MakeAFP

MakeAFP Formatter User's Guide

Version 4.0

This edition applies to the MakeAFP Formatter.

MakeAFP welcomes your comments and suggestions. You can send your comments and suggestions to:

support@makeafp.com

When you send information to MakeAFP, you grant MakeAFP a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Overview

MakeAFP Formatter is a powerful, yet simple-to-use Dynamic-Link Library and flexible AFP formatter for generating AFP files directly at extremely high speed. A simple C/C++ language program can be linked with the library to produce a native executable module that can generate AFP files directly without the help of any other application.

Contrast MakeAFP solutions with the typical methods of creating AFP files, which generally involves multi-stages of "work-flow" processes that require the development of applications to pre-process the raw data into the special text data format required by the formatting software's templates first. Then the text data is formatted into AFP by the formatting software. Such multi-stages involve high cost with long training cycles and will be more painful in the transfer of skills, and you are also suffering and bounded by the limited logical processing and data manipulation capabilities provided by their GUI (Graphical User Interface) templates. Most GUI formatters use their formats with images, overlays, and fonts, that need to be converted into AFP resource objects during formatting. Also, it is tedious, time-consuming, computing resource-intensive, and has a low formatting speed. With MakeAFP Formatter, you can take advantage of the flexibility and power of the C or C++ functions for high-speed data formatting, and you are fully capable of processing any type of raw data from multiple inputs, with a very short learning and skill transferring cycle.

MakeAFP Formatter is designed for complex, dynamic AFP presentation with an extremely fast formatting performance to your mission-critical AFP core business applications, providing an integrated AFP formatting and processing environment that is stable, highly scalable, and functionality-rich.

MakeAFP Formatter frees you from the internals and intricacies of AFP, offering many useful functions for composing texts, graphics, images, barcodes, indexes, and AFP resources in AFP, giving you absolute control over the AFP functions.

Using C language as your core programming language on MakeAFP Formatter is strongly recommended. C is the most important and popular programming language, it is a very efficient and fully compile-able language that works at a level that is fairly close to native machine code, yet it is simple, portable, powerful, flexible, and programmer-oriented language. C perfectly fits the procedure processing tasks with data formatting.

Comprehensive MakeAFP utilities are also provided as the optional AFP Toolkits and some enhancements to Infoprint Manager for Windows. With the combination of your native executable modules generated by MakeAFP Formatting functions and

AutoMakeAFP automation utility which runs as a multithreaded Windows service, you can achieve a maximized AFP production environment that is fully integrated and automated with the AFP document creations and production processes.

Understanding AFP

Advanced Function Presentation (AFP) founded by IBM is an integrated hardware and software architecture for generating high speed and high-quality presentation outputs. AFP is an industry de-facto open standard fully published by IBM for high-speed printing and presentation.

AFP is implemented with IBM MO:DCA (Mixed Object Document Content Architecture) data stream consisting of structure fields of presentation information. The MO:DCA architecture defines the data stream used by applications to describe documents and object envelopes for interchange with other applications and application services. Documents defined in the MO:DCA format may be archived in a content management database, then later retrieved, viewed, annotated, and printed in local or distributed systems' environments. Presentation fidelity is accommodated by including resource objects in the documents that reference them.

AFP documents can be made up of different kinds of data, such as text, graphics, images, and barcodes. AFP object content architectures describe the structure and content of each type of data format that can exist in a document or appear in a data stream. AFP objects can be either data objects or resource objects. An AFP data object contains a single type of presentation data, that is, texts, vector graphics, raster image, or barcodes, and all of the controls required to present the data. An AFP resource object is a collection of presentation instructions and data. These AFP objects are referenced by name in the presentation data stream and can be stored in system libraries so that multiple applications and print servers can use them. All AFP object content architectures (OCAs) are totally self-identifying and independently defined. When multiple AFP objects are composed on a page, they exist as peer objects, which can be individually positioned and manipulated to meet the needs of the presentation application.

AFP Document

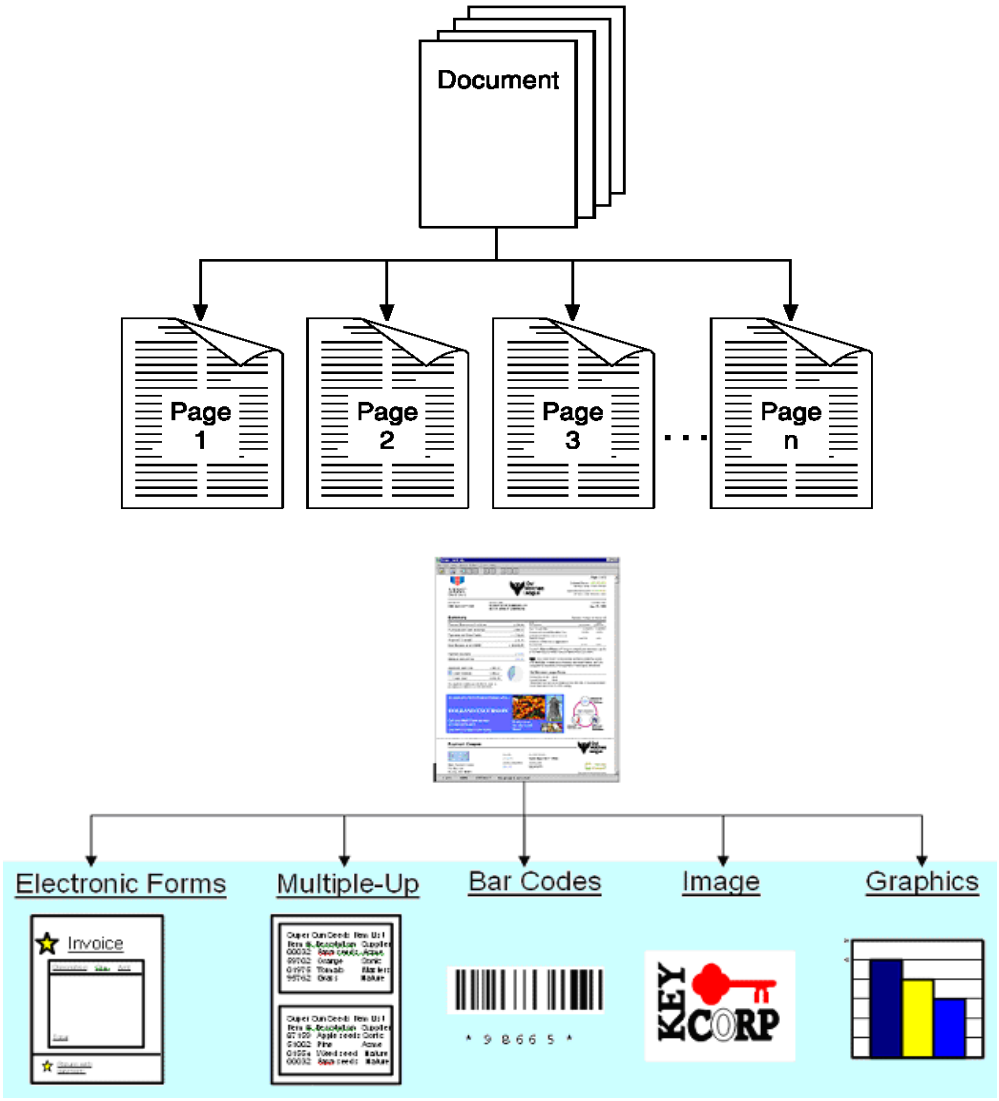
An AFP document consists of one or more AFP pages, as shown in the figure on the next page, each page contains the presentation of the text for that page, as well as the information about the placement of each text. In addition to text, the following types of data objects can also be included on a page:

- Graphic (line art, business chart, and graphic)
- Image (logo, scanned document image, signature)
- Lines, box, shading and color area
- Barcode
- Indexes for archiving and retrieving

AFP Data and Resource Objects

The AFP objects supported by MakeAFP are:

- Graphic objects in IBM Graphic Object Content Architecture (GOCA)
- Image objects in IBM Image Object Content Architecture (IOCA)
- Text Object in IBM Presentation Text Object Content Architecture (PTOCA)
- Barcode Object in IBM Barcode Object Content Architecture (BCOCA)
- Font Object in IBM Font Object Content Architecture (FOCA)



AFP and Non-AFP Resource Types

There are five types of AFP resources:

- Fonts
- Page Segment

- Overlays
- Form Definition
- Page Definition

AFP Fonts

A font is a collection of graphic characters of a given size and style which is used to present text. An AFP font can be ASCII, EBCDIC, DBCS-PC (GBK, BIG5, SJIS, KSC), DBCS-HOST, and UNICODE encoded. An AFP font is composed of 3 font-member types:

- A coded font member associates a code page and a font character set as a pair. A single-byte coded font contains 1 code page and font character-set pair. A double-byte raster coded font, which requires 2 bytes to identify each graphic character, contains 2 or more code page and font character-set pairs; each pair is called a font section. A double-byte outline font contains 1 code page and font-character-set pair.
- A code page member associates a code point and a graphic character identifier for each graphic character supported by the code page and specifies how code points that are not valid are to be processed.
- A font character set member contains a graphic character identifier and a raster pattern or outline vector information for each graphic character in the font or font section, as well as information about how the characters are to be printed.

You can use the following products to create AFP fonts:

- **MakeAFP Fonter**, a powerful and advanced font converter and editor for AFP.
- ISIS Papyrus Font Converter
- Elixir DesignPro Tools for AFP

AFP Page Segment

A page segment usually contains image data such as signatures, logos, or graphics in bi-level (monochrome), grayscale, and color AFP image format.

Page segments can also contain a mixture of text, image objects, and graphics data objects, and can be placed anywhere on a presentation page. Programs can request page segments for presentation on a page or overlay.

You can use the following products to create an AFP page segment:

- **MakeAFP Imager,** transforms a popular format image to AFP monochrome or color image in AFP IOCA FS11, and FS45 formats.
- Ricoh AFP Printer Driver for Windows (freeware contributed by IBM)
- InfoPrint Manager transforms – PS2AFP, PDF2AFP, TIFF2AFP, GIF2AFP, JPEG2AFP, PCL2AFP
- ISIS ImageEditor for AFP

– Elixir DesignPro Tools for AFP

AFP Overlay

The overlay is used as the electronic form, an overlay can contain the following elements:

- Vertical, horizontal, and diagonal lines with different widths and thicknesses
- Vector Color graphics and bitmap images
- Linear or 2D Barcodes
- Text in various positions with different font styles and orientation
- Different inline directions and page rotations
- Boxes with and without shading or color background, border thicknesses, and border types
- Circles and arcs with different shades and patterns, colors, border thicknesses, and border types
- Arbitrary shapes, called paths, with different shades and patterns, border thicknesses, and border types

You can use the following products to create AFP overlay:

– **MakeAFP Form Designer**, a powerful and advanced form design tool
– Ricoh AFP Printer Driver for Windows (freeware contributed by IBM)
– InfoPrint Manager transforms – PS2AFP, PDF2AFP, TIFF2AFP, GIF2AFP, JPEG2AFP, PCL2AFP
– Canon PRISMAtools - Document Designer Standard Suite
– ISIS AFP-Designer
– Elixir DesignPro Tools for AFP
– IBM legacy Overlay Generation Language (OGL)

AFP Form Definition

A form definition is a resource that specifies the physical attributes of the print output. You must specify a form definition for each job you want to print, either by specifying a form definition by name while you submit a print job or by using the default form definition setting defined by your AFP print server installation.

Each form definition can contain several subsets of page controls, called *copy groups (also called medium maps),* which can be changed between pages of a document to dynamically select the form-mapping controls for subsequent pages.

A form definition contains the following printing control information:

- Page origin, which is the top-left boundary for printing. When the printer is duplexing, page origin may be different for the front and back of the page
- Position of a logical page on a physical page
- Sheets on which medium overlays are to be printed
- Inclusion of overlays, which substitute for pre-printed forms
- Number of copies of each page to be printed
- Input and output paper tray

- Jog control (the offset stacking of cut-sheet output or copy marking on continuous-forms output)
- Simplex or duplex printing
- Constant control (allows front or back printing of a page without variable data)
- Data fields that are to be suppressed, that is, not printed
- Printed copy groups to be stacked offset from each other
- Page presentation in either portrait or landscape presentation
- N_UP Printing: Printing one, two, three, or four logical pages on a single side of a page
- Post-processing controls, such as finisher control of the cut-sheet printer, cutting, perforating

You can use the following products to create a form definition:

- IBM or Ricoh Page Printer Formatting Aid (PPFA)
- Canon PRISMAtools - Document Designer Standard Suite
- ISIS AFP-Designer
- Elixir DesignPro Tools for AFP

MakeAFP supplies general-purpose multi-functions form definitions within the standard package and also provides the creation of form definition as an optional service.

AFP Page Definition

Page definition specifies how to format simple line data into AFP by the following functions:

- Dimensions of the logical page
- The print direction of the logical page
- The print direction of text lines and fields relative to the logical page
- Conditional processing (different formats on different pages, based on the content of data)
- Text line spacing (number of lines per inch)
- Location of individual text lines and fields
- Number of text lines per page
- Page segments for inclusion in the printed output
- Overlays for inclusion in printed output (positioned anywhere on the page)
- Page-ejection points
- Fonts and font rotation used on a page
- Multiple-up printing (placing more than one sub-page on one side of a single sheet)
- Colors to be used for text, circle, box, and color area

You can use the following products to create page definition:

- IBM Page Printer Formatting Aid (PPFA)
- Canon PRISMAtools - Document Designer Standard Suite
- ISIS AFP-Designer
- Elixir DesignPro Tools for AFP

Page Definition is designed as a simple and easy means for formatting simple line data into simple output layouts. You have to use MakeAFP Formatter or other document formatters for AFP if you want to get dynamic documents to be formatted from your complex input raw data. With MakeAFP Formatter, page definition is not required since you can generate a fully composed AFP document file directly rather than a line data file.

Non-AFP Object

Non-AFP objects are defined in AFP as the data-object, the OpenType/TrueType fonts, and some popular image formats are supported in the new generation of the AFP system directly.

MakeAFP Formatter supports TrueType/TrueType Collection/ OpenType fonts and popular image formats of TIFF, JPEG, GIF in AFP.

Resources Access

AFP and non-AFP resources are necessary objects to the presentation of AFP, including fonts, images, form definitions, and overlays. The resources can be accessed separately from the resources' libraries/directories, or all of the resources necessary to represent an AFP document can be bundled in a single resource file, or be embedded inline as a resource group within the AFP file (resources must be in front of the AFP document) for transport.

The inline resources are treated as private resources in the AFP systems. The scope of an inline resource group is the AFP print file. Once the document in the AFP print file has been processed, the resources in the resource group are no longer available to the presentation system for use with another AFP print file. For performance reasons, picking up resources from the resources' libraries/directories directly is strongly recommended with AFP production printing, so that the resources can be retained in the IPDS printer's memory across the job boundaries.

Chapter 2. Installing MakeAFP Formatter for Windows

This chapter provides information about the Windows prerequisites and installation of the MakeAFP Formatter for Windows.

MakeAFP Formatter for Windows Prerequisites

Here are the prerequisites to run MakeAFP Formatter for Windows:

1. Windows 7 or above, 64-bit.
2. Windows Server 2010 or above, 64-bit.
3. Microsoft Visual C++ 2010 Service Pack 1 Redistributable Package, 64-bit.

Installing MakeAFP Formatter on Windows

To install MakeAFP Formatter on a Windows server or workstation:

1. Log on to the Windows system as an administrator.
2. Run the MakeAFP Formatter setup package you received.
3. Follow the instructions on the installation screens to install the package, the destination folder is **c:\MakeAFP**.

Applying License of MakeAFP Formatter for Windows

Installing Soft-license Key

1. Click the **Start** button, then select **Programs**, **MakeAFP Software**, **MakeAFP Formatter,** and **License Manager.**
2. On MakeAFP Formatter License Manager, copy the Serial Number and then paste it into an email to be sent to MakeAFP Support at support@makeafp.com to request a software license key for your system.
3. Once a MakeAFP Formatter License key is received, you need to run MakeAFP Formatter License Manager to apply the license key.

MakeAFP Formatter Demo License for Windows

Without the License key for MakeAFP Formatter, you can run MakeAFP Formatter in the demo mode, which allows you to generate up to 2000 AFP pages during developments. AFP files are not permitted to use for production printing.

Chapter 3. Installing MakeAFP Formatter for Linux

This chapter provides installation information of the MakeAFP Formatter for Linux.

Installing MakeAFP Formatteron Linux

To install MakeAFP Formatter on a Linux server:

If install on the MakeAFP Formatter default installation path **/usr/share/makeafp**, type the following command:

```
sudo sh makeafp_install.bin
```

If install on a user defines a specific path, type the following command:

```
sudo sh makeafp_install.bin your_makeafp_path_name
```

Applying License of MakeAFP Formatter for Linux

Applying license Key:

1. Type command:

   ```
   makeafplic
   ```

   Copy the Serial Number generated and then paste it into an email to be sent to MakeAFP Support at support@makeafp.com to request a software license key for your system.

2. Once your MakeAFP Formatter License key file **makeafp.lic** was received, copy it into your MakeAFP Formatter installation path, default path is:

   ```
   /usr/share/makeafp
   ```

Uninstalling MakeAFP Formatter on Linux

To uninstall MakeAFP Formatter on a Linux server:

If installed MakeAFP Formatter on the default installation path `/usr/share/MakeAFP Formatter`, type the following command:

```
sudo sh makeafp_install.bin --uninstall
```

If MakeAFP Formatter was installed on a user-defined specific path, type the following command:

```
sudo sh makeafp_install.bin your_makeafp_path_name –uninstall
```

MakeAFP Formatter Demo License for Linux

Without the License key for MakeAFP Formatter, you can run MakeAFP Formatter in the demo mode, which allows you to generate up to 2000 AFP pages during developments. AFP files are not permitted to use for production printing.

Chapter 4. Getting Started

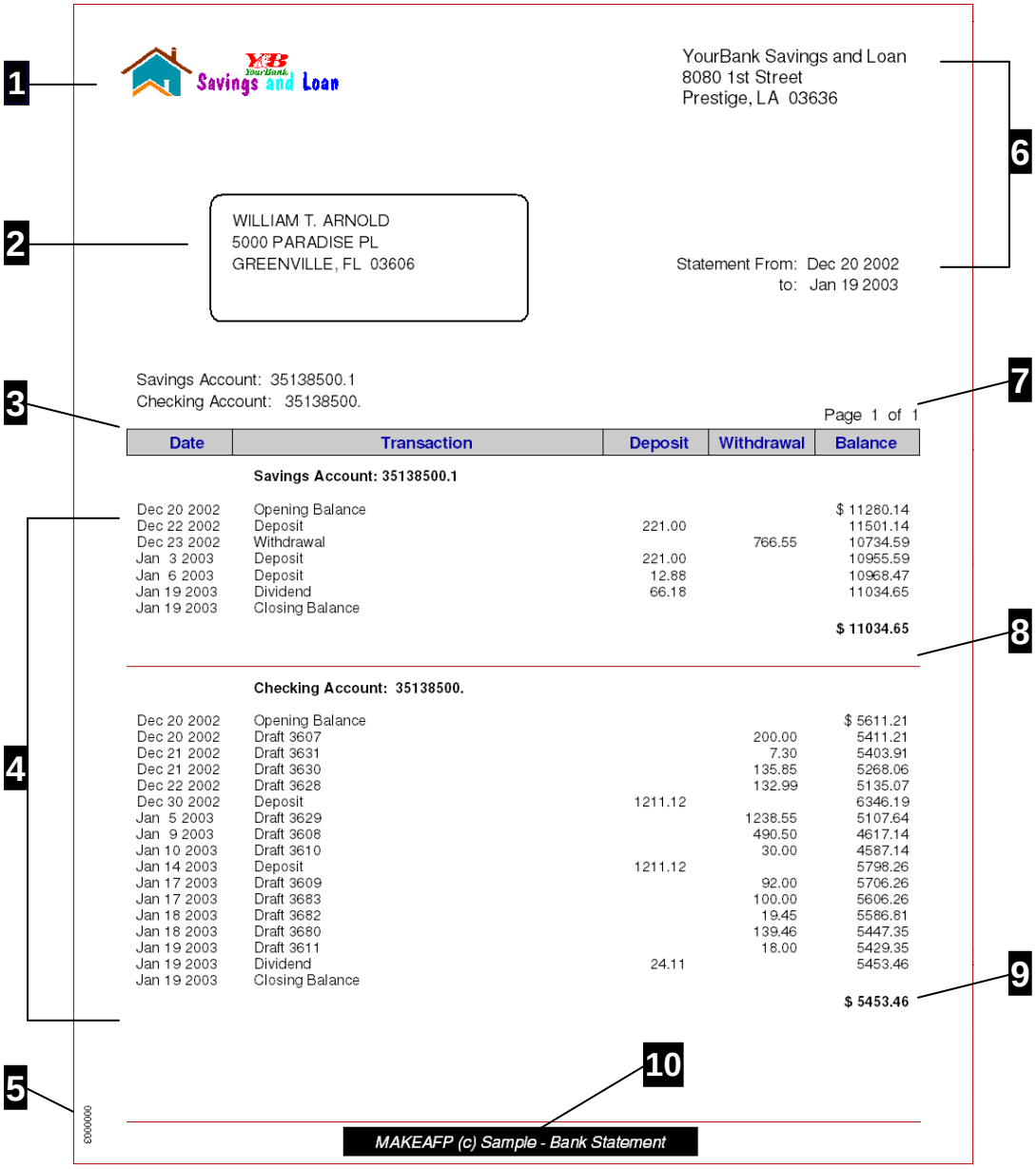
Applications that use MakeAFP Formatter functions must obey the hierarchy and structural rules of the MakeAFP Formatter language when building AFP compound documents that are very easy to understand. Writing applications according to these rules are quite straightforward.

This chapter describes the step-by-step processes on how to use MakeAFP Formatter with C to generate the AFP document.

Comprehensive samples in C/C++, C#, and Java are shipped along with the MakeAFP Formatter installation package, to provide you with realistic practice scenarios. MakeAFP understands that the sample code is one of the most important aspects of a toolkit for quick understanding and learning, and as such, we will try our best to include more samples within our installation package.

Understanding a Bank Statement

The following figure shows an example of a bank statement, which contains the client's address, banking transaction records during the statement period for the accounts owned by the client, boxes, lines, logo, and the page number.



1. The bank logo is included as an AFP page segment.
2. Client's address printed in a rounded box address window
3. The gray color box contains title texts
4. Detail transaction information sorted by date, text aligned by left or right
5. Running numbers used for a printing operation and reprint

6 Bank address and statement period

7 Pagination numbers on each page

8 A line is drawn to divide detailed transactions of a client's different accounts

9 Boldface to highlight the client's final balance

10 Inverse color text message for important information

If you want to format a file into AFP for both printing and archiving, you can build AFP indexes based on the account number, client's name, client's address and date, so that you can quickly retrieve the pages of a particular client.

Starting a MakeAFP Formatting Session

"**Start**" function starts and establishes the initiation of a MakeAFP Formatting session, allocates the memory required, parses the parameters defined in the MakeAFP definition file, retrieves all of the AFP resources or OpenType/TrueType fonts required by your program either by generating an external resource file or embedding resources inline within your AFP document files and also retrieves AFP font or OpenType/ TrueType fonts information required by MakeAFP Formatter for the text formatting alignments.

Sample:

```
FILE *fdin;

void main(void)
{
  Start();                  // Starts initiation, parse definition file
                            // and open AFP output files, retrieves
                            // AFP resources and information, as well
                            // as OpenType/TrueType font, allocates
                            // memory required

  fdin = OpenDataFile();    // Open default input data file specifying
                            // by command-line flag parameter -i input_file

      :
      :

}
```

Opening and Closing an AFP Document

You must call the "**Open Document**" function to initialize an AFP document before you open an AFP page, and you must close this AFP document by the "**Close Document**" function before ending your program.

MakeAFP Formatter allows the opening of up to 10 AFP document output files with the output file naming convention *afpfilename.nn.afp* (nn – AFP document file number). Most of the print bureau environments may need the printing AFP jobs to be separated by the number of pages per customer, required by its pre-post devices, inserter, or mailing system, for example, putting 1 – 5 pages per customer's AFP data in an AFP file, and putting 6 pages or more per customer's AFP data in another AFP file; another reason is that you may need to split your output into several segmented AFP files, so you can print the segmented AFP files quickly or sent them to several IPDS printers for the balancing of printing concurrently.

MakeAFP Formatter embeds AFP resources into each AFP output document file if the AFP resource inline parameter is specified in the MakeAFP definition file.

Sample 1: single AFP output file

```
void main( )
{
  Start();              // Starts initiation

  OpenDoc();            // Opens default AFP document

          :
          :

  CloseDoc();           // Closes default AFP document and its file

}
```

Sample 2: multiple AFP output files

```
void main( )
{
  Start();              // Starts initiation

  OpenDoc();            // Opens default AFP document, the first
                        // AFP document

  OpenDoc(2);           // Opens second AFP document

          :
          :

  CloseDoc()            // Closes default AFP document and its file

  CloseDoc(2);          // Closes second AFP document and its file

}
```

Setting Default Units

You can set default units by calling the "**Set Unit**" function before calling the "Open Document" function or at any time if you want to change the default measurement units.

MakeAFP Formatter default is IN_U600 if you do not call the "Set Unit" function.

Sample:

```
void main( )
{
  Start();

  SetUnit(IN_U300);       // Set default units to inch, 300 pixels
                          // per inch
                          // Set units Before OpenDoc() function

  OpenDoc();

  OpenPage(8.5,11);       // LETTER paper size, 8.5" x 11"

     :
     :

  ClosePage();

  CloseDoc();

}
```

Opening and Closing a Page

You must open and close every page within the AFP document. MakeAFP Formatter does not automatically close pages, page closing has to be done by your application control.

With MakeAFP Formatter, you can open multiple pages by the "**Open Page**" functions, and then process different pages in arbitrary order once each page is initialized, all the AFP data stream will be kept in memory buffers in page-level, and only to be written to one of the AFP document files you opened until the page is closed with the "**Close Page**" function.

Sample:

```
void main( )
{
  Start();

  SetUnit(IN_U600);

  OpenDoc();

  OpenPage(8.5,11);              // Open an AFP page
                                 // LETTER paper size, 8.5" x 11"
     :
     :
  ClosePage();                   // Close AFP page, write to default
                                 // AFP file - first AFP document

     :

  OpenDoc(2)                     // Open second AFP document

  OpenPage(21,29.7);             // A4 paper size, 21cm x 29.7cm

     :
     :
  ClosePage(2);                  // Close AFP page, write to second
                                 // AFP document file
     :
     :

  CloseDoc();                    // Close first AFP document

     :
     :

  CloseDoc(2);                   // Close second AFP document


}
```

Text Positioning

MakeAFP Formatter provides powerful text positioning functions and variables to help you position your data and plot graphics easily. After you called the "Open Page" function, you can use them to position your data and graphics quickly. The following table shows the functions and variables.

MakeAFP Formatter functions	Description
Margin (float value)	Sets the inline left margin
LineSpace (float value)	Sets baseline spacing
LPI (float value)	Sets baseline spacing by LPI (lines per inch)
NextLine ()	Skips to the next begin line position defined by Margin() and LineSpace() or LPI()
Skip (float value)	Skips lines then start from margin
Xpos (float value)	Sets absolute horizontal position
Ypos (float value)	Sets the absolute vertical position
Pos (float x, float y)	Sets absolute horizontal and vertical position
Xmove (float value)	Sets horizontal position relative to the current horizontal position, you can specify a negative value
Ymove (float value)	Sets vertical position relative to the current vertical position, you can specify a negative value
GPos (float x, float y)	Sets absolute horizontal and vertical position of GOCA graphic
Cm (float value)	Specifies a value in centimeters
mm (float value)	Specifies a value in millimeters
inch (float value)	Specifies a value in inches
GetXpos()	Gets current AFP text X position
GetYpos()	Gets current AFP text Y position

Sample 1:

```
OpenPage(8.5,11);            // Open AFP page, LETTER page size
Pos(1,1);                    // Position at (1",1")
Ltxt("Testing sample 1");    // Left put text at (1", 1")
Ymove(0.5);                  // Move Y baseline down 0.5"
Ltxt("Testing sample 2");    // Left put text at (1", 1.5")
Xpos(mm(15) );               // Set X position at 15 mm
YPos(cm(2.5) );              // Set Y position at 2.5 cm
Ltxt("Testing sample 3");    // Left put text at (15mm, 2.5cm)
ClosePage();                 // Close AFP page
```

Sample 2:

```
OpenPage(8.5,11);            // Open AFP page
                             // LETTER paper size, 8.5" x 11"
Ypos(1);                     // Set Y position to 1"
Margin(1.5)                  // Set left margin to 1.5"
LineSpace(0.25);             // Set baseline increment to 0.25"
Skip(25.4);                  // Skip 25.4 lines
Ltxt("Testing sample 4");    // Left put text at X = 1.5",
                             // y = 1" + (0.25" x 25) = 7.25"
ClosePage();                 // Close AFP page
```

Putting Color Text on the Page

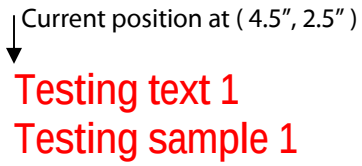
After you opened a page, you must specify the position and font before you can put color text into the page. You can specify IBM OCA standard color, RGB color, or CMYK color for your text.

Text string will be presented exactly as entered except alignment, if you want to do some manipulation on your text string, you can use string functions provided by C and C++ or MakeAFP Formatter before you put the text on the page.

MakeAFP Formatter provides the "Left Text", "Right Text", "Center Text" functions for putting your text on a page by the left, right, and center alignments relative to the current position, and it also provides alignment functions for DBCS-PC (GBK, GB18030, BIG5, SJIS, KSC), DBCS-HOST, as well as the new generation Unicode data string UTF-8 and UTF-16 by using OpenType/TrueType fonts. Refer to *MakeAFP Formatter Reference* for more details.

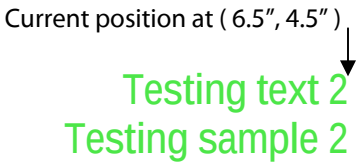
Left Alignment:

```
Pos( in(4.5), in(2.5) );
/* Use font 1 defined by MakeAFP */
/* definition file               */
Font(1);
/* OCA RED color */
Color(RED);
Ltxt("Testing text 1");
Ltxt("Testing sample 1");
```



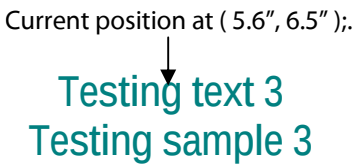
Right Alignment:

```
Pos( in(6.5), in(4.5) );
/* Use font 2 defined by MakeAFP */
/* definition file               */
Font(2);
/* RGB color */
ColorRGB(76, 230, 76);
Rtxt("Testing text 2");
Rtxt("Testing sample 2");
```



Center Alignment:

```
Pos( in(5.6), in(6.5) );
// Use font 3 defined by MakeAFP
// definition file
Font(3);
// CMYK color, by percentage
ColorCMYK(100,76,76,0);
Ctxt("Testing text 3");
Ctxt("Testing sample 3");
```



* OCA color data stream is supported by most of the IPDS printer controllers, RGB, and CMYK color data streams are only supported by the new IPDS printer controllers.

Formatting Paragraphs

Paragraphs are blocks of texts bounded in a rectangle area. With MakeAFP Formatter, you can specify the width of your paragraph, colors and underline your text, and also control text alignment. MakeAFP Formatter provides paragraph functions to handle ASCII or EBCDIC text, as well as DBCS-PC (GB18030, GBK, BIG5, SJIS, KSC), DBCS-HOST, UTF-8, and UTF-16.

Sample 1: fixed paragraph

```
const helv14 = 1;     // For your own convenience, you can define a constant
                      // variable as your local font alias name

  char *text = "AFP is an integrated hardware and software architecture";

  DefaultCode("ibm-437");       // Input data is USA ASCII with codepage 437

  DefaultLocale("en_US");       // language locale is USA English

  SetUnit(MM_U600);

  OpenDoc();

  OpenPage(210,297);

  LPI(5);                       // Set line spacing to 5 LPI
  Pos(20,10);
  Font(helv14);                 // Use local alias font-name helv14
  Color(CYAN);
  ParTxt(text, 45, LEFT);       // Left align paragraph, width = 45 mm

  Pos(20,30);
  Color(BLUE);
  ParTxt(text, 45, RIGHT);      // Right align paragraph

  Pos(80,10);
  Color(BLACK);
  BgnUscore();                  // Begin underscore
  ParTxt(text, 45, CENTER);     // Center align paragraph
  EndUscore();                  // End underscore

  Pos(80,30);
  Color(RED);
  ParTxt(text, 45, JUSTIFY);    // Justify align paragraph

  ClosePage();
  CloseDoc();
```

Output:

AFP is an integrated hardware and software architecture

AFP is an integrated hardware and software architecture

AFP is an integrated hardware and software architecture

AFP is an integrated hardware and software architecture

Sample 2: variable paragraph

```
char *name = "David B. Lee";

char *msg1 = "CONGRATULATIONS, ";
char *msg2 = ", because of your excellent credit rating, ";
char *msg3 = "you are now eligible for free credit insurance.";

SetUnit(MM_U600);

OpenDoc();

DefaultCode("ibm-437");           // Input data is USA ASCII with codepage 437

DefaultLocale("en_US");           // language locale is USA English

OpenPage(210,297);

Pos(20, 50);

LPI(4);                           // Set line spacing to 4 LPI

BgnParTxt(110, LEFT);             // Begin a variable paragraph, 110 mm
                                  // width, left aligned
Font(4);                          // Use font 4 defined in MakeAFP definition
PutParTxt(msg1, PINK);            // Put 1st text in pink color

Font(3);                          // Use font 3 defined in MakeAFP definition
PutParTxt(name, BLUE);            // Put client name in blue color

PutParTxt(msg2, BLACK);           // Put 3rd text in black color

PutParTxt(msg3, GREEN, ON);       // Put 4th text in green color, and
                                  // turn on underscore

EndParTxt();                      // End variable paragraph

ClosePage();
CloseDoc();

#ifdef _DEBUG                     // Only view AFP in debug mode

  ViewAFP();                      // View AFP file generated

#endif
```

Output:

CONGRATULATIONS, David B. Lee, because of your excellent credit rating, you are now eligible for free credit insurance.

Drawing Color Lines

With MakeAFP Formatter, you can draw the vertical and horizontal color lines in fixed or variable sizes and by plotting in GOCA graphics, you can also select line-type and plot slant lines.

Sample 1:

```
SetUnit(IN_U300);
OpenPage(8.5,11);

Hline(0.5, 10.6, 7.5, 0.02);      // Draw a horizontal line from
                                  // (0.5",10.6"), 7.5" length,
                                  // 0.02" width, red color
Vline(0.5, 0.5, 5.5, 0.01);       // Draw a vertical line from
                                  // (0.5",0.5"), 5.5" length,
                                  // 0.01" width, BLUE color
BgnVline(0.5, 0.5, 0.03);         // Draw a variable size vertical
                                  // black line from (0.5",0.5"),
                                  // 0.03" width, default is black
            :
            :
EndVline();                       // End vertical line at current
                                  // position
ClosePage();
```

Sample 2: using GOCA graphic

```
SetUnit(IN_U240);
OpenPage(8.5,11);

GLineWidth(2);                    // About 0.02" line width
GColor(RED);                      // Set GOCA OCA color to RED
GLineType(SOLID);                 // Set line type to solid line
GHline(1, 1, 3);                  // Plot horizontal line from
                                  // (1", 1"), 3" length

GColor(30,100,30);                // Set a GOCA RGB color
GLineType(LONGDASH);              // Set line type to long dash
GVline(2, 2, 5);                  // Plot vertical line from
                                  // (2", 2"), 5" length

GColor(65,100,45, 0);             // Set a GOCA CMYK color
GPos(3,3)                         // Set GOCA position to (3",3")
GLineTo(6,6);                     // Plot slant line from current
                                  // position to (6",6")

ClosePage();
```

* OCA color data stream is supported by most IPDS printer controllers, GOCA graphic, RGB, and CMYK color data streams are only supported by the new IPDS printer controllers.

Drawing Color Boxes

With MakeAFP Formatter, you can draw the color boxes in fixed or variable sizes with color shading, and you can also plot a rounded box with a color background by GOCA graphics.

Sample 1:

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);

Box(0.5, 4, 7.5, 0.4, 0.02);     // Draw box from (0.5",4"),
                                 // box width = 7.5", height = 0.4"
                                 // line thick = 0.02"
Shade(0.5, 4, 7.5, 0.4, 5);      // Draw shading from (0.5",4"),
                                 // shading width = 7.5", height = 0.4",
                                 // 5% shading
ClosePage();
CloseDoc();
```

Sample 2: by GOCA graphic

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(210, 297);

GColorRGB(165,255,165); // Set RGB light green color for color area
GBgnFill();             // Begin filling a GOCA area, default is solid
                        // pattern
GLineWidth(4);          // Line width is about 0.04"
GColor(DARKBLUE);       // Set OCA dark blue color for box line
GBox(50,50,50,30, 3);   // Box size 50 x 30 MM, rounded corners 3 mm radius
GEndFill();             // End filling GOCA area
Pos(57,67);             // Set position at (57,27) for text data
Color(RED);             // Set OCA color red for text
Font(1);                // Use font 1 defined in MakeAFP Definition file
Ltxt("Coloured Box");   // Left put the text

ClosePage();
CloseDoc();
```

Output:

* OCA color shading is supported by most IPDS printer controllers, GOCA graphic, RGB, and CMYK color data streams are only supported by the new IPDS printer controllers.

Including AFP Object and Data-Object

AFP resources are the objects that you have previously created with MakeAFP or other tools and are stored in MakeAFP AFP resource library directories. With MakeAFP Formatter, you can include an AFP overlay or AFP page segments at a fixed or dynamic position.

With the latest AFP Systems, you can Include a reference directly to an AFP object (image, graphic, barcode), or a non-AFP data-object(TIFF, JPEG, GIF, etc) at the specified position or current position, and specify the area size, rotation, mapping option for the object to be presented in high performance with a CMR (Color Management Resource) and specific color rendering intent.

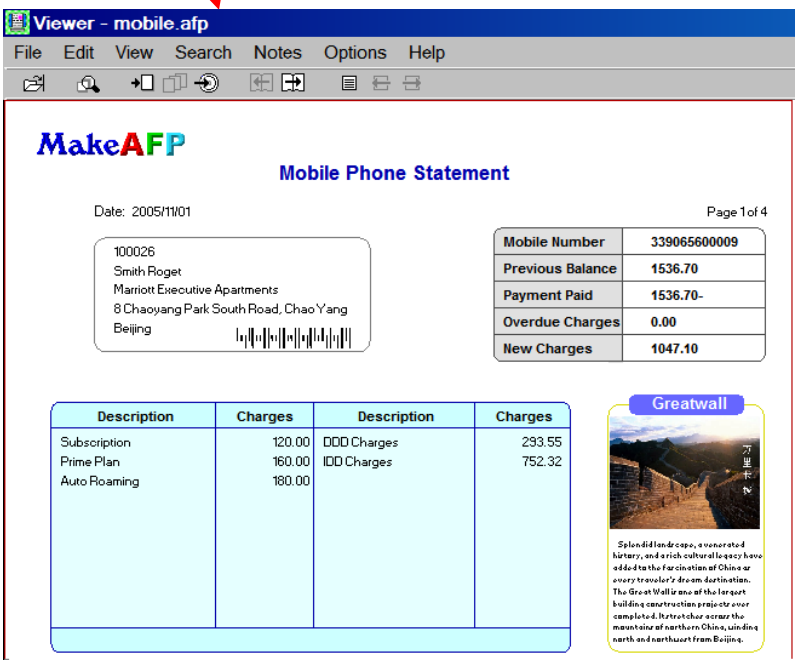
Using an object as a resource is more efficient when that object appears more than once in a print job; resources are downloaded to the printer just once and referenced as needed.

Besides, you also can directly import data-object images, AFP IOCA image objects, GOCA graphic objects inline within your AFP document for the dynamic images/graphic formatting.

Sample 1: Including an AFP page segment

```
SetUnit(IN_U600);

OpenPage(8.5,11);
        :

InclPseg("S1MKAFP",0.2,0.2);          // Include AFP page segment at
                                      // (0.2", 0.2")
        :

ClosePage();
```

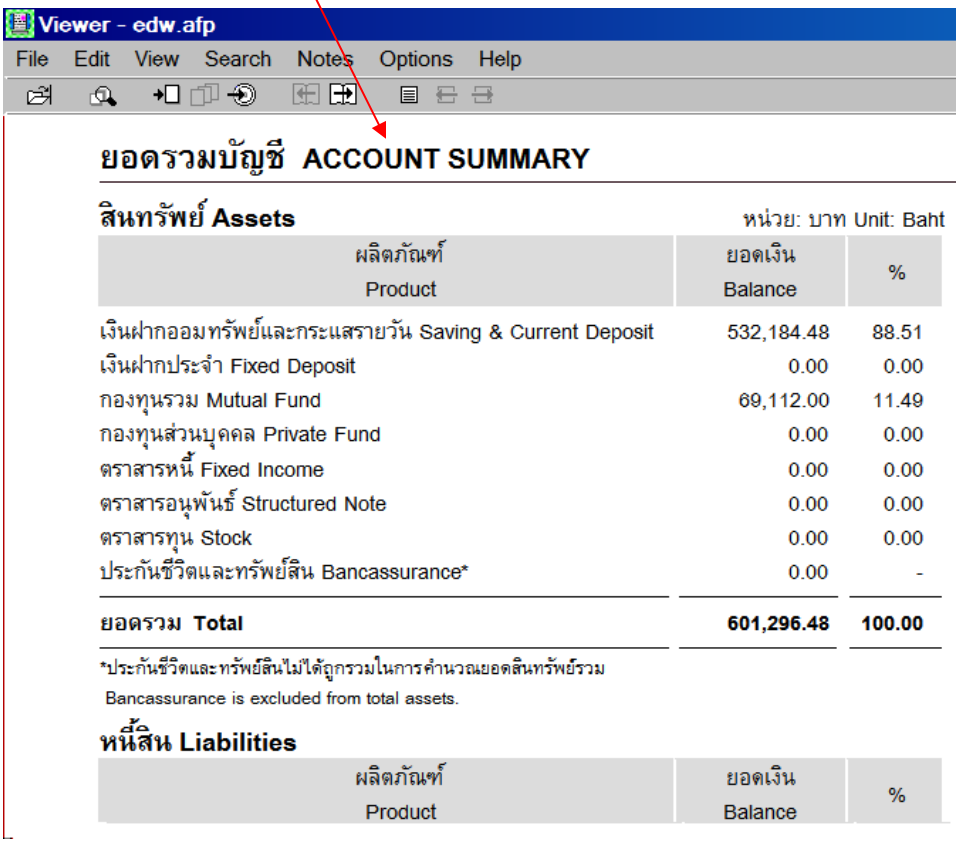
Output:



Sample 2: Including an AFP overlay

```
SetUnit(IN_U600);

OpenPage(8.5,11);

        :
        :

InclOvly("01SUMARY",0.2,0.3);        // Include AFP overlay at

        :                            // (0.2", 0.3")
        :

ClosePage();
```

Output:

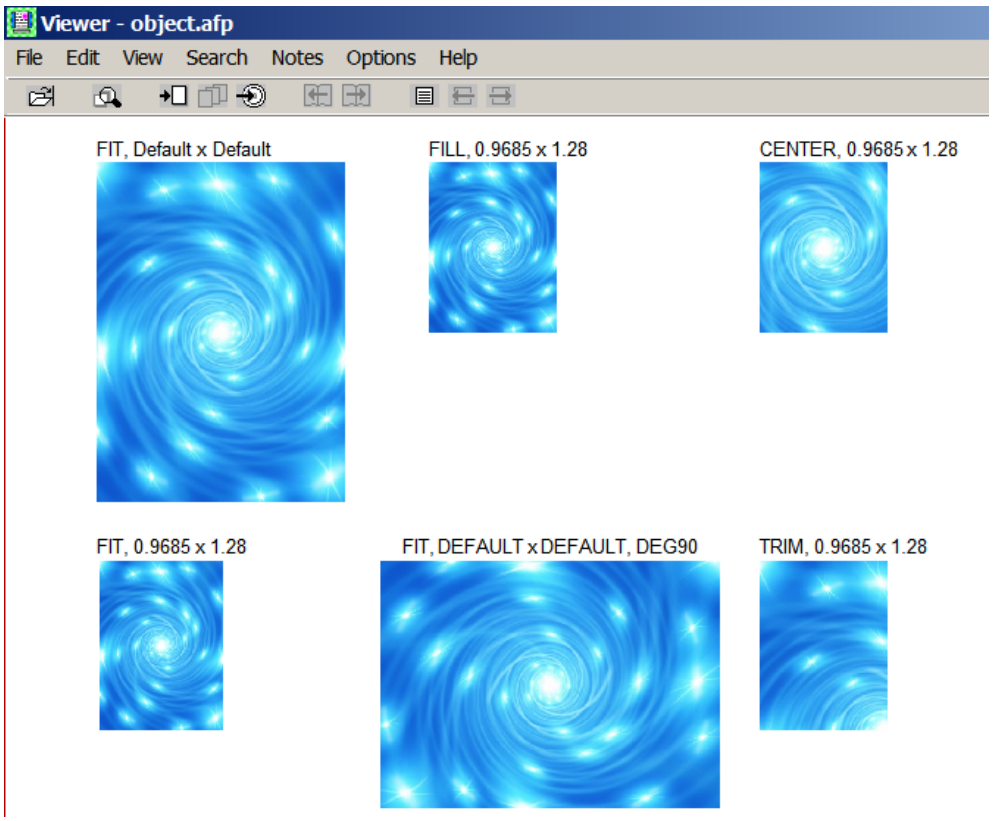


Sample 3: Including data-object images

```
SetUnit(IN_U600);              // Set default units to inch, 600 dpi

OpenPage(8.27, 11.67);         // A4 Paper size

Font(1);

Pos(0.7,0.95);
Ltxt( "FIT, Default x Default");

InclObjt("SAMPLE",0.7,1, DEFAULT, DEFAULT);   // Included an JPEG image,
                                              // image type is defined
```

```
Pos(3.2,0.95);                                  // in MakeAFP definition
Ltxt( "FILL, 0.9685 x 1.28");                   // file

InclObjt("SAMPLE",3.2,1, 0.9685, 1.28, FILL);

Pos(5.7,0.95);
Ltxt( "CENTER, 0.9685 x 1.28");

InclObjt("SAMPLE",5.7,1, 0.9685, 1.28, CENTER);

Pos(0.7,3.95);
Ltxt( "FIT, 0.9685 x 1.28");

InclObjt("SAMPLE",0.7,4, 0.9685, 1.28, FIT);

                :
                :

ClosePage();
```

Output:



* Data-object images, such as EPS, GIF, JPEG, PDF, and TIFF are only supported by the new IPDS printer controllers.

Plotting Bar Charts

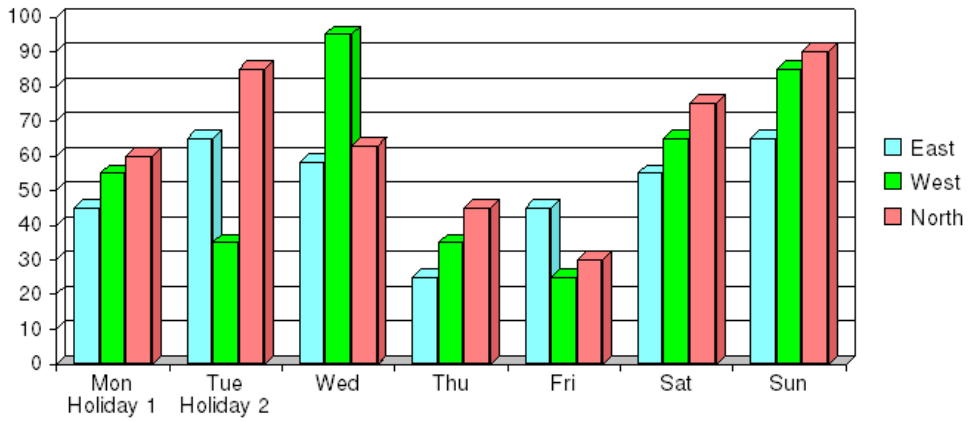
Powerful 2D and 3D bar chart functions are provided to generate business bar charts quickly. The bar chart can be generated either in high-quality AFP GOCA graphics or in legacy shading with a small AFP data stream size.

Sample:

```
float data[] = {45, 55, 60,
                65, 35, 85,
                58, 95, 63,
                25, 35, 45,
                45, 25, 30,
                55, 65, 75,
                65, 85, 90};

char *lbls[]  = {"Mon\nHoliday 1", "Tue\nHoliday 2", "Wed", "Thu", "Fri",
                 "Sat", "Sun"};
char *legend[] = { "East", "West", "North" };

SetUnit(IN_U1440);
OpenDoc();
OpenPage(8.5,11);
LPI(8);

GPos(1, 2.5);                // Set position of the barchart
Font(1);                     // select font 1 for barchart
GBarchart(2,                 // Max height of bar chart
          0.15,              // Width of each bar
          0.2,               // Gap between each bar set
          10,                // unit size of v-axis scale
          10,                // number of v-axis scales
          3,                 // number of bars per bar set
          7,                 // number of bar set
          data,              // data array of input data
          lbls,              // label texts below H-axis
          legend);           // legend texts on right side
ClosePage();
CloseDoc();
```

Newline control code is used to split labels' text

Output:



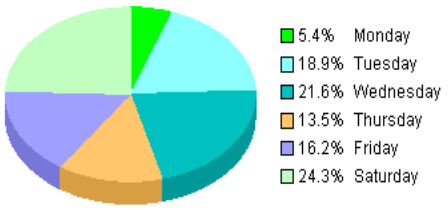
Plotting Piecharts

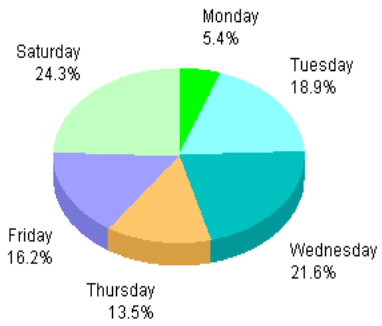
Powerful 2D and 3D pie chart functions are provided to help you generate business pie charts quickly. The pie chart is generated in a high-quality AFP GOCA graphic with a small AFP data stream size.

Sample:

```
float data[] = {15, 35, 40, 25, 30, 45, 55};
char *label[] = { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
                  "Saturday", "Sunday"};

Font(2);
GPos(1.5, 1.5);            // Set current GOCA position

GPiechart(1.6,             // Piechart width
          1,               // Piechart height
          0.13,            // Piechart shadow depth
          6,               // sectors of the piechart
          data,            // Input data array
          label,           // Label text array
          PCTLBL);         // Show percent and labels by the legend

GPos(2, 3.5);              // Set current GOCA position
GPiechart(1.6,             // Piechart width
          1,               // Piechart height
          0.13,            // Piechart shadow depth
          6,               // sectors of the piechart
          data,            // Input data array
          label);          // Label text array
```

Output:





Working with Bar Codes

MakeAFP Formatter supports all of the linear and 2D barcodes defined in AFP's latest BCOCA standard.

MakeAFP Formatter supports barcodes not only by IBM BCOCA object but also over 50 types of popular linear and 2D barcodes by MakeAFP barcode drawing with a small AFP data stream size.

Linear and 2D barcodes generated by MakeAFP Formatter drawing can be displayed and printed on any type of printer or presentation device with full fidelity and high print/display quality.

Sample 1: Code 128 bar code by AFP BCOCA

```
char *data = "1234567890";

SetUnit(IN_U1440);

OpenDoc();

OpenPage(8.5,11);

BBarCode(CODE128,          // BCOCA Bar code type is Code 128
         data,             // Bar code data variable
         1,                // Bar code x position at 1"
         1,                // Bar code Y position at 1"
         20,               // Bar code module width in mils (thousandths
                           // of an inch)
         0.5);             // Bar code element height 0.5"
                           // Other parameters use defaults

ClosePage();

CloseDoc();
```

Print output:

* BCOCA object is supported by the MakeAFP Viewer and new IPDS printer controllers, please check with your printer vendor whether its printer controller microcode level supports the bar code type you defined.

* Same BCOCA objects may be printed in different dimensions on different vendor's IPDS printers.

* Not every AFP viewer can view BCOCA objects, please check with the vendor of your AFP viewer.

Sample 2: Royal 4-state postal barcode by MakeAFP barcode drawing

```
char *data = "123456";
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
Barcode(RM4SCC,              // Royal 4-state postal barcode
        data,                // Bar code data variable
        1,                   // Bar code x position at 1"
        1,                   // Bar code Y position at 1"
        1.3,                 // Bar code dimension width 1.3"
        0.35);                // Bar code dimension height 0.35"
ClosePage();
CloseDoc();
```

Print / display output:

Sample 3: DataMatrix 2D barcode by MakeAFP barcode drawing

```
char *data = "1234567890 this is testing of DataMatrix";
SetUnit(IN_U600);
OpenDoc();
  OpenPage(8.5,11);
    DataMatrix(data,1.2,1.5);       // position at (1.2",1.5")
  ClosePage();
CloseDoc();
```

Print / display:

Working with Pagination Numbers

MakeAFP Formatter offers a powerful capability that allows you to generate your pagination numbers easily, such as “Page x of y”, on every page of your statements. With MakeAFP Formatter, you can open multiple pages by the “Open Page” functions, and then process different pages in an interleaved manner once each page is initialized, all of the AFP data streams will be kept in memory buffers at page-level. After you processed and counted all of the pages of a page-group, you can put your pagination numbers text, OMR, or barcode on each page just before you close the page with the “Close Page” function.

MakeAFP Formatter provides a special variable **PageNum**, with which you can switch to any AFP page directly regardless of if it is to be opened or already opened.

The C sample shown below was extracted from the program developed for the sample bank statement shown in the figure on page 8, It uses the MakeAFP Formatter function Ypos() to check if the current Y position of the data is greater than 10.3 inches on the page. If so, a new page will be opened and generated. Note that the current page is not ended because we still need to place the “Page x of y” string once on each page until we have counted the total pages.

Sample:

```
SetUnit(IN_U600);
MaxPaging = 1000;              // MaxPaging is a MakeAFP Formater special
                               // variable, to define the maximum page
                               // buffers as 1000, it must be defined
                               // before Start() function call
Start();                       // Start a MakeAFP session
OpenDoc();
               :
// Check to see if Y current position is at the bottom of the page, if
// so, then create a new page by create_new_page(client_record) routine
if (Ypos() > 10.3)                // if current Y position > 10.3 inches
{
   PageNum++;                        // Switch to next page buffer number
   create_new_page(client_record);
}
               :
// Before close up all of the pages of a client, on each page, move
// to a point just above the headings and then put the pagination numbers
// string "Page x of y"
unsigned short numpages = PageNum;  // Keep total pages of a client
char tmp[25];
for (int i = 0; i < numpages; i++)
{
   PageNum = i + 1;               // Switch to each page buffer
   sprintf(tmp, "Page %d  of  %d", PageNum, numpages);
   Pos(8.0,3.93);                 // Set position at (8", 3.93")
```

```
   Rtxt(tmp);                    // Right alignment of page number
                                 // on each page before end of page
   ClosePage();                  // close each page
}
                :
 PageNum = 1;                    // Reset AFP page buffer number to 1
                                 // for the next customer statement
```

Creatin Page Group Indexes

The AFP document pages can be organized into smaller, uniquely identifiable units, called page groups, each group represents a logical multi-page bundle. MakeAFP lets you divide a large AFP document into individual page groups, by inserting AFP indexing structured fields in the AFP file that define the group boundaries.

With the "**Begin Index**" and "**End Index**" functions, you can define the start and end of index page group boundaries within an AFP document, so the statement pages that belong to each client can be quickly navigated and retrieved by the AFP viewer and AFP archiving system, and to be used by MakeAFP reprint and sorting utilities.

With the "**Put Index**" function, you can identify a group of pages with the indexing tag containing an attribute name and value. For instance, an "Account Number" attribute is associated with the account number of each customer. This type of tag is called a group-level tag since it is associated with a group of pages.

Besides, MakeAFP Formatter allows you to generate a group of AFP index objects files, AFP resources files, and AFP document files, that can be loaded into IBM Content Manager OnDemand directly in high performance. Refer to MakeAFP Formatter Reference Chapter 2 for more details about using MakeAFP definition parameters RESTYPE and INDEXOBJ.

For group-level AFP indexing, a "Begin Index" function and "Put Index" functions must be called before writing the first page of each page group, and an "End Index" must be called after writing the last page of each page group.

In the following sample, the information for each client is indexed by the customer's name, and account number.

Sample:

```
                    :
  // Now all input data of a client are formatted into the AFP page
  // buffers, before writing out all of the pages of a client, we can insert
  // beginning of group index tag and index value tags

  unsigned short numpages = PageNum;      // Keep total pages of a client
  char tmp[25];

  BgnIdx(account_no);                     // Begin index page group
  PutIdx("Customer Name", client_name);   // Put group-level index tags,
  PutIdx("Account Number", account_no);   // BgnIdx and PutIdx must be called
                                          // before writing of the first page

  for (int i = 0; i < numpages; i++)      // of each page group
  {
    PageNum = i + 1;                      // Switch to each page buffer
    sprintf(tmp, "Page %d  of  %d", PageNum, numpages);
    Pos(8.0,3.93);                        // Set position at (8", 3.93")
    Rtxt(tmp);                            // Right alignment of page number
                                          // on each page before close each
    ClosePage();                          // page
  }
```

```
EndIdx();                              // End index page group, must be
                                       // called after writing of the last
                                       // page of each page group

PageNum = 1;                           // Reset AFP page buffer number to 1
                                       // for the next customer statement
```

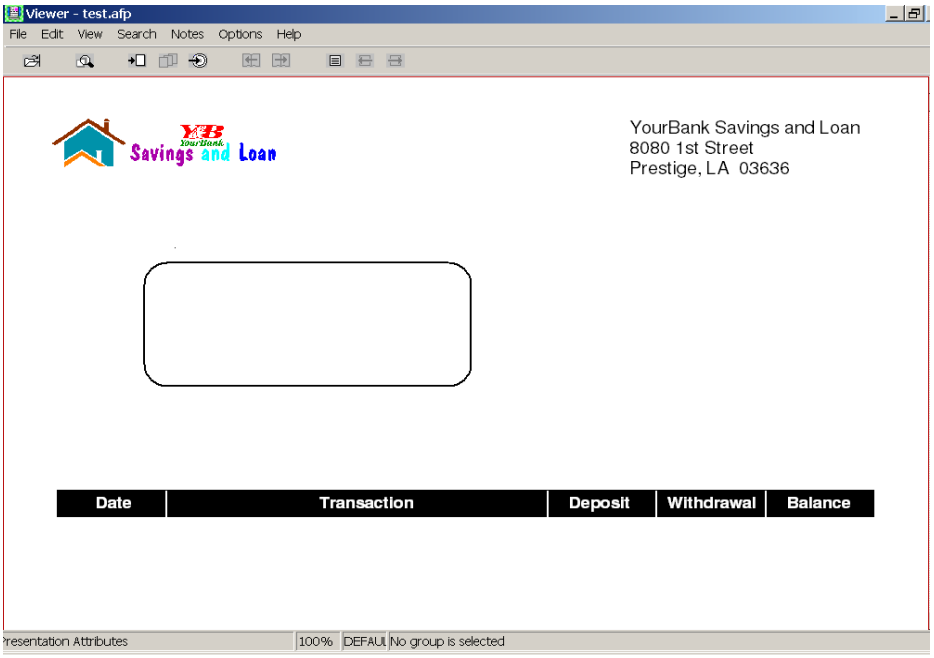
Working with MakeAFP Formatter Overlay Functions

MakeAFP Formatter provides two additional functions “Open Overlay” and “Close Overlay” for creating an AFP overlay with MakeAFP formatting programming, you can use them with all of the functions provided by MakeAFP for text formatting, drawing, graphic plotting, and barcode.

Sample:

```
void main( )
{
  Start();

  SetUnit(IN_U600);

  OpenOvly(8.5,11);                              // Open overlay, size 8.5"x11"

  InclPseg("S1YBLOGC",0.45,0.4);                 // Include an AFP page segment

  Font(1); Pos(5.75,0.55);                       // Put fixed address
  Ltxt("YourBank Savings and Loan");
  Pos(5.75,0.75); Ltxt("8080 1st Street");
  Pos(5.75,0.95); Ltxt("Prestige, LA  03636");

  GLineWidth(BOLD);                              // Set line width to bold
  GBox(1.3, 1.8, 3, 1.2, LARGE);                 // Plot a rounded box

  GColor(BLACK);                                 // set OCA black color
  GBgnFill();                                    // start filling of a box area
  GBox(0.5, 4, 7.5, 0.26, 0.02);                 // plot a box
  GEndFill();                                    // end filling of box area
  GColorRGB(255,255,255);                        // set RBG white color
  GVline(1.5, 4, 0.25);                          // plot vertical lines
  GVline(5, 4, 0.25);
  GVline(6, 4, 0.25);
  GVline(7, 4, 0.25);

  Font(3); ColorRGB(255,255,255);                // Put heading texts
  Pos(0.85, 4.17); Ltxt("Date");
  Xpos(2.9); Ltxt("Transaction");
  Xpos(5.20); Ltxt("Deposit");
  Xpos(6.1); Ltxt("Withdrawal");
  Xpos(7.2); Ltxt("Balance");

  CloseOvly();                                   // Close overlay and its file

  #ifdef _DEBUG                                  // Only view AFP in debug mode

    ViewAFP();                                   // View overlay

  #endif

}
```

Overlay display:


Viewer - test.afp
File Edit View Search Notes Options Help
YourBank Savings and Loan
8080 1st Street
Prestige, LA 03636
Date
Transaction
Deposit
Withdrawal
Balance
100%
No group is selected

MakeAFP Form Designer

MakeAFP Form Designer for Windows provides a user-friendly WYSIWYG graphical interface to empower you to visually create superior AFP overlays with a small AFP data stream quickly, reducing your design time and improving your productivity tremendously.

MakeAFP Form Designer offers a very useful and user-friendly feature helping you with your MakeAFP Formatter programming, it allows you to copy the parameter in C/++ syntax from the active design element/object, and then you can quickly paste the C/C++ source code to your MakeAFP Formatter program.

MakeAFP Form Designer provides the following advanced features and functions helping you to create your AFP overlays at ease:

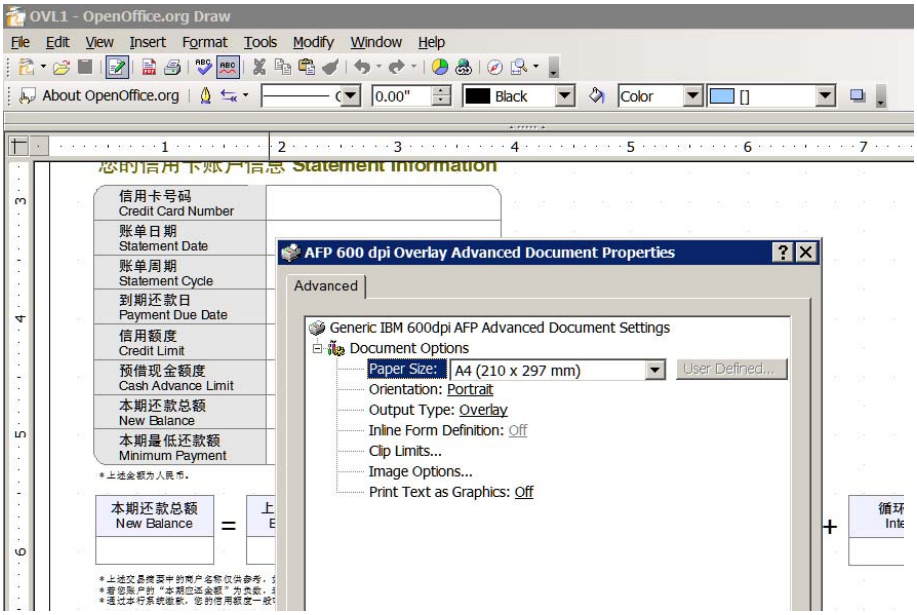
- WYSIWYG GUI in superior presentation quality and high performance.
- Supports legacy AFP ASCII/EBCDIC/DBCS-PC/DBCS-HOST encoding FOCA bitmap and outline fonts, as well as using the new generation of OpenType/ TrueType fonts directly, able select a font by the properties of typeface name, style, and size, able scale outline font width.
- Supports the legacy AFP page segment images as well as the conversion from the popular image formats.
- Supports the data-object containers in AFP page segment, GIF, JPEG, PDF, and TIFF formats.
- Supports ICC profile-based color management natively.
- Supports legacy lines, boxes and shading patterns LED, SCREEN, and STANDARD.
- Supports AFP GOCA vector graphic drawings, such as the color graphic table, box, rounded corner box, line, arrow-line, fillet, marker, ellipse, arc, circle, etc.
- Able to control GOCA vector line width and styles; area filling patterns and colors; curves of each corner of the rounded box.
- Able to repeat drawing of line, box, rounded box, shading, color area precisely.
- Accurate positioning of all types of elements/objects with mouse or keyboard input in units of inches, millimeters, centimeters, or points.
- Displays positioning/sizing parameters in real-time along with the element/object.
- Auto-snap positioning while aligning element/object with the relevant element/object.
- Supports ASCII/EBCDIC/DBCS-PC/DBCS-HOST and UTF-8/ UTF-16 texts, and paragraph rotations and alignments of left, center, right and justify.
- Able to enter universal texts by keyboard and copy/paste at ease.
- Supports zoom levels of the entire page, page width, zooming from 25% to 500%, or a user-specify zoom level.
- Able to use grids and guidelines to facilitate AFP overlay design.
- Supports copy/cut/paste, undo/redo, drag and drop elements/objects.
- Supports both legacy monochrome and full-color elements/objects.

- Supports the graphic rulers with a choice of units.
- Able to use the scan-in image as the form Template with choices of color control.
- Supports all popular types of 1D and 2D barcodes by AFP drawing, including the new US Postal 4-State OneCode barcode.
- Able to import IBM OGL source code.
- Able to export C/C++ source code for MakeAFP Formatter.

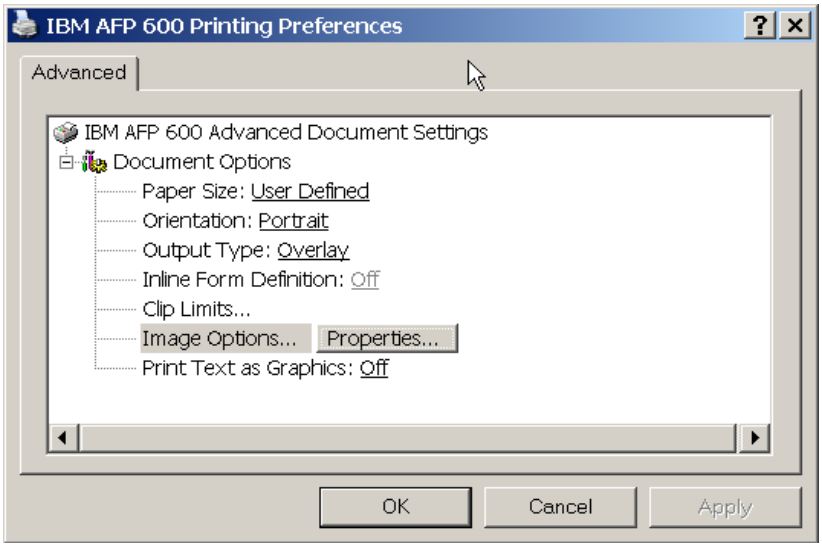
Exporting AFP Overlay from Windows Applications

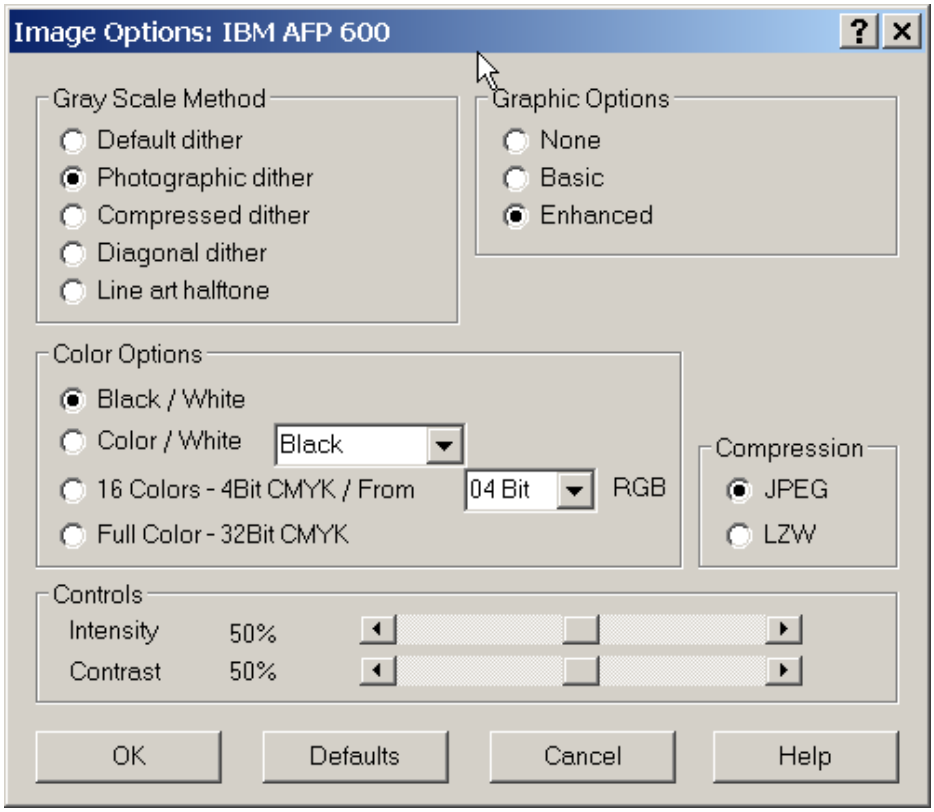
You may want to create AFP overlays with some freeware. Here are some recommendations.

You can export an AFP overlay is by using Ricoh AFP Printer Driver for Windows. With this printer driver, you can simply export your printout as an AFP overlay from your Windows application software. Freeware OpenOffice Draw from its website www.openoffice.org is the recommended form design tool.

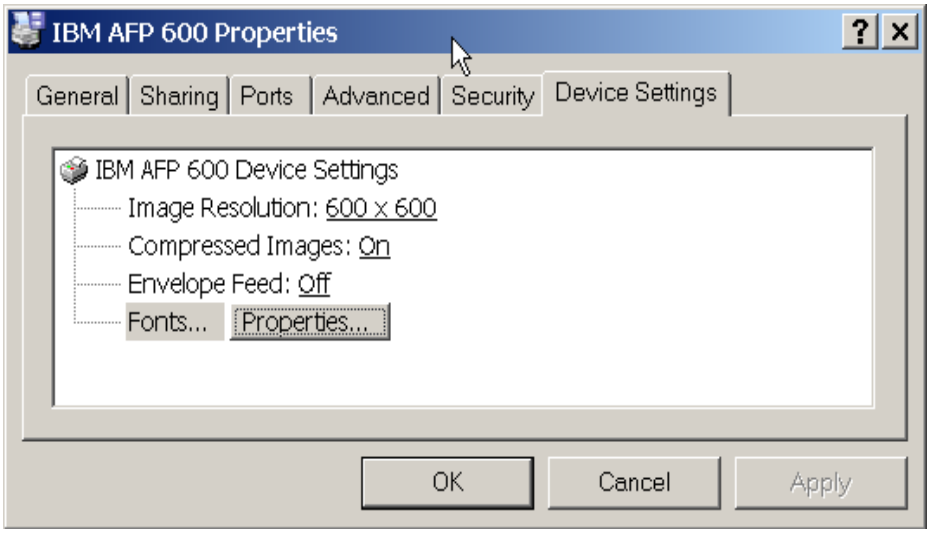


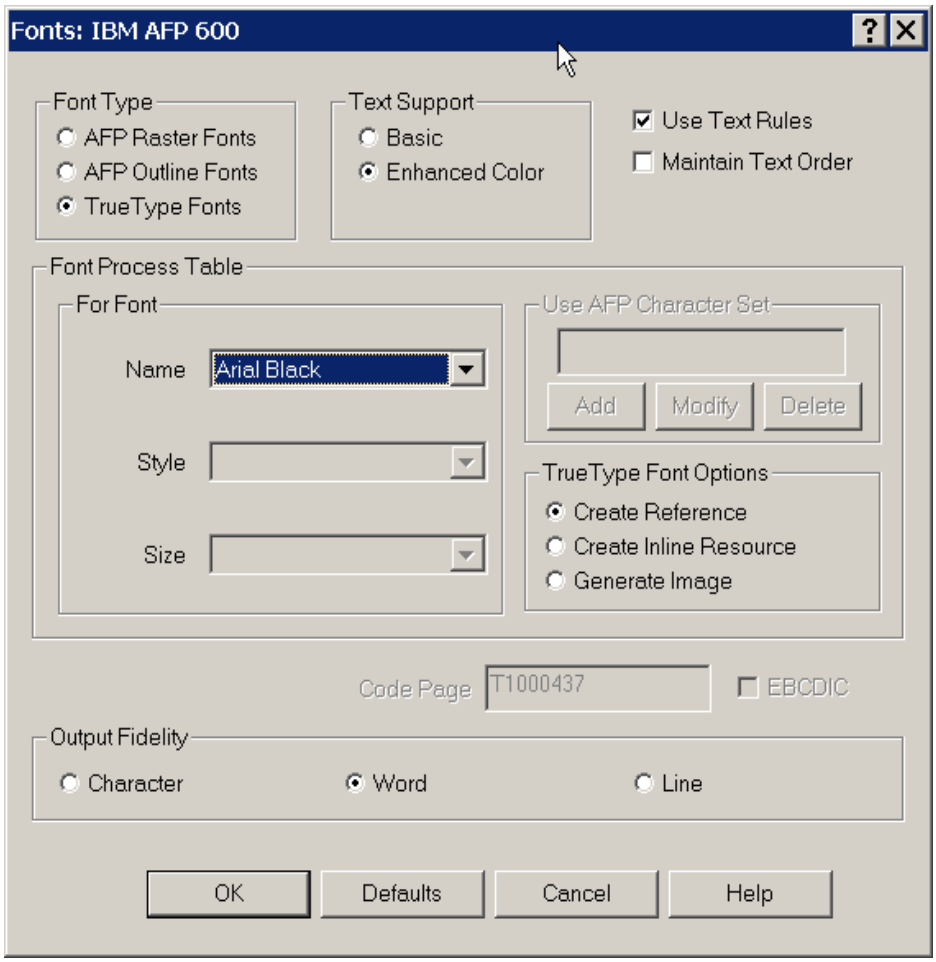
With Ricoh AFP Printer Driver for Windows, you can choose the following parameters with its properties, such as device settings and printing preferences if your IPDS printer supports AFP GOCA graphics:





If your IPDS printer supports OpenType/TrueType fonts, make sure the checkbox "TrueType Fonts" is checked and the fonts are available on your MakeAFP server and AFP print server; Or make sure the AFP outline or raster fonts are available on your MakeAFP server and AFP print server if the checkbox "AFP Outline Fonts" or "AFP Raster Fonts" is checked, IBM AFP Print Driver converts texts as AFP IOCA images if Windows fonts are not mapped to AFP fonts.





Here are some recommendations on how to create a nice overlay for East Asian languages to be used on your latest AFP systems in high performance and superior presentation quality:

Open AFP Printer Driver properties, with "Image Options", select "Gray Scale Method" as "Photographic Dither", and "Graphic Options" as "Enhanced"; with "Font Options", select "Font Type" as "TrueType Fonts",, and "Text Support" as "Enhanced Color".

The latest MS Office XP/2003/2007 (Word, PowerPoint, Excel) work fine with AFP Printer Driver for exporting of a DBCS overlay by using OpenType/TrueType fonts and color vector graphics.

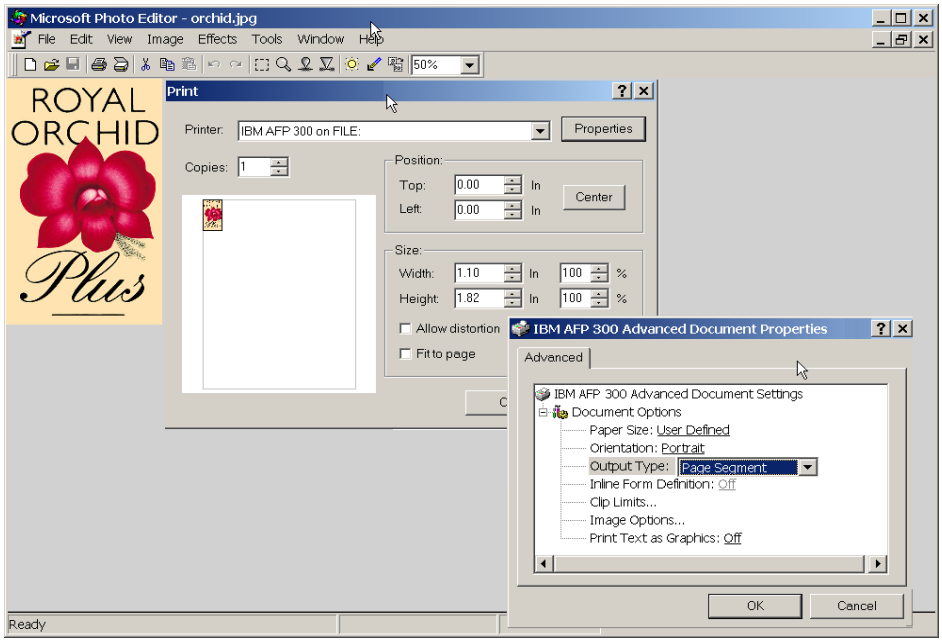
OpenOffice Draw is one of the most powerful vector drawing software, it works fine with AFP Printer Driver for Latin 1 to 8 characters, but for DBCS characters you need to enter them in MS-Word first, then copy them from MS-Word and paste as the "metafile" format to OpenOffice Draw, you can export them as the text stream in AFP UTF-16.

OmniForm is a powerful form design tool, it works fine with AFP Printer Driver for Latin characters, but for DBCS characters you need to enter them in MS-Word first, then copy them from MS-Word and paste as the "OLE object" format to OmniForm, you can export them as the text stream in AFP UTF-16.

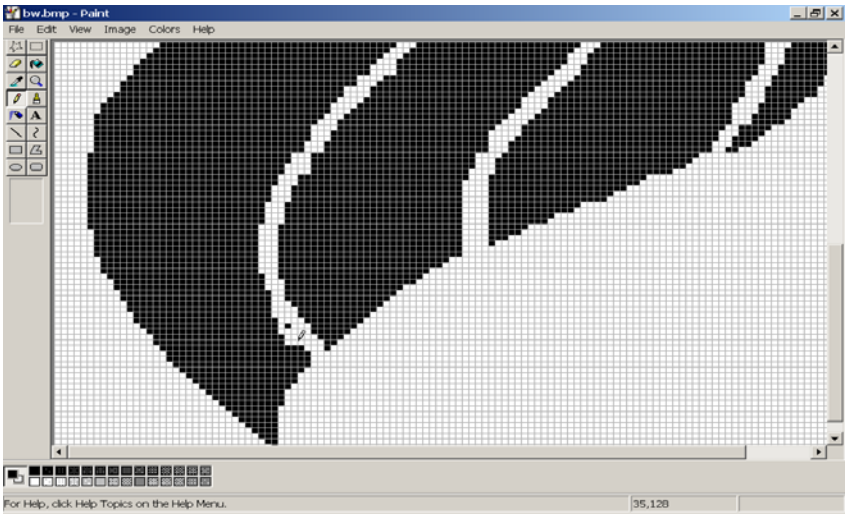
Working with AFP Page Segments

MakeAFP Imager provides a command-line tool to convert a popular format image to AFP page segment images in AFP IOCA monochrome FS11 or color FS45 format.

Another easy way to create an AFP page segment is to use Ricoh AFP Printer Driver for Windows. With this driver, you can simply export your printout as an AFP page segment from your Windows image software or other Windows software.



When you scan-in an image as a black-white image with a scanner, select options of monochrome line-art format and a resolution from the optional 240/300/480/600 dpi according to your printer resolution, then you need to touch-up the bitmap image dot-by-dot for better image quality with Windows image software, like MS Paint for Windows 7.



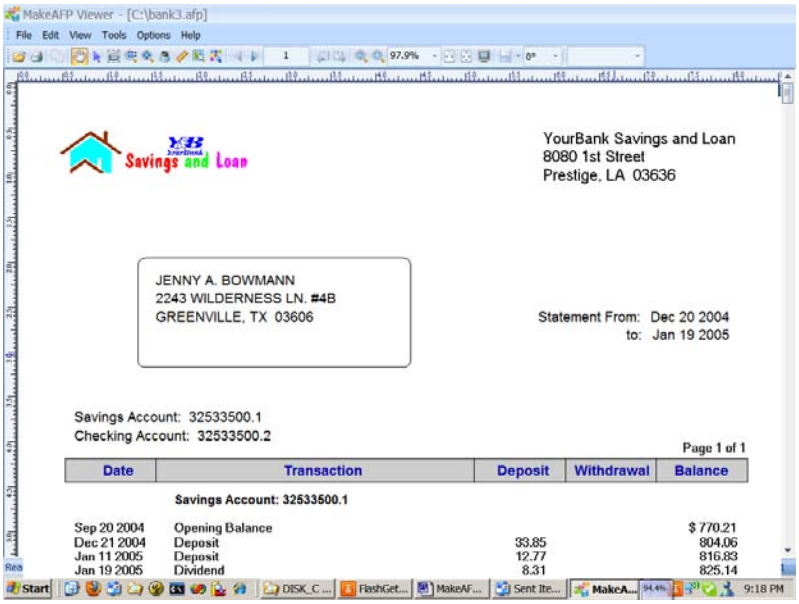
Viewing AFP File

MakeAFP Formatter provides the "View AFP" function with which you can immediately call an AFP viewer to view the AFP file you just generated in debug or execute mode, this will assist you greatly during your development.

AFP viewer for Windows can be integrated easily with MakeAFP Formatter by the "View AFP" function so that you can view a newly generated AFP file immediately during your development.

With Windows Explorer, you can select "Tools --> Folder Options --> File Types --> New" to associate the AFP type file to an AFP viewer. Once you defined the new AFP file type, MakeAFP Formatter will be able to call your AFP viewer to view the generated AFP file.

The "View AFP" function must be called after the "Close Document" function.



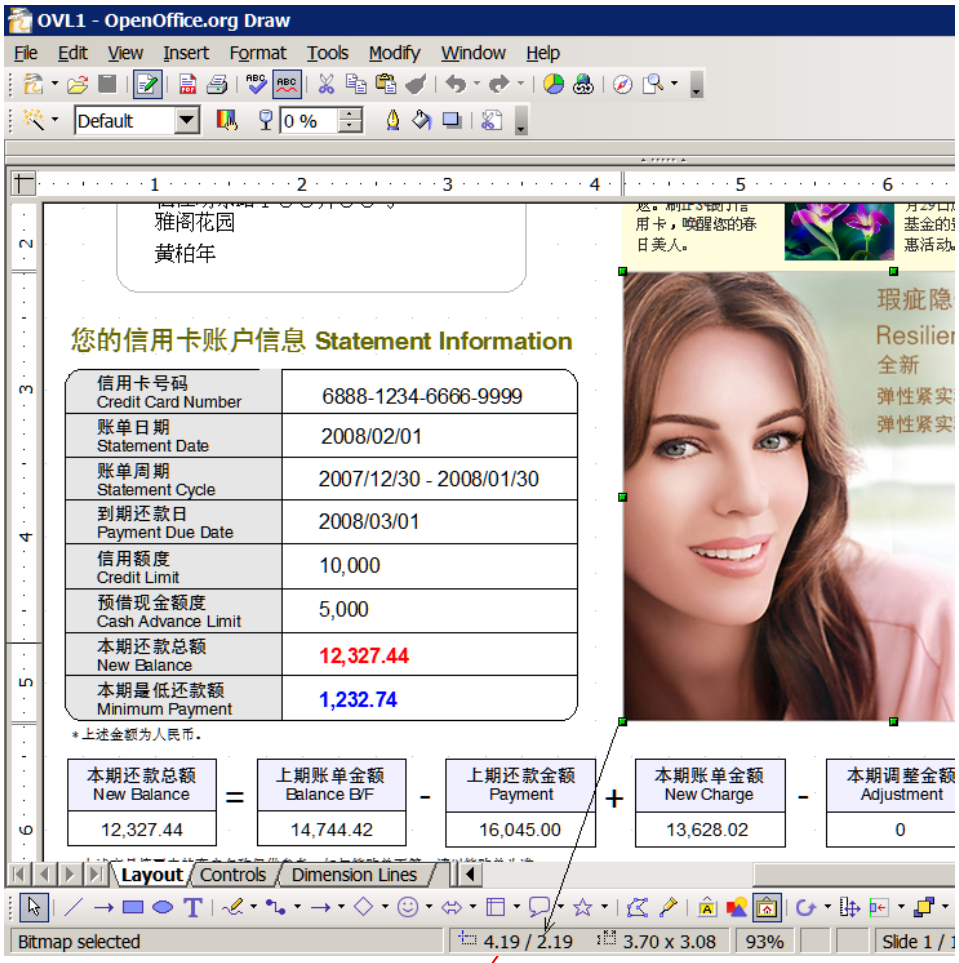
Sample:

```
void main( )
{
  Start();              // Start initiation, open default input,
                        // output and definition files, getting
                        // AFP resources and font information
  OpenDoc();
      :
      :
  CloseDoc();           // End AFP document and close it's file

  #ifdef _DEBUG         // Only view AFP in debug mode

    ViewAFP();          // view AFP file just generated, it must
                        // be called after CloseDoc() function
  #endif
}
```

Getting Help for WYSIWYG Data Positioning

After you have completed the development with Windows WYSIWYG form design software for your overlays, you can continue to rely on the form design tool you have used during your MakeAFP formatting programming for the positioning of the variable input data.

On your form design software, you can find such data fields and object positions from your design templates; or create a simple data field and then move it to the place where your variable data field will be placed. You can then find out its (x, y) position from the WYSIWYG screen, and continue moving this sample field around until you got all the data fields' positions needed for your MakeAFP data positioning.



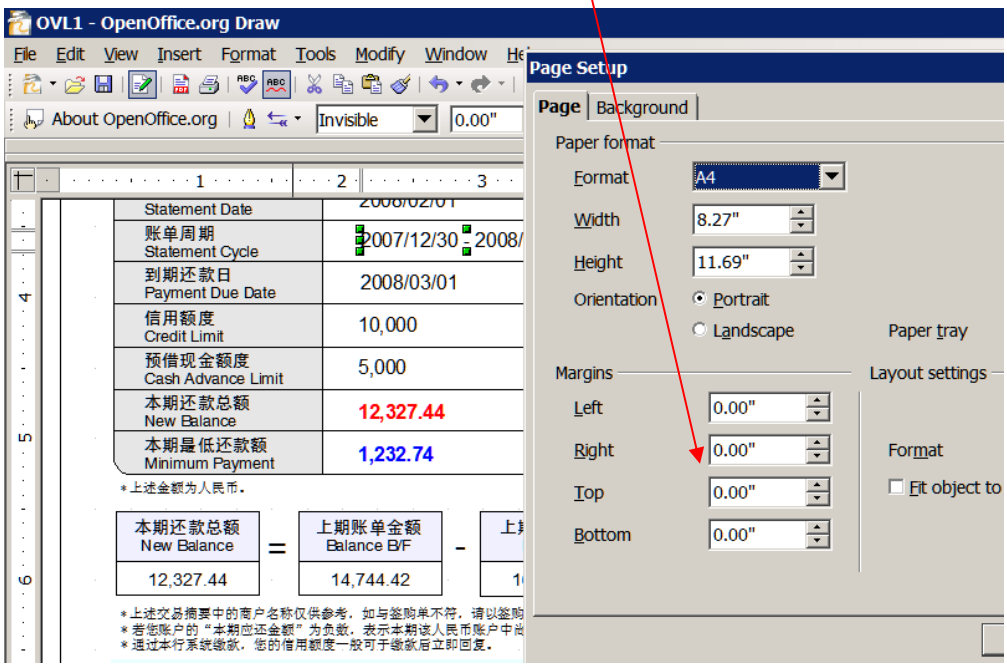
MakeAFP programming:

```
OpenDoc();

SetUnit(IN_U600);
OpenPage(8.5,11);
          :
InclObjt("CMB1", 4.19, 2.19);
          :
```

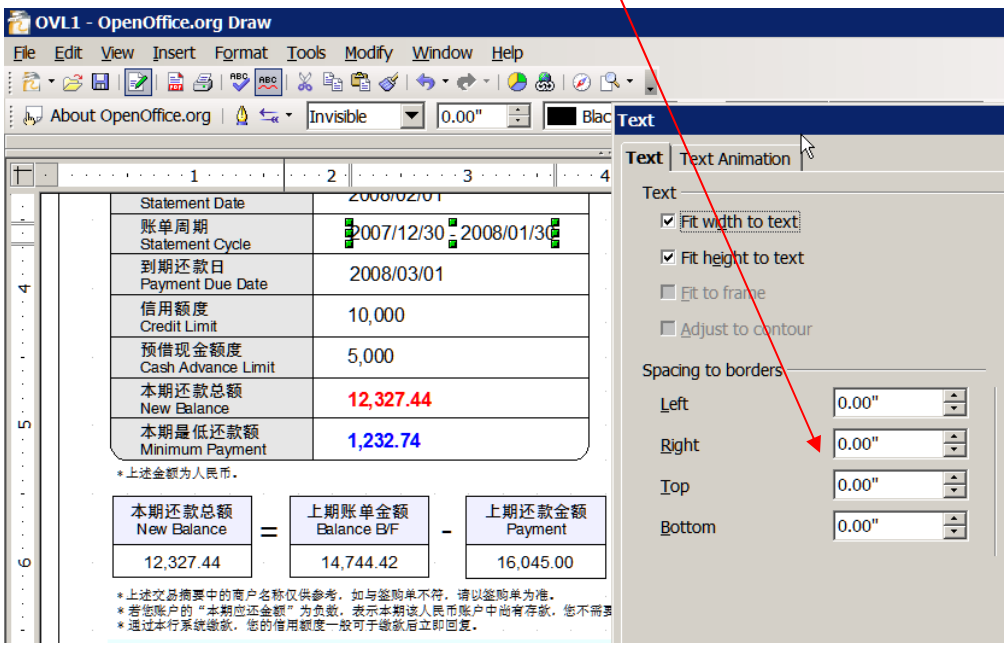
Getting Data Field Position on OpenOffice Draw

On OpenOffice/StarOffice Draw, you can get the actual text/data position to be used for the AFP data/text positioning.

With “Page Setup” settings, define page “Margins” to 0 values.

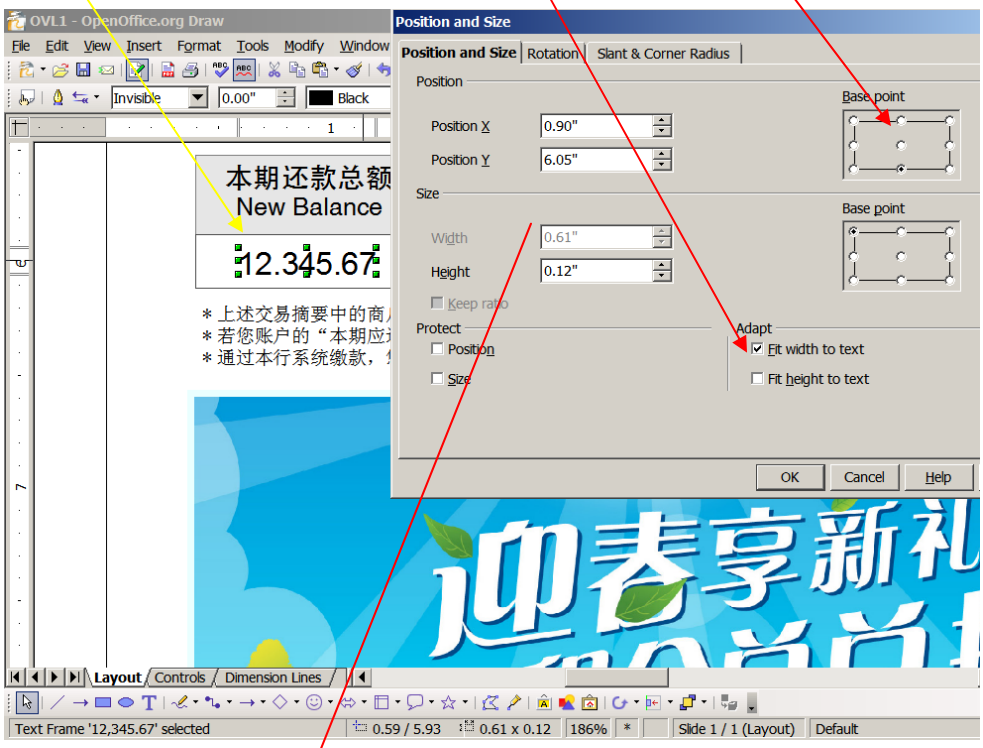


With “Text” settings, define “Spacing to borders” to 0 values.



With “Position and Size” settings, select the bottom left corner, bottom center, or bottom right corner as the “Base point” if you want to get a position for an AFP data/text that is to be left, center, or right-aligned, respectively.

You need to unclick the Adapt option “Fit Height to text” and then return the data field to adjust its baseline position manually for getting a precise baseline Y position.



MakeAFP programming:

```
OpenDoc();

SetUnit(IN_U600);

OpenPage(8.5,11);
          :
Font(1);
          :
Pos(0.9, 6.05);

Ctxt(new_balance);
          :
ClosePage();
          :
CloseDoc();
```

Working with MakeAFP Definition File

MakeAFP definition file is used to specify your resource names, TrueType/TrueType Collection/ OpenType fonts, locations of resource libraries, and whether you want to embed the resources required inline within your AFP document file or put them into a separate AFP resource file.

Once the MakeAFP definition file is defined for your program, you can call it with the command-line argument flag "-d definition_file" when you run your program in the batch mode.

Definition Sample 1:

```
restype=all,inline           // Retrieve all AFP resources and put them inline
fdeflib=c:\makeafp\reslib    // Form Definition directory
ovlylib=c:\makeafp\reslib    // Overlay directory
pseglib=c:\makeafp\reslib    // Page segment directory
fontlib=c:\makeafp\reslib    // Font directory
font1=czh200,t1000437,12.7   // Font 1, outline AFP font, point size is 12.7
font2=czh200,t1000437,11     // Font 2, by character set and code page
font3=xzn200,10              // Font 3, by coded font name
font4=c0d0gt10,t1000437      // Font 4, AFP raster font
pseg=s1yblogo                // AFP Page Segment
ovly=o1bank1                 // AFP Overlay
```

Definition Sample 2:

```
restype=fdef,ovly,pseg,objt  // Only retrieve form definitions, overlays,
                             // page segments and data-object, put them into
                             // a separate AFP resource file

reslib=c:\makeafp\reslib;d:\ipm2000\reslib   // Resource paths for overlays,
                                             // page segments, fonts and
                                             // form definitions
font1=czh200,t1000437,12
font2=czh200,t1000437,11.5
font3=xzn200,10
font4=c0d0gt10,t1000437
ovly=o1bank01
objt="orchid flower",jpeg                    // non-AFP image in JPEG
                                             // format
```

Definition Sample 3:

```
restype=none                 // Specifies that no resource file be created &
                             // no AFP resources inline within AFP document
fdeflib=c:\makeafp\reslib
ovlylib=c:\makeafp\reslib
pseglib=c:\makeafp\reslib
fontlib=c:\makeafp\reslib;c:\winnt\fonts   // Font directories of AFP fonts
                                           // and OpenType/TrueType fonts
font1=czh200,t1000437,12
font2=czh200,t1000437,11
font3=xzn200,10
font4=airialuni.ttf,T1000437,12            // Specifies using a TrueType font
                                           // "Arial Unicode MS", font size
                                           // 12, and encoding is ASCII codepage
                                           // T1000437 for USA English
```

Chapter 5. Working with Samples and Developments

Comprehensive samples in C/C++, C#, and Java are shipped along with the MakeAFP Formatter installation package, to provide you with realistic practice scenarios. MakeAFP understands that the sample code is one of the most important aspects of a toolkit for quick understanding and learning, and as such, we will try our best to include more samples within our installation package.

Learning from MakeAFP Formatter Samples

MakeAFP Formatter comprehensive samples are installed in its installation sub-path **`samples`**.

On Windows platforms, you can open the Visual Studio solution project files ***.sln** for C/C++, and C#, then press the **`F5`** key to execute the program or press the **`F10`** or **`F11`** (go into each sub-function) key to run the program in step-by-step debugging mode to familiarize yourself with MakeAFP Formatter.

On Linux platforms, you can use **`gcc`** compiler to compile all C++ sample files. You can use **`make`** command to build all sample files. Use "**`make all`**" to run all sample applications, or use "`make bank1`" to run bank1 application, for instance

Languages Bindings for Java

The usage of MakeAFP Formatter with Java is essentially the same as with C/C++.

On Windows platforms, to use MakeAFP Formatter with Java you need to add the **`c:\makeafp\include\makeafp.java`** file to your project. Samples for Java are provided in path **`c:\makeafp\samples\java\`**.

On Linux platforms, to use MakeAFP Formatter with Java you need to add the **`/usr/share/makeafp/include/makeafp.java`** file to your project. Samples for Java are provided in path **`/usr/share/makeafp/samples/java/`**.

Languages Bindings for Visual C#

The usage of MakeAFP Formatter with Visual C# is essentially the same as with C/C++.

To use MakeAFP Formatter with C#, you need to add the c:\makeafp\include\makeafp.cs file to your project, via the menu “Project” --> “Add Existing Element…”.

Chapter 6. Running MakeAFP Formatter in Batch Mode

Once you have completed the development of your custom MakeAFP Formatter program and generated the executable module in the release mode, you can run your own AFP formatting program in high performance by its command-line.

Running MakeAFP Formatter in Batch Mode

The program can run it in batch mode by using the following command syntax:

```
program_name -d definitionFile [-i inputFile] -o outputAfpFile [-l]
```

"-i input_file" is an optional parameter, for development testing or development of an overlay With MakeAFP Formatter, you may just key-in the data within your program, or you may want to control the opening of multiple input files on your own, or you may read the data from the STDIN (Standard Input) via system pipe from the STDOUT (Standard Output) of another application directly.

"-l" is an optional parameter, if you want to output the error message to STDERR (Standard Error) in DOS command-line mode, default is pop-up the error message by a message window.

Reading the Compressed or Encrypted Input Data

While you are running your MakeAFP formatting program in batch mode, you can either read raw data from an input file if "*-i input_file*" flag parameter is specified, or read the input data stream directly from the output of another application program on the fly in high speed via the system pipe.

Example 1: Reading WinZip AES 256-bit encrypted input file

Assuming your MakeAFP formatting program is stmt1.exe, and your zipped input file smt1.zip is encrypted and compressed by PKZIP AES 256-bit encryption with password *rainbow1*, you can process it directly on the fly with the MakeAFP munzip utility:

```
munzip -s -p rainbow1 stmt1.zip | stmt1 -d stmt1.def -o stmt1.afp
```

Example 2: Reading input file compressed in ZIP 2.0 standard format

Assuming your MakeAFP formatting program is stmt2.exe, and your zipped input file smt2.zip is compressed in ZIP 2.0 standard format, you can directly process it on the fly with freeware Info-ZIP utility UNZIP:

```
unzip -p stmt2.zip | stmt2 -d stmt2.def -o stmt2.afp
```

Info-ZIP at www.info-zip.org is a diverse, Internet-based workgroup to provide free, portable, high-quality versions of the Zip and UnZip compressor-archiver utilities that are compatible with ZIP 2.0 standard. Info-ZIP supports the widest system platforms with hardware from microcomputers all the way up to the IBM mainframe.

Example 3: Reading 7-ZIP formats input file

With 7-Zip freeware from www.7-zip.org, you can read compressed formats of 7Z, ZIP, CAB, RAR, ARJ, GZIP, BZIP2, TAR, CPIO, RPM, and DEB, it also supports its own AES 256 bits encryption and its LZMA compression algorithm offers ultra-high compression ratio.

Assuming your MakeAFP formatting program is stmt3.exe, and your input file smt3.7z is encrypted and compressed by 7-Zip utility with password *rainbow2*, you can read 7-Zip encrypted data directly on the fly with the command:

```
7zip e -prainbow2 -so stmt3.7z | stmt3 -d stmt3.def -o stmt3.afp
```

Example 4: Reading RAR format input file

With UNRAR freeware from www.rarlab.com, you can decompress RAR format compressed and encrypted files.

Assuming your MakeAFP formatting program is stmt4.exe, and your input file smt4.rar is encrypted and compressed by RAR shareware or WinRAR utility with password *bigsnow5*, you can read RAR encrypted data directly on the fly with the command:

```
unrar p –ierr –pbigsnow5 stmt4.rar | stmt4 -d stmt4.def -o stmt4.afp
```

MakeAFP Automation Utilities

With powerful MakeAFP automation utility AutoMakeAFP, once your data file, or compressed & encrypted input data file is received, it will automatically be dispatched to the appropriate AFP applications for subsequent processing, such as AFP formatting, AFP sorting by postal code and mail-piece, print submits, etc. Input raw data files can be either automatically deleted for data security reasons or retained into the appropriate directory after AFP formatting.

Refer to *MakeAFP Utilities User's Guide* for more details about MakeAFP utilities.

Chapter 7. Working With Form Definition

This chapter describes the form definitions that are supplied With MakeAFP Formatter, and contains some information on how to use AFP form definition with MakeAFP Formatter.

MakeAFP Supplied Form Definitions

A form definition specifies how the printer controls the processing of the physical sheets of paper, such as input paper bin switching, simplex or duplex printing, N-UP partitions, color rendering, and CMR (Color Management Resource) association for the whole print job or group pages, etc.

MakeAFP supplied AFP form definition object resources are stored in path c:\makeafp\reslib, PPFA source codes are stored in c:\makeafp\ppfa directory for your reference.

All of the form definitions supplied by MakeAFP are in across print direction with (0, 0) offset.

Form Definitions for cut-sheet printers, with multiple input paper bins (1, 2, 3, 4, 5, envelope, manual) defined:

Form Definition	Presentation	Sides	N_UP	Page Placement
F1LCD	Landscape	1, **2**	1	Default
F1LCDN2	Landscape	1, **2**	2	Default
F1LCS	Landscape	**1**, 2	1	Default
F1LCSN2	Landscape	**1**, 2	2	Default
F1LCT	Landscape	1, **Tumble**	1	Default
F1LCTN2	Landscape	1, **Tumble**	2	Default
F1PCD	Portrait	1, **2**	1	Default
F1PCDN2	Portrait	1, **2**	2	Default
F1PCDN2A	Portrait	**2**	2	Page 1 at 1 front Page 2 at 2 back Page 3 at 2 front Page 4 at 1 back
F1PCDN2B	Portrait	**2**	2	Page 1 at 1 front Page 2 at 2 front Page 3 at 1 back Page 4 at 2 back
F1PCDN2C	Portrait	**2**	2	Page 1 at 1 front Page 2 at 2 front Page 3 at 2 back Page 4 at 1 back
F1PCDN3A	Portrait	**2**	3	Page 1 at 1 front Page 2 at 2 front Page 3 at 3 front Page 4 at 1 back Page 5 at 2 back Page 6 at 3 back

F1PCDN3B	Portrait	**2**	3	Page 1 at 1 front Page 2 at 3 back Page 3 at 2 front Page 4 at 2 back Page 5 at 3 front Page 6 at 1 back
F1PCS	Portrait	**1**, 2	1	Default
F1PCSN2	Portrait	1, 2	2	Default
F1PCT	Portrait	1, Tumble	1	Default
F1PCTN2	Portrait	1, Tumble	2	Default
F1PCTN2P	Portrait	Tumble	2	Page 1 at 1 front Page 2 at 1 back Page 3 at 2 front Page 4 at 2 back

** In Sides column, bolded value is the default defined by default copy group.*

Form Definitions for continuous form (also call fan-fold) printers:

Form Definition	Presentation	Sides	N_UP	Page Placement
F1LFD	Landscape	2	1	Default
F1LFS	Landscape	1	1	Default
F1LFSN2	Landscape	1	2	Default
F1PFD	Portrait	2	1	Default
F1PFS	Portrait	1	1	Default
F1PFSN2	Portrait	1	2	Default
F1PFDN2A	Portrait	2	2	Page 1 at 1 front Page 2 at 2 back Page 3 at 2 front Page 4 at 1 back
F1PFDN2B	Portrait	2	2	Page 1 at 1 front Page 2 at 2 front Page 3 at 1 back Page 4 at 2 back
F1PFDN2C	Portrait	2	2	Page 1 at 1 front Page 2 at 2 front Page 3 at 2 back Page 4 at 1 back
F1PFDN2T	Portrait	Tumble	2	Page 1 at 1 front Page 2 at 1 back Page 3 at 2 front Page 4 at 2 back
F1PFDN3A	Portrait	2	3	Page 1 at 1 front Page 2 at 2 front Page 3 at 3 front Page 4 at 1 back Page 5 at 2 back Page 6 at 3 back
F1PFDN3B	Portrait	2	3	Page 1 at 1 front Page 2 at 3 back Page 3 at 2 front Page 4 at 2 back Page 5 at 3 front Page 6 at 1 back
F12UP180	Portrait, Rotation 180	2	2	Page 1 at 1 front Page 2 at 2 back Page 3 at 2 front Page 4 at 1 back

For concept and functions of form definition, refer to IBM *Page Printer Formatting Aid User's Guide* for more information.

How to Define a Copy Group in the MakeAFP Formatting

A single form definition contains one or more several subsets of page controls, called *copy groups*. Copy groups define each page that can be used in the printing job. For example, when we are printing some pages in duplex mode, we need a copy group that is defined using both sides of the paper.

With the parameter *FDEF=fdefname* defined in your MakeAFP Definition file, you can specify which form definition is to be used with your job; and with the MakeAFP "Copy Group" function call in your program, you can invoke a copy group previously defined within your form definition directly.

The following example illustrates how to invoke a copy group defined in a form definition directly:

FORMDEF F1LCD defined in MakeAFP definition file:

```
restype=all,inline
fdef=F1LCD                      // The FORMDEF object is compiled from
fdeflib=c:\makeafp\reslib       // the following PPFA source code
ovlylib=c:\makeafp\reslib
pseglib=c:\makeafp\reslib
fontlib=c:\makeafp\reslib
          :
```

PPFA source code for form definition F1LCD:

```
FORMDEF LCD                     // this form definition defined two
  PRESENT LANDSCAPE             // copygroups, first one for select tray
  DIRECTION ACROSS              // 1 for duplex, second one for select
  N_UP 1                        // tray 2 for simplex
  OFFSET 0 0
  REPLACE YES;

  COPYGROUP F2LCD1
    BIN 1                       // input paper bin is 1
    DUPLEX NORMAL;              // duplex printing

  COPYGROUP F2LCS2
    BIN 2                       // input paper bin is 2
    DUPLEX NO;                  // simplex printing
          :
```

MakeAFP formatting program source code:

```
void main( )
{
  Start();
  OpenDoc();
  SetUnit(MM_U600);

   OpenPage(210, 297);
   CopyGroup("F2LCS2");         // This page & subsequent pages will be
                                // printed on the paper from bin 2 in
                                // simplex mode defined by copy group F2LCS2

          :
  ClosePage();
```

```
        :
OpenPage(210, 297);
        :
ClosePage();
        :
OpenPage(210, 297);
CopyGroup("F2LCD1");      // This page & subsequent pages will be
                          // printed on the paper from bin 1 in duplex
                          // mode defined by copy group F2LCD1

        :
ClosePage();
        :
OpenPage(210, 297);
        :
ClosePage();
        :
        :
CloseDoc();
```

Appendix A. Understanding Data Format and Transferring

This chapter contains associated guidance information to help you understand the input data format and how to get input data from other systems.

Legacy Line Data Formats

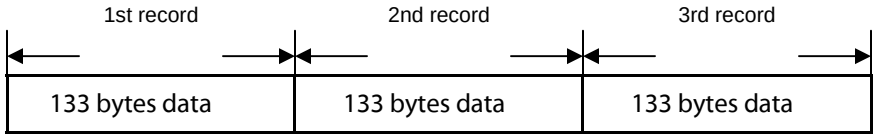
Since legacy line data is the data formatted for line printers, it generally has the following characteristics:

- Line data is text only
- Line data is stored as records or lines of data
- Line data always has some kind of carriage control commands that instruct the line printers on how to move down the forms, such as line feed and page break.

Line data is fundamentally record orientated in nature. A program reading the line data must have a way to identify the end of a record and the start of the next one. There are three types of line data formats widely used: fixed-length record, variable-length record, and delimited-record file from PC and UNIX open systems or IBM host systems.

Fixed-length Line Data Format

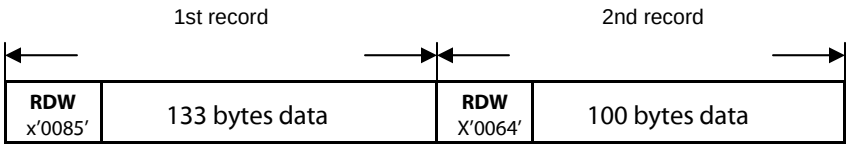
Fixed-length line data contains records that are all of the same lengths. There are no separator or prefix or self-identifying information existing that indicates the record length. This format is widely used by IBM host systems, We must know the exact record length from the mainframe programmer.



Variable-length Line Data Format

Variable-length line data uses a two bytes record length prefix (called RDW, Record Descriptor Word) in front of each record that provides the length of the record in the file. The length prefix is a 16-bit big-endian binary value. The value of the length prefix does not include the two bytes length prefix itself if the data is from IBM host systems.

Please note that Visual C++ on PC uses little-endian to store an integer value, so after you have read in RDW prefix, you must swap its two bytes from big-endian to little-endian before you can continue reading its variable-length data.



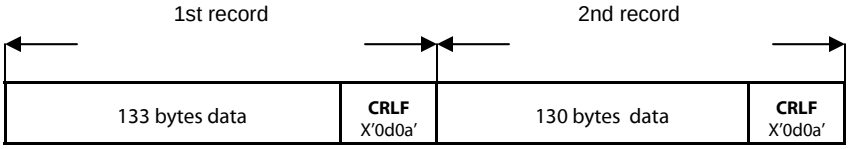
Delimited Record Data Format

The delimited record uses a separator or delimiter to indicate the end of a record.

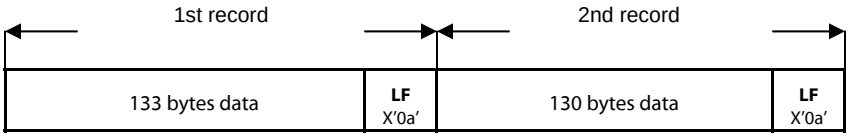
On UNIX, the new line delimiter is line-feed in hex code x'0a'; on Windows, the new line delimiter is carriage-return and line-feed in hex code x'0d0a'. The form-feed hex code is '0c'x.

If the line data uses EBCDIC encoding from IBM host systems, the default newline delimiter is x'25'.

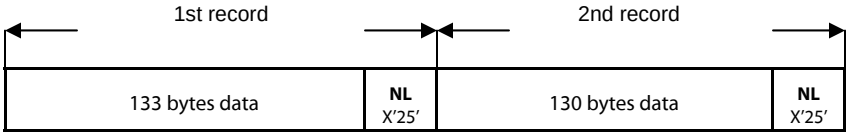
Windows/DOS format:



UNIX format:

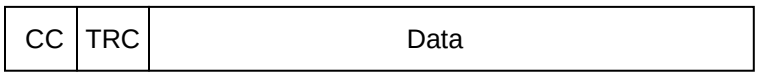


EBCDIC format:



Carriage Control Codes

Line data is the data usually formatted for printing on a line printer by IBM host systems or other computing systems. A line-data record can contain a 1-byte carriage control character and a 1-byte table reference character followed by the data to be printed. Both characters are optional and are defined as follows:



CC The carriage control character acts as a vertical tab command to position the paper at the start of a new page, at a specified line on the page, or to control the skipping to the next line

TRC Table Reference Character selects the font number ID in which the line is to be printed

The carriage control characters can be one of these two types: ANSI (American National Standards Institute) carriage control or machine carriage control.

ANSI Carriage Control Characters

The most universal carriage control is ANSI. The standard ANSI carriage control characters are:

Action	ANSI CC
Do not space the line and print	+
Single space the line and print	White-space
Float space the line and print	0
Triple space the line and print	-
Skip to Channel 1 (the top of the page)	1
Skip to Channel 2 position	2
Skip to Channel 3 position	3
Skip to Channel 4 position	4
Skip to Channel 5 position	5
Skip to Channel 6 position	6
Skip to Channel 7 position	7
Skip to Channel 8 position	8
Skip to Channel 9 position	9
Skip to Channel 10 position	A
Skip to Channel 11 position	B
Skip to Channel 12 position	C

Machine Carriage Control Codes

Machine carriage control codes were originally the actual hardware control commands for line printers, which are hex values, therefore, machine carriage control codes cannot be translated.

Typical IBM machine carriage control codes are:

Action	Action After Printing	Action (Immediate)
Print (no space)	x'01'	
Space 1 line	x'09'	x'0b'
Space 2 lines	x'11'	x'13'
Space 3 lines	x'19'	x'1b'
Skip to Channel 1	x'89'	x'8b'
Skip to Channel 2	x'91'	x'93'
Skip to Channel 3	x'99'	x'9b'
Skip to Channel 4	x'a1'	x'a3'
Skip to Channel 5	x'a9'	x'ab'
Skip to Channel 6	x'b1'	x'b3'
Skip to Channel 7	x'b9'	x'bb'
Skip to Channel 8	x'c1'	x'c3'
Skip to Channel 9	x'c9'	x'cb'

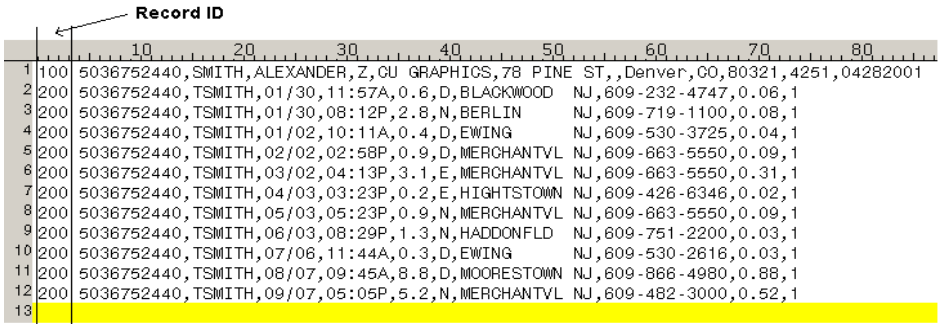
Skip to Channel 10	x'd1'	x'd3'
Skip to Channel 11	x'd9'	x'db'
Skip to Channel 12	x'e1'	x'e3'

Keyed Record Format Data

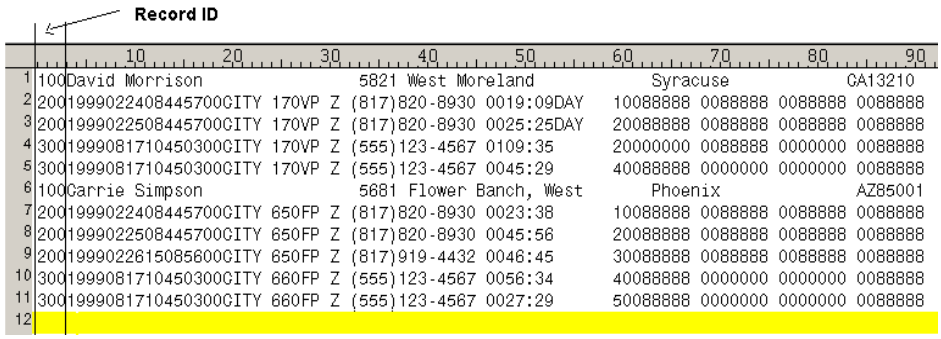
Keyed record format data is a new form of the effective raw data format used by open systems, such as CRM, SAP RDI (Raw Data Interface), Telecom billing raw data, etc. With this format, each record type must be identifiable by a unique key field, this record identification (RID) key field can appear anywhere within a record, but must be in the same position and have the same length in every record type, mostly this RID field is at the front of each data record that is used to achieve more flexible and faster data formatting capabilities, and also for connecting external text management systems to control individual requests.

The data fields that can be delimited are separated by the consistent character(s) that is(are) set aside to indicate the beginning and end of a data field. Most often, these character(s) is(are) inserted by your database software or your application.

The following picture shows the data fields separated by a delimiter character "**,**", the length of the data field can be variable.



Another type of data field can be the placeholder type, where each field starts at a fixed column position with a fixed length.



XML Data

XML data is self-describing and its data element contents can be easily extracted into the record and field structure for data formatting by programming

The following is an example of a client's information showing his "name" and his "address" fields placed together:

```
<Client>
  <name>
    <title>Dr.</title>
    <first>David</first>
    <middle>MJ</middle>
    <last>Greenpone</last>
  </name>
  <address>
    <street>2068</street>
    <street>Broadway Road</street>
    <city>Smithtown</city>
    <Country>USA</Country>
    <zip>34005</zip>
  </address>
</Client>
```

Unicode

The Unicode Standard is the universal character encoding scheme for international languages. It defines a consistent way of encoding multilingual text that enables the exchange of text data internationally and creates the foundation for global software. Unicode is implemented in all modern operating systems, and computer languages.

With Unicode, the information technology industry has replaced proliferating character sets with data stability, global interoperability, and data interchange, simplified software, and reduced development costs.

Unicode provides three popular encoding forms: an 8-bit form (UTF-8), a 16-bit form (UTF-16), and a 32-bit form (UTF-32). The 8-bit, byte-oriented form UTF-8, is designed for easy usage with existing ASCII-based systems.

Access the Internet for more information about Unicode.

Common Methods of File Transferring

There are a variety of methods to transfer the raw data files from other systems to the MakeAFP server. Methods commonly used to transfer files from other systems to MakeAFP are:

- Physical media (such as tape and CD-ROM)
- SNA file transfer software
- FTP, NFS, or LPR/LPD
- SMB Connections
- PSF Download feature for z/OS

Physical Media

Physical media is still used for data exchange across systems. Nowadays, we can easily exchange data across open systems by portable hard-disk, CD-ROM, and DVD; in some situations, we may still need to use tape media to get data from IBM mainframes and other UNIX systems.

A variety of software providers offer such tape utilities for Windows, with which you can read IBM mainframe format tape, as well as UNIX tar format tape. NovaXchange for Windows from NovaStor Corporation is one of them.

SNA File Transferring

You may transfer files from IBM mainframes to the MakeAFP server by using the IBM 3270 PC file transfer program (IND$FILE).

The following parameters are recommended:

- For the files with fixed-length records that contain only printable characters and either ANSI carriage control characters, or no carriage control characters, both TEXT or BINARY transferring can be used.
- For the files with variable-length records that contain only printable characters and either ANSI carriage control characters, or no carriage control characters, specify file transfer options ASCII and CRLF in your PC 3270 emulation software.
- For the files with machine carriage control codes, specify file transfer options BINARY and CRLF. The received data will be EBCDIC encoding with IBM machine carriage control codes and records separated by ASCII CRLF (x'0d0a').

Several software providers offer SNA RJE 3277 emulation software for Windows, with which you can get print line data files from IBM spool queues directly, such as POWER for VSE; JES2 or JES3 for z/OS. This solution works well for low-volume line data with less critical availability requirements.

FTP or NFS File Sharing

FTP is a high-performance TCP/IP file transfer function that has been implemented on many platforms, with FTP you can transfer the files across the systems by text or binary mode.

IBM FTP on z/OS enables you to transfer data sets between your z/OS and any server that supports TCP/IP.

Following is a sample JCL that shows usage of FTP for z/OS in a batch job, to transfer the data from a data set with EBCDIC to ASCII conversion into a PC/Unix server:

```
//STEPFTP  EXEC PGM=FTP,REGION=4M
//OUTPUT     DD SYSOUT=*
//INPUT      DD *
10.209.50.179
userid68
password68
ebcdic
```

```
put 'CPCPACT.CC1P121.TEST68'
quit
/*
```

For more information on IBM FTP for z/OS, refer to IBM *z/OS IP User's Guide*.

The latest FTP software supports both encryption and compression on the fly during file transferring, Serv-U FTP server and FTP Voyager client software are strongly recommended for FTP automation and float data protection even if your data has been encrypted by AES 256-bit.

For general-purpose usage, Network File System offers a flexible option for the exchange of data between like and unlike platforms that support the NFS protocol. NFS is the de-facto standard system for sharing directories and files over Internet Protocol (IP) networks. The shared data sets and disks appear to each remote host system as just another local dataset or disk.

Generally, NFS-mounted files are not translated. However, NFS for IBM mainframe may include a two-byte binary record length prefix (RDW) for variable-length records data.

Microsoft SFU (Service for Unix) 3.5 improved the overall performance of the Server for NFS, both by enhancing the performance of individual servers and by making the server for NFS cluster-aware. It is free and can be downloaded from the Microsoft SFU home page http://www.microsoft.com/windows/sfu/downloads/default.asp.

LPR/LPD File Transferring

LPR/LPD is widely used in open systems for printing via TCP/IP network, with the MakeAFP LPD server you can get data sending from remote spool systems, such as spool files from Unix and PC, JES2/JES3 spool files from z/OS via IP PrintWay or NPF for z/OS, POWER spool files from VSE via LPR, or SCS spool files from OS/400.

SMB Connections via NETBIOS over TCP/IP

Samba is a popular open-source/free software suite that provides seamless file and print services to SMB/CIFS clients, with Samba we can get files from open systems and IBM z/OS quickly.

IBM Fast Connect for AIX also provides fast seamless file sharing with Windows.

PSF Download for z/OS

If you have the IBM Download feature of PSF installed on z/OS, and InfoPrint Manager or IBM Content Manager OnDemand for Windows installed on the MakeAFP server, you can get z/OS JES spool files directly.

MVS Download provides high-speed data transfer for high-volume jobs and supports restart/resume from a checkpoint if the data transmission is interrupted.

Hints and Tips for Improve Performance

Although MakeAFP formatting program and utilities support reading compressed raw data and writing compressed AFP directly on the fly, with amazing formatting speed, tremendously reducing the disk I/O, printing job submission, and file transferring time, you still can do some tuning to improve its performance furthermore.

With mission-critical high-speed AFP formatting and printing environment, huge data I/O can be a constraining point that leads to little disk clutching. It is recommended that both MakeAFP and InfoPrint Manager are installed on the PC server that is configured with Dual Channel UltraSCSI controller with multiple UltraSCSI320 15K RPM hard disks, and use separate hard disks for storing input raw data files, output AFP files, and spooling of InfoPrint Manager, to maximize I/O throughput by avoiding read/write contention, by this method your AFP formatting and data transmission performance and speed will be tremendously improved.

If you need a RAID controller for your PC server, it is recommended that you check with your service provider for the best RAID configuration. RAID 1+0 (sometimes called RAID 10) seems to be popular, with good performance and cost compromises.

If your input raw data files are stored on another PC or Unix server, it is recommended that you use 1 Gb Ethernet LAN or 1 Gb dedicated cross-wired Ethernet cable with a direct link, it generally provides Ultra-fast data transmission.