing
Demo.bat (lines 1 to 4)

```
On Windows:
afp2web.exe -q -c -doc_col d -sp: addWatermark. pm samples\i nsure. afp

On Unix:
./afp2web -q -c -doc_col d -sp: addWatermark. pm samples/i nsure. afp
```

Results of addWatermark.pm

This example produces PDF files with watermarks added to each page. Please note that the demo version of AFP2web adds its own "Demo Version" watermark to the result.

addAnnotations.pm: Adding a Hypertext Annotation

Understanding addAnnotations.pm

In this example, we focus on the subroutines

<i>Subroutine</i>	<i>Description</i>
sub initialize()	Used to set initial values for the hyperlink appearance.
sub afp2web()	Main entry for AFP2web. In our example, this routine will check for the first page of a document and will add a hyperlink annotation to the text found in predefined position.

The subroutine initialize() is set initial values for the hyperlink appearance:

Listing
addAnnotations.pm - initialize()

```
sub initialize(){  
  
    # Assuming width of a character as 2.53 millimeter  
    # It is used to calculate Hyper Link Rectangle Width as below  
    # Hyper Link Rect Width (in millimeters)=Hyper Link Text Length*$CharWidth  
    $CharWidth = 2.53;  
  
    # Hyper Link Rectangle Height values in millimeters ---  
    $HPLHeight= 2.6;  
  
    # Hyper Link Rectangle Border Width value in millimeters (approximately  
    1.0 Point)  
    #--- Set BorderWidth to zero if border is not needed.  
    $BorderWidth = 0.35;  
  
    #--- Hyper Link Action URL  
    $UrlText = "http://afp2web.com";  
  
    # Hyper Link Border color as RGB value (default color blue is assigned  
    here)  
  
    $HPLColor=0x000000FF; #---- Blue (0x00RRGGBB)  
    # $HPLColor=0x00FF0000; #---- Red  
    # $HPLColor=0x0000FF00; #---- Green  
    # $HPLColor=0x00FFFFFF; #---- White  
    # $HPLColor=0x00000000; #---- Black  
  
}
```

The routine afp2web() is used to process each document page. This routine check for the first page of a document and will add a hyperlink annotation to the text found in predefined position. In the following we only show the command that inserts the annotation.

Listing
addAnnotations.pm - afp2web() - Command used to insert the annotation

```
$a2wPagePar->addAnnotation( $CheckXPosTmp, #---- Upper left X position of  
annotation box (in AFP page units)  
$CheckYPosTmp, #---- Upper left Y position of annotation box (in AFP page  
units)  
$WidthTmp, #---- Width of annotation box (in AFP page units)  
$HeightTmp, #---- Height of annotation box (in AFP page units)  
$UrlText, #---- URL  
$BorderWidthTmp, #---- Annotation border width  
$HPLColor); #---- Annotation border color (Format=0x00BBGGRR)
```

At this point we do not want to repeat the information, which you can find in the script itself. Please refer to the in-line comments of the script. There, explanations have been added for each attribute.

Starting the Script Example

You use the following command in the demo batch file to start AFP2web with the Scripting Facility:

Listing Demo Batch File

```
On Windows:
afp2web.exe -q -c -doc_col d -sp: addAnnotations.pm samples\insure.afp

On Unix:
./afp2web -q -c -doc_col d -sp: addAnnotations.pm samples/insure.afp
```

Results of addAnnotations.pm

You will find the output file in the output sub folder (the default will be in the subfolder /pdf). The hypertext annotation will look like:

Insurance Specialists Company will pay the benefits provider

We have issued this Policy to You in consideration of the pe
Your application. Your application is part of Your Policy.

Insured

Beth N McShine



<http://afp2web.com>

Policy Number **332-5767288-77**

afp2xml.pm: Script Used to Produce XML Out of AFP

Producing XML from AFP requires two steps to solve the problem: The first step is to use the Scripting Facility to extract anything you want from the AFP spool file. The second step is to define your XML. Using a DTD, you define which XML elements and attributes are valid and using basic XML functionality of Perl, you produce your XML.

afp2xml.pm shows how to extract text and index data from AFP documents to create text files in XML format.

Understanding afp2xml.pm

This script requires that following packages be included:

<i>Listing</i> <i>afp2xml.pm - Packages to Include</i>

<pre>use a2w: : Conf i g ; use a2w: : Document ; use a2w: : Font ; use a2w: : I n d e x ; use a2w: : Kernel ; use a2w: : L i n e ; use a2w: : Medi u m M a p ; use a2w: : N O P ; use a2w: : O v e r l a y ; use a2w: : P a g e ; use a2w: : P S E G ; use a2w: : T e x t ; #---- Required Perl Modules for XML output use IO: : F i l e ; use XML: : W r i t e r ;</pre>
--

This script uses XML::Writer, a simple Perl module for writing XML documents, which you can obtain at the Perl Comprehensive Network (www.cpan.org). We place this module in a new subfolder of the Perl installation lib\XML\.

A typical sequence of methods used to create an XML document: `my $writer = new XML::Writer(); $writer->startTag('greeting', 'type' => 'simple'); $writer->characters("Hello, world!"); $writer->endTag('greeting'); $writer->end();`

Please refer to the documentation of the XML::Writer, which you can find on the official CPAN website. There, you will find more information on the following methods:

<i>Function:</i>	<i>Description</i>
<code>\$xmlDocWriter->xmlDecl</code>	Add an XML declaration to the beginning of an XML document. The version will always be "1.0".
<code>\$xmlDocWriter->doctype</code>	Add a DOCTYPE declaration to an XML document. The declaration must appear before the beginning of the root element.
<code>\$xmlDocWriter->comment</code>	Add a comment to an XML document.
<code>\$xmlDocWriter->startTag</code>	Add a start tag to an XML document. Any arguments after the element name are assumed to be name/value pairs for attributes: the module will escape all '&', '<', '>', and '"' characters in the attribute values using the predefined XML entities
<code>\$xmlDocWriter->endTag</code>	Add an end tag to an XML document. The end tag must match the closest open start tag, and there must be a matching and properly-nested end tag for every start tag
<code>\$xmlDocWriter->dataElement</code>	Print an entire element containing only character data.
<code>\$xmlDocWriter->characters</code>	Add character data to an XML document. All '<', '>', and '&' characters in the \$data argument will automatically be escaped using the predefined XML entities
<code>\$xmlDocWriter->end</code>	Finish creating an XML document. This method will check that the document has exactly one document element, and that all start tags are closed

The basic subroutines which must be located in the script are:

- `sub afp2web()` - the main entry point for AFP2web
- `sub initialize()`
- `sub initializeDoc()`
- `sub initializePage()`
- `sub finalizePage()`
- `sub finalizeDoc()`
- `sub finalize()`

In this example, we focus on the subroutines:

<i>Subroutine</i>	<i>Description</i>
sub initializeDoc()	Creates the start tag for the Document element
sub afp2web()	Main entry for AFP2web. It creates start tags for PageList and Page and produces XML output for elements on this page.
sub finalizePage()	Creates the end tag for the current element Page
sub finalizeDoc()	Creates end tags for the elements PageList and Document

The routine initializeDoc() defines the root element for the document:

Listing
afp2xml.pm - initializeDoc() - Defining the Root Element of the XML Document

```
#---- Generating FileHandle for XML Writer (needed since a2wDocumentPar-
>getSimpleFilename() not available from initializeDoc())
$xml DocIdFilename = $svOutputFilePath . $DocIdTmp . ".xml ";
$xml DocOutput = new IO::File( ">" . $xml DocIdFilename );

#---- Generating XML Writer Object
$xml DocWriter = new XML::Writer(OUTPUT=> $xml DocOutput,
  NAMESPACES=> 1,
);

#---- Processing Instruction
$xml DocWriter->xml Decl ("ISO-8859-1");
$xml DocWriter->comment("Generated by $svScriptProc " . $Version . ", " .
  local time());
$xml DocWriter->doctype(' Document', "", ' afp2xml.dtd ');

#---- Starting with root element
$xml DocWriter->startTag(' Document', 'Id' => $DocIdTmp, # Doc Id element
  ' Name' => $a2wDocumentPar->getName(),
  ' Spool ' => $svSpool Filename,
  ' PID' => $a2wDocumentPar->getPID(),
);
```

The routine afp2web() is called for each document page and creates start tags for PageList , Page and produces XML output for elements on this page.

Listing
afp2xml.pm - afp2web()

```

#---- Add Document Indexes to Index List
if ($PageId == 1){
  addIndexes( $a2wDocumentPar );
  $xml DocWriter->startTag(' PageLi st'); # <PageList>
}

#---- Starting with Element Page
$xml DocWriter->startTag(' Page', ' Id' => $PageId, # Page Id element
' Wi dth' => $a2wPagePar->getWi dth(),
' Hei ght' => $a2wPagePar->getHei ght(),
' Resol uti on' => $a2wPagePar->getResol uti on()
);

#---- Add Page Applied Medium Map Info
addPageMedi umMapI nfo();

#---- Add Page Applied Medium Overlay List
addPageMedi umOverl ayLi st();

#---- Add Page Included Overlay List
addI ncl udedOverl ayLi st( $a2wPagePar );

#---- Add Page Included PageSegment List
addI ncl udedPageSegmentLi st( $a2wPagePar );

#---- Add Page Indexes to IndexList of Page
addI ndexes( $a2wPagePar );

#---- Add NOPS to NOPList of Page
addPageNOPs();

#---- Add Text Objects to TextObje ctLi st of Page
addTextObje cts( $a2wPagePar );

#---- Add Line Objects to LineObje ctLi st of Page
addLi neObje cts( $a2wPagePar );

```

For how to use the XML::Writer to create additional XML elements, please refer to the subroutines in the script example.

Starting the Script Example

Note: To run this example, you need to install Perl. Please refer to the system prerequisites in the installation chapter of the AFP2web User Guide.

The following listing shows a simple method of adding directories to the Perl search path. Please note that these directories must be modified for your environment. Important: This statement must come as the very first statement in the main module loaded by AFP2web (this is afp2xml.pm):

Listing
Adding directories to the Perl search path in the internal Perl array @INC

```
BEGIN {  
  unshift(@INC, ('C:\Perl\lib', 'C:\Perl\site\lib', 'C:\temp'));  
}
```

The command to start AFP2web and to load the main module afp2xml.pm are:

Listing
Demo Batch File

```
On Windows:  
FOR /F %i IN ( 'perl -e "{$,=';'; print @INC;}"' ) DO @SET PERLLIB=%i  
afp2web.exe -q -c -doc_cold -sp:afp2xml.pm samples\insure.afp  
  
On Unix:  
export PERLLIB=`perl -e "{$,=';'; print @INC;}"`  
./afp2web -q -c -doc_cold -sp:afp2xml.pm samples/insure.afp
```

Results of afp2xml.pm

One PDF file is created per AFP document and one XML file with document information. The XML representation of the AFP document will come as a text file without any formatting for readability:

```
0 10 20 30 40 50 60 70
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!-- Generated by afp2xml.pm v1.0.0, Fri Aug 19 18:10:49 2005 -->
3 <!DOCTYPE Document SYSTEM "mhtDoc_V3.0.dtd">
4 <Document Id="1" Name="insure" Spool="samples/insure.afp" PID="s1qg"><Index:
. " YPos="6176" CodedFont="" CharacterSet="C1DOGT10" CodePage="TIDOBASE" Type:
. trikeover="no">If You are not satisfied with Your Policy, You may cancel it
. ents">Additional Benefits</Index><Index Name="Contents">Premium Summary</I
. h="-1" Weight="Bold" Slant="Medium" Underline="no" Strikeover="no">.....
. ace="GOTHIC" Height="100" Width="-1" Weight="Normal" Slant="Medium" Underl:
. TlGI0361" Typeface="HELVETICA LATIN1" Height="100" Width="-1" Weight="Bold'
. -1" Weight="Normal" Slant="Medium" Underline="no" Strikeover="no">--</Text(
. h="-1" Weight="Bold" Slant="Medium" Underline="no" Strikeover="no">Part 1 .
. Height="100" Width="-1" Weight="Normal" Slant="Medium" Underline="no" Stril
. 61" Typeface="TIMES NEW ROMAN LATIN1" Height="100" Width="-1" Weight="Bold'
. 100" Width="-1" Weight="Normal" Slant="Medium" Underline="no" Strikeover="
. >2.</TextObject><TextObject XPos="4152" YPos="11407" CodedFont="" Character:
5
```

If you want a formatted view, we recommend using an XML editor. For this purpose, we have placed DTDs for the document and index data in the sub folder /pdf:

XML

Comment

Generated by afp2xml.pm v1.0.0, Fri Aug 19 18:10:49 2005

DOCTYPE Document

Document

Id

1

Name

insure

Spool

samples/insure.afp

PID

s1qg

IndexList

Index (2)

	Name	Abc Text
1	Insured	Geoffrey R Stephens
2	Policy	324-1443255-11

PageList

Page (3)

	Id	Width	Height	Resolution	MediumMap
1	1	12240	15840	1440	MediumMap Na...
2	2	12240	15840	1440	MediumMap Na...
3	3	12240	15840	1440	MediumMap Na...

A.3 PDF/A Support in AFP2web

AFP2web's Commitment to the PDF/A Standard

AFP2web produces PDF output, which conforms to the PDF/A-1b standard. The standard is defined in ISO 19005-1. Document management — Electronic document file format for long-term preservation — Part 1: Use of PDF 1.4 (PDF/A-1) Reference number ISO 19005-1:2005(E)

PDF/A-compliant documents are self contained PDF files, which are used for long-term archival. The standard is based on PDF Version 1.4 and imposes restrictions on the PDF functionality.

Adobe Systems Limited (www.adobe.com), the originator of this technology, states, "[PDF/A-compliant files] are primarily used for archiving. Because long-term preservation is the goal, the document must contain only what is needed for opening and viewing throughout the intended life of the document. For example, PDF/A-compliant files can contain only text, raster images, and vector objects; they cannot contain encryption and scripts. In addition, all fonts must be embedded so the documents can be opened and viewed as created."

What You Can Find in This Appendix

This appendix is a supplement to the AFP2web Version 3.x User Guide and Reference. It describes how to use AFP2web Version 3.x to produce PDF/A-compliant output.

Important Notes

AFP2web does not validate the output for compliance to PDF/A. It remains the responsibility of the user to set the AFP2web parameters correctly and to avoid features, which are not allowed for PDF/A, such as font referencing or transparency.

INI Parameters and Command Line Options

To produce output for PDF/A-1b, the following parameters are relevant:

<i>INI Parameter</i>	<i>Command Line Option</i>	<i>Description</i>
OutputFormat=PDFA	-pdfa	Specifies the output format, which meets the PDF/A conformance level: PDF/A-1b.
Strict=on	-strict	When running AFP2web to produce PDF/A output, you must use the processing parameter Strict=on. Doing this will ensure that AFP2web will stop processing if any resources (fonts or AFP resources) are missing.
CMYKtoRGB=on		<p>Ensures that all colors are converted to use only one color space. AFP2web converts all colors to RGB.</p> <p>Note: AFP2web uses ICC sRGB IEC61966-2-1 color profile as OutputIntent.</p> <p>Important: Do not use the command line option -nocmyktorgb, because this will overwrite the INI parameter.</p>

Fonts

For PDF/A the fonts must be embedded in the PDF file.

AFP2web supplies a mapping table for font processing. In this file the section [CHARSET RENDERING] is used to define how to process an AFP character set. The processing mode can either use the original AFP font or a substitution font.

<i>Configuration Setting</i>	<i>Description</i>
mapping.def, section [CHARSET RENDERING], Rendering flag	<p>Please set the rendering flag to either:</p> <p>0 (this is the default for AFP fonts)</p> <p>2 to embed substitution font. For PDF/A, AFP2web supports only Type 1 fonts.</p> <p>Important: Do not use the rendering flag 1 to reference fonts, because this setting is not allowed for PDF/A.</p>

Metadata in XMP Format

AFP2web generates metadata in XMP format automatically, embeds the metadata in PDF as a stream object, and references it using the "Metadata" entry in the "Catalog" dictionary.

AFP2web maps metadata as shown in the table below:

<i>INI File Parameter</i>	<i>XMP metadata</i>		<i>Values</i>
	<i>Field</i>	<i>Type</i>	
Title	dc:title	Text	Title as specified in the INI parameter
Subject	dc:subject	Text	Subject as specified in the INI parameter
Keywords	pdf:keywords	Text	Keywords as specified in the INI parameter
-----	dc:creator	Text	AFP2web Version ?? [Built for <OS> on <timestamp>]
-----	xmp:CreatorTool	Text	AFP2web Version ?? [Built for <OS> on <timestamp>]
-----	pdf:Producer	Text	Maas PDF Library Vx.x
-----	xmp:CreateDate	Date	Date and Time of PDF creation

Note: AFP2web also generates metadata for outline fonts.

Features Not to Be Used for PDF/A

The following table lists the features, which are not compliant to PDF/A, and suggests how to avoid them when using AFP2web to create PDF/A-compliant documents:

<i>Limitation</i>	<i>Solution</i>
No hyperlinks to external files.	Hyperlink annotations are not possible. Do not use the Scripting Facility "addAnnotation" interface of "a2w::Page".
Scripting Facility, a2w::Page->addText	Do not use the Scripting Facility "addText" interface of "a2w::Page". It does not embed the font.
Encryption	Encryption is not allowed for PDF/A. Therefore do not set PDF Security options (this applies to the INI parameter PDFSecurity or the command line option -ps).
Transparency	Do not use transparency. When producing output for PDF/A, AFP2web will remove the transparency.
Watermark	Do not use AFP2web to insert text as watermark. Do not use the INI parameter Watermark nor the command line option -wm.

A.4 Frequently Asked Questions (FAQs)

E088: Scripting Facility Error (rc=-998): Unable to parse the file

Problem:

AFP2web fails with the following error message:

```
Can't locate a2w/MyModule.pm in @INC (@INC contains: listofPerl-  
paths.) at NOPs.pm line 'n'.
```

```
E088: Scripting Facility Error (rc=-998): Unable to parse the file
```

Solution:

Check your settings for Perl. See the Scripting Facility user guide in this manual.

Index

Symbols

·? 149

A

a2w Config 214
 a2w Document 217
 a2w Font 225
 a2w Index 236
 a2w NOP 254
 a2w Overlay 257
 a2w Page 266
 a2w Text 287
 addAnnotations.pm 376
 addBookmarks.pm 370
 addWatermark.pm 374
 AFP
 curve smoothing 147
 AFP index 158
 AFP resource path 135
 AFP resources 46, 156
 AFP2web
 adding watermark images to pages 60
 application scenarios 29
 configuration overview 41
 core components 27
 creating bookmarks 59
 function overview 25
 important notices 19
 input formats supported 26
 installation 35
 introduction to font handling 65, 73
 overview of processing parameters 49
 product architecture 28
 quick start 47
 SDK option 28
 selecting documents for conversion 50
 starting AFP2web from the command line ..
 48
 stopping AFP2web via CLI 48
 AFP2web new features 13
 afp2web() 96
 afp2web.ini
 function and structure 119
 section AFPCOLORTable 140
 section FORMDEF 139
 section settings
 AFPComplianceLevel 120
 AFPFontPath 120
 Author 122

Autosplit 120
 CharSetExt 120
 CMYKTORGB 121
 CodedFontExt 121
 CodePage 121
 CodePageDefaultChar 121
 CodePageExt 121
 Color 122
 CpPath 122
 CurveSamplingFactor 122
 DocumentCount 122
 EndingDocument 123
 EndingPage 123
 ExceptionLoggingLevel 123
 ExtFontPath 123
 FileCreationMode 123
 FilenamePattern 124
 FormatType 125
 FormDefExt 125
 FormDefPath 125
 GenerateUniqueFile 125
 IndexFormat 126
 IndexPath 126
 IndexRecord 126
 InputFormat 126
 JPEGQuality 127
 Keywords 127
 LaunchPreview 127
 Licensee 127
 Logging 128
 LoggingFont 128
 LoggingLevel 128
 LogPath 129
 MaxCols 129
 MemoryOutputStream 130
 OutputFilePath 130
 OutputFormat 130
 OutputSize 130
 OverlayExt 130
 OverlayPath 131
 PageOutput 131
 PageRotation 131
 PageSegExt 131
 PageSegmentPath 131
 PDFBookmark 132
 PDFDocLimits 132
 PDFSecurity 132
 PDFUIOptions 133
 PDFWinOptions 134
 PrintableLineWidth 135
 Protocol 135
 Quietmode 135

- Resolution 135
 - ResPath 135
 - SaveOCDataOnDisk 136
 - ScriptArgument 136
 - ScriptProcedure 136
 - ScriptUnitBase 136
 - SerialNr 136
 - SkipObjectSize 137
 - SkipPage 137
 - StartingDocument 137
 - StartingPage 137
 - Statistic 137
 - Strict 137
 - Subject 138
 - TempPath 138
 - Title 138
 - Watermark 138
 - sections for user-defined FTP locations
 - HostName 141
 - Password 141
 - StartDir 141
 - Type 141
 - UserName 141
 - afp2web.pm 208
 - afp2xml.pm 379
 - AFPComplianceLevel 120
 - AFPFonPath 120
 - ALL 123
 - Author property for PDF 122
 - Autosplit 120
 - autosplit.pm 329
- B**
- Barcode support 57
 - bookmarks for PDF 59
 - bookmarks in PDF 132, 145
- C**
- character sets
 - mapping code pages 175
 - CharSetExt 120
 - clr 146
 - CMYKTORGB 121
 - code page default 121, 147
 - code page default character 121
 - code page files 175
 - code page mapping 175
 - code pages location 146
 - CodedFontExt 121
 - CodePage 121
 - CodePageDefaultChar 121
 - CodePageExt 121
 - color
 - CMYK to RGB for IOCA 121, 140, 146, 151
 - processing flag 122, 145
 - command line options
 - ? 149
 - ap 145
 - bm 145
 - c 145
 - clr 146
 - compression for TIFF 145
 - cp 146
 - csf 147
 - dc 147
 - dcp 147
 - ed 147
 - ell 147
 - fd 148
 - fdp 148
 - file creation modes 143
 - fnpt 148
 - fp 149
 - h 149
 - iASCII 149
 - if 149
 - input format 144
 - jq 150
 - l 150
 - lf 150
 - ll 150
 - lp 151
 - mc 151
 - nocmyklogb 151
 - op 152
 - os 152
 - output format 144
 - overview 111, 143
 - ovp 152
 - p 152
 - plw 152
 - po 152
 - pp 153
 - pr 153
 - ps 153
 - psp 154
 - pui 154
 - pv 154
 - pwn 154
 - q 155
 - r 155
 - rf 156
 - rp 156
 - s 156
 - sa 156
 - sd 156
 - si 156
 - so 156
 - sod 156
 - sp 157
 - std 157
 - Strict 157
 - tp 157
 - u 157

-v	157
-vall	157
-wm	158
-xf	158
-xp	158
command line options for fonts	68
compression for JPEG	127
compression for TIFF	145
console	
displaying optimal values for ASCII format-	
ting	149
quick help	149
suppressing messages to	135, 155
conversion PDF to AFP	13
-cp	146
CP files used for mapping code pages	175
CpPath	122
-csf	147
CurveSamplingFactor	122

D

-dc	147
-dcp	147
deprecated features	17
DOC	123
DOC_COLD	123
DOC_INDEX	123
DocumentCount	122
dumpIniParms.pm	325
dumpMediumMaps.pm	352
dumpNOPs.pm	356
dumpPageTextObjs.pm	339

E

-ed	147
-ell	147
EndingDocument	123
EndingPage	123
error handling	
enforcing rules	61, 137, 157
error messages	
AFP2web Error Messages	296
MHTIMG Library messages	308
ExceptionLoggingLevel	123
ExtFontPath	123
eyecatcher.pm	360

F

-fd	148
-fdp	148
feature overview	13
file extensions	
overview	45

file locations	
AFP font resources	145
AFP formdefs	148
AFP index	158
AFP overlay resources	131, 152
AFP page segments	131, 154
AFP resources	135, 156
code pages	122, 146
fonts for PDF	123, 149
Formdefs	125
index data output	158
index output	126
output files	130, 152
temporary work folder	138, 157
the INI file to use	149
the path to afp2web.ini	149
file name extensions	
AFP character sets	120
AFP code pages	121
AFP coded fonts	121
AFP formdefs	125
AFP overlay resources	130
AFP page segments	131
file name pattern	124
FileCreationMode	123
FilenamePattern	124
finalize()	97
finalizeDoc()	97
finalizePage()	97
-fnpt	148
font metric files	17
fonts	13, 68
AFP font naming convention	186
configuring AFP2web for fonts	44
introduction to the mapping.def	69
logging flags	128, 150
overview of AFP2web font handling ...	65,
73	
path to AFP font resources	145
path to substitution fonts	123
temporary work folder	138
formats	
compression for JPEG	127
improvements for TIFF	14
index	126
input formats	126, 144
input formats supported	13
output formats	130, 144
overview of formats supported	26
PDF	14
PDF/A	14
resolution for bitmap formats	135
TIFF compression	145
TIFF output	125
FormatType	125
formdef path	148
formdef used as standard	148
FormdefExt	125
FormDefPath	125
Formdefs	139

-fp 149
 FTP locations 141

G

GenerateUniqueFile 125

H

-h 149

I

-iASCII 149
 -if 149
 images
 optimizing 56
 processing Included Objects 55
 index
 AFP2web default processing 52
 custom index formats as input 99
 output format for CSV 126
 output path 126
 processing capabilities of the Scripting
 Facility 99
 processing flag 123
 processing options 52
 standard AFP index 99
 standard format 126
 IndexFormat 126
 IndexPath 126
 IndexRecord 126
 INI file 42
 parameter overview 111
 INI file parameters for fonts 68
 INI file specified via command line 149
 INI-file and command options 68
 initialize() 96
 initializeDoc() 96
 initializePage() 96
 input
 AFP index 158
 formats 126, 144
 input and output via stdin and stdout 157
 input from stdin 156
 InputFormat 126
 IOCA processing 55, 56

J

JPEGQuality 127, 150
 -jq 150

K

Keywords 127

L

-l 150
 LaunchPreview 127
 -lf 150
 Licensee 127
 line width in AFP 152
 -ll 150
 Logging 128
 logging
 flags 123, 128
 font information 177
 introduction to using log files 62
 processing statistics 156
 writing a processing protocol ... 135, 152
 logging flags 147, 150, 152
 LoggingFont 128
 LoggingLevel 128
 LogPath 129, 151
 -lp 151

M

Maas font metric files 17
 mapping.def
 overview 159
 role of the 69
 section AFPFONT 171
 section CHARSET 165
 section CHARSET RENDERING 161
 section CODEPG 173
 section FGID 166
 section FONTSUFFIX 167
 section PDFFONT 168
 section UNIXFONT 170
 section WINFONT 169
 section XFONT 164
 MaxCols 129
 -mc 151
 memory allocation for PDF processing 132
 messages
 AFP2web Error Messages 296
 ExceptionLoggingLevel 123
 logging flags 128
 logging output 129
 MHTIMG Library messages 308
 processing statistics 137
 suppressing 135
 Migrating to AFP2web Version 3.x. 21
 migration
 Scripting Facility modules to Version 3.x. ... 315

N

a2w Kernel	242
a2w Line	244
a2w MediumMap	251
a2w PSEG	264
Scripting Facility API	
a2w Kernel	242
a2w Line	244
a2w MediumMap	251
a2w PSEG	264
-nocmyktgb	151

O

one output file per document page for TIFF	152
-op	152
optimization flag	136, 156
-os	152
output	
ASCII page limits	151
bookmarks in PDF	132
compression for JPEG	127
displaying optimal values for ASCII format- ting	149
file name pattern	124
folders for output documents	54
formats	123, 130
for TIFF	125
logging	
flags	150
logging flags	147, 150
path for log files	129, 151
path for output files	130
processing statistics	137
standard index format	126
suppressing messages to console	135
target location for index data	158
output and input via stdin and stdout	157
output filename pattern	148
output in subfolders	122, 147
output to stdout	156
OutputFilePath	130
OutputFormat	130
OutputSize	130
OverlayExt	130
OverlayPath	131
Overview	
Conventions used in this document	207
overview	46, 94
-ovp	152

P

-p	152
page orientation	131

page rotation	155
PageOutput	131
PageRotation	131
PageSegExt	131
PageSegmentPath	131
paths	
AFP font resources	145
AFP formdefs	148
AFP index	158
AFP overlay resources	131, 152
AFP page segments	131, 154
AFP resources	135, 156
code pages	122, 146
fonts for PDF	123, 149
Formdefs	125
index data output	158
index output	126
output files	130, 152
temporary work folder	138, 157
the INI file to use	149
the path to afp2web.ini	149
paths to AFP resources	44
PDF processing memory	132
PDF properties	
Author	122
Keywords	127
Licensee	127
PDFSecurity	132
Subject	138
Title	138
PDF Security options	153
PDF viewer settings	133, 134, 154
PDF/A Support	385
PDFBookmark	132
PDFDocLimits	132
PDFSecurity	132
PDFUIOptions	133
PDFWinOptions	134
-plw	152
-po	152
-pp	153
-pr	153
PrintableLineWidth	135
Protocol	135
-ps	153
-psp	154
-put	154
-pv	154
-pwn	154

Q

-q	155
Quietmode	135

R

-r	155
Resolution	135, 153
ResPath	135
-rf	156
role of the	42
rotation	155
-rp	156

S

-s	156
-sa	156
SaveOCCDataOnDisk	136
ScriptArgument	136
Scripting Facility	
application scenarios	89
arguments	136, 156
default encoding	98
index processing overview	99
interaction with AFP2web	93
migrating to Version 3.x.	315
output to memory	108
overview of improvements	15
overview of methods	199
overview of routines	192
page coordinates	98
Perl settings	106
processing flag	123
script to use	136, 157
splitting spool file into documents and pages	100
started via command line option	143
unit of measurement for page parsing	136
using the	103
variable scopes	98
Scripting Facility API	
a2w Config	214
a2w Document	217
a2w Font	225
a2w Index	236
a2w NOP	254
a2w Overlay	257
a2w Page	266
a2w Text	287
afp2web.pm	208
Overview	
Conventions used in this document	207
Scripting Facility examples	
addAnnotations.pm	376
addBookmarks.pm	370
addWatermark.pm	374
afp2xml.pm	379
autosplit.pm	329
dumpIniParams.pm	325
dumpMediumMaps.pm	352

dumpNOPs.pm	356
dumpPageTextObjs.pm	339
eyecatcher.pm	360
sortPageTextObjs.pm	346
Tutorial	319
Scripting Facility interface	94
afp2web()	96
finalize()	97
finalizeDoc()	97
finalizePage()	97
initialize()	96
initializeDoc()	96
initializePage()	96
Scripting Facility parsing events	95
ScriptProcedure	136
ScriptUnitBase	136
-sd	156
SerialNr	136
-si	156
single page output for TIFF	
TIFF	
single page output	131
SkipObjectSize	137
SkipPage	137
-so	156
-sod	156
sortPageTextObjs.pm	346
-sp	157
specifying Formdefs	46
spool file	
first document to process	137, 156
first page to process	137
flag to skip empty pages	137
last document	123, 147
last page	123
page range to process	153
size of empty pages	137
splitting into documents and pages with the	
Scripting Facility	100
StartingDocument	137
StartingPage	137
Statistic	137, 156
-std	157
-Strict	157
Strict	137
Subject	138

T

temporary work folder	138, 157
TempPath	138
thumbnail images	130, 152
TIFF compression	145
Title	138
-tp	157
Tutorial	319

U

-u 157

V

-v 157

-vall 157

version number of AFP2web 157

version number of all AFP2web components ...
157

viewer for conversion results 127, 154

W

Watermark 138, 158

watermark images 60

-wm 158

X

-xf 158

-xp 158

