MakeAFP

MakeAFP Weaver Reference

Version 3.7

This edition applies to the MakeAFP Weaver.

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. MakeAFP Weaver Functions 1

Using the MakeAFP Weaver Function Calls to Build an AFP Document 1

Hierarchy of MakeAFP Weaver Calls 1

Session Level calls 2

Document Level Calls 2

Page-level Calls 3

Format of the Function Call Descriptions 4

2D Aztec Barcode by Drawing 5

2D DataMatrix Barcode by Drawing 7

2D MaxiCode Barcode by Drawing 9

2D MicroPDF417 Barcode by Drawing 11

2D PDF417 Barcode by Drawing 13

2D PDF417 Truncated Barcode by Drawing 15

2D QR Code Barcode by Drawing 17

Barcode (Linear) by Drawing 19

BCOCA Barcode (Linear) 22

BCOCA MaxiCode 2D Barcode 28

BCOCA PDF417 2D Barcode 31

BCOCA QR Code 2D Barcode 34

Begin of Index Name Group of Input AFP (Checking) 37

Begin Index Group 38

Begin Overstrike 41

Begin Underscore 42

Blank Page (Checking) 43

Box Drawing 44

Center Align 1-Byte Text 45

Center Align Japanese 46

Center Align Korean 47

Center Align Simplified Chinese 48

Center Align Traditional Chinese 49

Center Align SBCS-HOST/DBCS-HOST 50

Center Align UTF-16 Text 51

Center Align UTF-16 Text Converting from Legacy String 52

Center Align UTF-8 Text 53

Center Align UTF-8 Text Converting from Legacy String 54
Center Align UTF-8 Text Converting from UTF-16LE 55
Centimeter Value 56
Close Document 57
Close Page 58
Color for Text 59
Color Management Resource Association 60
Copy Group 62
Default Language Locale 64
Encoding of AFP and PC Native 65
End Index Group 67
End of Document File of Input AFP (Checking) 68
End of Index Name Group of Input AFP (Checking) 69
End Overstrike 70
End Underscore 71
Font Definition 72
Font ID Query 74
Get Index Value 75
Get No-Operation Value 77
Get Page – Getting an Existing AFP Page 78
Get Text Field by a Location 80
Get Text Field by a Location Area 83
Get Text Field by a Location Area and a Pettern 86
Get Text Field by a X-location Range and a Pettern 89
Get Text Field by a Y-location Range and a Pattern 91
Get Text Field Position 93
Goto Page 95
Horizontal Line 96
Horizontal Lines 97
Inch Value 98
Include Data-Object 99
Include Overlay 103
Include Page Segment 104
Left Align ASCII / EBCDIC Text 105
Left Align Japanese 106
Left Align Korean 107
Left Align Simplified Chinese 108
Left Align Traditional Chinese 109
Left Align SBCS-HOST/DBCS-HOST 110
Left Align UTF-16 Text 111
Left Align UTF-16 Text Converting from Legacy String 112
Left Align UTF-8 Text 113

Left Align UTF-16 Text Converting from Legacy String 114
Left Align UTF-8 Text Converting from UTF-16LE 115
Lines Per Inch 116
Line Spacing 117
Margin of Inline Text 118
Mask an Area 119
Mask a Text Field 120
Mask Text Field by a Location Area 121
Mask Text Field by a Location Area and a Pattern 122
Mask Text Field by an X-location Range and a Pattern 124
Mask Text Field by a Y-location Range and a Pattern 126
Maximum Pagination 128
Millimeter Value 129
Move AFP Page between AFP Page Buffers 130
Next Line 132
Open Document 133
Opening Page – Adding New Page 134
Paragraph of 1-Byte Text 136
Paragraph of Japanese 138
Paragraph of Korean 140
Paragraph of Simplified Chinese 142
Paragraph of Traditional Chinese 144
Paragraph of UTF-16 Text 146
Paragraph of UTF-16 Text Converting from Legacy String 148
Paragraph of UTF-8 Text 150
Paragraph of UTF-8 Text Converting from Legacy String 152
Paragraph of UTF-8 Text Converting from UTF-16LE 154
Point Value 156
Position of Text 157
Print AFP File 158
Put Index Tag 160
Rendering Color Intent 163
Right Align 1-Byte Text 165
Right Align Japanese 166
Right Align Korean 167
Right Align Simplified Chinese 168
Right Align Traditional Chinese 169
Right Align SBCS-HOST/DBCS-HOST 170
Right Align UTF-16 Text 171
Right Align UTF-16 Text Converting from Legacy String 172
Right Align UTF-8 Text 173
Right Align UTF-8 Text Converting from Legacy String 174

Right Align UTF-8 Text Converting from UTF-16LE 175
Set Default Unit 176
Skip Lines 177
Start Session 178
Text Orientation 179
Trigger by a Location and a Pettern 180
Trigger by a location arear and a Pattern 183
Trigger by an X-location Ranger and a Pattern 186
Trigger by a Y-location Ranger and a Pettern 187
Trigger by Name of Copy Group 188
Trigger by Name of Data-Object Image 189
Trigger by Name of Overlay 190
Trigger by Name of Page Segment 193
Vertical Line 194
Vertical Lines 195
View AFP File 196
X Absolute Position 198
X Current Position (Query) 199
X Move Relative Position 200
Y Absolute Position 201
Y Current Position (Query) 202
Y Move Relative Position 203

Chapter 2. MakeAFP Weaver Parameters 204

Conventions Used in This Chapter 204
CMR – Specifies a Colo Management Resource 205
CPGID – Specifies a Code Page Identifier 205
FDEF – Sepcifies a Form Fefintion 206
FDEFLIB – Specifies the Library Path of Form Defintions 206
FONT – Specifies an AFP FOCA Raster or Outline Font 207
FONT – Specifies an OpenType/TrueType Font 208
FONTLIB – Specifies the Library Path of Fonts 209
INDEXOBJ – Specifies Generating of the AFP Index Object File 210
OBJT – Specifies an AFP Object or non-AFP Object 210
OBJTLIB – Specifies the Library Path of Image and CMR Objects 211
OVLY – Specifies an Overlay 212
OVLYLIB – Specifies the Library Path of Overlays 212
PRMODE – Specifies the Type of Input Data and Processing Option 213
PSEG – Specifies a Page Segment 213
PSEGLIB – Specifies the Library Path of Page Segments 214
RESLIB – Specifies the Library Path of AFP Resources and Objects 215
RESTYPE - Specifies the Types of Resources to be Retrieved 215

Chapter 3. MakeAFP Weaver Variables 217

$Bng – Begin of AFP Page Group Index 217

$Eng – End of AFP Page Group Index 217

$Edt – End of AFP Document 217

$MaxPaging – Maximum Number of AFP Page Buffers 217

$Page – Current AFP Page Buffer Number 218

Chapter 4. String Manipulation Functions 219

Comma Float 219

Comma Integer 220

Comma Digital String 220

Delete Characters 221

Extract Substrings 221

Find String 223

First Character 223

In Substitution Table 224

Insert String 224

Is Empty 225

Last Character 225

Left Copy 226

Left Copy and Pad 226

Left Copy and Right Trim 227

Left Trim 227

Left Trim for EBCDIC 228

Left Trim and Concatenate Strings 228

Match String Comparing 229

Pattern Searching 230

Remove String 230

Replace String 231

Reverse Find String 232

Right Copy 233

Right Copy and Left Trim 233

Right Copy and Pad 234

Right Copy and Right Trim 234

Right Trim 235

Right Trim and Concatenate Strings 235

Right Trim for EBCDIC 236

Strings Concatenate 236

String Pad 237

Substitute String 237

Substitute Change 238

Substring 238

Substring Left Trim 239
Substring Pad 240
Substring Right Trim 241
Substring Trim Both Sides 242
System Date and Time 242
Title String 243
Thai Compose 244
Translate Digits to Simplified Chinese Figures 244
Translate Digits to Traditional Chinese Figures 245
Trim Both Sides 245
Trim Both Sides for EBCDIC 246
Trim and Concatenate Strings 246

Chapter 5. Conversion Functions 247

Codepage/Charset TO UTF-16 Conversion 247
Codepage/Charset TO UTF-8 Conversion 247
Default Encoding Names 248
Universal Conversion 248
UTF-16 to Codepage/Charset Conversion 249
UTF-16 to UTF-8 Conversion 250
UTF-32 to UTF-16 Conversion 250
UTF-32 to UTF-8 Conversion 251
UTF-8 to UTF-16 Conversion 251
UTF-8 to Codepage/Charset Conversion 252
Vietnamese Codepage/Charset Codepage/Charset Conversion 253

Appendix A. ASCII/EBCDIC AFP Code Pages and CPGID Summary 254

Appendix B. SBCS/DBCS AFP Code Pages and CPGID Summary 258

Appendix C. Combined SBCS/DBCS CPGID Summary 259

Appendix D. Encoding Names and Alias 260

Appendix E. How to Specify a Locale 269

Appendix F. Visual Studio C++ Run-Time Routines 276

Buffer-Manipulation Routines 276
Character-Classification Routines 276
Console and Port I/O Routines 277
Data-Conversion Routines 278
Directory-Control Routines 279
File-Handling Routines (File Descriptor) 279
File-Handling Routines (Path or Filename) 279
File-Handling Routines (Open File) 280
Low-Level I/O Functions 280

Stream I/O Routines 281
String-Manipulation Routines 282
Time Routines 284
Memory and Other Routines....................285

Chapter 1. MakeAFP Weaver Functions

This chapter describes the functions for AFP provided by MakeAFP Weaver. To use these functions, you must obey certain structural rules which are very easy to understand, otherwise, MakeAFP Weaver reports an error message if a problem is detected during your development in debug or execute mode of MS Visual Studio C++.

Using the MakeAFP Weaver Function Calls to Build an AFP Document

A typical sequence of MakeAFP Weaver function calls for an AFP application is as follows:

1. Initialize a MakeAFP Weaver environment by calling the “Start” function. The “Start” session function must be called before using any other MakeAFP Weaver functions.
2. Call the “Open Document” function.
3. Set default measurement units for the whole job by calling the “Set Unit” function.
4. Call the “Begin Page Group Index” function if needed for adding AFP indexes.
5. Put page group level index tags if needed for adding AFP indexes.
6. Call the “Open Page” or “Get Page” function by either adding a new AFP page or reading an existing AFP page.
7. Specify trigger to determine where to locate a text or page.
8. Get the text string by its location, to be used for reengineering purposes, like to create AFP indexes or dynamic new barcodes.
9. Specify the position where to put the next data on the page by calling functions of set X and Y absolute or relative positions, left margin, next line, and skip lines.
10. If needed, specify any changes to the attributes of the data by calling the “Set Font” and “Set Color” functions before placing the data.
11. Make enhancements to an AFP page using the appropriate function call:
    - Add a line of ASCII/EBCDIC/DBCS/Unicode text string in left, right, or center alignment.
    - Add a text paragraph in left, right, center, and fully justify alignment.
    - Add line or box at a fixed or dynamic position.
    - Add 1D and 2D barcode at a fixed or dynamic position.
    - Include AFP Object, such as overlay or Page segment at a fixed or dynamic position.
    - Include external non-AFP Objects, such as JPEG/TIFF/GIF images at a fixed or dynamic position.
12. Repeat some of the above steps until the AFP page is processed.
13. Call the “Close Page” function.
14. Call the “End Page Group Index” if needed for adding AFP indexes.
15. Repeat steps 4 through 15 until all pages are done.
16. Call the “Close Document” function.

Hierarchy of MakeAFP Weaver Calls

MakeAFP Weaver has three levels of function calls:

- **Session Level Calls**
  These calls start the MakeAFP Weaver session, set overall session measurement units, and are issued only once for each program. Print and view the AFP file if needed.
- **Document Level Calls**
  These calls open and close an AFP document and place data (such as page group level indexes) at the document level.
- **Page-Level Calls**
  These calls open and close an AFP page, and format data within individual pages.

Session Level calls

Set Default Measurement Units
Defines the default measurement units for the whole job.

Set Maximum Page Buffers for Pagination
Defines the maximum buffers for keeping the AFP data stream in the AFP page buffers.

Start Session
Starts the MakeAFP Weaver session.

Print AFP File
Submits the generated AFP file to AFP/IPDS Print Server, which must be specified after the “Close Document” request.

View AFP File
Views the generated AFP file, must be specified after the “Close Document” request.

Document Level Calls

Open Document
Opens an AFP document.

Begin Page Group Index
Begins a page group level index.

Put Page Group Level Index Tag
Put an indexing tag in the document for use by AFP archiving systems, AFP utilities or postprocessor applications.

Close Page Group Index
Ends a page group level index.

Invoke Copy-Group
Invokes an AFP copy-group.

Associating Color Management Resource
Associates CMR (Color Management resource) with the subsequent pages.

Associating Color Management Resource and Color Rendering Intent
Defines color rendering intent for the subsequent pages

Close Document
Closes an AFP document.

Page-level Calls

Open or Get Page
Opens an AFP page by either read an existing AFP page or create a new AFP page.

Draw Box
Draws a fixed size box from the current position.

Set Color
Sets a color for the subsequent data or graphic.

Put SBCS / DBCS / Unicode Text
Puts a line of SBCS / DBCS / Unicode text.

Put Fixed Paragraph
Puts a boxed text paragraph.

Draw Horizontal Line
Draws a fixed-length horizontal line.

Draw Vertical Line
Draws a fixed-length vertical line.

Set Horizontal X Position
Specifies a horizontal X position.

Horizontal Move
Moves horizontally relative to the current X coordinate position.

Set Vertical Y Position
Specifies a vertical Y position.

Query Position
Queries current X or Y position.

Vertical Move
Moves vertically, relative to the Y current position.

Next Line or Skip Lines
Advances one or more line(s) from the current position.

Set Font
Specifies the font for subsequent legacy ASCII, EBCDIC, Unicode data.

Set Fonts for Mixed SBCS/DBCS Text
Specifies a pair of SBCS and DBCS fonts for subsequent SBCS / DBCS text data.

Set Text Orientation
Sets text orientation for the subsequent text.

Begin Underscore
Begins an underscore.

End Underscore
Ends an underscore.

Put Linear Barcode Data
Puts a linear barcode data.

Put 2D Barcode Data
Puts a 2D barcode data.

Mask Area

Hides a rectangle area.

Get Index Value

Gets index tag value's strings that can be used to create the string of barcode.

Set Trigger

Specifies trigger to determine where to locate a text string or page.

Get Data Field

Gets a text string by its location, to be used to create AFP indexes or string of barcode.

Close Page

Closes the page.

Format of the Function Call Descriptions

The function descriptions are listed in alphabetic order. Each function calls description includes the following sections:

Function

A description of the major purpose of the function.

Syntax

A diagram showing the function parameters.

Parameters

Explanation of each parameter.

Function Call Samples

Provides samples for using the function. All sample functions assume that prerequisite calls and variable definitions have been made before the sample function call.

Default Values

In C++, you may assign a default value to a function's parameter, which will be used automatically if no corresponding argument is specified when the function is called. The default value is specified in a manner syntactically similar to a variable initialization.

A default argument is specified by providing an explicit initializer for the parameter in the parameter list. We may define defaults for one or more parameters. However, if a parameter has a default argument, all the parameters that follow it also must have default arguments, In other words, you cannot omit a middle parameter.

MakeAFP provides default values to rarely used parameters to simplify the use of the MakeAFP Weaver function.

2D Aztec Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest AFP BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

This function generates Aztec 2D barcode in small size of AFP IOCA image data stream which is device resolution-independent. Compares with the 2D barcode by AFP font used by some AFP software, this solution offers much better advantages to the e-output.

Syntax

```
void Aztec(
          char*       data,
          float       x_pos,
          float       y_pos,
          ushort      symbol_size = 0,
          ushort      security_mode = 0,
          float       scale = 1.0,
          degree      degree = DEG0,
          ocaColor    color = BLACK
         );
```

Parameters

data
The null-terminated extended ASCII character data up to a maximum length of approximately 3823 numeric or 3067 alphabetic characters or 1914 bytes of data.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

symbol_size
The size of the symbol can be specified a value between 1 and 36. The default value is 0, the symbol size is chosen automatically according to the number of input characters and security level.

security
The desired security level for the symbol, the valid value is 1 through 4. The higher the security level, the more error correction will be added to the symbol, the use default value is recommended, the symbol will be produced with the default amount of error correction.

scale
Specifies the scale to adjust 2D barcode image size, default value is 1.0, the maximum value allowed is 5.0.

degree
The rotation of 2D barcode image. The valid values are:

`DEG0`	The barcode image is not rotated.
`DEG90`	The barcode image is rotated 90 degrees clockwise
`DEG180`	The barcode image is rotated 180 degrees clockwise
`DEG270`	The barcode image is rotated 270 degrees clockwise

ocaColor
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample:

```
char *data = "1234567890 this is testing of Aztec";
SetUnit(IN_U300);
OpenDoc();
  OpenPage(8.5,11);
    Aztec(data,1.2,1.5);

  ClosePage();
CloseDoc();
```

Print / display:

2D DataMatrix Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest IBM BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

This function generates DataMatrix 2D barcode in small size of AFP IOCA image data stream which is device resolution-independent. Compares with the 2D barcode by AFP font used by some AFP software, this solution offers much better advantages to the e-output.

Syntax

```
void DataMatrix(
               char*        data,
               float        x_pos,
               float        y_pos,
               ushort       symbol_size = 0,
               ushort       security_mode = 0,
               float        scale = 1.0,
               degree       degree = DEG0,
               ocaColor     color = BLACK,
              );
```

Parameters

data
The null-terminated ASCII string up to 780 characters. Symbol size is determined by the length of the input data and error correction auto-added.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

symbol_size
The size of the symbol can be specified as a value between 1 and 15. The default value is 0, the symbol size is chosen automatically according to the number of input characters and security level.

Security_mode
The desired security level for the symbol, the valid value is 1 through 6. The higher the security level, the more error correction will be added to the symbol, the use default value is recommended, the symbol will be produced with the default amount of error correction.

scale
Specifies the scale to adjust 2D barcode image size, default value is 1.0, the maximum value allowed is 5.0.

degree
The rotation of 2D barcode image. The valid values are:

`DEG0` The barcode image is not rotated.
`DEG90` The barcode image is rotated 90 degrees clockwise
`DEG180` The barcode image is rotated 180 degrees clockwise
`DEG270` The barcode image is rotated 270 degrees clockwise

ocaColor
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample:

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

  ClosePage();
CloseDoc();
```

Print / display:

2D MaxiCode Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest IBM BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

This function generates MaxiCode 2D barcode in small size of AFP IOCA image data stream which is device resolution-independent. Compares with the 2D barcode by AFP font used by some AFP software, this solution offers much better advantages to the e-output.

Syntax

```
void MaxiCode(
               char*        data,
               float        x_pos,
               float        y_pos,
               mode         symbol_mode = 4,
               char*        postal_data = NULL,
               degree       degree = DEG0,
               ocaColor     color = BLACK,
             );
```

Parameters

data
The null-terminated ASCII string up to 93 upper letters or up to 135 digits.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

symbol_mode
Symbol mode, Valid mode values are:

Mode		Maximum Data Length for Capital Letters	Maximum Data Length for Numeric Digits	Number of Error Correction Codewords
2	Structured Carrier Message for additional numeric postal code	84	126	50
3	Structured Carrier Message for additional alphanumeric postal code	84	126	50
4	Standard symbol (default value) for numeric and alphanumeric character sequences (includes Standard Error Correction)	93	135	50
5	Full ECC, like MODE4 but with Enhanced Error Correction	77	110	66
6	Reserved for the maintenance of scanner hardware	93	135	50

postal_data
Structured postal data can be composed by Mode 2 or Mode 3, it consists of a structured data field which includes various data about the package being sent, the format is given in the following table:

Characters	Meaning
1-9	Postcode data can consist of up to 9 digits (for mode 2) or up to 6 alphanumeric characters (for mode 3). The remaining unused characters should be filled with the BLANK character (ASCII '20'x)
10-12	Three-digit country code according to ISO 3166
13-15	Three-digit service code. This depends on your parcel courier.

degree
The rotation of 2D barcode image. The valid values are:

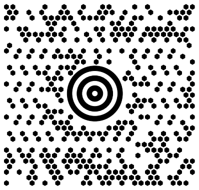
`DEG0`	The barcode image is not rotated.
`DEG90`	The barcode image is rotated 90 degrees clockwise.
`DEG180`	The barcode image is rotated 180 degrees clockwise.
`DEG270`	The barcode image is rotated 270 degrees clockwise.

color
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample

```
char *data = "1234567890 This is testing of MaxiCode";

SetUnit(IN_U1440);
OpenDoc();
OpenPage(8.5,11);
          :
          :
MaxiCode(data,           // Barcode data
          1,             // Barcode x position to 1"
          1,             // Barcode Y position to 1"

          :
          :

ClosePage();
CloseDoc();
```

Print / display:



2D MicroPDF417 Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest IBM BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

This function generates MicroPDF417 2D barcode in small size of AFP IOCA image data stream which is device resolution-independent. Compares with the 2D barcode by AFP font used by some AFP software, this solution offers much better advantages to the e-output.

Syntax

```
void MPDF417(
             char*          data,
             float          x_pos,
             float          y_pos,
             float          width = 0,
             float          scale = 1.0,
             degree         degree = DEG0,
             ocaColor       color = BLACK
            );
```

Parameters

data
The null-terminated ASCII string up to 250 alphanumeric characters or 366 digits.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

width
The columns of MicroPDF417 symbols, valid values are 1 through 4. 34 pre-defined symbol sizes are available with 1 - 4 columns and 4 - 44 rows.

The default value is 0, the symbol size is chosen automatically according to the number of input characters and security level.

scale
Specifies the scale to adjust 2D barcode image size, default value is 1.0, the maximum value allowed is 5.0.

degree
The rotation of 2D barcode image. The valid values are:

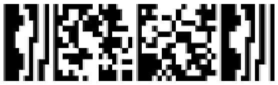
`DEG0`	The barcode image is not rotated.
`DEG90`	The barcode image is rotated 90 degrees clockwise.
`DEG180`	The barcode image is rotated 180 degrees clockwise.
`DEG270`	The barcode image is rotated 270 degrees clockwise.

ocaColor
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample:

```
char *data = "12345566787 MicroPDF417";
SetUnit(IN_U600);
OpenDoc();
  OpenPage(8.5,11);
    MDF417(data,1.2,1.5);

  ClosePage();
CloseDoc();
```

Print / display:



2D PDF417 Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest IBM BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

This function generates PDF417 2D barcode in small size of AFP IOCA image data stream which is device resolution-independent. Compares with the 2D barcode by AFP font used by some AFP software, this solution offers much better advantages to the e-output.

Syntax

```
void PDF417(
            char*         data,
            float         x_pos,
            float         y_pos,
            float         width = 0,
            ushort        security = 0,
            float         scale = 1.0,
            degree        degree = DEG0,
            ocaColor      color = BLACK
           );
```

Parameters

data
The null-terminated ASCII string up to 1850 characters or 2710 digits.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

width
The columns of PDF417 symbols, valid values are 1 through 30. The default value is 0, the symbol size is chosen automatically according to the number of input characters and security level.

security
The desired security level for the symbol is an integer from 0 (only error recognition) to 8 (highest). The higher the security level, the more error correction codewords will be added to the symbol. The default value is 0, the security level is chosen automatically according to the number of input characters.

scale
Specifies the scale to adjust 2D barcode image size, default value is 1.0, the maximum value allowed is 5.0.

degree
The rotation of 2D barcode image. The valid values are:

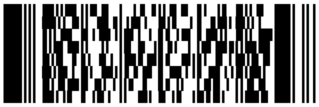
`DEG0`	The barcode image is not rotated.
`DEG90`	The barcode image is rotated 90 degrees clockwise.
`DEG180`	The barcode image is rotated 180 degrees clockwise.
`DEG270`	The barcode image is rotated 270 degrees clockwise.

ocaColor
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample:

```
char *data = "1234567890 this is testing of PDF417";
SetUnit(IN_U600);
OpenDoc();
  OpenPage(8.5,11);
    PDF417(data,1.2,1.5);   // position to (1.2",15")
  ClosePage();
CloseDoc();
```

Print / display:



2D PDF417 Truncated Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest IBM BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

This function generates PDF417 Truncated 2D barcode in small size of AFP IOCA image data stream which is device resolution-independent. Compares with the 2D barcode by AFP font used by some AFP software, this solution offers much better advantages to the e-output.

Syntax

```
void PDF417T(
             char*        data,
             float        x_pos,
             float        y_pos,
             float        width = 0,
             ushort       security = 0,
             float        scale = 1.0,
             degree       degree = DEG0,
             ocaColor     color = BLACK
            );
```

Parameters

data
The null-terminated ASCII string up to 1850 characters or 2710 digits.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

width
The columns of the PDF417 symbol, valid values are 1 through 30. The default value is 0, the symbol size is chosen automatically according to the number of input characters and security level.

security
The desired security level for the symbol is an integer from 0 (only error recognition) to 8 (highest). The higher the security level, the more error correction codewords will be added to the symbol. The default value is 0, the security level is chosen automatically according to the number of input characters.

scale
Specifies the scale to adjust 2D barcode image size, default value is 1.0, the maximum value allowed is 5.0.

degree
The rotation of 2D barcode image. The valid values are:

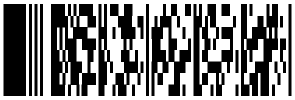
`DEG0`	The barcode image is not rotated.
`DEG90`	The barcode image is rotated 90 degrees clockwise
`DEG180`	The barcode image is rotated 180 degrees clockwise
`DEG270`	The barcode image is rotated 270 degrees clockwise

ocaColor
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample:

```
char *data = "1234567890 this is testing of PDF417 Truncated";
SetUnit(IN_U600);
OpenDoc();
  OpenPage(8.5,11);
    PDF417(data,1.2,1.5);       // position to (1.2",15")
  ClosePage();
CloseDoc();
```

Print / display:



2D QR Code Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest IBM BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

This function supports barcode data in Chinese, Japanese and Korean also, it generates the QR Code 2D barcode in small size of the AFP IOCA image data stream which is device resolution-independent. Compares with the 2D barcode by AFP font used by some AFP software, this solution offers much better advantages to the e-output.

Syntax

```
void  QRCode(
             char*        data,
             float        x_pos,
             float        y_pos,
             float        symbol_size = 0,
             ushort       security = 0,
             float        scale = 1.0,
             encoding     encode = TOUTF8,
             degree       degree = DEG0,
             ocaColor     color = BLACK,
            );
```

Micro QR Code for short messages:

```
void  MQRCode(
               char*       data,
               float       x_pos,
               float       y_pos,
               float       symbol_size = 0,
               ushort      security = 0,
               float       scale = 1.0,
               encoding    encode = TOUTF8,
               degree      degree = DEG0,
               ocaColor    color = BLACK,
            );
```

Parameters

data
The null-terminated ASCII string up to 7089 numeric digits, 4296 alphanumeric characters or mixed 2953 bytes of data.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

symbol_size
The size of the symbol, valid values are 1 through 40. The default value is 0, the symbol size is chosen automatically according to the number of input characters and security level.

security
Error Correction Level. It specifies the level of error correction to be used for the symbol. Valid values are 1 through 4. The default value is 0, the security level is chosen automatically according to the number of input characters.

scale
Specifies the scale to adjust 2D barcode image size, default value is 1.0, the maximum value allowed is 5.0.

encode
The encoding of the input data, the valid values are:

`TOUTF8`	Converts legacy encoding data to UTF-8 by MakeAFP Weaver, make sure that the default input data encoding is defined properly by the function of DefaultCode( ) first, otherwise the default input data encoding "Windows-1252" is being used for the internal data encoding conversion.
`UTF8`	Input data is in UTF-8 encoding

degree
The rotation for the barcode. The valid values are:

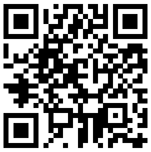
`DEG0`	The barcode is not rotated.
`DEG90`	The barcode is rotated 90 degrees clockwise
`DEG180`	The barcode is rotated 180 degrees clockwise
`DEG270`	The barcode is rotated 270 degrees clockwise

color
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY, Default is BLACK color.

Sample:

```
char *data = "1234567890 this is testing of QR Code";
SetUnit(IN_U600);
OpenDoc();
  OpenPage(8.5,11);
    QRCode(data,1.2,1.5);        // position to (1.2",15")
  ClosePage();
CloseDoc();
```

Print / display:



Barcode (Linear) by Drawing

Although MakeAFP Weaver supports the 1D barcodes defined in the latest IBM BCOCA standard for ultra-fast speed formatting with a very small AFP BCOCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCA object into other formats, like PDF, HTML, and XML, the same BCOCA objects may be printed in different dimensions on different vendors' IPDS printers.

The absolute best way to create bar codes is to use the vector drawing which is device independent, a vector barcode drawing contains a sequence of drawing instructions that describe how to render the bars. Over 50 types of popular linear barcodes are supported by MakeAFP vector drawing, which generates a small size of AFP data stream, and can be presented on any type of printer or presentation system with full fidelity and high print/display quality.

Barcode drawing function does not control the presentation of HRI (human-readable interpretation) characters, but it returns the text string of HRI with auto-calculated check-digits if required, to allow you to take full control of the HRI presentation, such as text position, font style, character size, and text orientation.

Syntax

```
Char* BarCode(
              type        barcode_type,
              char*       data,
              float       x_pos,
              float       y_pos,
              float       width,
              float       height,
              degree      degree = DEG0,
              ocaColor    color = BLACK
             );
```

Parameters

Barcode_type
The barcode encoding, followings are supported:

CDB2OF7	AIM USS-Codabar, Codabar 2-of-7
CODE11	Code 11
CODE128	CODE 128, A, B, and C auto-switching mode
CODE128B	CODE 128, Set B, for suppress mode C in favor of mode B
CODE32	Code 32, up to 8 digits
CODE39	AIM USS-39, Code 39 (3 of 9)
CODE39E	Code 39 (3 of 9) Extended (full text)
CODE93	Code 93
DL2OF5	Data Logical Code 2 of 5
DPIDENT	Deutsche Post Identcode, 11 digits
DPLEIT	Deutsche Post Leitcode, 13 digits
EAN128	EAN 128
EAN14	EAN-14, 13 digits
IATA2OF5	IATA Code 2 of 5
IND2OF5	Industrial Code 2 of 5
ITF14	ITF-14, 13 digits
ITL2OF5	Interleaved Code 2-of-5

LOGMARS	LOGMARS
MAT2OF5	Matrix Code 2-of-5
MSI	MSI Plessey
PHARMA	Pharmacode One-Track
PLESSEY	PLESSEY (an older code still popular in some industries)
TELEPENA	Telepen Alpha
TELEPENN	Telepen Numeric

APOST	Australia Post Standard customer, allows 8 Digits, 8 digits followed by 5 characters, 16 digits, 8 digits followed by 10 characters, 23 digits
APOSTRD	Australia Post Redirection, 8 digits
APOSTRP	Australia Post Reply Paid, 8 digits
APOSTRT	Australia Post Routing, 8 digits
DPOST	Dutch Post KIX, 11 characters
JPOST	Japan Postal barcode
KPOST	Korea Postal barcode
POSTNET	PostNet
PLANET	Planet
RM4SCC	Royal Mail 4 State
SPOST4	Singapore Postal 4-state barcode
USPS4S	USPS 4-state postal barcode, 20 digits, 5, 9, or 11 digits zip code can be appended using the dash (-) character

EAN	EAN, EAN-2/EAN-5/EAN-8/EAN-13, 2, 5, 7, and 12 digits, EAN-2 2 digits or EAN-5 5 digits can be appended using the plus (+) character
UPCA	UPC-A, 11 digits, EAN-2 2 digits or EAN-5 5 digits can be appended using the plus (+) character
UPCE	UPC-E, 6 digits, also 7 digits starting with 1, EAN-2 2 digits or EAN-5 5 digits can be appended using the plus (+) character

data

Either ASCII or EBCDIC input data. Make sure the PRMODE parameter is specified in your MakeAFP Weaver definition file if your input data is in EBCDIC encoding, in this case, EBCDIC data will be converted into ASCII before being encoded in barcode encodings.

x_pos, y_pos

The position of the top left corner of the leftmost element of the barcode symbol.

width, height

The width and height of barcode dimension.

degree

The rotation for the barcode. The valid values are:

DEG0	The barcode is not rotated.
DEG90	The barcode is rotated 90 degrees clockwise
EG180	The barcode is rotated 180 degrees clockwise
DEG270	The barcode is rotated 270 degrees clockwise

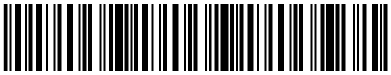
ocaColor

Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample:

```
char *data = "123456789012345678901234567890";

  SetUnit(IN_U1440);

  OpenDoc();

  OpenPage(8.5,11);
                 :
                 :
  BarCode(CODE128C,                 // Barcode type is Code 128 set C
           data,                   // Input data field
           1,                      // barcode X position to 1"
           1,                      // barcode Y position to 1"
           2,                      // barcode dimension width 2"
           0.35)                    // barcode dimension height 0.35"
                 :
                 :

  ClosePage();
  CloseDoc();
```

Print / display:



BCOCA Barcode (Linear)

Function

Generates data in IBM BCOCA linear barcode format, you should be familiar with the standard linear barcode programming techniques and values. Invalid data and length or invalid barcode symbology values may result in errors when your document is printed.

Please read your printer hardware documentation before using bar codes. The documentation should indicate which bar code types, modifiers, module width, element heights, and ratio values are valid for the printer. MakeAFP does minimal verification of the bar code values. If you are using the parameters of modifiers, element height, module width, and ratio, ensure the values you specified are valid for your IPDS printer.

As required by BCOCA standard, if your input data is ASCII, for UPS and EAN barcodes, the data will be translated from IBM ASCII code page 877 to IBM EBCDIC code page 893; for barcode 128, the data will be translated from IBM ASCII code page 819 to IBM EBCDIC code page 1303; and for the remaining linear barcode types, the data will be translated from IBM ASCII code page 819 to IBM EBCDIC code page 500.

Make sure the PRMODE parameter is specified in your MakeAFP Weaver definition file if your input data is in EBCDIC encoding, in this case, EBCDIC data will not be converted.

Note: BCOCA linear barcode requires appropriate printer microcode support.

Syntax

```
void BBarCode(
              type          barcode_type,
              char*         data,
              float         x_pos,
              float         y_pos,
              ushort        module_width = DEFAULT,
              float         element_height = DEFAULT,
              ushort        degree = DEG0,
              ushort        present_HRI = DEFAULT,
              bool          asterix = ON,
              ushort        fontid = DEFAULT,
              ushort        modifier = DEFAULT,
              ushort        height_multiplier = 1,
              ushort        wide_to_narrow_ratio = DEFAULT,
              ushort        oca_color = BLACK,
              ushort        cmr_id = 0,
              ushort        process_mode = AUDIT
             );
```

Parameters

barcode_type
The type of linear barcode symbol generated. Valid values are:

APOST	Postal barcode for Australia
CDB2OF7	AIM USS-Codabar, Codabar 2-of-7
CODE128	CODE 128, AIM USS-128
CODE39	AIM USS-39, Code 39 (3 of 9)
CODE93	Code 93
EAN13	EAN-13 (includes JAN-standard)
EAN2SUP	EAN Two-digit supplemental

EAN5SUP	EAN Five-digit supplemental
EAN8	EAN-8 (includes JAN-short)
IND2OF5	Industrial 2-of-5
ITL2OF5	Interleaved 2-of-5, AIM USS-I 2/5
JPOST	Postal 4-State barcode for Japan
MAT2OF5	Matrix 2-of-5
MSI	Modified Plessey
POSTNET	POSTNET
RM4SCC	Royal Mail 4 State
UPCA	UPC/CGPC Version A
UPCE	UPC/CGPC Version E
UPC2SUPP	UPC - two digit supplemental
UPC5SUPP	UPC - five digit supplemental
USPS4S	USPS Intelligent 4-State barcode for USA postal, 20, 25, 29 or 31 digits required

data
The null-terminated single-byte input data string.

x_pos
The X position of the top left corner of the leftmost element of the barcode symbol.

y_pos
The Y position of the top left corner of the leftmost element of the barcode symbol. Zero is not valid. If you specify HRI (human-readable interpretation) to be presented on top of the barcode, the offset position must allow enough room for the text.

module_width
The width in mils (thousandths of an inch, 0.001 inches) of the smallest defined linear barcode element. Some barcode symbologies refer to this value as the unit or X-dimension width. The widths of all symbol elements (bars and spaces) are normally expressed as multiples of the module width. Specify DEFAULT to use the default module width of the presentation device.

element_height
The height of the bar code symbol. The element height and height multiplier values are used to compute the total bar and space height of the bar code symbol. Specify DEFAULT to use the default element height of the presentation device.

degree
The rotation for the barcode. The valid values are:

DEG0	The barcode symbol is not rotated
DEG90	The barcode symbol is rotated 90 degrees clockwise
DEG180	The barcode symbol is rotated 180 degrees clockwise
DEG270	The barcode symbol is rotated 270 degrees clockwise

present_HRI
Specifies whether the human-readable interpretation of the barcode data should be printed and the location of the HRI. Some bar code types ignore the HRI request. Valid values are ON, OFF, ABOVE, BELOW. The default value is DEFAULT that is to use device default.

asterisk
Specifies whether an asterisk should be presented as the HRI for Code 39 barcode start and stop characters. This value is ignored for other bar code types. Possible values are ON and OFF. Default is ON.

fontid
The ID number of the font to be used when HRI (human-readable interpretation) is requested. Specify DEFAULT to use your device's default font.

Some bar code types have specific requirements for the type of HRI font used, like the UPC and EAN symbologies specify OCR-B for HRI, and some bar code types do not allow HRI at all, for example, Japan Postal barcode, POSTNET, and RM4SCC, where this field is ignored.

modifier

The modifier gives additional processing information about the bar code symbol generated. For example, it indicates whether a check-digit is generated for the barcode symbol. The meaning of the modifier values will vary depending on the type of bar code symbol. Specify DEFAULT to use barcode default. Refer to IBM *Bar Code Object Content Architecture Reference* for more details. Valid values are (bolded is the default):

Austrailia postal	**'\x01'** through '\x08'
AIM USS-39, Code 39 (3 of 9)	**'\x01'** and '\x02'
AIM USS-Codabar, Codabar 2–of-7	**'\x01'** and '\x02'
Code 128, AIM USS-128, UCC/EAN128	**'\x02'** through '\x05'
Code 93	**'\x00'**
EAN-8 (includes JAN-short)	**'\x00'**
EAN-13 (includes JAN-standard)	**'\x00'**
EAN two-digit supplemental	**'\x00'** and '\x01'
EAN five-digit supplemental	**'\x00'** and '\x01'
Industrial 2–of–5	**'\X01'** and '\x02'
Interleaved 2–of-5, AIM USS-I 2/5	**'\X01'** and '\x02'
Japan postal	**'\x00'** and '\x01'
Matrix 2–of–5	**'\x01'** and '\x02'
Modified Plessey	**'\x01'** through '\x09'
POSTNET, PLANET	**'\x00'** through '\x04'
Royal mail	**'\x00'** and '\x01'
UPC/CGPC Version A	**'\x00'**
UPC/CGPC Version E	**'\x00'**
UPC – 2 digit supplemental	**'\x00'** through '\x02'
UPC – 5 digit supplemental	**'\x00'** through '\x02'
USPS 4-State OneCode	'\x00' through '\x03'

height_multiplier

Specifies a value that, when multiplied by the element height, yields the total bar and space height presented. Valid values are 1 to 255, the default value is 1.

wide_to_narrow_ratio

The ratio of the wide-element dimension to the narrow-element dimension for a two-level linear bar code symbol. For example, if you want a ratio of 1.65 to 1, set this field to 165. Specify DEFAULT to use the device default. This parameter is ignored for POSTNET, EAN, UPC type linear bar codes, but a value must still be specified since all parameters are required on C function calls.

ocaColor

Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

cmr_id

The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Weaver definition file with a CMR parameter. Default value 0 specifies that CMR is not being defined.

process_mode

Specifies the processing mode for the CMR:

`AUDIT` The audit processing mode. Refers to processing that has already been applied to a resource. In most cases, audit CMRs describe input data and are similar to ICC input profiles. The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

INSTR

The instruction processing mode. Refers to processing that is done to prepare the resource for a specific printer using a certain paper or another device. Generally, instruction CMRs refer to output data and are similar to ICC output profiles.

The instruction processing mode is used with color conversion, tone transfer curve, and halftone CMRs. In instruction processing mode, these CMRs indicate how the system must convert a resource so it prints correctly on the target printer. The manufacturer of your printer should provide ICC profiles or a variety of CMRs that you can use. Those ICC profiles and CMRs might be installed in the printer controller, included with the printer on a CD, or available for download from the manufacturer's Web site.

Sample

```
char *data = "1234567890";

OpenPage(8.5,11);
BBarCode(data,            // Barcode data variable
        1,                // Barcode x position to 1"
        1,                // Barcode Y position to 1"
        CODE128,          // Barcode type is CODE 128
        20,               // Barcode module width in mils
        0.5);             // Barcode element height 0.5"
                          // Other parameters use AFP BCOCA defaults
ClosePage();
CloseDoc();
```

Valid Linear Barcode Characters and Data Lengths

Bar Code Type	Valid Characters	Valid Data Length
Australia Post Bar Modifier '\X01'	0123456789	Symbology: 8 digits BCOCA range: 8 digits
Australia Post Bar Modifier '\X02'	0123456789	Symbology: 8–16 digits BCOCA range: 8–16 digits
Australia Post Bar Modifier '\X03'	0123456789 ABCDEFGHIJKLM NOPQRSTUVWXYZ abcdefghijklm nopqrstuvwxyz (space), # (number sign)	Symbology: 8–13 characters BCOCA range: 8–13 characters
Australia Post Bar Modifier '\X04'	0123456789 for sorting code 0–3 for customer information	Symbology: 8–24 digits BCOCA range: 8–24 digits
Australia Post Bar Modifier '\X05'	0123456789	Symbology: 8–23 digits BCOCA range: 8–23 digits
Australia Post Bar Modifier '\X06'	0123456789 ABCDEFGHIJKLM NOPQRSTUVWXYZ abcdefghijklm nopqrstuvwxyz (space), # (number sign)	Symbology: 8–18 digits BCOCA range: 8–18 digits
Australia Post Bar Modifier X'07'	0123456789 for sorting code 0–3 for customer information	Symbology: 8–39 digits BCOCA range: 8–39 digits
Australia Post Bar Modifier '\X08'	0123456789	Symbology: 8 digits BCOCA range: 8 digits
Code 128, AIM USS-128	All characters defined in the Code 128 code page	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
Code 39 (3-of-9 Code), AIM USS-39	0123456789 ABCDEFGHIJKLM NOPQRSTUVWXYZ -.$/+% (space) A total of 43 valid characters	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
Code 93	0123456789 ABCDEFGHIJKLMNOP QRSTUVWXYZ-.$/+% space character a – representing Shift 1 b – representing Shift 2 c – representing Shift 3 d – representing Shift 4	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
EAN-8 (includes JAN-short)	0123456789	7 characters
EAN-13 (includes JAN-standard)	0123456789	12 characters
EAN Two-digit Supplemental	0123456789	2 characters for Modifier X'00' 14 characters for Modifier '\X01'

EAN Five-digit Supplemental	0123456789	5 characters for Modifier X'00' 17 characters for Modifier ‘\X01’
Industrial 2-of-5	0123456789	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
Interleaved 2-of-5, AIM USS-I 2/5	0123456789	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
Japan Postal Bar Code (Modifier X'00')	0123456789 ABCDEFGHIJKLM NOPQRSTUVWXYZ - (hyphen)	Symbology: 7 or more. BCOCA range: 7 to 50 characters, some printers may support a larger data length.
Japan Postal 4-State Bar Code (Modifier ‘\X01’)	0123456789 CC1,CC2,CC3,CC4, CC5,CC6,CC7,CC8 - (hyphen), start, stop	No length checking done.
Matrix 2-of-5	0123456789	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
MSI (modified Plessey code)	0123456789	3 to 15 characters for Modifier ‘\X01’ 2 to 14 characters for Modifier ‘\X02’ 1 to 13 characters for all other modifiers
POSTNET	0123456789	5 characters for Modifier X'00' 9 characters for Modifier ‘\X01’ 11 characters for Modifier ‘\X02’ BCOCA range for Modifier ‘\X03’: 0 to 50 characters, some printers may support a larger data length
Royal Mail (RM4SCC, modifier X'00')	0123456789 ABCDEFGHIJKLM NOPQRSTUVWXYZ	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
Royal Mail (Dutch KIX variation, modifier ‘\X01’)	0123456789 ABCDEFGHIJKLM NOPQRSTUVWXYZ abcdefghijklm nopqrstuvwxyz	Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.
UPC/CGPC Version A	0123456789	11 characters
UPC/CGPC Version E	0123456789	10 characters
UPC Two-digit Supplemental (Periodicals)	0123456789	2 characters for Modifier X'00' 13 characters for Modifier ‘\X01’ 12 characters for Modifier ‘\X02’
UPC Five-digit Supplemental Paperbacks)	0123456789	5 characters for Modifier X'00' 16 characters for Modifier ‘\X01’ 15 characters for Modifier ‘\X02’
USPS 4-State	0123456789	20 digits for modifier X’00’ 25 digits for modifier ‘\X01’ 29 digits for modifier ‘\X02’ 31 digits for modifier ‘\X03’

BCOCA MaxiCode 2D Barcode

Function

Generates data in BCOCA MaxiCode 2D barcode format, you should be familiar with the MaxiCode 2D barcode programming techniques and values. Invalid data and values may result in errors when your document is printed.

As required by BCOCA standard, if your input data is EBCDIC, the data will be translated from IBM EBCDIC code page 500 to IBM ASCII code page 819.

Make sure the PRMODE parameter is specified in your MakeAFP Weaver definition file if your input data is in EBCDIC encoding, in this case, EBCDIC data will be converted into ASCII before being encoded in the barcode encodings.

Note: BCOCA BCOCA MaxiCode 2D barcode requires appropriate printer microcode support.

Syntax

```
void BMaxiCode(
               char*             data,
               float             x_pos,
               float             y_pos,
               ushort            symbol_mode = MODE4,
               bool              zipper = FALSE,
               bool              NoESC = TRUE,
               ushort            degree = DEG0,
               ushort            oca_color = BLACK,
               ushort            cmr_id = 0,
               ushort            process_mode = AUDIT,
               ushort            seqcount = 0,
               ushort            seqind = 0
              );
```

Parameters

data
The null-terminated up to 138 characters of ASCII or EBCDIC input data string.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

symbol_mode
Symbol mode, Valid values are:

MODE2	Structured Carrier Message — numeric postal code
MODE3	Structured Carrier Message — alphanumeric postal code
MODE4	Standard symbol (default value)
MODE5	Full ECC (Enhanced Error Correction) Symbol
MODE6	The bar code data is used to program the bar code reader system

zipper
Specifies whether or not a vertical zipper-like test pattern and contrast block is to be printed to the right of the symbol. The zipper provides a quick visual check for printing distortions. If the bar code is rotated, the zipper and block are rotated along with the symbol. Default is FALSE, a zipper pattern is not printed.

noESC
Specifies whether a backslash character '\' within the bar code data should be treated as an escape sequence or not. Specify FALSE if the backslash should be treated as an escape, an escape sequence is useful if you need to encode control characters like Carriage Return into the barcode. The default value is TRUE, each backslash character within the bar code data is treated as character data.

degree
The rotation for the barcode. The valid values are:

`DEG0`	The barcode image is not rotated
`DEG90`	The barcode image is rotated 90 degrees clockwise
`DEG180`	The barcode image is rotated 180 degrees clockwise
`DEG270`	The barcode image is rotated 270 degrees clockwise

color
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

cmr_id
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Weaver definition file with a CMR parameter. Default value 0 specifies that CMR is not being defined.

process_mode
Specifies the processing mode for the CMR:

`AUDIT`

The audit processing mode. Refers to processing that has already been applied to a resource. In most cases, audit CMRs describe input data and are similar to ICC input profiles. The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

`INSTR`

The instruction processing mode. Refers to processing that is done to prepare the resource for a specific printer using a certain paper or another device. Generally, instruction CMRs refer to output data and are similar to ICC output profiles.

The instruction processing mode is used with color conversion, tone transfer curve, and halftone CMRs. In instruction processing mode, these CMRs indicate how the system must convert a resource so it prints correctly on the target printer. The manufacturer of your printer should provide ICC profiles or a variety of CMRs that you can use. Those ICC profiles and CMRs might be installed in the printer controller, included with the printer on a CD, or available for download from the manufacturer's Web site.

Seqcount and seqind
MaxiCode bar code symbols can be linked together logically to encode large amounts of data. This is called a structured append sequence. The logically linked symbols can be printed separately and then recombined logically after they are scanned. 2 to 8 MaxiCode symbols can be linked together. The Sequence Count specifies the number of

symbols to be linked. The Sequence Indicator specifies for each bar code symbol where it fits logically into the sequence (i.e. a number from 1 to 8). The Sequence Indicator must be 1 if the Sequence Count is 0 or 1.

If default 0 is specified, this symbol is not part of a structured append.

Sample

```
char *data = "1234567890";

SetUnit(IN_U1440);
OpenDoc();
OpenPage(8.5,11);
           :
           :
BMaxiCode(data,           // Barcode data
           1,             // Barcode x position to 1"
           1);            // Barcode Y position to 1"

           :
           :

ClosePage();
CloseDoc();
```

BCOCA PDF417 2D Barcode

Function

Generates data in BCOCA PDF417 2D barcode format, you should be familiar with the PDF417 2D barcode programming techniques and values. Invalid data and values may result in errors when your document is printed.

As required by BCOCA standard, if your input data is EBCDIC, the data will be translated from IBM EBCDIC code page 500 to IBM ASCII code page 437 subset GL 0.

Make sure the PRMODE parameter is specified in your MakeAFP Weaver definition file if your input data is in EBCDIC encoding, in this case, EBCDIC data will be converted into ASCII before being encoded in the barcode encodings.

Note: BCOCA PDF417 2D barcode requires appropriate printer microcode support.

Syntax

```
void BPDF417(
             char*          data,
             float          x_pos,
             float          y_pos,
             ushort         numrows = 0,
             ushort         rowsize = 10,
             ushort         modifier = 0x00,
             bool           NoESC = TRUE,
             ushort         security = 0,
             ushort         degree = DEG0,
             ushort         oca_color = BLACK,
             ushort         cmr_id = 0,
             ushort         process_mode = AUDIT,
             char*          macro = NULL
            );
```

Parameters

data
The null-terminated up to 2710 characters of ASCII or EBCDIC input data string.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

numrows
The desired number of rows in the generated bar code symbol. 3 to 90 rows can be requested, or specify 0 as the number of rows to have the printer generate the minimum number of rows necessary. The number of rows times the number of data symbol characters per row cannot exceed 928. The actual number of rows generated by the printer depends on the amount of data and the security level selected. If more rows are requested with this parameter than necessary, the symbol is padded to fill the requested number. If not enough rows are specified, extra rows will be inserted by the printer to produce the symbol.

Default is 0 which lets the printer generates the minimum number of rows.

rowsize
The number of data symbol characters per row. Each row consists of a start pattern, a left row indicator codeword, 1 to 30 data symbol characters, a right row indicator

codeword, and a stop pattern. The number of rows times the number of data symbol characters per row cannot exceed 928. The default value is 10.

modifier
Specifies additional processing information about the bar code symbol to be generated (for example, it specifies whether a check-digit should be generated for the bar code symbol). Valid values for PDF417 are 0 or 1, default value is 0.

noESC
Specifies whether a backslash character '\' within the bar code data should be treated as an escape sequence or not. Specify FALSE if the backslash should be treated as an escape, an escape sequence is useful if you need to encode control characters like Carriage Return into the barcode. The default value is TRUE, each backslash character within the bar code data is treated as character data.

security
The desired security level for the symbol as an integer from 0 (only error recognition) to 8 (highest). The higher the security level, the more error correction codewords will be added to the symbol. Default is Security level 0.

degree
The rotation for the barcode. The valid values are:

`DEG0`	The barcode is not rotated
`DEG90`	The barcode overlay is rotated 90 degrees clockwise
`DEG180`	The barcode overlay is rotated 180 degrees clockwise
`DEG270`	The barcode overlay is rotated 270 degrees clockwise

color
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

cmr_id
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Weaver definition file with a CMR parameter. Default value 0 specifies that CMR is not being defined.

process_mode
Specifies the processing mode for the CMR:

`AUDIT`
The audit processing mode. Refers to processing that has already been applied to a resource. In most cases, audit CMRs describe input data and are similar to ICC input profiles. The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

`INSTR`
The instruction processing mode. Refers to processing that is done to prepare the resource for a specific printer using a certain paper or another device. Generally, instruction CMRs refer to output data and are similar to ICC output profiles.

The instruction processing mode is used with color conversion, tone transfer curve, and halftone CMRs. In instruction processing mode, these CMRs indicate how the system must convert a resource so it prints correctly on the target printer. The manufacturer of your printer should provide ICC profiles or a variety of CMRs that you can use. Those ICC profiles and CMRs might be installed in the printer controller, included with the printer on a CD, or available for download from the manufacturer's Web site.

macro
PDF417 Macro data. The total length of the macro text is limited to 2,710 bytes. This is the maximum number of symbols that can be displayed using PDF417 symbology and all numbers in the data. The data for this macro must adhere to the format defined in section G.2 of the Uniform Symbology Specification PDF417. MakeAFP does not verify the macro contents, please make sure your macro data is correct, otherwise, you may end up with errors at print time.

Sample

```
char *data = "1234567890";

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

          :

BPDF417(data,             // Barcode data
        1,                // Barcode x position to 1"
        1);               // Barcode Y position to 1"

          :

ClosePage();
CloseDoc();
```

BCOCA QR Code 2D Barcode

Function

Generates data in BCOCA QR Code 2D barcode format, you should be familiar with the QR Code 2D barcode programming techniques and values. Invalid data and values may result in errors when your document is printed.

Note: BCOCA QR Code 2D barcode requires appropriate printer microcode support.

Syntax

```
void BQRCode(
             char*           data,
             float           x_pos,
             float           y_pos,
             ushort          size = 0,
             ushort          codepage = CP897,
             ushort          specfunc = USERDEF,
             ushort          appind = 0,
             bool            NoESC = TRUE,
             ushort          eclvl = MEDIUM,
             ushort          degree = DEG0,
             ushort          color = BLACK,
             ushort          cmr_id = 0,
             ushort          process_mode = AUDIT,
             ushort          parity = 0,
             ushort          seqcount = 0,
             ushort          seqind = 0
            );
```

Parameters

data
The null-terminated up to 7089 characters of ASCII or EBCDIC input data string.

x_pos, y_pos
The position of the top left corner of the leftmost element of the barcode symbol.

size
The desired size, the allowable values are from 21 to 177 increments of 4. See IBM *Bar Code Object Content Architecture Reference* for details.

Specify default value 0 as the number of sizes to have the printer generate a minimum number of rows based on the amount of symbol data.

codepage
Code page that encodes the QR Code 2D barcode data, default is CP897 for the barcode data encoded in ASCII, code pages supported for EBCDIC are CP500 (international #5), CP290 (Japanese Katakana Extended), and CP1027 (Japanese Latin Extended).

specfunc
This parameter is used to request special functions that can be used with QR Code 2D symbols. Valid values are:

`FNC1UCC` UCC/EAN1 alternate data type identifier indicates that this QR Code symbol conforms to the specific industry or application specifications previously agreed with AIM International. When this standard is selected, an application indicator must be specified.

FNC1IND	Industry FNC1 alternate data type identifier indicates that this bar code symbol conforms to the specific Industry or application specifications previously agreed with AIM International. When this standard is selected, an application indicator must be specified.
USERDEF	Default value**,** None of the above. This is a user-defined symbol with either no significance or "user-defined" significance assigned to all FNC1 characters appearing in the symbol.

appind
Application indicator for Industry FNC1. This parameter is required when FNC1IND is coded by a special function parameter. It is coded as a single upper or lower case alphabetic character, or a 1 byte of the hex value. Default is 0, this parameter is ignored

noESC
Specifies whether a backslash character '\' within the bar code data should be treated as an escape sequence or not. Specify FALSE if the backslash should be treated as an escape, an escape sequence is useful if you need to encode control characters like Carriage Return into the barcode. The default value is TRUE, each backslash character within the bar code data is treated as character data.

eclvl
Error Correction Level. It specifies the level of error correction to be used for the symbol. Each higher level of error correction causes more error correction code words to be added to the symbol and therefore leaves fewer code words for the data. Four different levels of Reed-Solomon error correction can be defined:

LOW	Allows recovery of 7% of symbol code words
MEDIUM	Allows recovery of 15% of symbol code words, default
QUARTIL	Allows recovery of 25% of symbol code words
HIGH	Allows recovery of 30% of symbol code words

degree
The rotation for the barcode. The valid values are:

DEG0	The barcode is not rotated.
DEG90	The barcode is rotated 90 degrees clockwise
DEG180	The barcode is rotated 180 degrees clockwise
DEG270	The barcode is rotated 270 degrees clockwise

color
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY, Default is BLACK color.

cmr_id
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Weaver definition file with a CMR parameter. Default value 0 specifies that CMR is not being defined.

process_mode
Specifies the processing mode for the CMR:

AUDIT	The audit processing mode. Refers to processing that has already been applied to a resource. In most cases, audit CMRs describe input data and are similar to ICC input profiles. The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

`INSTR`
The instruction processing mode. Refers to processing that is done to prepare the resource for a specific printer using a certain paper or another device. Generally, instruction CMRs refer to output data and are similar to ICC output profiles.

The instruction processing mode is used with color conversion, tone transfer curve, and halftone CMRs. In instruction processing mode, these CMRs indicate how the system must convert a resource so it prints correctly on the target printer. The manufacturer of your printer should provide ICC profiles or a variety of CMRs that you can use. Those ICC profiles and CMRs might be installed in the printer controller, included with the printer on a CD, or available for download from the manufacturer's Web site.

parity
This parameter specifies parity data for a structured-append symbol, it is used for the QR Code barcode only when it has linked the structured-append symbol. Valid values are ‘00’x to ‘FF’x, default value is 0.

seqcount, seqind
QR bar code symbols can be logically linked together to encode large amounts of data. This is called a structured append sequence. The logically linked symbols can be printed separately and then logically recombined after they are scanned. From 2 to 16 QR Code symbols can be linked together. The Sequence Count specifies the number of symbols to be linked. The Sequence Indicator specifies for each bar code symbol where it fits logically into the sequence (a number from 1 to 16).

seqconut is the total number of structured-append sequences, which acceptable range of values is 2 to 16. The default value is 0, this symbol is not part of a structured-append.

seqind is the structured-append sequence indicator, allowed values are 1 to 16. The default value is 0, this symbol is not part of a structured-append.

Sample

```
char *data = "1234567890";

SetUnit(IN_U1440);

OpenDoc();
OpenPage(8.5,11);

           :
           :

BQRCode(data,           // Barcode data
        1,              // Barcode x position to 1"
        1);             // Barcode Y position to 1"

           :
           :

ClosePage();
CloseDoc();
```

Begin of Index Name Group of Input AFP (Checking)

Function

Tests for begin of index name group of input AFP file.

Returns 1, if the beginning of the name group of AFP index boundary has been detected, after the “Get Page” function is called for the reading of an AFP page from the input AFP file or returns 0 if it is still not.

This function is mainly developed for calling from other programming languages; with Visual C++, you can use $Bng variable directly.

Syntax

```
Bool Bng(void);
```

Parameters

None.

Sample

None.

Begin Index Group

Function

Begins an index page group.

With the “Begin Index” and “End Index” functions, you can define the beginning and the ending of the index page group boundaries within an AFP document, so the statement pages belonging to each client can be quickly navigated and retrieved by AFP viewer, AFP archiving system, MakeAFP reprint, and sorting utilities, or other software.

The index group name should be unique within a document. Groups of pages cannot be overlapped or nested, and each index group must end before another can begin.

Syntax

```
void BgnIdx(
              char*         groupname,
              ushort        docNo = 1
              bool          autoConvert = true
            );
```

Parameters

groupname
The name of the indexing group. The null-terminated group name should be unique within a document. The maximum number of characters in the group name is 8, blanks are allowed as part of the group name.

docNo
Specified to which AFP document to insert the index information, valid values are 1 through 10, the default value is 1.

autoConvert
Specifies whether let MakeAFP Weaver determine a conversion from the native PC ASCII encoding to the target AFP index string encoding is needed automatically. The default value is TRUE lets MakeAFP Weaver to auto-decide a conversion is required. MakeAFP Weaver calls converter by the encodes specified by the “Encoding” function. Make sure the “Encoding” function is called if a conversion is required.

Sample

```
/*****************************************************************************/
/* This sample shows how to capture a trigger by an overlay name and        */
/* data fields from page 1, add AFP indexes and barcode to existing         */
/* AFP                          AFP is encoded in CP-037, USA EBCDIC        */
/*****************************************************************************/

int main( )
{
   unsigned int i, grpPages, pageSN, groups;
   char tmp[80], mobileNo[20], custName[60];
   bool bog = 0;

   $MaxPaging = 50;          // Maximum paging is up to 50 pages

   SetUnit(IN_U600);                // Set default unit to inch

   Start();                  // Start initiation, open default input,
                             // output and definition files, retrieves
```

```
                          // AFP resources, allocate memory

Encoding("ibm-037","ibm-437");

OpenDoc();                // Open AFP document
$Page = 1;                // Set AFP page buffer number to 1 for the first
                          // page of AFP file

GetPage();                // Get first page of AFP file

while ($Edt == 0)         // Until end of AFP document
{
  GetField(660, 1080, custName);      // Get customer name

  GetField(4050, 900, mobileNo);      // Get customer mobile number

  do {

     $Page++;            // Point to next AFP page buffer

     GetPage();          // Get next page

     // detecting if it is the first page of a group,
     // overlay O1OVL1E only used by at first page of
     // each page group
     bog = TriggerOvly("O1OVL1E");

   } while (!bog && !$Edt);      // Until beginning of next page group or
                                        // End of AFP file

  bog = 0;                       // Reset it for next group

  // Now got all pages of a page group and first page of next group, now
  it  // is ready to process new AFP output

  if (!$Edt)                     // If not end of AFP document
     grpPages = $Page -1 ;       // Keep total number of pages per group,
                                 // need to minus 1 page of the first
                                 // page of next group
  sprintf(tmp, "%08d", ++groups);
  BgnIdx(tmp);          // Auto-converts ASCII to EBCDIC for indexes
  PutIdx("Customer Name", custName);
  PutIdx("Mobile Number", mobileNo);

  for (i = 0; i < grpPages; i++)
  {
     $Page = i + 1;              // Point to page buffer number to be
  opened

     sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
     BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90);   // Add 1D barcode

     ClosePage();                // End of AFP page, write to AFP file
  }

  EndIdx();                      // End of group level index

  MovePage(1, grpPages + 1);     // As we got first page of next group
                                        // previously, now need move its
  contents
                                 // to page buffer 1 for the next page
  group

  $Page = 1;                     // Reset page buffer to 1 for next group
}

CloseDoc();                      // End of AFP document, close AFP output
```

```
    return 0;
}
```

Begin Overstrike

Function

Begins overstriking of text on the page, you can end overstriking of text by the “End Overstrike” Function.

Syntax

```
void BgnOstrike( );
```

Parameters

No parameter to be specified.

Sample

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

BgnOstrike();                        // begin overstriking of text

Ltxt("This is an overstrike text");  // text will be overstriked
              :
              :

EndOstrike()                         // end overstriking of text
              :
ClosePage();
CloseDoc();
```

Begin Underscore

Function

Begins underscoring of text on the page, you can end underscoring of text by the “End Underscore” Function.

Syntax

```
void BgnUscore( );
```

Parameters

No parameter to be specified.

Sample

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

BgnUscore();                          // begin underscoring of text

Ltxt("This is an underscore text");   // text will be underscored
                :
                :

EndUscore()                           // end underscoring of text
                :
ClosePage();
CloseDoc();
```

Blank Page (Checking)

Function

Tests if the current page is a blank page that has no text presented.

Syntax

```
bool BlankPage(void);
```

Parameters

No parameter to be specified.

Sample

```
SetUnit(IN_U600);

OpenDoc();

GetPage();               // Read-in an AFP page

              :
              :

if ( !BlankPage() )      // Output this AFP page if it is not a blank
  ClosePage();           // page

              :
              :


CloseDoc();
```

Box Drawing

Function

Draws a box at the specified position using the specified line thickness. Ensure that the box you have specified fits on the page.

Syntax

```
void Box(
        float        x_pos,
        float        y_pos,
        float        box_width,
        float        box_height,
        float        line_thickness,
       );
```

Parameters

x_pos
The X position of the top left corner of the box.

y_pos
The Y position of the top left corner of the box.

box_width
The width of the box.

box_height
The height of the box.

line_thickness
The thickness of the lines of the box.

Sample

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

Box(1,1,5,2,0.02);                  // Draw box from (1",1"), size 5 x 2",
                                    // red line thickness is 0.02"

Box(1,5,5,1,0.01);                  // Draw box from (1",5"), size 5 x 1",
                                    // blue line thickness is 0.01"
              :
              :

ClosePage();
CloseDoc();
```

Center Align 1-Byte Text

Function

Center aligns a single-line of the 1-byte text string at the current position.

You need to define an ASCII or EBCDIC encoded font with the “Font” function. MakeAFP Weaver converts data encoding internally, according to the encoding of AFP font defined, however for a better formatting performance, using ASCII encoding font is recommended to avoid such ASCII to EBCDIC conversion.

If the font using is an EBCDIC encoded font, then you must make sure that the default input data encoding is defined properly by the function of DefaultCode( ) first, otherwise the default input data encoding “Windows-1252” is being used for internal data encoding conversion.

Syntax

```
void Ctxt(
          char*    data,
          bool     same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated ASCII data string.

same_pos
Indicates whether the current X position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current X position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
             :
Font(3);                            // assume font 3 is ASCII font
             :
Pos(2,2);                           // current position at (2",2")
Ctxt("text is center aligned");     // Center text at (2",2")
             :

ClosePage();
CloseDoc();
```

Center Align Japanese

Function

Center aligns a single-line of the Japanese text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be an SJIS-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Cjp(
         char*       data,
         bool        same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated SJIS data string.

same_pos
Indicates whether the current X position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current X position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
             :
Font2(3,4);                          // assume font 3 is ASCII font,
                                     // and font 4 is SJIS font

             :
Pos(2,2);                                 // position at (2",2")
Cjp("Alphabet が混在した文章のサンプルです");  // Center SJIS text at
                                          // (2",2")

             :
             :

ClosePage();
CloseDoc();
```

Center Align Korean

Function

Center aligns a single-line of the Korean text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be a KSC-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Ckr(
          char*        data,
          bool         same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated KSC data string.

same_pos
Indicates whether the current X position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current X position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
Font2(3,4);                              // assume font 3 is ASCII font,
                                         // and font 4 is KSC font

               :
Pos(2,2);                                // position at (2",2")
Ckr("IBM 소프트웨어 솔루션");            // Center KSC text at
                                         // (2",2")

               :
               :

ClosePage();
CloseDoc();
```

Center Align Simplified Chinese

Function

Center aligns a single-line of the Simplified Chinese text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be a GBK-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Csc(
         char*     data,
         bool      same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated GBK data string.

same_pos
Indicates whether the current X position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current X position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
Font2(3,4);                         // assume font 3 is ASCII font,
                                    // and font 4 is Gb18030 font
              :
Pos(2,2);                           // current position at (2",2")
Csc("实现 Win2000 与 Linux 的双引导");    // Center GBK text at (2",2")

              :
              :

ClosePage();
CloseDoc();
```

Center Align Traditional Chinese

Function

Center aligns a single-line of the Traditional Chinese text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be a BIG5-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Ctc(
          char*     data,
          bool      same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated BIG5 data string.

same_pos
Indicates whether the current X position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current X position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
Font2(3,4);                              // assume font 3 is ASCII font,
                                         // and font 4 is BIG5 font

               :
Pos(2,2);                                // current position at (2",2")
Ctc("實現 Win2000 與 Linux 的双引导");     // Center BIG5 text at (2",2")


               :
               :

ClosePage();
CloseDoc();
```

Center Align SBCS-HOST/DBCS-HOST

Function

Center aligns a single-line of the EBCDIC/DBCS-HOST text at the current position.

You need to call a pair of fonts with the “Font2” function, the first parameter must be an EBCDIC font, and the second one must be a DBCS-HOST font.

Syntax

```
void Cdbcs(
            char*   data,
            bool    same_pos = TRUE
           );
```

Parameters

data
The NULL-terminated SBCS-HOST/DBCS-HOST data string.

same_pos
Indicates whether the current X position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current X position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
Font2(3,4);                              // assume font 3 is EBCDIC font,
                                         // and font 4 is DBCS-HOST font

               :
Pos(2,2);                                // current position at (2",2")
Cdbcs("实现  Win2000 与 Linux 的双引导"); // Center DBCS text at (2",2")

               :
               :

ClosePage();
CloseDoc();
```

Center Align UTF-16 Text

Function

Center aligns a single-line of the UTF-16 string at the current position. Native UTF-16 string on Windows is in litter-endian (UTF-16LE) encoding, this function converts it to UTF-16BE that is used by AFP.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with the data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Cu16(
          UChar*       data,
          bool         same_pos = TRUE
         );
```

Parameters

data
The UTF-16 NULL-terminated UTF-16 litter-endian string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
 /* UTF-16 string, "test" and CJK characters "测试"  */
 UChar    data1[20] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
             :
             :
Pos(2,2);                         // current position at (2",2")

Font(2);                          // Assume font 2 is a TrueType font
                                  // with data type UTF16BE defined

Cu16(data1);                      // center put UTF-16 at (2",2")

             :
             :

ClosePage();
CloseDoc();
```

Center Align UTF-16 Text Converting from Legacy String

Function

Center aligns a single-line of the UTF-16BE string converting from the legacy codepage/charset string, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with the data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Cu16c(
           char*        data,
           char*        fromcode = NULL,
           bool         same_pos = TRUE
          );
```

Parameters

data
The NULL-terminated legacy codepage string.

fromcode
The encoding name of the source string to be converted into UTF-16. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();

DefaultCode("GB18030");             // set default codepage of input data

OpenPage(8.5,11);
             :
             :
Pos(2,2);                           // set current position at (2",2")

Font(2);                            // Assume font 2 is a TrueType font
                                    // with data type UTF16BE defined

Cu16c("test 测试");                  // center put UTF-16 converting from
                                    // Chinese GB18030
             :
             :

ClosePage();
CloseDoc();
```

Center Align UTF-8 Text

Function

Center aligns a single-line of the UTF-8 string at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with the data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Cu8(
          UChar8*       data,
          bool          same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated UTF-8 string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
/* UTF-8 string, "test" and CJK characters "测试"  */
UChar8    data1[20] = "test\xe6\xb5\x8b\xe8\xaf\x95";

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
               :
Pos(2,2);                           // current position at (2",2")

Font(2);                            // Assume font 2 is a TrueType font
                                    // with data type UTF8 defined

Cu8(data1);                         // center put UTF-8 at (2",2")

               :
               :

ClosePage();
CloseDoc();
```

Center Align UTF-8 Text Converting from Legacy String

Function

Center aligns a single-line of the UTF-8 string converting from the legacy codepage/charset string, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with the data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Cu8c(
        char*         data,
        char*         fromcode = NULL,
        bool          same_pos = TRUE
       );
```

Parameters

data
The NULL-terminated legacy codepage string.

fromcode
The encoding name of the source string to be converted into UTF-8. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
             :
             :
Pos(2,2);                       // Current position at (2",2")

Font(2);                        // Assume font 2 is a TrueType font
                                // with data type UTF8 defined

Cu8c("test 测试","GB18030");     // Center put UTF-8 converting from
                                // Chinese GB18030

             :
             :

ClosePage();
CloseDoc();
```

Center Align UTF-8 Text Converting from UTF-16LE

Function

Center aligns a single-line of the UTF-8 string converting from the UTF16-LE text, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with the data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Cu8u(
          UChar*        u16_data,
          bool          same_pos = TRUE
         );
```

Parameters

u16_data
The NULL-terminated UTF-16LE text string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position to which the next character would be placed.

Sample

```
/* UTF-16 string, "test" and CJK characters "测试"  */
UChar    data1[] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
               :
Pos(2,2);                           // current position to (2",2")

Font(2);                            // Assume font 2 is a TrueType font
                                    // with data type UTF8 defined

Cu8u(data);                         // center put UTF-8 converting from
                                    // UTF16-LE

               :
               :

ClosePage();
CloseDoc();
```

Centimeter Value

Function

Specifies a value in centimeters.

Syntax

```
float cm(
          float     value
        );
```

Parameters

value
The value in centimeters.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8,11);
               :
               :
Pos(2.5,4);                   // set X and Y position to (2.5",4")
               :
               :
Pos(cm(2),3.5);               // set X position to 2 cm and Y position to
                              // 3.5"
               :
               :

ClosePage();
CloseDoc();
```

Close Document

Function

Closes the AFP document previously opened with an "Open Document" call.

You must issue the "Close Page" function request for all pages still opened before issue the "Close Document" function request, otherwise the pages will not be placed into the AFP document output.

The AFP document file will be closed once this function is requested.

CloseDoc( ) deletes an empty AFP document file if there is no page written in that AFP output document.

Syntax

```
void CloseDoc(
               ushort      docNo = 1
              );
```

Parameters

docNo
Specifies which AFP document to be ended, valid values are 1 through 10, the default value is 1.

Sample

```
SetUnit(IN_U600);

OpenDoc();

OpenPage(8.5,11);

    :

    :


ClosePage();

CloseDoc();
```

Close Page

Function

Closes an AFP page previously opened with an “Open Page” call, once the page formatting is completed, you need to close the page with the “Close Page” function to write that AFP page into the AFP file.

With MakeAFP, you can open multiple pages by either the “Open Page” or the “Get Page” function requests, and then process different pages in an interleaved manner once each page is initialized, all the entire MO:DCA 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.

With $MaxPaging variable or the “Maximum Paging” function, you can define the maximum number of AFP page buffers. For generating OMR and page pagination, such as “Page 347 of 1000”, we need to keep composed AFP data in the AFP page buffers first. With MakeAFP 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 the composed AFP data stream will be kept in memory buffers in page-level, and after you have completed all the formatting and counted all the pages of a page group, you can finally put your OMR and pagination text on each page just before you close the page with the “Close Page” function.

With $Page variable, you can indicate which AFP page buffer is to be opened with the “Open Page” function, or switch to the page buffer again before you further format that page, or close that page.

Syntax

```
void ClosePage(
                ushort       docNo = 1
               );
```

Parameters

docNo
Specifies to which AFP document to output AFP page, valid values are 1 through 10, the default value is 1.

Sample

```
void main( )
{
  Start();
  SetUnit(CM_U600);
  OpenDoc();                    // Open first AFP document
     :
  $Page = 3;                    // Indicate to open page buffer 3
  GetPage();                    // Get an AFP page to page buffer 3
     :
  ClosePage();                  // Close AFP page 3, write to AFP file
     :
  CloseDoc();                   // Close AFP document and its file
}
```

Color for Text

Function

Specifies the color for the subsequent texts and legacy text lines/boxes.

Syntax

For OCA color:

```
Color(
      ocacolor        ocacolor = BLACK
     );
```

For RGB color:

```
ColorRGB(
         UCHAR        red_color,
         UCHAR        green_color,
         UCHAR        blue_color
        );
```

For CYMK color:

```
ColorCMYK(
          UCHAR       cyan_color_ percentage,
          UCHAR       yellow_color_ percentage,
          UCHAR       magenta_color_ percentage,
          UCHAR       black_color_ percentage
         );
```

Parameters

ocacolor
Any of the defined MO:DCA OCA color values: BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, MEDIUM or WHITE, and GRAY, the default value is BLACK.

RGB values
Valid RGB intensity range values for each component are 0 through 255.

CYMK color percentage values
Valid CYMK percentage range values for each component are 0 through 100.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
              :
Pos(5,5);                              // current position at (5,5) mm
ColorRGB(255,0,0);                     // RGB red color
Ltxt("RGB Red Color Text");
Pos(5,10.);
ColorCMYK(0,0,0,100);                  // CYMK black color
Ltxt("CYMK Black Color Text");
Pos(5,15.);
Color(CYAN);                           // AFP OCA CYAN color
Ltxt("AFP OCA CYAN Color Text");
              :
ClosePage();
CloseDoc();
```

Color Management Resource Association

Function

Associates a CMR (Color Management Resource) with the subsequent pages or an overlay created by MakeAFP Weaver.

This function can be repeated to associate all CMRs required.

Color management resources (CMRs) are the foundation of color management in AFP print systems. They are AFP resources that provide all the color management information, such as ICC profiles and halftones, that an AFP system needs to process a print job and maintain consistent color from one device to another.

IPDS printer manufacturers and groups that support AFP color standards create CMRs that you can use in your color printing systems.

Syntax

Invokes CMR Association:

```
void CMR(
         ushort        cmr_id,
         mode          process_mode = AUDIT
        );
```

Revokes CMR Association:

```
void RevokeCMR( );
```

Parameters

cmr_id
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Weaver definition file with a CMR parameter. Default value 0 specifies that CMR is not being defined.

process_mode
Specifies the processing mode for the CMR:

`AUDIT` The audit processing mode. Refers to processing that has already been applied to a resource. In most cases, audit CMRs describe input data and are similar to ICC input profiles. The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

`INSTR` The instruction processing mode. Refers to processing that is done to prepare the resource for a specific printer using a certain paper or another device. Generally, instruction CMRs refer to output data and are similar to ICC output profiles.

The instruction processing mode is used with color conversion, tone transfer curve, and halftone CMRs. In instruction processing mode, these CMRs indicate how the system must convert a resource so it prints correctly on the target printer. The manufacturer of your printer should provide ICC profiles or a variety of CMRs that you can use. Those ICC profiles and CMRs might be installed in the printer controller, included with the printer on a CD, or available for download from the manufacturer's Web site.

Sample

```
SetUnit(MM_U600);

OpenDoc();

               :

CMR(1, INSTR);                  // Invoke a CMR association for the
                                // subsequent pages, ID 1 of CMR
                                // was predefined in the MakeAFP
definition
                                // file with parameter CMR1

CMR(2, AUDIT);

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

               :

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

RevokeCMR();                    // revoke CMR association

CloseDoc();
```

Copy Group

Function

Invokes an AFP copy group name that was previously defined in the form definition.

With copy groups (also called medium map) predefined in the form definition to be called, you can select the form-mapping controls dynamically (such as input paper bin, duplex, control N-UP partition, etc) for the subsequent pages, and define color rendering and CMR(Color Management Resource) association for the whole AFP file or group pages, refer to latest IBM *Page Printer Formatting Aid User's Guide* for more information.

Syntax

```
void CopyGroup(
               char *     copygroup_name,
               ushort     docNo
              );
```

Parameters

copygroup_name
The copygroup name with a maximum of up to 8 characters for the current page and subsequent pages. Make sure the copygroup name matches exactly with the name of the copy group that was previously defined in your AFP form definition, which must be called during your print job submission.

docNo
Specifies to which AFP document to insert the command of invoking copy group to, valid values are 1 through 10, the default value is 1.

Sample

```
SetUnit(MM_U600);

OpenDoc();
             :
             :
OpenPage(210,297);

             :

ClosePage();

OpenPage(210,297);

CopyGroup("F2TRAY2");          // Call copy group F2TRAY2 you defined in
                               // form definition, for use the paper
                               from
                               // input paper tray 2 for this page and
                               // subsequence pages

             :

ClosePage();
             :
             :
```

```
        CloseDoc();
```

Default Language Locale

Function

Defines the Locale name of your language, to be used to control the text boundary-breaking of a paragraph.

Make sure you have defined a correct locale name before calling paragraph functions.

Syntax

```
void     DefaultLocale(
                          char        *localeName = "en_US"
                        );
```

Parameters

localName
The Locale name of your language, MakeAFP Weaver default is “en_US” if this function is not called.

Refer to MakeAFP document *How to specify a Locale for* more details about the locale names.

Encoding of AFP and PC Native

Function

Defines the default encoding names of your AFP document and your PC native, so that MakeAFP Weaver converts your non-PC native encoded AFP index values or text fields to PC native encoding automatically.

This function must be called before processing your non-PC native encoded AFP or can be recalled again for any of the default encoding changes if needed.

Syntax

```
void      Encoding(
                    char          *afp_code,
                    char          *pc_code
                   );
```

Parameters

afp_code
The name of the default encoding of your AFP in EBCDIC, mixed SBCS-HOST/DBCS-HOST, UTF-8, and UTF-16. Refer to MakeAFP document *Encoding Names for* more details about the available names.

pc_code
The name of the native default encoding of your PC in ASCII, mixed SBCS-PC/DBCS-PC. Refer to MakeAFP document *Encoding Names for* more details about the available names.

Sample

```
/*************************************************************************/
/* This sample shows how to mask an area, capture an index value        */
/* as the part of string for add a barcode, and add a page segment       */
/*                                                                       */
/* Indexed AFP was encoded in CP-037, USA EBCDIC                         */
/*************************************************************************/

int main( )
{
  unsigned int i, grpPages, pageSN = 0;
  char tmp[80], policyNo[20];

  $MaxPaging = 50;            // Maximum paging is up to 50 pages

  SetUnit(IN_U600);              // Set default unit to inch

  Start();                    // Start initiation, open default input,
                              // output and definition files, retrieves
                              // AFP resources, allocate memory

  Encoding("ibm-037","ibm-437");   // AFP – CP037, PC - CP437

  OpenDoc();                  // Open an AFP document

  while ($Edt == 0)           // Until end of AFP document
  {
    $Page = 0;                // Reset AFP page buffer number

    do {
      $Page++;                // Point to next AFP page buffer

      GetPage();              // Get a page from existing AFP file

    } while ($Eng == 0);      // Until end of each page group


    // Now got all pages of a page group, now it is
    // ready to compose the new AFP output

    grpPages = $Page;         // keep total number of pages per group

    for (i = 0; i < grpPages; i++)
    {
      $Page = i + 1;          // Point to page buffer number to be opened
   again

      InclPseg("S1OWL", 0.3, 0.25);    // Add a new page segment image

      MaskArea(5, 0.4, 2, 0.75); // Mask an area on every page

      sprintf(tmp, "Page %d of %d", $Page, grpPages); // Generate pagination
      Font(1);  Pos(8, 0.45);  Rtxt(tmp);

      sprintf(tmp, "%06d", ++pageSN);  // generate page serial number

      Font(2);  Pos(0.2, 10.8);  Ltxt(tmp);  // With MakeAFP Weaver, you can
                                                                   // use a
   font encoded in ASCII                                                  //
   directly

      GetIdx("Policy", policyNo);            // MakeAFP Weaver does auto-
                                             // conversion with the
                                             // encoding names defined with
                                                    // Encoding() function

      sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);

      BarCode(CODE128, tmp, 0.3, 2, 2, 0.2, DEG90);  // Add 1D barcode 128
      DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);           // Add 2D DataMatrix

      ClosePage();            // Close AFP page, write each page to AFP file
    }
  }

  CloseDoc();                 // Close AFP document and its file

  #ifdef      _DEBUG
    ViewAFP();                // Only view AFP output in debug mode
  #endif

  return 0;
}
```

End Index Group

Function

Ends an index page group previously started with a “Begin Index Group” call.

Index Groups cannot be nested or overlapped. Each index group must be ended before another can begin.

Syntax

```
void EndIdx(
            ushort      docNo = 1
           );
```

Parameters

docNo
Specifies to which AFP document to insert the index information, valid values are 1 through 10, the default value is 1.

Sample

```
 // Now got all AFP pages in the AFP page buffers, before write
        // out all of the pages of a page group, we can insert
 // beginning of group index tag and index value tags


unsigned short numpages = $Page;        // Keep total pages of a client
char tmp[25];
int groups;
sprintf(tmp, "%08d", ++groups);

BgnIdx(tmp);                            // 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
{
  $Page = i + 1;                        // Switch to each page buffer
  sprintf(tmp, "Page %d  of  %d", $Page, 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 each
  ClosePage();                                 // page
}
EndIdx();                               // End index page group, must be
                                               // called after writing of
the last
                                        // page of each page group

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

End of Document File of Input AFP (Checking)

Function

Tests for end of document of input AFP file.

Returns 1, if the end of AFP document file has been detected, after the “Get Page” function is called for the reading of an AFP page from the input AFP file, or 0 if it is not.

This function is mainly developed for calling from other programming languages; with Visual C++, you can use $Edt variable directly.

Syntax

```
Bool Edt(void);
```

Parameters

None.

Sample

None.

End of Index Name Group of Input AFP (Checking)

Function

Tests for end of index name group of input AFP file.

Returns 1, if the end of name group of AFP index boundary has been detected, after “Get Page” function is called for the reading of an AFP page from the input AFP file, or 0 if it is not.

This function is mainly developed for calling from other programming languages; with Visual C++, you can use $Eng variable directly.

Syntax

```
Bool Eng(void);
```

Parameters

None.

Sample

None.

End Overstrike

Function

Ends overstriking of text previously started with a “Begin Overstrike” function call.

Syntax

```
void EndOstrike();
```

Parameters

No parameter to be specified.

Sample

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

BgnOstrike();                         // begin overstriking of text

Ltxt("This is an overstrike text");   // text will be overstriked
               :
               :

EndOstrike()                          // end overstriking of text
               :
ClosePage();
CloseDoc();
```

End Underscore

Function

Ends underscoring of text previously started with a “Begin Underscore” function call.

Syntax

```
void EndUscore(
                void
               );
```

Parameters

No parameter to be specified.

Sample

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

BgnUscore();                         // begin underscoring of text

Ltxt("This is an underscore text");  // text will be underscored
              :
              :

EndUscore()                          // end underscoring of text
              :
ClosePage();
CloseDoc();
```

Font Definition

Function

Define the font ID number(s) to be used as your current font(s) for the subsequent texts. You must define your font(s) in your MakeAFP Weaver definition file before you call the font ID numbers.

For your convenience, you can define a constant variable as your local alias name for each font ID, refer to the following sample for more details.

Syntax

For AFP output in encoding of ASCII, EBCDIC, UTF-8 and UTF-16:

```
void Font(
          int       fontid
         );
```

For AFP output in encoding of mixed ASCII/DBCS-PC, EBCDIC/DBCS-HOST:

```
void Font2(
           int      SBCS_fontid,
           int      DBCS_fontid
          );
```

Parameters

fontid
The ID number of the ASCII / EBCDIC / UTF-8 / UTF16BE font, which is defined in your MakeAFP Weaver definition file with FONT parameter.

SBCS_fontid
The ID number of the ASCII / EBCDIC font, which is defined in your MakeAFP Weaver definition file with FONT parameter.

DBCS_fontid
The ID number of the DBCS-PC / DBCS-HOST font, which is defined in your MakeAFP Weaver definition file with FONT parameter.

Sample

AFP fonts defined in the MakeAFP definition file:

```
fontlib = c:\makeafp\reslib      // Font resources directory
font1 = czh200,t1000437,11       // Font 1, SBCS font, Helvetica, point size is 11
font2 = czsong,t11385,11         // Font 2, DBCS font for Simplified-Chinese
font3 = czn400,t1000437,14       // Font 3, SBCS font, TimesNewRoman, 14 points
```

AFP fonts' calls in the MakeAFP Weaver program:

```
const in helv11 = 1;          // define local alias name helv11 for font 1
const int song11 = 2;         // define local alias name song11 for font 2
const int times14 = 3;        // define local alias name times14 for font
               :
Pos(4.5, 2.5 );
Font(times14);                // use font times14 for the subsequent texts
Ltxt("Testing text");
Pos(6, 4);
Font2(helv11, song11);        // use fonts helv11 and song11 for the
                              // subsequent SBCS-HOST/DBCS-HOST output
```

Csc2("实现 Win2000 与 Linux 的双引导");

Font ID Query

Function

Queries the current font ID number.

Syntax

`int FontID();`	Returns the current SBCS or UTF-8 / UTF-16 font ID
`int FontID2();`	Returns the current DBCS font ID

Sample

```
SetUnit(IN_U600);

OpenDoc();

OpenPage(8.27, 11.67);
               :
               :
if ( FontID() == 2 )                // if current Font ID is 2
{
               :
               :
}
else
{
               :
               :

}

ClosePage();
CloseDoc();
```

Get Index Value

Function

Retrieves the index values of page group level or page level index from an indexed AFP.

This function can be called to retrieve the text strings of an index value, after an indexed page group, or the first page of page group or a page is read-in by the “Get Page” function.

MakeAFP Weaver converts the index value of non-PC native encoded AFP to the ASCII or ASCII/DBCS-PC native encoding if AFP and PC encoding are specified by the “Encoding” function. Make sure the “Encoding” function is called before this function is called.

Syntax

```
void *GetIdx(
              char           *index_name,
              char           *index_value
              );
```

Parameters

Index_name
Specifies an index name of page group level index or page level index.

With MakeAFP ShowIDX utility, you can dump index information of index names and values from an indexed AFP file.

Index_value
Specifies a variable to store the text string of index value retrieved, make sure the “Encoding” function is previously called so that the non PC native AFP text or index string can be converted to native ASCII or ASCII/DBCS-PC encoding automatically.

Sample

```
/*************************************************************************/
/* This sample shows how to mask an area, capture an index value         */
/* as the part of string for add a barcode, and add a page segment       */
/*                                                                       */
/* AFP was encoded in CP-037, USA EBCDIC                                 */
/*************************************************************************/

int main( )
{
  unsigned int i, grpPages, pageSN = 0;
  char tmp[80], policyNo[20];

  $MaxPaging = 50;            // Maximum paging is up to 50 pages

  SetUnit(IN_U600);              // Set default unit to inch

  Start();                    // Start initiation, open default input,
                              // output and definition files, retrieves
                              // AFP resources, allocate memory

  Encoding("ibm-037","ibm-437");   // AFP – CP037, PC - CP437

  OpenDoc();                  // Open AFP document

  while ($Edt == 0)           // Until end of AFP document
  {
}

    $Page = 0;                // Reset AFP page buffer number

    do {

      $Page++;                // Point to next AFP page buffer

      GetPage();              // Get a page from existing AFP file

    } while ($Eng == 0);      // Until end of each page group


    // Now got all pages of a page group, now it is
    // ready to compose the new AFP output

    grpPages = $Page;         // keep total number of pages per group

    for (i = 0; i < grpPages; i++)
    {
      $Page = i + 1;          // Point to page buffer number to be opened
    again

      InclPseg("S1OWL", 0.3, 0.25);    // Add a new page segment image

      MaskArea(5, 0.4, 2, 0.75); // Mask an area on every page

      sprintf(tmp, "Page %d of %d", $Page, grpPages); // Generate pagination

      Font(1);  Pos(8, 0.45);  Rtxt(tmp);    // With MakeAFP Weaver, you can
                                                                   // use a
    font encoded in ASCII                                                     //
    directly


      sprintf(tmp, "%06d", ++pageSN);  // generate page serial number
      Font(2);  Pos(0.2, 10.8);  Ltxt(tmp);

      GetIdx("Policy", policyNo);            // MakeAFP Weaver does auto-
                                             // conversion with the
                                             // encoding names defined with
                                                    // Encoding() function

      sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);

      BarCode(CODE128, tmp, 0.3, 2, 2, 0.2, DEG90);  // Add 1D barcode 128
      DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);          // Add 2D DataMatrix

      ClosePage();            // Close AFP page, write each page to AFP file
    }
  }

  CloseDoc();                 // Close AFP document and its file

  #ifdef      _DEBUG
    ViewAFP();                // Only view AFP output in debug mode
  #endif

  return 0;
}
```

Get No-Operation Value

Function

Gets a NOP (No Operation) stream from your input AFP page without any data string conversion.

It must be called after the GetPage( ) function and can be called multiple times to get multiple NOP values.

The No Operation AFP structured field may be used to carry comments or any other type of special stream or instruction, such as carry semantic data or command in private or exchange data streams.

Syntax

```
Char *GetNop(
              char *nop_value
             );
```

Parameters

nop_value
The NOP value retrieving from AFP, GetNop ( ) does not perform any data conversion, it should be done by your coding if it is required.

Sample

```
Char nop_value1[512], nop_value2[512];

SetUnit(IN_U600);

$MaxPaging = 100;              // Maximum page buffers are 100, must be
                               // defined before Start() function call

Start();                       // Start a MakeAFP Weaver session

OpenDoc();
             :

$Page = 3;                     // Get AFP page into page buffer 3
GetPage();

GetNop(nop_value1);            // Get first NOP value from page 3

GetNop(nop_value2);            // Get second NOP value from page 3


             :
ClosePage();                   // Close page 3 and write to AFP output
file
             :
$Page = 15;                    // Get AFP page into page buffer 15
GetPage();
             :
ClosePage();                   // Close page 15 and write to AFP output
file
```

Get Page – Getting an Existing AFP Page

Function

Gets an existing AFP page from the AFP input file. You can write out the page with the “Close Page” function once you have completed the processes to the page.

If your input AFP stream is encoded in the non-PC native encoding. You have to make sure the “Encoding” function is called before the “Open Page” function so that the conversion from the AFP encoding to the native ASCII or ASCII/DBCS-PC encoding can be performed automatically for the strings of index values and text fields to be retrieved

With the $MaxPaging variable or the “Maximum Paging” function, you can define the maximum number of AFP page buffers. For generating OMR and page pagination, such as “Page 347 of 1000”, we need to keep composed AFP data in the AFP page buffers first.

With MakeAFP Weaver, you can open multiple pages by either the “Get Page” or the “Open Page” functions, and then process different pages in an interleaved manner once each page is initialized, all the composed AFP data stream will be kept in memory buffers in page-level, and finally, after you have completed all the formatting and counted all the pages of a page group, you can put your OMR and pagination text on each page just before you close the page with the “Close Page” function.

With $Page variable, you can indicate which AFP page buffer is to be opened with the “Open Page” function, or switch to the page buffer again before you further format or end that page.

Syntax

```
void GetPage(
              bool          remove_imm = false
             );
```

Parameters

remove_imm
Specifies whether to remove the IMM (Invoke Medium Map, also called copy-group) from the AFP output, so that you can insert your new copy-group.

Sample

```
SetUnit(IN_U600);

$MaxPaging = 100;              // Maximum page buffers are 100, must be
                               // defined before Start() function call

Start();                       // Start a MakeAFP Weaver session

OpenDoc();
            :

$Page = 3;                     // Get AFP page into page buffer 3
GetPage();
            :
ClosePage();                   // Close page 3 and write to AFP output
file
            :
$Page = 15;                    // Get AFP page into page buffer 15
GetPage();
            :
```

```
ClosePage();                   // Close page 15 and write to AFP output
file
```

Get Text Field by a Location

Function

Captures a text field string by the coordinate location of the data field on an AFP page.

This function can be called to capture the text strings of a data field after the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

MakeAFP Weaver converts the data field of non-PC native encoded AFP to the ASCII or ASCII/DBCS-PC native encoding if AFP and PC encoding are specified by the “Encoding” function. Make sure the “Encoding” function is called before this function is called.

Syntax

```
char *GetField(
                 ushort       x,
                 ushort       y,
                 char*        field_value,
                 ushort       field_no = 1
              );
```

Parameters

x
Specifies the X position of data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y
Specifies the Y position of a data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

field_value
Specifies a variable to store the text string of the data field captured, make sure the “Encoding” function is previously called so that the non-PC native AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically.

field_no
Specifies the order number of the AFP data field from which the data field is being captured from, the default value is 1, but sometimes several fields can be referenced from the same (x, y) position, in this case, you must make sure which field is being captured.

Sample

```
/***************************************************************************/
/* This sample shows how to detect a trigger by an overlay name and        */
/* capture data fields from page 1, add barcode to existing AFP            *
/*                                                                         */
/* AFP was encoded in CP-037, USA EBCDIC                                   */
/***************************************************************************/

int main( )
```

```
                      {
                         unsigned int i, grpPages, pageSN = 0;
                         char tmp[80], mobileNo[20];
                         bool bog = 0;

                         $MaxPaging = 50;        // Maximum paging is up to 50 pages

                         SetUnit(IN_U600);       // Set default unit to inch

                         Start();                // Start initiation, open default input,
                                                 // output and definition files, retrieves
                                                 // AFP resources, allocate memory

                         Encoding("ibm-037","ibm-437");

                         OpenDoc();              // Open AFP document

                         $Page = 1;              // Set AFP page buffer number to 1 for the first
                                                 // page of AFP file

                         GetPage();              // Get first page of AFP file

                         while ($Edt == 0)       // Until end of AFP document
                         {
                           GetField(4050, 900, mobileNo);      // Get customer mobile number

                           do {

                             $Page++;            // Point to next AFP page buffer

                             GetPage();          // Get next page

                             // Detecting if it is the first page of a group,
                             // overlay O1OVL1E only used by at first page of
                             // each page group
                             bog = TriggerOvly("O1OVL1E");

                           } while (!bog && !$Edt);  // Until beginning of next page group or end
of
                                                     // AFP file

                           bog = 0;                  // reset it for next group

                           // Now got all pages of a page group and first page of next group, now
it                                // is ready to compose the new AFP output

                           if (!$Edt)                // If not end of AFP document
                             grpPages = $Page -1 ;   // keep total number of pages per group, need
to
                                                     // minus 1 page of the first page of next
group

                           for (i = 0; i < grpPages; i++)
                           {
                             $Page = i + 1;          // point to page buffer number to be opened
again

                             if (*mobileNo)
                             {
                               sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
                               BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90);
                             }

                             ClosePage();              // Close AFP page, write to AFP file
                           }
```

```
                          MovePage(1, grpPages + 1);  // As we got first page of next group
previously,
                                                      // now need to move its contents to page
                                                      // buffer 1 for the next page group
                          $Page = 1;
                        }

                        CloseDoc();                   // Close AFP document and its file

                        return 0;
                     }
```

Get Text Field by a Location Area

Function

Captures a text field string by the coordinate location range of the data field on an AFP page.

This function can be called to capture the text strings of a data field after the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

MakeAFP Weaver converts the data field of non-PC native encoded AFP to the ASCII or ASCII/DBCS-PC native encoding if AFP and PC encoding are specified by the “Encoding” function. Make sure the “Encoding” function is called before this function is called.

Syntax

```
char *GetField2(
                ushort          x1,
                ushort          x2,
                ushort          y1,
                ushort          y2,
                char*           field_value,
                ushort          field_no = 1
               );
```

Parameters

x1, x2
Specifies the X position range of data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y1, y2
Specifies the Y position range of data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

field_value
Specifies a variable to store the text string of the data field captured, make sure the “Encoding” function is previously called so that the non-PC native AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically.

field_no
Specifies the order number of the AFP data field from which the data field is being captured, the default value is 1, but sometimes several fields can be referenced from the same (x, y) position, in this case, you must make sure which field is being captured.

Sample

```
      /***************************************************************************/
                  /* This sample shows how to capture a trigger from last page of each group,
*/
```

```
                      /* get a feild from page 1 for add a barcode, mask an area and add a page
*/
                      /* segment.
*/
                      /*
*/
                      /* AFP was encoded in CP-037, USA EBCDIC
*/

       /*****************************************************************************/

                      int main( )
                      {
                        unsigned int i, grpPages, pageSN = 0;
                        char tmp[80], policyNo[20];
                        bool eog = 0;

                        $MaxPaging = 50;            // Maximum paging is up to 50 pages

                        SetUnit(IN_U600);           // Set default unit to inch

                        Start();                    // Start initiation, open default input,
                                                    // output and definition files, retrieves
                                                    // AFP resources, allocate memory

                        Encoding("ibm-037","ibm-437");   // AFP - CP037, PC - CP437

                        OpenDoc();                  // Open AFP document

                        while ($Edt == 0)           // Until end of AFP document
                        {
                          $Page = 0;                // Reset AFP page buffer number

                          do {

                          $Page++;                  // Point to next AFP page buffer

                          GetPage();                // Get a page from existing AFP file

                          if ($Page == 1)           // Get policy number from page 1
                            GetField2(2448, 2448, 6080, 6110, policyNo);

                          if ($Page > 2)
                            eog = Trigger(3744, 2338, "Part 1");  // detecting if it is a last page
of                                                               // a page group, "Part 1" text
                                                                        // string only appears at
last page
                                                                 // of each page group

                        } while (!eop);             // Until end of each page group

                        eop = 0;                    // reset it for next group

                        // Now got all pages of a page group, now it is
                        // ready to compose the new AFP output

                        grpPages = $Page;           // keep total number of pages per group

                        for (i = 0; i < grpPages; i++)
                        {
                          $Page = i + 1;            // point to page buffer number to be opened
again

                          InclPseg("S1OWL", 0.3, 0.25);  // Add a page segment image
                          MaskArea(5, 0.4, 2, 0.75);     // Mask an area on every page
                          sprintf(tmp, "Page %d of %d", $Page, grpPages); // generate pagination

                          Font(1);  Pos(8, 0.45);  Rtxt(tmp);     // With MakeAFP Weaver you can
use
                                                                  // an ASCII encoded font directly

                          sprintf(tmp, "%06d", ++pageSN);         // generate page serial number
                          Font(2);  Pos(0.2, 10.8);  Ltxt(tmp);

                          sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);
                          BarCode(CODE128, tmp, 0.3, 2, 2, 0.2, DEG90);   // Add 1D and 2D barcodes
                          DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);

                          ClosePage();                // Close each AFP page, write to AFP file
                        }
                      }

                      CloseDoc();                     // Close AFP document and its file

                      return 0;
                    }
```

Get Text Field by a Location Area and a Pettern

Function

Captures a text field string by the coordinate location range of the data field on an AFP page and a matching pattern of symbols.

This function can be called to capture the text strings of a data field after the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

MakeAFP Weaver converts the data field of non-PC native encoded AFP to the ASCII or ASCII/DBCS-PC native encoding if AFP and PC encoding are specified by the “Encoding” function. Make sure the “Encoding” function is called before this function is called.

Syntax

```
char *GetField3(
               ushort          x1,
               ushort          x2,
               ushort          y1,
               ushort          y2,
               char*           field_value,
               char*           pattern
              );
```

Parameters

x1, x2
Specifies the X position range of data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y1, y2
Specifies the Y position range of data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

field_value
Specifies a variable to store the text string of the data field captured, make sure the “Encoding” function is previously called so that the non-PC native AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically.

pattern
Specifies a character string or a pattern of symbols to be used for picking up a set of the character string that matches with the specified pattern. Valid pattern symbols are:

'@’	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

Sample

```
       /***************************************************************************/
       /* This sample shows how to capture a trigger and field from page 1 only, */
       /* for adding barcode to existing AFP.   AFP was encoded in CP-437, ASCII */
       /***************************************************************************/
       int main( )
       {
         unsigned int i, grpPages, pageSN = 0;
         char tmp[80], savingsNo[20];
         bool bog = 0;

         $MaxPaging = 50;        // Maximum paging is up to 50 pages

         SetUnit(IN_U600);       // Set default unit to inch

         Start();                // Start initiation, open default input,
                                 // output and definition files, retrieves
                                 // AFP resources, allocate memory

         OpenDoc();              // Open AFP document

         $Page = 1;              // Set AFP page buffer number to 1 for the first
page
                                 // of AFP file

         GetPage();              // Get first page of AFP file

         while ($Edt == 0)       // Until end of AFP document
         {
                                 // Get Savings A/C number from page 1 by a mask
           GetField3(180, 180, 1080, 1120, savingsNo, "###-####-####");

           do {

             $Page++;            // Point to next AFP page buffer

             GetPage();          // Get next page

             // detecting if it is the first page of a group,
             // "Page 1 of" text string only appears at
             // first page of each page group
             bog = Trigger2(2187, 2187, 1120, 1200, "Page 1 of");

          } while (!bog && !$Edt);  // Until beginning of next page group or end of
                                           // AFP file

          bog = 0;                  // Reset it for next group

          // Now got all pages of a page group and first page of next group, now it
is
          // ready to compose new AFP output

          if (!$Edt)
            grpPages = $Page -1 ;   // keep total number of pages per group, need to
                                    // minus 1 page of the first page of next group

          for (i = 0; i < grpPages; i++)
          {
            $Page = i + 1;          // Point to page buffer number to be opened
again

            sprintf(tmp, "%d %d %s", ++pageSN, $Page, savingsNo);
            DataMatrix(tmp, 0.2, 2.2, 0.4, 0.4);

            ClosePage();            // Close AFP page, write to AFP file
          }

          MovePage(1, grpPages + 1);  // As we got first page of next group previously,
                                      // now need to move its contents to page buffer
1
                                      // for next page group
          $Page = 1;                  // Reset page buffer number for next group page
1
        }

        CloseDoc();                   // Close AFP document and its file

        return 0;
      }
```

Get Text Field by a X-location Range and a Pettern

Function

Captures a text field string by the X-location range of the data field on an AFP page and a matching pattern of symbols.

This function can be called to capture the text strings of a data field once the AFP page is read-in by the “Get Page” function.

With “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

MakeAFP Weaver converts the data field of non PC native encoded AFP to the ASCII or ASCII/DBCS-PC native encoding, if AFP and PC encoding are specified by the “Encoding” function. Make sure the “Encoding” function is called before this function is called.

Syntax

```
char *GetFieldX(
               ushort          x1,
               ushort          x2,
               char*           field_value,
               char*           pattern
              );
```

Parameters

x1, x2
Specifies the X position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

field_value
Specifies a variable to store the text string of the data field captured, make sure the “Encoding” function is previously called so that the non-PC native AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically.

pattern
Specifies a character string or a pattern of symbols to be used for picking up a set of the character string that matches with the specified pattern. Valid pattern symbols are:

'@’	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

Sample

None.

Get Text Field by a Y-location Range and a Pattern

Function

Captures a text field string by the Y-location range of the data field on an AFP page and a match pattern of symbols.

This function can be called to capture the text strings of a data field once the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

MakeAFP Weaver converts the data field of non-PC native encoded AFP to the ASCII or ASCII/DBCS-PC native encoding if AFP and PC encoding are specified by the “Encoding” function. Make sure the “Encoding” function is called before this function is called.

Syntax

```
char *GetFieldY(
                ushort          y1,
                ushort          y2,
                char*           field_value,
                char*           pattern
              );
```

Parameters

y1, y2
Specifies the Y position range of data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

field_value
Specifies a variable to store the text string of the data field captured, make sure the “Encoding” function is previously called so that the non-PC native AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically.

pattern
Specifies a character string or a pattern of symbols to be used for picking up a set of the character string that matches with the specified pattern. Valid pattern symbols are:

'@’	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

Sample

None.

Get Text Field Position

Function

Gets the location of a text field. It returns a TRUE bool if the text field is found.

Syntax

```
bool GetFieldPos(
                 ushort        x1,
                 ushort        x2,
                 ushort        y1,
                 ushort        y2,
                 char *        mask
                 ushort &      x_found,
                 ushort &      y_found
                );
```

Parameters

x1, x2
Specifies the X position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y1, y2
Specifies the Y position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

mask
Specifies a native text string or a pattern of symbols to be used to identify the data field captured from the AFP page. Make sure the “Encoding” function is previously called so that the non-ASCII or non-ASCII/ DBCS-PC AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically for the comparison with the string or pattern of symbols specified in the native encoding. Valid pattern symbols are:

'@’	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

x_found, y_found
The variables returning X and Y position of the text field in PELS.

Sample

```
int main( )
{
  UINT grpPages;
  USHORT x_found = 0, y_found = 0;
  bool eog = 0;
  char account[20];

  $MaxPaging = 100;                        // Maximum paging is up to 100
pages
```

```
     SetUnit(IN_U600);        // Set default unit to INCH

     Start();                // Start initiation, open default input,
                             // output and definition files, retrieves
                             // AFP resources, allocate memory

     OpenDoc();              // Open AFP document

     while ($Edt == 0)       // Process until end of AFP document
     {
       $Page = 0;

       do {

         $Page++;             // Point to next AFP page buffer

         GetPage();          // Get a page from existing AFP file

         if ($Page == 1)
         {
           GetField(443, 447, account, 1);     // Get A/C number from page 1
           Box(0.7, 1.3, 2.5, 1.23, 0.007);    // draw a box around the address

           // Add 1d and 2D barcode
           BarCode(CODE128, account, 8.77, 2.5, 1.5, 0.37, DEG90);
           QRCode(account, 8.4, 1);
         }
         else
         {
           if(GetFieldPos(136, 136, 700, 2000, "Subtotal", x_found, y_found))
           {
             Vline(0.324, 2.1, (float )y_found/300 - 2, 0.007);   // AFP is 300
PELS
             Vline(8.04, 2.1, (float )y_found/300 - 2, 0.007);
             Hline(0.324, (float )y_found/300.0 + 0.1, 7.72, 0.007);
           }
         }

       } while ($Eng == 0);      // Until end of each indexed page group

      // Now got all pages of a page group, now it is
      // ready to compose the new AFP output

      grpPages = $Page;         // keep total number of pages per group

      for ($Page = 1; $Page <= grpPages; $Page++)
        ClosePage();            // Closeof each AFP page, write to AFP file
   }

   CloseDoc();                  // Close AFP document and its file

   #ifdef _DEBUG
     ViewAFP();                 // Only view AFP output in debug mode
   #endif

  return 0;
}
```

Goto Page

Function

Indicates which AFP page buffer is to be opened with the “Open Page” function, or switch to the page buffer again before you further format that page, or close that page.

This function is mainly developed for calling from other programming languages; with Visual C++, you can switch to any AFP page buffer directly by the MakeAFP Weaver $Page variable.

Syntax

```
void GotoPage(
              ushort      pageNo
             );
```

Sample

```
SetUnit(IN_U600);

MaxPaging(1000);               // Sets maximum of page buffers to 1000,
                               // it must be called before Start()
function

Start();

OpenDoc();
            :
GotoPage(3);                     // switch to page buffer 3

OpenPage(8.5,11);
            :
ClosePage();
            :

GoPage(15);                    // switch to page buffer 15

OpenPage(8.5,11);
            :
ClosePage();
```

Horizontal Line

Function

Draws a horizontal line.

Syntax

```
void Hline(
            float          x_pos,
            float          y_pos,
            float          length,
            float          thickness,
           );
```

Parameters

x_pos
The X starting position of the line, specify CP if you want to use the current position.

y_pos
The Y starting position of the line, specify CP if you want to use the current position.

length
The length of the line.

thickness
The thickness of the line.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
              :
              :
Color(RED);                      // defines color for the legacy line

Hline(10,10,100,1);              // draws a horizontal blue line from
                                 // (10,10)mm, its length is 100 mm,
                                 // thickness is 1 mm
              :
              :

ClosePage();
CloseDoc();
```

Horizontal Lines

Function

Repeat drawing of horizontal lines.

Syntax

```
void Hlines(
            float        x_pos,
            float        y_pos,
            float        length,
            float        thickness,
            ushort       repeat,
            float        space,
            ushort       direction = DOWN
           );
```

Parameters

x_pos
The X starting position of the line, specify CP if you want to use the current position.

y_pos
The Y starting position of the line, specify CP if you want to use the current position.

length
The length of the line.

thickness
The thickness of the line.

repeat
The number of additional lines to be repeated.

space
The gap space between the lines.

direction
The direction of line repeating, valid values are ACROSS and DOWN, default is DOWN.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
             :
             :

Color(BLUE);                    // defines color for texts and legacy
                                   lines

Hlines(10,10,100,1,7,5,BLUE);   // draw 8 horizontal blue line from
                                // (10,10)mm, its length is 100 mm,
                                // thickness is 1 mm, space is 5mm
             :
             :

ClosePage();
CloseDoc();
```

Inch Value

Function

Specifies a value in inches.

Syntax

```
float inch(
          float     value
         );
```

Parameters

value
The value in inches.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
            :
            :
Pos(10,10);                  // set X and Y position to (10,10)mm
            :
            :
Pos( inch(2), 35);           // set X position to 2" and Y position to
                             // 35mm
            :
            :

ClosePage();
CloseDoc();
```

Include Data-Object

Function

Includes a reference to an AFP object(image, graphic, barcode), or non-AFP data-object(TIFF, JPEG, GIF, etc) at the specified position or current position, and specifies the area size, rotation, mapping option, color rendering intent, and a CMR (Color Management Resource) for the object to be printed.

Using a data-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.

Note: This feature requires appropriate AFP print servers and IPDS printer microcodes support.

Syntax

```
void InclObjt(
              char*              object_name,
              float              object_xpos,
              float              object_ypos,
              float              object_width = DEFAULT,
              float              object_height = DEFAULT,
              ushort             dpi_resolution = DEFAULT,
              mapping            object_mapping = FIT,
              ushort             cmr_id = 0,
              cmr_mode           process_mode = AUDIT,
              render_type        rendering_intent = NONE,
              rotate             object_rotation = DEG0,
              ocacolor           object_color = NONE
             );
```

Parameters

obecjt _name
The name of the data-object previously defined in your MakeAFP Weaver definition file with an OBJT parameter, you must use your data-object file base name exclusive of filename-extension as the data-object name, maximum of 125 characters are allowed, and valid characters are A-Z, 0-10, _ (underscore), #, and @. The data-object file must be available to MakeAFP at the time of formatting.

When MakeAFP Weaver finds more than one data-object image with the same base filename in the same object directory, it selects the matching data-object image by the following file extension search order:

1. No filename extension
2. JPG
3. TIF
4. GIF
5. JP2
6. EPS
7. PDF
8. BMP
9. PCX
10. PCL
11. OBJ

Note: Some file extensions may not be supported by your AFP print server.
Using legacy AFP object naming is recommended, which allows one to eight characters as the base filename. Your AFP print server may support the data-object resource file that has No filename extension or with extension .obj.

If the name of the data-object is more than 8 bytes and it is not embedded inline in AFP, then it must be installed in a resource library using software such as AFP Resource Installer.

obecjt_xpos
The X position of the object..

object_ypos
The Y position of the object.

object_width
The width of the object placement area, the DEFAULT is the width specified in the object.

object_height
The height of the object placement area, the DEFAULT is the height specified in the object.

dpi_resolution
Defines the correct resolution of your data-object image. Your data-object image resource files may not gave or gave wrong resolution information. The DEFAULT is the image resolution specified in the object image.

object_mapping
The mapping of the object to the object placement area, DEFAULT is the mapping option within the object is used. If the object does not contain a mapping option, then the AFP print server sets the default for each object type. The default value is FIT, valid options are:

`CENTER` Specifies that the center of the object is to be positioned at the center of the object placement area. Any portion of the object that falls outside the object placement area is trimmed.

`LEFT` Specifies that the object is positioned at the upper, left-hand corner of the object placement area, an object that falls outside the object placement area as defined by the object_width & object_height parameters is not trimmed and could cause an exception condition by the IPDS printer.

`FILL` Specifies that the center of the object is to be positioned coincident with the center of the object placement area. The object is then scaled so that it fills the object placement area in both the horizontal and vertical directions. This may require that the object be asymmetrically scaled by different scale factors in both horizontal and vertical directions.

`FIT` Specifies scale to fit. The object is to be scaled to fit within the object placement area, as defined by the object_width & object_height parameters. The center of the object is placed in the center of the object placement area and the object is scaled up or down to fit the area. Scaling in the horizontal and vertical directions is symmetrical. This parameter ensures that the object is not being trimmed and presented in the object placement area with the largest possible size.

`REPEAT` Specifies that the origin of the object is to be positioned with the origin of the object placement area. The object is then replicated in horizontal and vertical directions. If the last replicated data does not fit in the object area, it is trimmed to fit.

`TRIM` Specifies position and trim. The object is positioned at the upper, left-hand corner of the object placement area. Any portion of the object that falls outside the object placement area as defined by the object_width & object_height parameters is trimmed.

All object mapping types are allowed with AFP Page Segment image object; The FILL, FIT, CENTER, REPEAT, and TRIM parameters are allowed with IOCA, GOCA, and non-AFP objects; only the LEFT parameter is allowed with AFP BCOCA barcode object.

cmr_id
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Weaver definition file with a CMR parameter. Default value 0 specifies that CMR is not being defined.

process_mode
Specifies the processing mode for the CMR:

`AUDIT` The audit processing mode. Refers to processing that has already been applied to a resource. In most cases, audit CMRs describe input data and are similar to ICC input profiles. The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

The audit processing mode is used primarily with color conversion CMRs. In audit processing mode, those CMRs indicate which ICC profile must be applied to convert the data into the Profile Connection Space (PCS).

`INSTR` The instruction processing mode. Refers to processing that is done to prepare the resource for a specific printer using a certain paper or another device. Generally, instruction CMRs refer to output data and are similar to ICC output profiles.

The instruction processing mode is used with color conversion, tone transfer curve, and halftone CMRs. In instruction processing mode, these CMRs indicate how the system must convert a resource so it prints correctly on the target printer. The manufacturer of your printer should provide ICC profiles or a variety of CMRs that you can use. Those ICC profiles and CMRs might be installed in the printer controller, included with the printer on a CD, or available for download from the manufacturer's Web site.

render_intent
Specify the rendering intent for the above object:

`PERCP` The Perceptual rendering intent. With this rendering intent, gamut mapping is vendor-specific, and colors are adjusted to give a pleasing appearance. This intent is typically used to render continuous-tone images.

`SATUR` The Saturation rendering intent. With this rendering intent, gamut mapping is vendor-specific, and colors are adjusted to emphasize saturation. This intent results in vivid colors and is typically used for business graphics.

`PELCM` The Media-relative colorimetric rendering intent. In-gamut colors are rendered accurately, and out-of-gamut colors are mapped to the nearest value within the gamut. Colors are rendered concerning the source white point and are adjusted for the media white point. Therefore colors printed on two different media with different white points won't match colorimetrically but may match visually. This intent is typically used for vector graphics.

ABSCM — The ICC-absolute colorimetric rendering intent. In-gamut colors are rendered accurately, and out-of-gamut colors are mapped to the nearest value within the gamut. Colors are rendered only concerning the source white point and are not adjusted for the media white point. Therefore colors printed on two different media with different white points should match colorimetrically, but may not match visually. This intent is typically used for logos.

object_rotation
The rotation of the object clockwise around the object's origin. The valid values are:

DEG0	The overlay is not rotated
DEG90	The overlay is rotated 90 degrees clockwise
DEG180	The overlay is rotated 180 degrees clockwise
DEG270	The overlay is rotated 270 degrees clockwise

object_color
The color to be used as the default color or initial color for the object placement area. This parameter is used only for AFP objects of the PSEG, GOCA, BCOCA, and IOCA type. If the object type is non-AFP, this parameter is ignored. Colors specified must be one of the standard AFP OCA color, valid values are NONE, DEFAULT, BLACK, BLUE, BROWN, GREEN, RED, PINK (or MAGENTA), TURQ (or CYAN), YELLOW, DARKBLUE (or DBLUE), ORANGE, PURPLE, MUSTARD, GRAY, DARKGREEN (or DGREEN), DARKTURQ (or DTURQ), and DARKCYAN (or DCYAN).

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
              :
              :

InclObjt("Orchid Flower",10,10);   // Include an JPEG image at (10
                                    mm,10mm),
              :                     // image type JPEG is defined in the
              :                     // MakeAFP defintion file

InclObjt("FLOWER02");               // Include a TIFF image at current
                                    // position, image type TIFF is
                                    defined
                                    // in the MakeAFP defintion file
              :
              :
ClosePage();
CloseDoc();
```

Include Overlay

Function

Includes a reference to an overlay at the specified position or current position. You can include up to 127 unique page overlays on a page.

Syntax

```
void InclOvly(
              char*        overlay_name,
              float        x_pos,
              float        y_pos,
              degree       degree = DEG0
             );
```

Parameters

overlay_name
The name of the page overlay in the MakeAFP overlay directory. The overlay may need to be available to MakeAFP at the time of formatting. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, _ (underscore), #, and @, for example, O1TEST01.

x_pos
The X position of the overlay.

y_pos
The Y position of the overlay.

degree
The rotation for the overlay. The valid values are:

DEG0	The overlay is not rotated
DEG90	The overlay is rotated 90 degrees clockwise
DEG180	The overlay is rotated 180 degrees clockwise
DEG270	The overlay is rotated 270 degrees clockwise

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
               :
               :
InclOvly("O1TEST01",10,10);        // Include an overlay at (10,10)
               :
               :
InclOvly("O1TEST02");              // Include an overlay at current
                                   // position
               :
               :
ClosePage();
CloseDoc();
```

Include Page Segment

Function

Includes a reference to a page segment at the specified position or current position. You can include up to 127 unique page segments on a page.

Syntax

```
void InclPseg(
              char*       psegname,
              float       x_pos,
              float       y_pos
             );
```

Parameters

psegname
The name of the page segment in the MakeAFP page segment directory. The page segment may need to be available to MakeAFP at the time of formatting. Names can have a maximum of eight characters; valid characters are A-Z, 0-10, _ (underscore), #, and @, for example, S1TEST01.

x_pos
The X position of the page segment.

y_pos
The Y position of the page segment.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
              :
              :
InclPseg("S1TEST01",10,10);        // Include a PSEG at (10,10)
              :
              :
InclPseg("S1TEST02");              // Include a PSEG at current position
              :
              :
ClosePage();
CloseDoc();
```

Left Align ASCII / EBCDIC Text

Function

Left aligns a single-line of the 1-byte text string at the current position.

You need to define an ASCII or EBCDIC encoded font with the “Font” function. MakeAFP Weaver converts data encoding internally, according to the encoding of AFP font defined, however for a better formatting performance, using ASCII encoding font is recommended to avoid such ASCII to EBCDIC conversion.

If the font using is an EBCDIC encoded font, then you must make sure that the default input data encoding is defined properly by the function of DefaultCode( ) first, otherwise the default input data encoding “Windows-1252” is being used for internal data encoding conversion.

Syntax

```
void Ltxt(
          char*         data,
          bool          same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated ASCII data string.

Make sure your default input data encoding is defined properly by the function of DefaultCode( ) before calling this function with *toCode* parameter, otherwise default input data encoding is “Windows-1252”.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
Pos(2,2);                           // current position at (2",2")

Font(3);                            // assume font 3 is ASCII font

Ltxt("text is left aligned");       // left put text at (2",2")
              :

ClosePage();
CloseDoc();
```

Left Align Japanese

Function

Left aligns a single-line of the Japanese text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC font, and the second one must be an SJIS-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and SJIS-PC fonts is recommended.

Syntax

```
void Ljp(
        char*           data,
        bool            same_pos = TRUE
       );
```

Parameters

data
The NULL-terminated SJIS data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
            :
            :
Pos(2,2);                              // position at (2",2")

Font2(3,4);                             // assume font 3 is ASCII font,
                                       // and font 4 is SJIS font

Ljp("Alphabetが混在した文章のサンプルです");  // left put SJIS text at
                                       // (2",2")

            :
            :

ClosePage();
CloseDoc();
```

Left Align Korean

Function

Left aligns a single-line of the Korean text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC font, and the second one must be a KSC-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and KSC-PC fonts is recommended.

Syntax

```
void Lkr(
        char*           data,
        bool            same_pos = TRUE
       );
```

Parameters

data
The NULL-terminated KSC data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
             :
             :
Pos(2,2);                           // position at (2",2")

Font2(3,4);                         // assume font 3 is ASCII font,
                                    // and font 4 is KSC font

Lkr("IBM 소프트웨어 솔루션");          // left put KSC text
                                    // at (2",2")

             :
             :

ClosePage();
CloseDoc();
```

Left Align Simplified Chinese

Function

Left aligns a single-line of the Simplified Chinese text string at the current position.

You need to define a pair of AFP fonts with the "Font2" function, the first parameter must be an ASCII or EBCDIC font, and the second one must be a GBK-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and GBK-PC fonts is recommended.

Syntax

```
void Lsc(
         char*            data,
         bool             same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated GB18030 data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
             :
             :
Font2(3,4);                          // assume font 3 is ASCII font,
                                     // and font 4 is Gb18030 font

Pos(2,2);                            // current position at (2",2")

Lsc("实现 Win2000 与 Linux 的双引导");    // left place GBK text at (2",2")


             :
             :

ClosePage();
CloseDoc();
```

Left Align Traditional Chinese

Function

Left aligns a single-line of the Traditional Chinese text string at the current position.

You need to define a pair of AFP fonts with the "Font2" function, the first parameter must be an ASCII or EBCDIC font, and the second one must be a BIG5-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and BIG5-PC fonts is recommended.

Syntax

```
void Ltc(
          char*             data,
          bool              same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated BIG5 data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
               :
Pos(2,2);                                // current position at (2",2")

Font2(3,4);                              // assume font 3 is ASCII font,
                                         // and font 4 is BIG5 font

Ltc("實現Win2000與Linux的双引导");       // left put BIG5 text at (2",2")


               :
               :

ClosePage();
CloseDoc();
```

Left Align SBCS-HOST/DBCS-HOST

Function

Left aligns a single-line of the SBCS-HOST/DBCS-HOST text string at the current position.

You need to call a pair of fonts with the “Font2” function, the first parameter must be an EBCDIC font, and the second one must be a DBCS-HOST font.

With OpenType/TrueType fonts, the data type EBCDIC_T1xxxxxx (with a codepage in EBCDIC encoding) must be defined for the first font, and DBCS_T1xxxxxx (with a codepage in DBCS-HOST encoding) must be defined for the second font, by the FONT parameters in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Ldbcs(
            char*         data,
            bool          same_pos = TRUE
          );
```

Parameters

data
The NULL-terminated SBCS-HOST/DBCS-HOST data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
               :
Pos(2,2);                                // current position at (2",2")

Font2(3,4);                              // assume font 3 is EBCDIC font,
                                         // and font 4 is DBCS-HOST font

Ldbcs("实现  Win2000 与 Linux 的双引导"); // left put DBCS text at (2",2")

               :
               :

ClosePage();
CloseDoc();
```

Left Align UTF-16 Text

Function

Left aligns a single-line UTF-16 string at the current position. Native UTF-16 string on Windows is in litter-endian (UTF-16LE) encoding, this function converts it to UTF-16BE that is used by AFP.

Before calling this function, make sure the font ID you called with the "Font" function, was defined with data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Lu16(
          UChar*       data,
          bool         same_pos = TRUE
         );
```

Parameters

data
The UTF-16 NULL-terminated UTF-16 litter-endian string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
/* UTF-16 string, "test" and CJK characters "测试"  */
UChar  data1[20] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
Pos(2,2);                          // current position at (2",2")

Font(2);                           // Assume font 2 is a TrueType font
                                   // with data type UTF16BE defined

Lu16(data1);                       // left put UTF-16 at (2",2")

              :
              :

ClosePage();
CloseDoc();
```

Left Align UTF-16 Text Converting from Legacy String

Function

Left aligns a single-line UTF-16BE string converting from the legacy codepage/charset string, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Lu16c(
          char*        data,
          char*        fromcode = NULL,
          bool         same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated legacy codepage string.

fromcode
The encoding name of the source string to be converted into UTF-16. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();

DefaultCode("GB18030");           // set default codepage of input data

OpenPage(8.5,11);
             :
             :
Pos(2,2);                         // set current position at (2",2")

Font(2);                          // Assume font 2 is a TrueType font
                                  // with data type UTF16BE defined

Lu16c("test 测试");               // left put UTF-16 converting from
                                  // Chinese GB18030
             :
             :

ClosePage();
CloseDoc();
```

Left Align UTF-8 Text

Function

Left aligns a single-line UTF-8 string at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Lu8(
          UChar8*       data,
          bool          same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated UTF-8 string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
/* UTF-8 string, "test" and CJK characters "测试"  */
UChar8    data1[20] = "test\xe6\xb5\x8b\xe8\xaf\x95";

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
Pos(2,2);                          // current position at (2",2")

Font(2);                           // Assume font 2 is a TrueType font
                                   // with data type UTF8 defined

Lu8(data1);                        // left put UTF-8 at (2",2")

              :
              :

ClosePage();
CloseDoc();
```

Left Align UTF-16 Text Converting from Legacy String

Function

Left aligns a single-line UTF-8 string converting from the legacy codepage/charset string, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Lu8c(
          char*         data,
          char*         fromcode = NULL,
          bool          same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated legacy codepage string.

fromcode
The encoding name of the source string to be converted into UTF-8. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();

DefaultCode("GB18030");             // set default codepage of input data

OpenPage(8.5,11);
               :
               :
Pos(2,2);                           // set current position at (2",2")

Font(2);                            // Assume font 2 is a TrueType font
                                    // with data type UTF16BE defined

Lu8c("test 测试");                  // left put UTF-8 converting from
                                    // Chinese GB18030
               :
               :

ClosePage();
CloseDoc();
```

Left Align UTF-8 Text Converting from UTF-16LE

Function

Left aligns a single-line UTF-8 string converting from the UTF16-LE text, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with the data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Lu8u(
          UChar*        u16_data,
          bool          same_pos = TRUE
         );
```

Parameters

u16_data
The NULL-terminated UTF-16LE text string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position to which the next character would be placed.

Sample

```
    /* UTF-16 string, "test" and CJK characters "测试"  */
    UChar    data1[] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};

    SetUnit(IN_U600);
    OpenDoc();
    OpenPage(8.5,11);
                   :
                   :
    Pos(2,2);                         // current position to (2",2")

    Font(2);                          // Assume font 2 is a TrueType font
                                      // with data type UTF8 defined

    Lu8u(data);                       // Left put UTF-8 converting from
                                      // UTF16-LE

                   :
                   :

    ClosePage();
    CloseDoc();
```

Lines Per Inch

Function

Defines the default vertical baseline spacing in terms of lines per inch for the subsequent text.

Syntax

```
void LPI(
         float      lines
        );
```

Parameters

lines
The lines per inch to set up the default line spacing for the subsequent text.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
               :
               :
LPI(8);                   // subsequent texts will be in 8 LPI
               :
               :
LPI(6.5)                  // subsequent texts will be in 6.5 LPI
               :
               :
ClosePage();
CloseDoc();
```

Line Spacing

Function

Defines the default vertical baseline spacing in terms of the measurement unit for the subsequent text.

Syntax

```
void LineSp(
             float         increment
            );
```

Parameters

increment
The baseline increment in terms of the measurement unit for the subsequent text.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
              :
              :
LineSp(4);              // subsequent baseline spacing will be 4 mm
              :
              :
LineSp(inch(0.4);       // subsequent baseline spacing will be 0.4 inch

              :
              :
ClosePage();
CloseDoc();
```

Margin of Inline Text

Function

Sets the inline left margin for the subsequent text to be positioned with the “Next Line” and “Skip Lines” function calls

Syntax

```
void Margin(
             float         margin
            );
```

Parameters

margin
The left inline margin for the text in terms of the measurement unit.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
lpi(8);

Margin(0.8);        // left margin for the text is 0.8"

Skip(10);           // skip 10 lines

              :
              :

ClosePage();
CloseDoc();
```

Mask an Area

Function

Hides a hidden area from an AFP page.

You can hide areas that you do not want to display or print, for instance, you might hide an area that contains old OMR lines and then create a new barcode in the same area.

Syntax

```
void MaskArea(
              float         x_pos,
              float         y_pos,
              float         width,
              float         height
             );
```

Parameters

x_pos
The X position of the top left corner of the hidden area.

y_pos
The Y position of the top left corner of the hidden area.

width
The width of the hidden area.

height
The height of the hidden area.

Sample

None.

Mask a Text Field

Function

Masks a text field string by the coordinate location of the data field on an AFP page. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the "Get Page" function.

With the "Trigger" function, we can define the indication information that indicates which AFP page containing the data fields need to be masked. The trigger must be consistent as a milepost throughout the AFP document.

Make sure the "Encoding" function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```
void MaskField(
               ushort       x,
               ushort       y,
               ushort       column,
               ushort       length,
               char         maskChar = '*',
               ushort       field_no = 1
             );
```

Parameters

x
Specifies the X position of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y
Specifies the Y position of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

column
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

length
Specifies the number of contiguous bytes (characters), starting at the *column*, that is to be masked.

maskChar
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

field_no
Specifies the order number of the AFP data field from which the data field is being masked, the default value is 1, but sometimes several fields can be referenced from the same (x, y) position, in this case, you must make sure which field is being masked.

Sample

None.

Mask Text Field by a Location Area

Function

Masks a text field string by the coordinate location range of the data field on an AFP page. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields need to be masked. The trigger must be consistent as a milepost throughout the AFP document.

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```
void MaskField2(
               ushort          x1,
               ushort          x2,
               ushort          y1,
               ushort          y2,
               ushort          column,
               ushort          length,
               char            maskChar = '*',
               ushort          field_no = 1
              );
```

Parameters

x1, x2
Specifies the X position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y1, y2
Specifies the Y position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

column
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

length
Specifies the number of contiguous bytes (characters), starting at the *column*, that is to be masked.

maskChar
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

field_no
Specifies the order number of the AFP data field from which the data field is being masked, the default value is 1, but sometimes several fields can be referenced from the same (x, y) position, in this case, you must make sure which field is being masked.

Sample

None.

Mask Text Field by a Location Area and a Pattern

Function

Masks a text field string by the coordinate location range of the data field on an AFP page and a matching pattern of symbols. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields need to be masked. The trigger must be consistent as a milepost throughout the AFP document.

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```
void MaskField3(
               ushort           x1,
               ushort           x2,
               ushort           y1,
               ushort           y2,
               ushort           column,
               ushort           length,
               char*            pattern,
               char             maskChar = '*',
              );
```

Parameters

x1, x2
Specifies the X position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y1, y2
Specifies the Y position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

column
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

length
Specifies the number of contiguous bytes (characters), starting at the *column*, that is to be masked.

pattern
Specifies a character string or a pattern of symbols to be used for picking up a set of the character string that matches with the specified pattern. Valid pattern symbols are:

'@′	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

maskChar
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

Sample

None.

Mask Text Field by an X-location Range and a Pattern

Function

Masks a text field string by the X-location range of the data field on an AFP page and a matching pattern of symbols. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields need to be masked. The trigger must be consistent as a milepost throughout the AFP document.

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```
void MaskFieldX(
               ushort           x1,
               ushort           x2,
               ushort           column,
               ushort           length,
               char*            pattern,
               char             maskChar = '*',
              );
```

Parameters

x1, x2
Specifies the X position range of the data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

column
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

length
Specifies the number of contiguous bytes (characters), starting at the *column*, that is to be masked.

pattern
Specifies a character string or a pattern of symbols to be used for picking up a set of the character string that matches with the specified pattern. Valid pattern symbols are:

'@′	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

maskChar
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

Sample

None.

Mask Text Field by a Y-location Range and a Pattern

Function

Masks a text field string by the Y-location range of the data field on an AFP page and a match pattern of symbols. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields need to be masked. The trigger must be consistent as a milepost throughout the AFP document.

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```
void MaskFieldY(
                ushort          y1,
                ushort          y2,
                ushort          column,
                ushort          length,
                char*           pattern,
                char            maskChar = '*',
              );
```

Parameters

y1, y2
Specifies the Y position range of data field in PELS. With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

column
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

length
Specifies the number of contiguous bytes (characters), starting at the *column*, that is to be masked.

pattern
Specifies a character string or a pattern of symbols to be used for picking up a set of the character string that matches with the specified pattern. Valid pattern symbols are:

'@’	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

maskChar
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

Sample

None.

Maximum Pagination

Function

Defines the maximum number of AFP page buffers.

This function is mainly developed for calling from other programming languages, with Visual C++, you can set MakeAFP variable $MaxPaging directly.

You must call this function to set the value for MakeAFP variable $MaxPaging before calling the “Start” function which allocates the required memory for your paging buffers.

For the generation of pagination, such as "Page 347 of 1000", we need to keep composed AFP data in the AFP page buffers first. With MakeAFP Weaver, you can open multiple pages with the “Open Page” functions, and then process different pages in an interleaved manner once each page is initialized, all the composed AFP data stream will be kept in memory buffers in page-level, and finally, after you have completed all the formatting and counted all the pages of a page group, you have to put your pagination text in each page just before you close the page with the “Close Page” function.

Syntax

```
void MaxPaging(
                uint        maxPaging
              );
```

Parameters

maxPaging
The maximum number of AFP page buffers. Big value takes up a big memory, only define this value as big as your maximum number required for the pagination. The default value is 1, MakeAFP reports an error message if this value is not enough for your AFP formatting.

Sample

```
SetUnit(IN_U600);

MaxPaging(1000);            // Sets maximum of page buffers to 1000,
                            // it must be called before Start() function
Start();

OpenDoc();

OpenPage(8.5,11);
              :
              :
ClosePage();

CloseDoc();
```

Millimeter Value

Function

Specifies a value in millimeters.

Syntax

```
float mm(
         float        value
        );
```

Parameters

value
The value in millimeters.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8,11);
              :
              :
Pos(2.5,4);                   // set X and Y position to (2.5",4")
              :
              :
Pos(mm(20),3.5);              // set X position to 20 mm and Y position to
                              // 3.5"
              :
              :

ClosePage();
CloseDoc();
```

Move AFP Page between AFP Page Buffers

Function

Moves an AFP page from an AFP page buffer in which your AFP page was stored previously to another buffer.

Syntax

```
void  MovePage(
               ushort      toPage,
               ushort      fromPage
              );
```

Parameters

toPage
Specifies an AFP page buffer number to move an AFP page to.

fromPage
Specifies an AFP page buffer number to move an AFP page from.

Sample

```
/***************************************************************************/
/* This sample shows how to capture a trigger by an overlay name and       */
/* data fields from page 1, add AFP indexes and barcode to existing        */
/* AFP                          AFP is encoded in CP-037, USA EBCDIC       */
/***************************************************************************/

int main( )
{
   unsigned int i, grpPages, pageSN, groups;
   char tmp[80], mobileNo[20], custName[60];
   bool bog = 0;

   $MaxPaging = 50;          // Maximum paging is up to 50 pages

   SetUnit(IN_U600);                // Set default unit to inch

   Start();                  // Start initiation, open default input,
                             // output and definition files, retrieves
                             // AFP resources, allocate memory

   Encoding("ibm-037","ibm-437");

   OpenDoc();                // OPne AFP document

   $Page = 1;                // Set AFP page buffer number to 1 for the first
                             // page of AFP file

   GetPage();                // Get first page of AFP file

   while ($Edt == 0)         // Until end of AFP document
   {
     GetField(660, 1080, custName);      // Get customer name

     GetField(4050, 900, mobileNo);      // Get customer mobile number

     do {

        $Page++;            // Point to next AFP page buffer
```

```
         GetPage();          // Get next page

         // detecting if it is the first page of a group,
         // overlay O1OVL1E only used by at first page of
         // each page group
         bog = TriggerOvly("O1OVL1E");

      } while (!bog && !$Edt);       // Until beginning of next page group or
                                            // End of AFP file

      bog = 0;                       // Reset it for next group

      // Now got all pages of a page group and first page of next group, now
      it  // is ready to process new AFP output

      if (!$Edt)                     // If not end of AFP document
         grpPages = $Page -1 ;       // Keep total number of pages per group,
                                     // need to minus 1 page of the first
                                     // page of next group
      sprintf(tmp, "%08d", ++groups);

      BgnIdx(tmp);                   // Auto-converts ASCII to EBCDIC for
      indexes
      PutIdx("Customer Name", custName);
      PutIdx("Mobile Number", mobileNo);

      for (i = 0; i < grpPages; i++)
      {
         $Page = i + 1;              // Point to page buffer number to be
      opened

         sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
         BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90);   // Add 1D barcode

         ClosePage();                // Close AFP page, write to AFP file
      }

      EndIdx();                      // End of group level index

      MovePage(1, grpPages + 1);     // As we got first page of next group
                                            // previously, now need move its
      contents
                                     // to page buffer 1 for the next page
      group

      $Page = 1;                     // Reset page buffer to 1 for next group
   }

   CloseDoc();                       // Close AFP document and its file

   return 0;
}
```

Next Line

Function

Starts a new text line from the left inline margin defined by the “Margin” function call, it increments the current baseline coordinate position by the amount of baseline increment defined by either the “Lines Per Inch” or “Line Spacing” function call.

Syntax

```
void NextLine (
              void
             );
```

Parameters

No parameter to be specified.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
lpi(8);

Margin(0.8);         // left margin for the text is 0.8"

NextLine();          // Jump to next new line position

              :
              :

ClosePage();
CloseDoc();
```

Open Document

Function

Opens an AFP document, you must call this 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 you end your program.

MakeAFP Weaver transfers AFP resources into each AFP output document file if the AFP resource inline is specified by the MakeAFP definition file.

Syntax

```
void OpenDoc(
             ushort        docNo = 1
            );
```

Parameters

docNo
Specifies which AFP document to be started, valid values are 1 through 10, the default value is 1.

Sample

C Sample:

```
void main( )
{
  Start();                 // Start initiation, open default input,
                           // output and definition files, getting
                           // AFP resources

  OpenDoc();               // Open an document, open its AFP file

           :
           :

  CloseDoc()               // Close AFP document and iys AFP file

}
```

Opening Page – Adding New Page

Function

Opens a new AFP page. Once the page formatting is completed, you can close the page with the “Close Page” function. The initial current position is at the page origin (the top left corner of the logical page specified by the Form Definition).

With the $MaxPaging variable or the “Maximum Paging” function, you can define the maximum number of AFP page buffers. For generating OMR and page pagination, such as “Page 347 of 1000”, we need to keep composed AFP data in the AFP page buffers first.

With MakeAFP Weaver, you can open multiple pages by either the “Get Page” or the “Open Page” functions, and then process different pages in an interleaved manner once each page is initialized, all the composed AFP data stream will be kept in memory buffers in page-level, and finally, after you have completed all the formatting and counted all the pages of a page group, you can put your OMR and pagination text on each page just before you close the page with the “Close Page” function.

With $Page variable, you can indicate which AFP page buffer is to be opened with the “Open Page” function, or switch to the page buffer again before you further format or end that page.

Syntax

```
void OpenPage(
              float        page_width,
              float        page_height
            );
```

Parameters

page_width
Width of the page.

page_lenght
Length of the page.

Sample

```
SetUnit(IN_U600);

$MaxPaging = 100;            // Maximum page buffers are 100, must be
                             // defined before Start() function call

    Start();                 // Start a MakeAFP Weaver session

OpenDoc();
             :
$Page = 3;                   // switch to page 3 for open page 3
OpenPage(8.5,11);
             :
ClosePage();                 // Close page 3 and write to AFP output
file
             :
$Page = 15;                  // switch to page 15 for open page 15
OpenPage(8.5,11);
             :
```

```
ClosePage();                    // Close page 15 and write to AFP output
file
             :
```

Paragraph of 1-Byte Text

Function

Formats a line of the 1-byte texts into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

You need to define an ASCII or EBCDIC encoded font with the “Font” function. MakeAFP Weaver converts data encoding internally, based on the encoding of AFP font defined.

If the font using is an EBCDIC encoded font, then you must make sure that the default input data encoding is defined properly by the function of DefaultCode( ) first, otherwise the default input data encoding “Windows-1252” is being used for internal data encoding conversion.

Make sure your default input data encoding and language locale are defined properly by the functions of DefaultCode( ) and DefaultLocale( ) before calling this function, otherwise the default encoding “Windows-1252” and locale “en_US” is being used for the paragraph internal processing.

Syntax

```
void ParTxt(
        char*         text,
        float         paragraph_width,
        alignmode     alignment = LEFT,
        bool          same_pos = FALSE
        );
```

Parameters

text
The ASCII or EBCDIC text to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\x0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

`[F=xx}` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

`[U=01}` Turns on underscore

`[U=00}` Turns off underscore

`[C=xx}` xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

`[C=rrggbb}` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

`[C=ccmmyykk}` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

Make sure your default input data encoding is defined properly by the function of DefaultCode( ) before calling this function with *toCode* parameter, otherwise default input data encoding is “Windows-1252”.

paragraph_width
The width of the paragraph.

alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

LEFT	Text is left aligned
RIGHT	Text is right aligned
CENTER	Text is centered
JUSTIFY	Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The paragraph of text will be right-aligned nicely";

Encoding("ibm-037", "ibm-437"); // PC codepage 437, USAN ASCII
                                // AFP codepage 037, USA EBCIDC

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


OpenPage(8.5,11);

    :

LineSp(0.25);                   // Line spacing is 0.25", 4 LPI

ParTxt(msg,3,RIGHT);            // text is right aligned into 3" width
                                // paragraph

    :

ClosePage();
```

Paragraph of Japanese

Function

Formats a line of the Japanese text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC font, and the second one must be an SJIS-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and SJIS-PC fonts is recommended.

Syntax

```
void ParJp (
             char*          text,
             float          paragraph_width,
             alignmode      alignment = LEFT,
             bool           same_pos = FALSE
            );
```

Parameters

text
The SJIS Japanese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\x0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

`[F=xx}` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

`[U=01}` Turns on underscore

`[U=00}` Turns off underscore

`[C=xx}` xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

`[C=rrggbb}` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

`[C=ccmmyykk}` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the Japanese paragraph.

alignment
Specifies how the Japanese text in the fixed paragraph should be formatted. The valid values are:

`LEFT` Japanese text is left-aligned

RIGHT	Japanese text is right-aligned
CENTER	Japanese text is centered
JUSTIFY	Japanese text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The Japanese ひらがな、漢字、数字  will be center-aligned";

SetUnit(IN_U600);
OpenDoc( );
OpenPage(8.5,11);
               :
LineSp(0.25);              // Line spacing is 0.25", 4 LPI

Font2(1,2);

ParJp(msg,3,CENTER);       // Japanese is center aligned into 3" width
                           // paragraph
               :
ClosePage();
CloseDoc();
```

Paragraph of Korean

Function

Formats a line of the Korean text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC font, and the second one must be a KSC-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and KSC-PC fonts is recommended.

Syntax

```
void ParKr(
            char*          text,
            float          paragraph_width,
            alignmode      alignment = LEFT,
            bool           same_pos = FALSE
          );
```

Parameters

text
The KSC Korean to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\x0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

`[F=xx}` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

`[U=01}` Turns on underscore

`[U=00}` Turns off underscore

`[C=xx}` xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

`[C=rrggbb}` rr, gg, bb are the RGB values in two characters of hex code value respectively, valid value is from “00” through “FF”.

`[C=ccmmyykk}` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, valid value is from “00” through “64”.

paragraph_width
The width of the Korean paragraph.

alignment
Specifies how the Korean text in the fixed paragraph should be formatted. The valid values are:

`LEFT` Korean text is left-aligned

`RIGHT`	Korean text is right-aligned
`CENTER`	Korean text is centered
`JUSTIFY`	Korean text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
          char *msg = "The Korean 온 가족의 티셔츠가 내 폼에  will be center-
aligned";
          OpenPage(8.5,11);
                        :
          LineSp(0.25);              // Line spacing is 0.25", 4 LPI

          Font2(1,2);

          ParKr(msg,3,CENTER);       // Korean is center aligned into 3" width
                                     // paragraph
                        :
          ClosePage();
```

Paragraph of Simplified Chinese

Function

Formats a line of the Simplified Chinese text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC font, and the second one must be a GBK-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and GBK-PC fonts is recommended.

Syntax

```
void ParSc(
            char*          text,
            float          paragraph_width,
            alignmode      alignment = LEFT,
            bool           same_pos = FALSE
          );
```

Parameters

text
The GBK Simplified Chinese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\x0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

`[F=xx}` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

`[U=01}` Turns on underscore

`[U=00}` Turns off underscore

`[C=xx}` xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

`[C=rrggbb}` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

`[C=ccmmyykk}` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the Chinese paragraph.

alignment
Specifies how the Chinese text in the fixed paragraph should be formatted. The valid values are:

`LEFT`	Chinese text is left-aligned
`RIGHT`	Chinese text is right-aligned
`CENTER`	Chinese text is centered
`JUSTIFY`	Chinese text is justified

same_pos

Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The Chinese 越来越多的电脑用户  will be center-aligned";

OpenPage(8.5,11);
               :
LineSp(0.25);              // Line spacing is 0.25", 4 LPI

Font2(1,2);

ParSc(msg,3,CENTER);       // Chinese is center aligned into 3" width
                           // paragraph
               :
ClosePage();
```

Paragraph of Traditional Chinese

Function

Formats a line of the Traditional Chinese text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC font, and the second one must be a BIG5-PC or DBCS-HOST font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts to be used. To avoid the internal data encoding conversion, using a pair of ASCII and BIG5-PC fonts is recommended.

Syntax

```
void ParTc(
             char*           text,
             float           paragraph_width,
             alignmode       alignment = LEFT,
             bool            same_pos = FALSE
           );
```

Parameters

text
The BIG5 Traditional Chinese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\x0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

`[F=xx}` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

`[U=01}` Turns on underscore

`[U=00}` Turns off underscore

`[C=xx}` xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

`[C=rrggbb}` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

`[C=ccmmyykk}` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the Chinese paragraph.

alignment
Specifies how the Chinese text in the fixed paragraph should be formatted. The valid values are:

LEFT	Chinese text is left-aligned
RIGHT	Chinese text is right-aligned
CENTER	Chinese text is centered
JUSTIFY	Chinese text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The Chinese 越来越多的電脑用户  will be center-aligned";

OpenPage(8.5,11);
               :
LineSp(0.25);             //  Line spacing is 0.25", 4 LPI

Font2(1,2);

ParTc(msg,3,CENTER);      //  Chinese is center aligned into 3" width
                          //  paragraph
               :
ClosePage();
```

Paragraph of UTF-16 Text

Function

Formats a line of the Unicode UTF-16LE text into a fixed-width paragraph. Native UTF-16 string on Windows is in litter-endian (UTF-16LE) encoding, this function internally translates it to UTF-16BE which is used by AFP.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your default language locale is defined properly by the function DefaultLocale( ) before calling this function, otherwise the default locale is “en_US”.

Syntax

```
void ParU16(
            UChar*          text,
            float           paragraph_width,
            alignmode       alignment = LEFT,
            bool            same_pos = FALSE
           );
```

Parameters

text
The UTF-16 text to be aligned into a fixed-width paragraph. The following escape control codes in UTF-16LE can be inserted into your text data for dynamic control of font, color, and underscore switching:

[F=xx} xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

[U=01} Turns on underscore

[U=00} Turns off underscore

[C=xx} xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

[C=rrggbb} rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

[C=ccmmyykk} cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the paragraph.

alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

LEFT	Text is left aligned
RIGHT	Text is right aligned
CENTER	Text is centered
JUSTIFY	Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";

UChar msg16[128];                       // Defines a buffer UTF-16


DefaultCode("gb18030");                 // Text is in Chinese GB18030

DefaultLocale("zh_CN");                 // Simplified Chinese of China

ChartoU16(msg16, 128, msg);             // Converts GB18030 to UTF-16LE


OpenPage(8.5,11);

               :

LPI(4);                                 // Line spacing is 4 lines per
                                    inch
                                        // for paragraph

ParU16(msg16, 3, CENTER);               // Chinese text is right aligned
                                        // into 3" width paragraph

               :

ClosePage();
```

Paragraph of UTF-16 Text Converting from Legacy String

Function

Formats a line of the Unicode UTF-16LE text converting from the legacy codepage/charset string into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your default PC encoding and language locale are defined properly by the functions of DefaultCode( ) and DefaultLocale( ) before calling this function, otherwise default PC encoding is “Windows-1252” and locale is “en_US”.

Syntax

```
void ParU16c(
             char*            text,
             float            paragraph_width,
             alignmode        alignment = LEFT,
             bool             same_pos = FALSE
            );
```

Parameters

text

The legacy codepage string to be converted to UTF-16 and aligned into a fixed-width paragraph. The following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

[F=xx} xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

[U=01} Turns on underscore

[U=00} Turns off underscore

[C=xx} xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

[C=rrggbb} rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

[C=ccmmyykk} cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width

The width of the paragraph.

alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

LEFT	Text is left aligned
RIGHT	Text is right aligned
CENTER	Text is centered
JUSTIFY	Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";

DefaultCode("gb18030");                 // Text is in Chinese GB18030

DefaultLocale("zh_CN");                 // Simplified Chinese of China


OpenPage(8.5,11);

               :

LPI(4);                                 // Line spacing is 4 lines per
                                    inch
                                        // for paragraph

ParU16c(msg, 3, CENTER);                // Chinese text is right aligned
                                        // into 3" width paragraph

               :

ClosePage();
```

Paragraph of UTF-8 Text

Function

Formats a line of the Unicode UTF-8 text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your default language locale is defined properly by the function DefaultLocale( ) before calling this function, otherwise the default locale is “en_US”.

Syntax

```
void ParU8(
            UChar8*         text,
            float           paragraph_width,
            alignmode       alignment = LEFT,
            bool            same_pos = FALSE
          );
```

Parameters

text
The UTF-8 text to be aligned into a fixed-width paragraph. The following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

[F=xx} xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc

[U=01} Turns on underscore

[U=00} Turns off underscore

[C=xx} xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

[C=rrggbb} rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

[C=ccmmyykk} cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the paragraph.

alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

LEFT Text is left aligned

RIGHT	Text is right aligned
CENTER	Text is centered
JUSTIFY	Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";

UChar8 msg8[256];                       // Defines a buffer UTF-8


DefaultCode("gb18030");                 // Text is in Chinese GB18030

DefaultLocale("zh_CN");                 // Simplified Chinese of China

ChartoU8(msg8, 256, msg);               // Converts GB18030 to UTF-8


OpenPage(8.5,11);

               :

LPI(4);                                 // Line spacing is 4 lines per
                                    inch
                                        // for paragraph

ParU8(msg8, 3, CENTER);                 // Chinese text is right aligned
                                        // into 3" width paragraph

               :

ClosePage();
```

Paragraph of UTF-8 Text Converting from Legacy String

Function

Formats a line of the Unicode UTF-8 text converting from the legacy codepage/charset string into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your default PC encoding and language locale are defined properly by the functions of DefaultCode( ) and DefaultLocale( ) before calling this function, otherwise default PC encoding is “Windows-1252” and locale is “en_US”.

Syntax

```
void ParU8c(
          char*             text,
          float             paragraph_width,
          alignmode         alignment = LEFT,
          bool              same_pos = FALSE
         );
```

Parameters

text
The legacy codepage string to be converted to UTF-8 and aligned into a fixed-width paragraph. The following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

`[F=xx}`	xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
`[U=01}`	Turns on underscore
`[U=00}`	Turns off underscore
`[C=xx}`	xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

`[C=rrggbb}`	rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
`[C=ccmmyykk}`	cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the paragraph.

alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

LEFT	Text is left aligned
RIGHT	Text is right aligned
CENTER	Text is centered
JUSTIFY	Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";


DefaultCode("gb18030");                 // Text is in Chinese GB18030

DefaultLocale("zh_CN");                 // Simplified Chinese of China


OpenPage(8.5,11);

               :

LPI(4);                                 // Line spacing is 4 lines per
                                    inch
                                        // for paragraph

ParU8c(msg, 3, CENTER);                 // Chinese text is right aligned
                                        // into 3" width paragraph

               :

ClosePage();
```

Paragraph of UTF-8 Text Converting from UTF-16LE

Function

Formats a line of the Unicode UTF-8 text converting from the Unicode UTF-16LE string into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your default language locale is defined properly by the function DefaultLocale( ) before calling this function, otherwise the default locale is “en_US”.

Syntax

```
void ParU8u(
             UChar*              u16_text,
             float               paragraph_width,
             alignmode           alignment = LEFT,
             bool                same_pos = FALSE
            );
```

Parameters

u16_text
The UTF-16LE text to be aligned into a fixed-width paragraph. The following escape formatting control codes in UTF-16LE can be inserted into your text data for dynamic control of font, color, and underscore switching:

[F=xx} xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
[U=01} Turns on underscore
[U=00} Turns off underscore
[C=xx} xx are the color ID in two characters of hex code value:

BLUE	"01"	RED	"02"
PINK	"03"	MAGENTA	"03"
GREEN	"04"	CYAN	"05"
TURQ	"05"	YELLOW	"06"
BLACK	"08"	DARKBLUE	"09"
BROWN	"10"	ORANGE	"0A"
PURPLE	"0B"	DARKGREEN	"0C"
DARKCYAN	"0D"	DARKTURQ	"0D"
MUSTARD	"0E"	GRAY	"0F"

[C=rrggbb} rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

[C=ccmmyykk} cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the paragraph.

alignment

Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

LEFT	Texts are left-aligned
RIGHT	Texts are right-aligned
CENTER	Texts are center-aligned
JUSTIFY	Texts are justify-aligned

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position to which the next text would be placed.

Sample

```
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";

UChar msg16[128];                       // Defines a buffer UTF-16


DefaultCode("gb18030");                 // Text is in Chinese GB18030

DefaultLocale("zh_CN");                 // language locale is Simplified
                                        // Chinese


ChartoU16(msg16, 128, msg);             // Converts GB18030 to UTF-16LE


OpenPage(8.5,11);

:

LPI(4);                                 // Line spacing is 4 lines per
inch
                                        // for paragraph

ParU8u(msg16, 3, CENTER);               // Chinese texts are right aligned
                                        // into 3" width paragraph

:

ClosePage();
```

Point Value

Function

Specifies a value in point.

Syntax

```
float pt(
          float     value
        );
```

Parameters

value
The value in point.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8,11);
              :
              :
Pos(2.5,4);                  // set X and Y position to (2.5",4")
              :
              :
Pos(pt(20),3.5);             // set X position to 20 pointsand Y position to
                             // 3.5"
              :
              :

ClosePage();
CloseDoc();
```

Position of Text

Function

Sets the absolute horizontal position (X) and absolute vertical position (Y) for the output text on the page. The origin position on the page is at (0, 0).

Syntax

```
void Pos(
         float     x_position,
         float     y_position
        );
```

Parameters

x_position

The value of the absolute horizontal position from the page origin. Negative values are not valid.

y_position

The value of the absolute vertical position from the page origin. Negative values are not valid.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
              :
              :
Pos(5,10);                        // set x position at 5 mm and
                                  // y position at 10 mm

              :
              :

ClosePage();
CloseDoc();
```

Print AFP File

Function

Submits the generated AFP file directly to your AFP/IPDS print server. It calls a printing to submit command provided by your AFP/IPDS Print server software.

It must be specified after the “Close Document” function request.

Notes: With your debug property setting "Program Arguments", you may have to define a fully-qualified AFP output filename with the flag parameter "-o", for example:

-d afp2pcl.def -i afp2pcl.txt -o c:\makeafp\samples\test\afp2pcl\afp2pcl.afp

To let PrintAFP() function to submit the AFP output file from a specific path during your development debug running.

Syntax

```
void PrintAFP(
              char*       print_command,
              ushort      docNo = 1
              char*       winPrinter = NULL
             );
```

Parameters

print_command
The printing submit command is provided by your AFP/IPDS print server or its client software. You must install the client software provided by your vendor if you want to submit the AFP file remotely.

docNo
Specifies which AFP document to be submitted to print, valid values are 1 through 10, the default value is 1.

winPrinter
Optional, only to be used for print AFP to a Windows PCL printer. Specifies the name of your Windows PCL printer, default is print to your Windows default printer if it is not specified.

Sample

Submit the generated AFP file to IBM Infoprint Manager:

```
Start();
SetUnit(IN_U600);
OpenDoc( );
           :
           :
CloseDoc();

// Call IBM Infoprint Manager print command:  pdpr
// job attribute file is: d:\ipmdata\att\test01.att
// IPDS printer name is:  prt1

PrintAFP("pdpr -X d:\\ipmdata\\att\\test01.att -p prt1");

// Call IBM AFP Workbench Viewer "Print It" program
// to print AFP to a Windows PCL printer named
// "Infoprint 1145 PCL by IP"

PrintAFP("\"d:\\AFP Viewer\\ftdwprt\" /p", 1, "Infoprint 1145 PCL by IP");
```

Put Index Tag

Function

Creates an indexing tag in the AFP document for use by an AFP viewer, AFP archiving systems, and MakeAFP reprint and sorting utilities. It generates an AFP Tag Logical Element (TLE) structured field at the page group.

Syntax

```
void PutIdx(
              char*         index_name,
              char*         index_value,
              ushort        docNo = 1,
              bool          autoConvert = true
            );
```

Parameters

MakeAFP Weaver puts the characters strings of index_name and index_value “as is” without any conversion, you may need to call one of the MakeAFP conversion functions to convert the string before you put it into AFP, for instance, to convert ASCII into EBCDIC for indexing in EBCDIC encoding instead of ASCII. Make sure the CPGID parameter is defined in your MakeAFP Weaver definition file properly.

Index_name
The name of the index, up to 250 characters, including blanks, for example, “Account Number”.

Index_value
The value of the index, up to 250 characters, including blanks, for example, “1234-567-4567”.

docNo
Specifies to which AFP document to insert the AFP indexing information, valid values are 1 through 10, the default value is 1.

autoConvert
Specifies whether let MakeAFP Weaver determine a conversion from the native PC ASCII encoding to the target AFP index string encoding is needed automatically. The default value is TRUE lets MakeAFP Weaver auto-decide a conversion is required. MakeAFP Weaver calls converter by the encodes specified by the “Encoding” function. Make sure the “Encoding” function is called if a conversion is required.

Sample

```
/****************************************************************************/
/* This sample shows how to capture a trigger by an overlay name and       */
/* data fields from page 1, add AFP indexes and barcode to existing        */
/* AFP                          AFP is encoded in CP-037, USA EBCDIC       */
/****************************************************************************/

int main( )
{
   unsigned int i, grpPages, pageSN, groups;
   char tmp[80], mobileNo[20], custName[60];
   bool bog = 0;

   $MaxPaging = 50;          // Maximum paging is up to 50 pages

   SetUnit(IN_U600);         // Set default unit to inch

   Start();                  // Start initiation, open default input,
                             // output and definition files, retrieves
                             // AFP resources, allocate memory

   Encoding("ibm-037","ibm-437");

   OpenDoc();                // Open AFP document

   $Page = 1;                // Set AFP page buffer number to 1 for the first
                             // page of AFP file

   GetPage();                // Get first page of AFP file

   while ($Edt == 0)         // Until end of AFP document
   {
     GetField(660, 1080, custName);      // Get customer name

     GetField(4050, 900, mobileNo);      // Get customer mobile number

     do {

        $Page++;            // Point to next AFP page buffer

        GetPage();          // Get next page

        // detecting if it is the first page of a group,
        // overlay O1OVL1E only used by at first page of
        // each page group
        bog = TriggerOvly("O1OVL1E");

      } while (!bog && !$Edt);      // Until beginning of next page group or
                                           // End of AFP file

     bog = 0;                       // Reset it for next group

     // Now got all pages of a page group and first page of next group, now
     it  // is ready to process new AFP output

     if (!$Edt)                     // If not end of AFP document
        grpPages = $Page -1 ;       // Keep total number of pages per group,
                                    // need to minus 1 page of the first
                                    // page of next group
     sprintf(tmp, "%08d", ++groups);

     BgnIdx(tmp);                   // Auto-converts ASCII to EBCDIC for indexes
     PutIdx("Customer Name", custName);
     PutIdx("Mobile Number", mobileNo);

     for (i = 0; i < grpPages; i++)
     {
        $Page = i + 1;              // Point to page buffer number to be
     opened

        sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
        BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90);   // Add 1D barcode

        ClosePage();                // End of AFP page, write to AFP file
     }

     EndIdx();                      // End of group level index

     MovePage(1, grpPages + 1);     // As we got first page of next group
                                           // previously, now need move its
     contents
                                    // to page buffer 1 for the next page
     group

     $Page = 1;                     // Reset page buffer to 1 for next group
   }
```

```
   CloseDoc();                      // End of AFP document, close AFP output

   return 0;
}
```

Rendering Color Intent

Function

Specifies the color rendering intent for the subsequent AFP pages or an overlay created by MakeAFP Weaver, to modify the final appearance of the color object.

This function can be repeated to define the rendering intents for all object types.

Syntax

Invokes Color Rendering:

```
void Render(
             objt_type          object_type,
             render_type        render_intent
           );
```

Revokes Color Rendering:

```
void RevokeRender( );
```

Parameters

object_type

Specify the object type to which the rendering intent applies:

IOCA	The AFP IOCA image object.
OBJT	The non-AFP data-object, such as JPEG/TIFF/GIF, etc.
PTOCA	The AFP PTOCA text object.
GOCA	The AFP GOCA vector graphic object.

render_intent

Specify the rendering intent for the above object:

PERCP	The Perceptual rendering intent. With this rendering intent, gamut mapping is vendor-specific, and colors are adjusted to give a pleasing appearance. This intent is typically used to render continuous-tone images.
SATUR	The Saturation rendering intent. With this rendering intent, gamut mapping is vendor-specific, and colors are adjusted to emphasize saturation. This intent results in vivid colors and is typically used for business graphics.
PELCM	The Media-relative colorimetric rendering intent. In-gamut colors are rendered accurately, and out-of-gamut colors are mapped to the nearest value within the gamut. Colors are rendered concerning the source white point and are adjusted for the media white point. Therefore colors printed on two different media with different white points won't match colorimetrically but may match visually. This intent is typically used for vector graphics.
ABSCM	The ICC-absolute colorimetric rendering intent. In-gamut colors are rendered accurately, and out-of-gamut colors are mapped to the nearest value within the gamut. Colors are rendered only concerning the source white point and are not adjusted for the media white point. Therefore colors printed on two different media

with different white points should match colorimetrically, but may not match visually. This intent is typically used for logos.

Sample

```
SetUnit(IN_U600);
OpenDoc();

                  :             // specify color rendering for the
                                // subsequent pages
Render(IOCA, PERCP);            // perceptual rendering to IOCA images
Render(GOCA, SATUR);            // saturation rendering to GOCA
                                // graphics

OpenPage(8.5,11);
                  :
                  :
ClosePage();
                  :
                  :

OpenPage(8.5,11);

                  :
                  :
ClosePage();

RevokeRender();                 // revoke color rendering

CloseDoc();
```

Right Align 1-Byte Text

Function

Right aligns a single-line of the 1-byte text string at the current position.

You need to define an ASCII or EBCDIC encoded font with the “Font” function. MakeAFP Weaver converts data encoding internally, according to the encoding of AFP font defined, however for a better formatting performance, using ASCII encoding font is recommended to avoid such ASCII to EBCDIC conversion.

If the font using is an EBCDIC encoded font, then you must make sure that the default input data encoding is defined properly by the function of DefaultCode( ) first, otherwise the default input data encoding “Windows-1252” is being used for internal data encoding conversion.

Syntax

```
void Rtxt(
           char*         data,
          );
```

Parameters

data
The NULL-terminated ASCII data string.

Make sure your default input data encoding is defined properly by the function of DefaultCode( ) before calling this function with *toCode* parameter, otherwise default input data encoding is “Windows-1252”.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
Font(3);                           // assume font 3 is an ASCII font

              :
Pos(2,2);                          // current position at (2",2")
Rtxt("text is right aligned");     // right align text at (2",2")

              :
              :
ClosePage();
CloseDoc();
```

Right Align Japanese

Function

Right aligns a single-line of the Japanese text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be an SJIS-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Rjp(
         char*        data,
         bool         same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated SJIS data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
Font2(3,4);                             // assume font 3 is ASCII font,
                                        // and font 4 is SJIS font

              :
Pos(2,2);                                     // position at (2",2")
Rjp("Alphabet が混在した文章のサンプルです");  // right align SJIS text at
                                              // (2",2")

              :
              :
ClosePage();
CloseDoc();
```

Right Align Korean

Function

Right aligns a single-line of the Korean text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be a KSC-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Rkr(
          char*         data,
          bool          same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated KSC data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
Font2(3,4);                              // assume font 3 is ASCII font,
                                         // and font 4 is KSC font

               :
Pos(2,2);                                // position at (2",2")
Rkr("IBM 소프트웨어 솔루션");                    // right align KSC text
                                         // at (2",2")

               :
               :
ClosePage();
CloseDoc();
```

Right Align Simplified Chinese

Function

Right aligns a single-line of the Simplified Chiense text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be a GBK-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Rsc(
          char*     data,
          bool      same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated GBK data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
Font2(3,4);                             // assume font 3 is ASCII font,
                                        // and font 4 is Gb18030 font
               :
Pos(2,2);                               // current position at (2",2")
Rsc("实现Win2000与Linux的双引导");      // right align GBK text at (2",2")

               :
               :
ClosePage();
CloseDoc();
```

Right Align Traditional Chinese

Function

Right aligns a single-line of the Traditional Chinese text string at the current position.

You need to define a pair of AFP fonts with the “Font2” function, the first parameter must be an ASCII or EBCDIC encoded font, and the second one must be a BIG5-PC or DBCS-HOST encoded font. MakeAFP Weaver converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```
void Rtc(
          char*     data,
          bool      same_pos = TRUE
         );
```

Parameters

data
The NULL-terminated BIG5 data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
Font2(3,4);                               // assume font 3 is ASCII font,
                                          // and font 4 is BIG5 font

               :
Pos(2,2);                                 // current position at (2",2")
Rtc("實現 Win2000 與 Linux 的双引导");     // right align BIG5 text at (2",2")


               :
               :

ClosePage();
CloseDoc();
```

Right Align SBCS-HOST/DBCS-HOST

Function

Right aligns a single-line of the SBCS-HOST/DBCS-HOST text string at the current position.

You need to call a pair of fonts with the “Font2” function, the first parameter must be an EBCDIC font, and the second one must be a DBCS-HOST font.

With OpenType/TrueType fonts, the data type EBCDIC_T1xxxxxx (with a codepage in EBCDIC encoding) must be defined for the first font, and DBCS_T1xxxxxx (with a codepage in DBCS-HOST encoding) must be defined for the second font, by the FONT parameters in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Rdbcs(
            char*       data,
            bool        same_pos = TRUE
           );
```

Parameters

data
The NULL-terminated SBCS-HOST/DBCS-HOST data string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
Font2(3,4);                              // assume font 3 is EBCDIC font,
                                         // and font 4 is DBCS-HOST font

               :
Pos(2,2);                                // current position at (2",2")
Rdbcs("实现  Win2000 与 Linux 的双引导"); // right align DBCS text at (2",2")

               :
               :
ClosePage();
CloseDoc();
```

Right Align UTF-16 Text

Function

Right aligns a single-line of the UTF-16 string at the current position. Native UTF-16 string on Windows is in litter-endian (UTF-16LE) encoding, this function converts it to UTF-16BE that is used by AFP.

Before calling of this function, make sure the font ID you called with the “Font” function, was defined with data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Ru16(
           UChar*       data,
           bool         same_pos = TRUE
          );
```

Parameters

data
The UTF-16 NULL-terminated UTF-16 litter-endian string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
/* UTF-16 string, "test" and CJK characters "测试"  */
UChar    data1[20] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
Pos(2,2);                          // current position at (2",2")

Font(2);                           // Assume font 2 is a TrueType font
                                   // with data type UTF16BE defined

Ru16(data1);                       // right put UTF-16 at (2",2")

              :
              :

ClosePage();
CloseDoc();
```

Right Align UTF-16 Text Converting from Legacy String

Function

Right aligns a single-line of the UTF-16BE string converting from the legacy codepage/charset string, at the current position.

Before calling this function, make sure the font ID you called with the "Font" function, was defined with data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Ru16c(
          char*        data,
          char*        fromcode = NULL,
          bool         same_pos = TRUE
          );
```

Parameters

data
The NULL-terminated legacy codepage string.

fromcode
The encoding name of the source string to be converted into UTF-16. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();

DefaultCode("GB18030");               // set default converter for input data

OpenPage(8.5,11);
               :
               :
Pos(2,2);                             // set current position at (2",2")

Font(2);                              // Assume font 2 is a TrueType font
                                      // with data type UTF16BE defined

Ru16c("test 测试");                    // right put UTF-16 converting from
                                      // Chinese GB18030
               :
               :

ClosePage();
CloseDoc();
```

Right Align UTF-8 Text

Function

Right aligns a single-line of the UTF-8 string at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Ru8(
         UChar8*       data,
         bool          same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated UTF-8 string.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
/* UTF-8 string, "test" and CJK characters "测试"  */
UChar8    data1[20] = "test\xe6\xb5\x8b\xe8\xaf\x95";

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
Pos(2,2);                          // current position at (2",2")

Font(2);                           // Assume font 2 is a TrueType font
                                   // with data type UTF8 defined

Ru8(data1);                        // right put UTF-8 at (2",2")

              :
              :

ClosePage();
CloseDoc();
```

Right Align UTF-8 Text Converting from Legacy String

Function

Right aligns a single-line of the UTF-8 string converting from the legacy codepage/charset string, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Ru8c(
         char*         data,
         char*         fromcode = NULL,
         bool          same_pos = TRUE
        );
```

Parameters

data
The NULL-terminated legacy codepage string.

fromcode
The encoding name of the source string to be converted into UTF-8. Default is NULL, default encoding name predefined by the DefaultCode() function is used. Refer to *MakeAFP document Encoding Names for* more details about the available names.

same_pos
Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
DefaultCode("GB18030");          // set default converter for input data

Pos(2,2);                        // current position at (2",2")

Font(2);                         // Assume font 2 is a TrueType font
                                 // with data type UTF8 defined

Ru8c("test 测试");                // right put UTF-8 converting from
                                 // Chinese GB18030

              :
              :

ClosePage();
CloseDoc();
```

Right Align UTF-8 Text Converting from UTF-16LE

Function

Right aligns a single-line of the UTF-8 string converting from the UTF16-LE text, at the current position.

Before calling this function, make sure the font ID you called with the “Font” function, was defined with the data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```
void Ru8u(
           UChar*       u16_data,
          );
```

Parameters

u16_data
The NULL-terminated UTF-16LE text string.

Sample

```
/* UTF-16 string, "test" and CJK characters "测试"  */
UChar    data1[] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
               :
               :
Pos(2,2);                          // current position to (2",2")

Font(2);                           // Assume font 2 is a TrueType font
                                   // with data type UTF8 defined

Ru8u(data);                        // Right put UTF-8 converting from
                                   // UTF16-LE

               :
               :

ClosePage();
CloseDoc();
```

Set Default Unit

Function

Sets default measurement unit and IPDS printer default units per inch, it must be called before calling the “Open Document” or “Open Page” function.

MakeAFP Weaver default is IN_U600 if you do not call this function.

Syntax

```
void SetUnit(
             unit      makeafp_unit
            );
```

Parameters

makeafp_unit
You can specify one of the following value:

CM_U240	CM, 240 units per inch
CM_U300	CM, 300 units per inch
CM_U360	CM, 360 units per inch
CM_U480	CM, 480 units per inch
CM_U600	CM, 600 units per inch
CM_U720	CM, 720 units per inch
CM_U1440	CM, 1440 units per inch
MM_U240	MM, 240 units per inch
MM_U300	MM, 300 units per inch
MM_U360	MM, 360 units per inch
MM_U480	MM, 480 units per inch
MM_U600	MM, 600 units per inch
MM_U720	MM, 720 units per inch
MM_U1440	MM, 1440 units per inch
IN_U240	Inch, 240 units per inch
IN_U300	Inch, 300 units per inch
IN_U360	Inch, 360 units per inch
IN_U480	Inch, 480 units per inch
IN_U600	Inch, 600 units per inch
IN_U720	Inch, 720 units per inch
IN_U1440	Inch, 1440 units per inch
PT_U240	Point, 240 units per inch
PT_U300	Point, 300 units per inch
PT_U360	Point, 360 units per inch
PT_U480	Point, 480 units per inch
PT_U600	Point, 600 units per inch
PT_U720	Point, 720 units per inch
PT_U1440	Point, 1440 units per inch

Sample

None.

Skip Lines

Function

Skips baseline position by a specific number of lines, and begins a new text line from left inline margin defined by the “Margin” function call, it increments the current baseline coordinate position by the number of lines times the baseline increment defined by either the “Lines Per Inch” or “Line Spacing” function call.

Syntax

```
void Skip(
          float    lines
         );
```

Parameters

lines
The number of lines to skip.

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
              :
              :
LineSp(0.25);       // Line spacing is 0.25", 4 LPI

Margin(0.8);        // left margin for the text is 0.8"

Skip(10.5);         // Skip 10.5 lines, baseline increment is
                    // 0.25" x 11.5 = 2.625"

              :
              :

ClosePage();
CloseDoc();
```

Start Session

Function

Starts a MakeAFP Weaver session before calling any other MakeAFP Weaver functions.

This function starts and establishes initiation of a MakeAFP Weaver session, opens a default input data file either in text or binary mode and output AFP document file in binary mode, parses the parameters defined in the MakeAFP Weaver definition file, and merges all the AFP resources and OpenType/TrueType fonts required by your program either by generating external AFP resource file or putting them inline within the output AFP document file; and it also retrieves AFP and OpenType/TrueType fonts information required by MakeAFP for text formatting and alignments.

Syntax

```
char* Start(
           char*          command_line_arguments = NULL,
          );
```

Parameters

command_line_arguments
It is mainly provided for calling from other programming languages, with which you may want to specify the command-line arguments directly, instead of specifying arguments while issuing commands.

Refer to *Chapter 2. Running MakeAFP Weaver in Batch Mode, MakeAFP Weaver Users' Guide,* for more details about command-line flag-arguments supported by MakeAFP Weaver.

Sample

```
void main( )
{

  Start();               // Start initiation, open default AFP input,
                         // AFP output and definition files, getting
                         // AFP resources and AFP font Information
      :
      :
      :

}
```

Text Orientation

Function

Sets the combination of inline and baseline orientations in which the subsequent text will be presented.

Syntax

```
void TextOrient(
                orientation      orientation = I0B90
               );
```

Parameters

orientation
The combination of inline and baseline orientations. The valid values are:

`I0B90` Text is rotated zero degrees clockwise. The text origin is at the upper left corner of the page. This is the default value.

`I0B270` Text is rotated zero degrees clockwise. The text origin is at the lower-left corner of the page.

`I90B180` Text is rotated 90 degrees clockwise. The text origin is at the upper-right corner of the page.

`I90B0` Text is rotated 90 degrees clockwise. The text origin is at the upper-left corner of the page.

`I180B270` Text is rotated 180 degrees clockwise. The text origin is at the lower- right corner of the page.

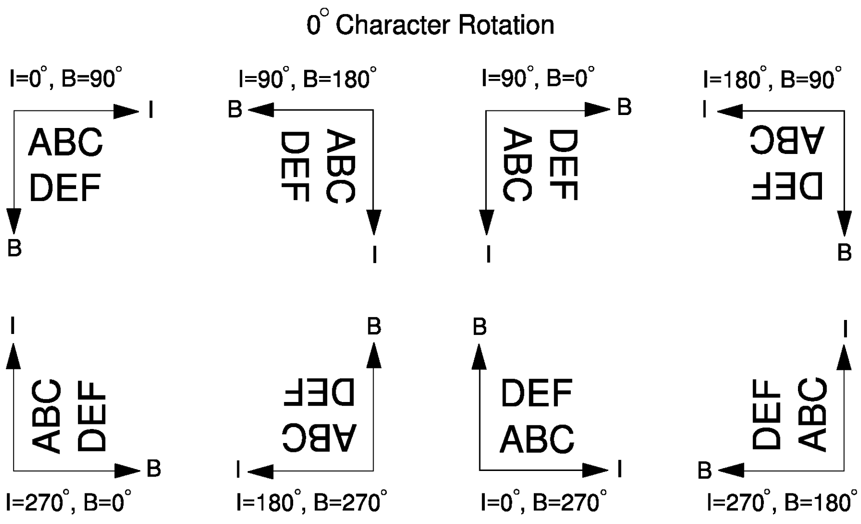
`I180B90` Text is rotated 180 degrees clockwise. The text origin is at the upper right corner of the page.

`I270B0` Text is rotated 270 degrees clockwise. The text origin is at the lower-left corner of the page.

`I270B180` Text is rotated 270 degrees clockwise. The text origin is at the lower right corner of the page.

Sample

This figure illustrates changes in orientation with no change in character rotation.



Trigger by a Location and a Pettern

Function

Defines a location and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

Syntax

```
bool Trigger(
              ushort      x_pos,
              ushort      y_pos,
              char*       mask
             );
```

Parameters

x_pos
Specifies the X position of the data field in PELS. With the MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y_pos
Specifies the Y position of the data field in PELS. With the MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

mask
Specifies a native text string or a pattern of symbols to be used to identify the data field captured from the AFP page. Make sure the “Encoding” function is previously called so that the non-ASCII or non-ASCII/ DBCS-PC AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically for the comparison with the string or pattern of symbols specified in the native encoding. Valid pattern symbols are:

'@’	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

You can specify an empty string as the mask if you only need to detect a trigger by its unique position without comparing of text string.

Sample

```
/*****************************************************************************/
```

```
                      /* This sample shows how to capture a trigger from last page of each group,
*/
                      /* get a feild from page 1 for add a barcode, mask an area and add a page
*/
                      /* segment.
*/
                      /*
*/
                      /* AFP was encoded in CP-037, USA EBCDIC
*/

       /*****************************************************************************/
                      int main( )
                      {
                        unsigned int i, grpPages, pageSN = 0;
                        char tmp[80], policyNo[20];
                        bool eog = 0;

                        $MaxPaging = 50;            // Maximum paging is up to 50 pages

                        SetUnit(IN_U600);           // Set default unit to inch

                        Start();                     // Start initiation, open default input,
                                                    // output and definition files, retrieves
                                                    // AFP resources, allocate memory

                        Encoding("ibm-037","ibm-437");   // AFP - CP037, PC - CP437

                        OpenDoc();                   // Open AFP document

                        while ($Edt == 0)           // Until end of AFP document
                        {
                          $Page = 0;                // Reset AFP page buffer number

                          do {

                          $Page++;                  // Point to next AFP page buffer

                          GetPage();                // Get a page from existing AFP file

                          if ($Page == 1)           // Get policy number from page 1
                            GetField2(2448, 2448, 6080, 6110, policyNo);

                          if ($Page > 2)
                           eog = Trigger(3744, 2338, "Part 1");  // detecting if it is a last page
of                                                              // a page group, "Part 1" text
                                                                       // string only appears at
last page
                                                                // of each page group

                        } while (!eop);            // Until end of each page group

                        eop = 0;                   // reset it for next group

                        // Now got all pages of a page group, now it is
                        // ready to compose the new AFP output

                        grpPages = $Page;          // keep total number of pages per group

                        for (i = 0; i < grpPages; i++)
                        {
                          $Page = i + 1;           // point to page buffer number to be opened
again

                          InclPseg("S1OWL", 0.3, 0.25);  // Add a page segment image
                          MaskArea(5, 0.4, 2, 0.75);     // Mask an area on every page

                          sprintf(tmp, "Page %d of %d", $Page, grpPages); // generate pagination
```

```
                         Font(1);  Pos(8, 0.45);  Rtxt(tmp);     // You can use an ASCII encoded
font
                                                                 // directly with MakeAFP Weaver
                         sprintf(tmp, "%06d", ++pageSN);         // generate page serial number
                         Font(2);  Pos(0.2, 10.8);  Ltxt(tmp);

                         sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);
                         BarCode(CODE128, tmp, 0.3, 2, 2, 0.2, DEG90);   // Add 1D and 2D barcodes
                         DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);

                         ClosePage();               // Close AFP page, write to AFP file
                       }
                     }

                     CloseDoc();                    // Close AFP document and its file

                     return 0;
                   }
```

Trigger by a location arear and a Pattern

Function

Defines a location area and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the "Get Field" function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the "Trigger" function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

Syntax

```
bool Trigger2(
            ushort     x1,
            ushort     x2,
            ushort     y1,
            ushort     y2,
            char*      mask
           );
```

Parameters

x1, x2
Specifies the X position range of the data field in PELS. With the MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

y1, y2
Specifies the Y position range of the data field in PELS. With the MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

mask
Specifies a native text string or a pattern of symbols to be used to identify the data field captured from the AFP page. Make sure the "Encoding" function is previously called so that the non-ASCII or non-ASCII/ DBCS-PC AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically for the comparison with the string or pattern of symbols specified in the native encoding. Valid pattern symbols are:

'@'	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any "@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

Sample

```
/****************************************************************************/
```

```
                      /* This sample shows how to capture a trigger from last page of each group,
*/
                      /* get a feild from page 1 for add a barcode, mask an area and add a page
*/
                      /* segment.                  AFP was encoded in CP-037, USA EBCDIC
*/

       /******************************************************************************/

                      int main( )
                      {
                        unsigned int i, grpPages, pageSN = 0;
                        char tmp[80], policyNo[20];
                        bool eog = 0;

                        $MaxPaging = 50;            // Maximum paging is up to 50 pages
                        SetUnit(IN_U600);            // Set default unit to inch

                        Start();                     // Start initiation, open default input,
                                                     // output and definition files, retrieves
                                                     // AFP resources, allocate memory

                        Encoding("ibm-037","ibm-437");   // AFP - CP037, PC - CP437

                        OpenDoc();                   // Open AFP document

                        while ($Edt == 0)            // Until end of AFP document
                        {
                          $Page = 0;                 // Reset AFP page buffer number

                          do {

                          $Page++;                   // Point to next AFP page buffer

                          GetPage();                 // Get a page from existing AFP file

                          if ($Page == 1)            // Get policy number from page 1
                            GetField2(2448, 2448, 6080, 6110, policyNo);

                          if ($Page > 2)
                           eog = Trigger(3744, 2338, "Part ##-#");  // detecting if it is a last
page                                                                       // a page group, like
"Part 68-1"                                                                // only appears at last
page

                                                                    // of each page group

                        } while (!eop);             // Until end of each page group

                        eop = 0;                    // reset it for next group

                        // Now got all pages of a page group, now it is
                        // ready to compose the new AFP output

                        grpPages = $Page;           // keep total number of pages per group

                        for (i = 0; i < grpPages; i++)
                        {
                          $Page = i + 1;            // point to page buffer number to be opened
again

                          InclPseg("S1OWL", 0.3, 0.25);  // Add a page segment image
                          MaskArea(5, 0.4, 2, 0.75);     // Mask an area on every page

                          sprintf(tmp, "Page %d of %d", $Page, grpPages); // generate pagination

                          Font(1);  Pos(8, 0.45);  Rtxt(tmp);     // You can use an ASCII encoded
font
```

```
                                              // directly with MakeAFP Weaver

      sprintf(tmp, "%06d", ++pageSN);         // generate page serial number
      Font(2);  Pos(0.2, 10.8);  Ltxt(tmp);

      sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);
      BarCode(CODE128, tmp, 0.3, 2, 2, 0.2, DEG90);   // Add 1D and 2D barcodes
      DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);

      ClosePage();               // Close AFP page, write to AFP file
    }
  }

  CloseDoc();                    // Close AFP document and its file

  return 0;
}
```

Trigger by an X-location Ranger and a Pattern

Function

Defines an X-location range and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

Syntax

```
bool TriggerX(
                ushort      x1,
                ushort      x2,
                char*       mask
             );
```

Parameters

x1, x2
Specifies the X position range of the data field in PELS. With the MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

mask
Specifies a native text string or a pattern of symbols to be used to identify the data field captured from the AFP page. Make sure the “Encoding” function is previously called so that the non-ASCII or non-ASCII/ DBCS-PC AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically for the comparison with the string or pattern of symbols specified in the native encoding. Valid pattern symbols are:

'@′	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

Sample

```
None.
```

Trigger by a Y-location Ranger and a Pettern

Function

Defines a Y-location range and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

Syntax

```
bool TriggerY(
               ushort      y1,
               ushort      y2,
               char*       mask
             );
```

Parameters

y1, y2
Specifies the Y position range of the data field in PELS. With the MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.

mask
Specifies a native text string or a pattern of symbols to be used to identify the data field captured from the AFP page. Make sure the “Encoding” function is previously called so that the non-ASCII or non-ASCII/ DBCS-PC AFP text string can be converted to the native ASCII or ASCII/DBCS-PC encoding automatically for the comparison with the string or pattern of symbols specified in the native encoding. Valid pattern symbols are:

'@’	A single alphabetic character (A to Z or a to z)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

Sample

```
None.
```

Trigger by Name of Copy Group

Function

Defines an AFP copy-group (also called medium map) name to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

This function must be called after an AFP page has been read-in from the existing AFP input file with the “Open Page” function so that you can check if the copy-group name is invoked before this page.

Syntax

```
bool TriggerCopygroup(
                          char*     copygroup
                         );
```

Parameters

copygroup
Specifies an AFP copy-group (also called medium map) name to uniquely identify the end of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

With the MakeAFP ShowPTX utility, you can find out which copy-group name and where it is invoked in your existing AFP.

Sample

```
None.
```

Trigger by Name of Data-Object Image

Function

Defines a data object image (like JPEG/TIFF/GIF) name to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

This function must be called after an AFP page has been read-in from the existing AFP input file with the “Open Page” function so that you can check up if the data-object name is included in this page.

Syntax

```
bool TriggerObjt(
                  char*       data_object
                 );
```

Parameters

data_object
Specifies a data-object name to uniquely identify the first page of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

With the MakeAFP ShowPTX utility, you can find out which data-object name and where it is included in your existing AFP.

Sample

```
None.
```

Trigger by Name of Overlay

Function

Defines an overlay name to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the “Trigger” function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

This function must be called after an AFP page has been read-in from the existing AFP input file with the “Open Page” function so that you can check up if the overlay name is included in this page.

Syntax

```
bool TriggerOvly(
                   char*        overlay
                  );
```

Parameters

overlay
Specifies an overlay name to uniquely identify the first page of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

With the MakeAFP ShowPTX utility, you can find out which overlay name and where it is included in your existing AFP.

Sample

```
/***************************************************************************/
/* This sample shows how to capture a trigger by an overlay name and       */
/* data fields from page 1, add AFP indexes and barcode to existing        */
/* AFP                                                                     */
/*                                                                         */
/* AFP was encoded in CP-037, USA EBCDIC                                   */
/***************************************************************************/

int main( )
{
   unsigned int i, grpPages, pageSN, groups;
   char tmp[80], mobileNo[20], custName[60];
   bool bog = 0;

   $MaxPaging = 50;          // Maximum paging is up to 50 pages

   SetUnit(IN_U600);                // Set default unit to inch

   Start();                  // Start initiation, open default input,
                             // output and definition files, retrieves
                             // AFP resources, allocate memory
```

```
Encoding("ibm-037","ibm-437");


OpenDoc();                // Open AFP document

$Page = 1;                // Set AFP page buffer number to 1 for the first
                          // page of AFP file

GetPage();                // Get first page of AFP file

while ($Edt == 0)       // Until end of AFP document
{
  GetField(660, 1080, custName);      // Get customer name

  GetField(4050, 900, mobileNo);      // Get customer mobile number

  do {

     $Page++;            // Point to next AFP page buffer

     GetPage();          // Get next page

     // detecting if it is the first page of a group,
     // overlay O1OVL1E only used by at first page of
     // each page group
     bog = TriggerOvly("O1OVL1E");

   } while (!bog && !$Edt);      // Until beginning of next page group or
                                        // End of AFP file

  bog = 0;                       // Reset it for next group

  // Now got all pages of a page group and first page of next group, now
  it  // is ready to process new AFP output

  if (!$Edt)                     // If not end of AFP document
     grpPages = $Page -1 ;       // Keep total number of pages per group,
                                 // need to minus 1 page of the first
                                 // page of next group
  sprintf(tmp, "%08d", ++groups);

  BgnIdx(tmp);                   // Auto-converts ASCII to EBCDIC for indexes
  PutIdx("Customer Name", custName);
  PutIdx("Mobile Number", mobileNo);

  for (i = 0; i < grpPages; i++)
  {
     $Page = i + 1;              // Point to page buffer number to be
  opened

     sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
     BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90);   // Add 1D barcode

     ClosePage();                // Close AFP page, write to AFP file
  }

  EndIdx();                      // End of group level index

  MovePage(1, grpPages + 1);     // As we got first page of next group
                                        // previously, now need move its
  contents
                                 // to page buffer 1 for the next page
  group
```

```
     $Page = 1;                     // Reset page buffer to 1 for next group
   }

   CloseDoc();                      // Close AFP document and its file

   #ifdef _DEBUG
     ViewAFP();                     // Only view AFP output in debug mode
   #endif

   return 0;
}
```

Trigger by Name of Page Segment

Function

Defines a page segment name to uniquely identify the first page of a page group or a specific page, or to be used with the "Get Field" function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

With the "Trigger" function, we can define the indication information that indicates which AFP page containing the data fields we need. The trigger must be consistent as a milepost throughout the AFP document.

The data fields are associated with triggers and contain the information that will be used for the AFP indexing or repurposes. Fields are defined by location or location range relative to the
Indication of the trigger.

This function must be called after an AFP page has been read-in from the existing AFP input file with the "Open Page" function so that you can check up if the page segment name is included in this page.

Syntax

```
bool TriggerPseg(
                   char*      page_segment
                 );
```

Parameters

page_segment
Specifies a page segment name to uniquely identify the first page of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

With the MakeAFP ShowPTX utility, you can find out which page segment name and where it is included in your existing AFP.

Sample

None.

Vertical Line

Function

Draws a vertical line.

Syntax

```
void Vline(
            float         x_pos,
            float         y_pos,
            float         length,
            float         thickness,
           );
```

Parameters

x_pos
The X starting position of the line, specify CP if you want to use the current position.

y_pos
The Y starting position of the line, specify CP if you want to use the current position.

length
The length of the line.

thickness
The thickness of the line.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
               :
               :
Color(RED);                     // defines color for the legacy line

Vline(10,10,100,1);             // draw a vertical blue line from
                                // (10,10)mm, its length is 100 mm,
                                // thickness is 1 mm
               :
               :
ClosePage();
CloseDoc();
```

Vertical Lines

Function

Repeat drawing vertical lines.

Syntax

```
void Vlines(
            float        x_pos,
            float        y_pos,
            float        length,
            float        thickness,
            ushort       repeat,
            float        space,
            ushort       direction = ACROSS
           );
```

Parameters

x_pos
The X starting position of the line, specify CP if you want to use the current position.

y_pos
The Y starting position of the line, specify CP if you want to use the current position.

length
The length of the line.

thickness
The thickness of the line.

repeat
The number of additional lines to be repeated.

space
The gap space between the lines.

direction
The direction of line repeating, valid values are ACROSS and DOWN, default is ACROSS.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
             :
             :
Color(BLUE);                     // defines color for texts and legacy line

Vlines(10,10,100,1,7,5,BLUE);    // draw 8 vertical blue line from
                                 // (10,10)mm, its length is 100 mm,
                                 // thickness is 1 mm, space is 5mm
             :
             :

ClosePage();
CloseDoc();
```

View AFP File

Function

Views the generated AFP file, it must be specified after the "Close Document" function request.

AFP viewer for Windows can be easily integrated with MakeAFP Weaver by the "View AFP" function, so that you can view the AFP file just generated immediately during your development or before printing.

With Windows Explorer, you can select "Tools → Folder Options → File Types → New" to link the AFP type file to an AFP viewer.

During your development, you can run the program in debug or execute mode with your MS Visual Studio C++ compiler. In your project settings, you can define the "Working directory" in which you can keep your input file and MakeAFP definition file, and then define "Program arguments" as -d definition_file -i input_file -o output_afp_file.

> * "-i input_file" is an optional parameter, for your development testing or for developing the overlay with MakeAFP, you may just key in the data within your program.

MakeAFP Weaver calls the AFP viewer automatically if an error message has taken place during your development or production, or once the 100 pages limitation is reached if it is running in demo mode without any software license key or hardware key.

Syntax

```
void ViewAFP(
             ushort      docNo = 1,
             char*       AFPviewer = NULL
            );
```

Parameters

docNo
Specifies which AFP document to be opened by AFP Viewer, valid values are 1 through 10, the default value is 1.

AFPviewer
The program name of the AFP viewer, fully qualified with path name in your hard disk, default is using your default AFP Viewer on the Windows system, if this parameter is not specified.

Sample

```
Start();

SetUnit(IN_U600);
OpenDoc( );
          :
          :
CloseDoc();                  // ViewAFP() must be called after AFP file is
                             // closed by CloseDoc() function

#ifdef DEBUG
```

```
  ViewAFP(1, "d:\\AFP Viewer\\ftdwinvw.exe");    // only view AFP in debug
#endif                                            // mode
```

X Absolute Position

Function

Sets the new horizontal absolute position (X) for the output text on the page. The origin position on the page is at (0, 0).

Syntax

```
void Xpos(
          float    x_position
        );
```

Parameters

x_position

The value of the absolute horizontal position from the page origin. Negative values are not valid.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
             :
             :
Xpos(5);                               // Set x position at 5 mm

             :
             :

ClosePage();
CloseDoc();
```

X Current Position (Query)

Function

Queries the current horizontal position on the page.

Syntax

```
float GetXpos( );
```

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.27, 11.67);
              :
              :
if ( GetXpos() > 5.5 )              // if current X position is more than
                                    // 5.5"
{
              :
              :
}
else
{
              :
              :

}

ClosePage();
CloseDoc();
```

X Move Relative Position

Function

Moes horizontal position (X) relative to the current horizontal coordinate position.

Syntax

```
void Xmove(
           float   x_move
          );
```

Parameters

x_move

The value of horizontal movement relative to the current presentation horizontal position (X). Positive value moves the position to the right; negative value moves the position to the left.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
              :
              :
Xmove(25);                           // Move 25 mm to the right
              :
              :
Xmove(-10);                          // Move 10 mm to the left


ClosePage();
CloseDoc();
```

Y Absolute Position

Function

Sets the new vertical absolute position for the output text on the page. The origin position on the page is at (0, 0).

Syntax

```
void Ypos(
           float      y_position
          );
```

Parameters

y_position
The value of the vertical position absolute from the page origin. Negative values are not valid.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
              :
              :
Ypos(15);                              // Set Y position at 15 mm

              :
              :

ClosePage();
CloseDoc();
```

Y Current Position (Query)

Function

Queries the current vertical position on the page.

Syntax

```
float GetYpos( );
```

Sample

```
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.27, 11.67);
               :
               :
if ( GetYpos() > 11.2 )             // if current Y position is more than
                                    // 11.2", may need to do page-breaking
{
               :
               :
}
else
{
               :
               :

}

ClosePage();
CloseDoc();
```

Y Move Relative Position

Function

Moves vertical position (Y) relative to the current vertical coordinate position.

Syntax

```
void Ymove(
            float     y_move
          );
```

Parameters

y_move

The value of vertical movement relative to the current presentation vertical position (Y). Positive values move the position down; negative values move the position up.

Sample

```
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
              :
              :
Ymove(25);                          // Move 25 mm down
              :
              :
Ymove(-10);                         // Move 10 mm up


ClosePage();
CloseDoc();
```

Chapter 2. MakeAFP Weaver Parameters

This chapter describes the MakeAFP Weaver parameters to be defined in the MakeAFP definition file, including the syntax rules and values.

Conventions Used in This Chapter

Highlighting

This chapter uses the following highlighting conventions:

- **Bold** Identifies commands, keywords, and other items, whose names are predefined
  by the MakeAFP or must be entered as-is.
- ***Italic*** Identifies parameters whose actual names or values you supply.

Syntax Notation

This chapter uses the following syntax notation:

- Italics within a command represent variables for which you must supply a value for. For instance:

  FONTLIB=*pathname*

  means that you enter **FONTLIB**= as shown and then replace the variable *pathname* with a value that represents any valid path name.

- Do not enter the following symbols as part of the command:

Vertical bar	**|**
Braces	**{ }**
Brackets	**[ ]**
Underscore	_

  The above symbols have the following meanings:

  - A vertical bar, **|**, between values, indicates that you can only enter one of the values with the command. For instance:

    PRMODE= { EBCDIC | SOSI1 | SOSI2 }

    means that when you enter **PROMODE=**, you can only specify one of the values.

  - Braces, { }, around values indicate a required value.

  - Brackets, [ ], around parameters indicate that they are optional. For instance:

    FONT1 = { *CDF* | *CHS, CDP* } [*,height_point*] [*,scale_ratio*]

means that height_point and scale_ration are the optional parameters.

- An underscore, _, indicates the default value, which MakeAFP uses if you do not specify the parameter with a non-default value. For instance:

  RESTYPE = { NONE | ALL | ......

  means that if the **RESTYPE** parameter is not entered, MakeAFP Weaver uses the default value of **NONE** for the RESTYPE parameter.

CMR – Specifies a Colo Management Resource

Function

Specifies a CMR for the AFP color management.

A separate CMR parameter is required for each CMR file, up to a maximum of 16 CMRs can be specified in a MakeAFP definition file.

Syntax

CMR*n* = *cmr_file*

Parameter

n
The CMR identifier number, when adding a CMR parameter, it is recommended that you use the next available number, beginning with 1 (one), a maximum of 16 CMRs can be specified in a MakeAFP definition file.

cmr_file
Specifies the file name of an AFP CMR with or without file name extension of *.cmr.

Sample

`objtlib = d:\afp_cmr`	Specify the object library path where CMRs are stored
`cmr1 = EUROISOCC001000.cmr`	Specify CMR of Europe ISO Coated
`cmr2 = JPSTD2CC001000.cmr`	Specify CMR of Japan Standard V2

CPGID – Specifies a Code Page Identifier

Function

Specifies the two through four digits Code Page Global Identifier that defines an IBM-registered code page ID, which is required whenever the index values and attribute names are specified with the MakeAFP Weaver indexing functions.

The Code Page Global Identifier is used by an AFP viewer or AFP archiving system client software, which must display indexing information. This software use this identifier with code page translation tables to represent the index attribute and value data.

For more information about IBM code pages, refer to *IBM AFP Fonts: Technical Reference for Code Pages*, S544–3802 and MakeAFP's documents *IBM CodePage Name and CPGID Summary, and Encoding Alias Names.*

Syntax

CPGID = *codepageID*

Parameter

codepageID
Any valid code page ID, which is a three through the four-character decimal value that defines an IBM-registered code page ID.

If this parameter is not specified, MakeAFP Weaver uses code page ID 850 (Personal Computer - Multilingual Page ASCII) as the default.

Sample

`cpgid = 437`	Code page ID for US English ASCII

FDEF – Sepcifies a Form Fefintion

Function

Specifies the file name of form definition to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver. Once it is specified, then the form definition resources from input AFP file is being removed from the output AFP.

The form definition defines the placement of the page on the form, the input and output bins to use, duplex printing, and so on. You must call a form definition when you print your job. If the AFP file doesn't contain an inline form definition, then you can either specify a form definition by name while you submit your print job or use the default form definition set up by your AFP print server installation.

Syntax

FDEF = ***fdefname***

Parameter

fdefname
Any valid form definition name. The form definition name can be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix F1, if there is one.

Sample

```
fdef = F1TEST01
```

FDEFLIB – Specifies the Library Path of Form Defintions

Function

Specifies the directories in which form definitions are stored.

Syntax

FDEFLIB = ***pathlist***

Parameter

pathlist
Any valid search path. You must use a semicolon (;) to separate multiple paths. MakeAFP Weaver searches the paths in the order you specified.

When MakeAFP Weaver finds more than one form definition with the same base filename in the same directory, it selects the matching form definition by the following file extension search order:

1. No filename extension
2. FDE
3. FIL
4. FDEF38PP

Some FROMDEF file extensions may not be supported by your AFP print server.

Sample

```
fdeflib = c:\makeafp\reslib;d:\ipmwin\reslib
```

FONT – Specifies an AFP FOCA Raster or Outline Font

Function

Specifies a single-byte or double-byte font to be used by the MakeAFP Weaver for the AFP text data stream to be added.

A separate FONT parameter is required for each font, up to a maximum of 32 fonts can be specified in a MakeAFP definition file.

Syntax

FONTn = { *CDF* | *CHS, CDP* } [,*height_point*] [,*scale_ratio*]

Parameter

n
The Font identifier number, when adding a font parameter, it is recommended that you use the next available number, beginning with 1 (one), a maximum of 127 fonts can be specified in a MakeAFP definition file.

CDF
Any valid AFP coded font name, up to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix X0 for AFP raster font or XZ for AFP outline font.

CHS
Any valid AFP character set name, up to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix C0 for AFP raster font or CZ for AFP outline font.

CDP
Any valid AFP coded page name, up to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix T1. Make sure you select the correct coded page for your input data, refer to Appendix A to Appendix B for more information.

height_point
Specifies the height of the AFP outline font in points (Each point is equal to 1/72 of one inch).

scale_ratio

Optional, specifies the ratio of font width scaling in percent with an outline font. For instance, specifying scale ratio 200 yields a font with characters string width twice as wide (200% as wide) as normal.

** Font width scaling may not be supported by some AFP viewers.*

Sample

`font1 = x0gt10`	Specify an AFP raster font by coded font name
`font2 = c0d0gt12,t1d0base`	Specify an AFP raster font by character set and coded page name
`font3 = xzhe00,12.5`	Specify an AFP outline font by coded font name and size
`font4 = czh210,t1v10037,10`	Specify an AFP outline font by character set, coded page name and point size

FONT – Specifies an OpenType/TrueType Font

Function

Specifies an OpenType/TrueType font with font type extension (.ttf, .otf or .ttc) to be used in MakeAFP formatting, and also indicates the user's input data type to be used with the font.

A separate FONT parameter is required for each OpenType/TrueType font, up to a maximum of 127 fonts can be specified in a MakeAFP definition file.

Syntax

FONTn = { *ttf_filename* | (*ttf_filename*, *font_index*) }, *ecoding, height_point* [,*scale_tatio*]

Parameter

n
The Font identifier number, when adding a font parameter, it is recommended that you use the next available number, beginning with 1 (one), a maximum of 127 fonts can be specified in a MakeAFP definition file.

ttf_filename
Any valid filename of the OpenType/TrueType font or TrueType font collection with file extensions of.ttf,.otf and.ttc.

ttf_index
The index number of a TrueType font within a TrueType Collection (*.ttc), which includes multiple TrueType fonts in a single file, default is 0 referring to the first TrueType font in TTC. TTC is mostly used for the East Asian CJK fonts.

encoding
Specifies the encoding to be used to use OpenType/TrueType fonts.

Most of the legacy AFP data stream is encoded by the EBCDIC-based and ASCII-based encoding schemes, although now OpenType/TrueType fonts are encoded in Unicode UTF-16, you can continue using the legacy encoding with AFP, but you must indicate its legacy encoding scheme by an AFP code page name. Code point conversions from the legacy encodings, such as from ASCII, EBCDIC, and DBCS-HOST to UTF-16BE, are performed in the presentation device for AFP, for example, by the IPDS printer or AFP viewer.

The following encoding types are allowed:

`T1xxxxxx`	ASCII, EBCDIC, DBCS-HOST codepage name defined by IBM, refer to Appendix A to B for more details about IBM codepage name.

`UTF8`	Unicode data is encoded in UTF-8. Simple code point conversions from UTF8 to UTF-16BE, is performed quickly in the presentation device for AFP.
`UTF16BE`	Unicode data is encoded in UTF-16 big-endian.

height_point
Specifies the height of the outline font in points (Each point is equal to 1/72 of one inch).

scale_ratio
Optional, specifies the ratio of font width scaling in percent with an outline font. For instance, specifying scale ratio 200 yields a font with characters string width twice as wide (200% as wide) as normal.

** Font width scaling may not be supported by some AFP viewers.*

Sample

`font1=tahoma.ttf,T1000437,12`	Specifies a TrueType font, height 12 and encoding is by ASCII codepage T1000437 (US English for PC)
`font2=(simsun18030.ttc,1),UTF16BE,11,120`	Specifies the second font in the TrueType font collection, height 11, width scale 120% and encoding is by Unicode UTF-16BE
`font3 = xzhe00,12.5`	Specifies an AFP FOCA outline font by coded font name and font height

FONTLIB – Specifies the Library Path of Fonts

Function

Specifies the directories in which AFP fonts and OpenType/TrueType fonts are stored.

Syntax

FONTLIB = *pathlist*

Parameter

pathlist
Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.

When MakeAFP Weaver finds more than one AFP font with the same base filename in the same directory, it selects the matching AFP font by the following file extension search order:

1. No filename extension
2. OLN
3. 600
4. 480
5. 360
6. 300
7. 240
8. ECP
9. CDP
10. CHS
11. CDF
12. CFT
13. FONTOLN
14. FONT240
15. FONT300

16. FONT38PP

Note: Some file extensions may not be supported by your AFP print server.

Sample

```
fontlib=c:\winnt\fonts;c:\makeafp\reslib;d:\ipmwin\fontlib
```

INDEXOBJ – Specifies Generating of the AFP Index Object File

Function

Specifies whether the AFP index object file is to be generated or not. MakeAFP Weaver puts group-level index entries into the index object file.

To achieve the best AFP data loading performance with an AFP archiving system, you need the AFP index file to be loaded together with the AFP document file and AFP resource file.

Refer to IBM *Content Manager OnDemand for Multiplatforms Administration Guide for* more details about loading a previously indexed AFP file directly.

Syntax

INDEXOBJ = { YES | <u>NO</u> }

Parameter

YES
Specifies that the AFP index object file is generated to be used by an AFP archiving system or AFP Viewer. MakeAFP Weaver generates the AFP index object file with the file name extension .ind.

Make sure the CPGID parameter is defined in your MakeAFP Weaver definition file properly.

NO
This is the default value, there is no AFP Index object file to be generated. The Index Object file is not required for printing.

OBJT – Specifies an AFP Object or non-AFP Object

Function

Specifies the file name of the AFP or non-AFP object to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver.

A separate OBJT parameter is required for each object, and a maximum of 127 objects can be specified in a MakeAFP definition file.

If you want objects to be loaded to the printer before the page begins printing, or if objects are used repeatedly and need to be available in the printer memory during printing, then you must define them with OBJT parameters to let MakeAFP Weaver build a catalog of objects being used in the AFP file to hard load them into the printer memory before printing starts.

Syntax

OBJT = *objtname, type*

Parameter

objtname
Any valid object name exclusive of the filename extension. The float -quoted object name can be 1 to 125 alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

type
Indicates type of the object:

`BCOCA`	AFP BCOCA barcode object
`GOCA`	AFP GOCA graphic object
`IOCA`	AFP IOCA image object
`PSEG`	AFP Page Segment image object
`BMP`	Windows Device Dependent Bit Map
`EPS`	Encapsulated Postscript
`EPSTR`	EPS with Transparency
`GIF`	Graphics Interchange Format
`PCX`	Paintbrush Picture File Format
`JPEG`	JPEG file Interchange Format
`JPEG2`	JPEG2000 file Interchange Format
`PCL`	PCL Page Object
`PDF`	PDF Single Page Object
`PDFSPOTR`	PDF Single Page Object with Transparency
`TIFF`	Tag Image File Format

The above objects require appropriate support of the IPDS printer and AFP print server to print.

Sample

```
objt = FLOWER1,JPEG
objt = "Orchid Flower",TIFF
```

OBJTLIB – Specifies the Library Path of Image and CMR Objects

Function

Specifies the directories in which AFP objects, non-AFP objects, and CMRs (Color Management Resources) are stored.

Syntax

OBJTLIB = ***pathlist***

Parameter

pathlist
Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.

When MakeAFP Weaver finds more than one data-object image with the same base filename in the same directory, it selects the matching data-object image by the following file extension search order:

12. No filename extension
13. JPG
14. TIF
15. GIF

16. JP2
17. EPS
18. PDF
19. BMP
20. PCX
21. PCL
22. OBJ

Note: Some file extensions may not be supported by your AFP print server.

Sample

```
objtlib = c:\makeafp\imglib;d:\ipmwin\reslib
```

OVLY – Specifies an Overlay

Function

Specifies the file name of the overlay to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver.

A separate OVLY parameter is required for each overlay, and a maximum of 127 overlays can be specified in a MakeAFP definition file.

If you want overlays to be loaded to the printer before the page begins printing, or if overlays are used repeatedly and need to be available in the printer memory during printing, then you must define them with OVLY parameters to let MakeAFP Weaver build a catalog of overlays being used in the AFP file to hard load them into printer memory before printing starts.

Syntax

OVLY = *ovlyname*

Parameter

ovlyname
Any valid overlay name. The overlay name can be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix O1, if there is one.

Sample

```
ovly = O1CDP01
ovly = O1CDP02
```

OVLYLIB – Specifies the Library Path of Overlays

Function

Specifies the directories in which AFP overlays are stored.

Syntax

OVLYLIB = *pathlist*

Parameter

pathlist
Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.

When MakeAFP Weaver finds more than one overlay with the same base filename in the same directory, it selects the matching overlay by the following file extension search order:

1. No filename extension
2. 600
3. 480
4. 360
5. 300
6. 240
7. OVL
8. OLY
9. OVR
10. OVLY38PP
11. AFP

Some file extensions may not be supported by your AFP print server.

Sample

```
ovlylib = c:\makeafp\reslib;d:\ipmwin\reslib
```

PRMODE – Specifies the Type of Input Data and Processing Option

Function

Specifies the type of input data and whether MakeAFP Weaver must perform optional processing on that data or not.

MakeAFP Weaver default is ASCII input data if you do not specify this parameter.

Syntax

PRMODE= { EBCDIC | SOSI1 | SOSI2 }

Parameter

EBCDIC
Specifies that input data is EBCDIC encoding from IBM mainframes.

SOSI1
Specifies that input data is SBCS-HOST/DBCS-HOST encoding and each SO(shift-out), SI(shift-in) code is to be converted to a white-space character.

SOSI2
Specifies that input data is SBCS-HOST/DBCS-HOST encoding and each SO(shift-out), SI(shift-in) code is to be escaped.

Sample

```
prmode = sosi1
```

PSEG – Specifies a Page Segment

Function

Specifies the file name of the page segment to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver.

A separate PSEG parameter is required for each page segment, and a maximum of 127 page segments can be specified in a MakeAFP definition file.

If you want page segments to be loaded to the printer before the page begins printing, or if page segments are used repeatedly and need to be available in the printer memory during printing, then you must define them with PSEG parameters to let MakeAFP Weaver build a catalog of page segments being used in the AFP file to hard load them into printer memory before printing starts.

Syntax

PSEG = *psegname*

Parameter

psegname
Any valid page segment name. The page segment name can be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix S1, if there is one.

Sample

```
pseg = S1CDP01
pseg = S1CDP02
```

PSEGLIB – Specifies the Library Path of Page Segments

Function

Specifies the directories in which page segments are stored.

Syntax

PSEGLIB = *pathlist*

Parameter

pathlist
Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.

When MakeAFP Weaver finds more than one-page segment with the same base filename in the same directory, it selects the matching page segment by the following file extension search order:

1. No filename extension
2. 600
3. 480
4. 360
5. 300
6. 240
7. PSG
8. PSE
9. PSEG38PP
10. AFP

Some file extensions may not be supported by your AFP print server.

Sample

```
pseglib = c:\makeafp\reslib;d:\ipmwin\reslib
```

RESLIB – Specifies the Library Path of AFP Resources and Objects

Function

Specifies the directories in which form definitions, overlays, page segments, AFP & non-AFP objects, AFP fonts, and OpenType/TrueType fonts are stored.

Syntax

RESLIB = ***pathlist***

Parameter

pathlist
Any valid search path. You must use a semicolon (;) to separate multiple paths. MakeAFP Weaver searches the paths in the order you specified.

Sample

```
reslib = c:\makeafp\reslib;d:\ipmwin\reslib;c:\winnt\fonts
```

RESTYPE - Specifies the Types of Resources to be Retrieved

Function

Specifies the types of resources that should be transferred from the input AFP file and retrieved from the resource directories if any new resourced is used, and whether the resources are being embedded inline within the AFP output document or as a separated AFP resource file.

Syntax

RESTYPE = { <u>NONE</u> | ALL | [,CMR] [,FDEF] [,FONT] [,OBJT] [,OVLY] [,PSEG] } [,INLINE]

Parameter

MakeAFP Weaver supports the specification of the parameters in any combination.

NONE
Specifies that no AFP resources file has been created or AFP resources are written inline within the AFP output document. This is the default, make sure that all AFP resources are available on the AFP print server.

ALL
Specifies that all AFP resources, OpenType/TrueType fonts, and non-AFP data-objects are embedded in the resource file or inline within the AFP output document.

CMR
Specifies that all CMRs (Color Management Resources) are embedded in the resource file or inline within the AFP document file.

FDEF
Specifies that the form definition is embedded in the resource file or inline within the AFP output document.

FONT
Specifies that all AFP fonts and OpenType/TrueType fonts are embedded in the resource file or inline within the AFP output document.

OBJT

Specifies that all AFP objects or non-AFP objects are embedded in the resource file or inline within the AFP output document.

OVLY
Specifies that all overlays are embedded in the resource file or inline within the AFP output document.

PSEG
Specifies that all page segments are embedded in the resource file or inline within the AFP output document.

INLINE
Specifies that resources are embedded inline within the AFP document file, otherwise a separate AFP resource file with the filename extension .res is being generated, which can be used by an AFP archiving system directly, like IBM Content Manager OnDemand.

AFP print server treats inline resources as the private AFP resources, and they will be purged from IPDS printer memory automatically after the job is printed successfully.

Sample

Include all AFP resources inline:

```
restype = all,inline
```

Include form definition, overlays, non-AFP object and page segments inline for viewing by IBM AFP viewer and IBM DB2 Contect Manager OnDemand:

```
restype = fdef,ovly,pseg,objt,inline
```

Chapter 3. MakeAFP Weaver Variables

In addition to the MakeAFP Weaver functions described in Chapter 1, there are several variables maintained by MakeAFP Weaver for internal use or exchanging of information during the data formatting. Some of the MakeAFP Weaver variables described below are accessible from your program.

$Bng – Begin of AFP Page Group Index

Indicates whether the “Begin of Name Group” of AFP index boundary has been detected after the “Get Page” function is called for the reading of an AFP page from the input AFP file. The “Get Page” function reads an AFP page from the AFP input file and also sets $Bng variable to TRUE if a “Begin of Name Group” boundary is detected.

$Eng – End of AFP Page Group Index

Indicates whether the “End of Name Group” of AFP index boundary has been detected after the “Get Page” function is called for the reading of an AFP page from the input AFP file. The “Get Page” function reads an AFP page from the AFP input file and also sets $End variable to TRUE if an “End of Name Group” boundary is detected.

$Edt – End of AFP Document

Indicates whether the “End of AFP Document” has been detected after the “Get Page” function is called for the reading of an AFP page from the input AFP file. The “Get Page” function reads an AFP page from the AFP input file and also sets $Edt variable to TRUE if the “End of Document” boundary is detected.

$MaxPaging – Maximum Number of AFP Page Buffers

Defines the maximum number of AFP page buffers. For generating page pagination, such as “Page 347 of 1000”, we need to keep composed AFP data in the AFP page buffers first. With MakeAFP Weaver, you can open multiple pages by “Open Page” function calls, and then process different pages in an interleaved manner once each page is initialized, all the composed AFP data stream will be kept in memory buffers in page-level, and finally, after you have completed all the formatting and counted all the pages of a page group, you have to put your pagination text and OMR in each page just before you end the page with “Close Page” function.

Big value takes big memory, only define this value as big as your maximum requirement for pagination. MakeAFP Weaver default value is 1, you can override its value before you start a MakeAFP Weaver session by calling the function of “Start();”. MakeAFP Weaver reports an error message if its value is not enough for your job.

$Page – Current AFP Page Buffer Number

Defines the current AFP page buffer number. With $Page variable, you can directly switch to any AFP page buffer is to be opened with the “Open Page” function request, or access it again.

Chapter 4. String Manipulation Functions

Although MS Visual Studio C++ provides comprehensive powerful functions for file input & output handling, searching and sorting, memory buffer manipulation, data conversion, string manipulation, directory control, etc, you still may need some of MakeAFP's complementary functions specially developed for data formatting requirements to assist your MakeAFP application developments.

Refer to Microsoft MSDN library for more detailed information about the functions provided by MS Visual Studio C++ in its run-time library routines, iostream library and standard C++ library,

The descriptions of the MakeAFP Weaver functions for string manipulation are listed in alphabetic order. The description of Each function includes the following sections:

Function
A description of the major purpose of the function.

Syntax
A diagram showing the function parameters.

Parameters
Explanation of each parameter.

Function Call Samples
Provides samples for using the function. All sample functions assume that prerequisite calls and variable definitions have been made before the sample function call.

Default Values
When calling these functions, every parameter must be specified in the order shown in this chapter. MakeAFP provides default values to some parameters for simplifying the use of the function, so you can omit them by default values when you invoke the function, but when your program omits parameters for a function that provides default values, your program must omit all the parameters that follow. In other words, you cannot omit a parameter in the middle.

Comma Float

Function

Formats a float using commas as the thousandth separators and a specified number of significant fractional digits.

Syntax

```
char *CommaFloat(
                 double   float _value,
                 ushort   fraction_digits
                );
```

Parameters

float _value
Source float value.

fraction_digits
The number of significant fractional digits.

Sample

```
float total = 129894.5698;
printf("Total Amount: %s", CommaFloat(total,2));
```

Output: `Total Amount: 129,894.57`

Comma Integer

Function

Formats a 64-bit integer using commas as the thousandth separators.

Syntax

```
char *CommaInt(
              _int64     integer_value
             );
```

Parameters

integer_value
Source 64-bit integer value.

fraction_digits
The number of significant fractional digits.

Sample

```
_int64 total = 1298945698123;
printf("Total Amount: %s", CommaInt(total));
```

Output: `Total Amount: 1,298,945,698,123`

Comma Digital String

Function

Formats a digital string using commas as the thousandth separators and a specified number of significant fractional digits, removing the leading zeros.

Syntax

```
char *CommaDigit(
                 str*     digital_string,
                 ushort   fraction_digits
                );

char *CommaDigit2(
                  str*    digital_string,
                 );
```

Parameters

digital_string
Source digital data string.

fraction_digits
The number of significant fractional digits.

Sample

```
char data1[20] = "001298945698123";
printf("Total Amount: %s", CommaDigit(data1,2));
```

Output:
```
Total Amount: 12,989,456,981.23
```

```
char data2[10] = "12989.49";
printf("Total Amount: %s", CommaDigit2(data2);
```

Output:
```
Total Amount: 12,989.49
```

Delete Characters

Function

Deletes a range of characters from the string.

Syntax

```
char *Delete(
           char*       string,
           ushort      start_col,
           ushort      length
          );
```

Parameters

string
Source data string.

start_col
Starting character position to delete.

length
The length of characters to be deleted.

Sample

```
char data[30] = "This is a string testing";
printf("After Deleted: %s", Delete(data,11,7));
```

Output:

```
    After Deleted: This is a testing
```

Extract Substrings

Function

Extracts a substring or multiple substrings delimited by the given separator(s).

Syntax

Extract once by a delimiter:

```
char *Extract1(
               char*     srcStr,
               ushort    order_pos,
               char*     delimiter
              );
```

Extract multiple time by a delimiter:

```
void Extract(
             char*        dstStr_array[],
             char*        srcStr,
             char         delimiter,
             char         qualifier
            );

void Extract2(
             char*        dstStr_array[],
             char*        srcStr,
             char*        delimiters
            );
```

Parameters

dstStr_array
Destination array of the strings extracted.

srcStr
Source string comprising of delimited character(s) substrings.

order_pos
Order position number of the substring to be extracted.

delimiters
Set of delimiter characters.

delimiter
A delimiter character.

qualifier
A character as the qualifier.

Sample

```
char src1[256] = "substring1;substring2;substring3;substring4";
char src2[256] = "field1:,123,456,000.00:,field3:,12,341.00:,field5";
char src3[256] = "test1,'168,456,000.00',test3,'88,666.00'";
char *dst[20];

printf("Extracted 3rd substrings is: %s", Extract1(src1, 3,";"));

Extract(dst,src2,":,");

printf("Extracted substrings 1 are: %s %s %s %s",
       dst[0], dst[1], dst[2], dst[3]);

Extract(dst, src3, ',', '\'');

printf("Extracted substrings 2 are: %s %s %s %s",
       dst[0], dst[1], dst[2], dst[3]);
```

Output:

```
Extracted 3rd substrings is: substring3
Extracted substrings 1 are: field1 123,456,000.00 field3 12,341.00
Extracted substrings 2 are: test1 168,456,000.00 test3 88,666.00
```

Find String

Function

Checks whether a string is in the data string returns its position if found, otherwise returns 0.

Syntax

```
int Find1(
          char*        str,
          char*        search,
          int          start_pos
         )

int Find2(
          char*        str,
          char*        search,
          int          start_pos,
          int          stop_pos
         )
```

Parameters

str
Source data string.

search
Search string.

start_pos
The position to start the search.

stop_pos
The position to stop the search.

Sample

```
char str[80] = "This is data string search testing";

int pos = Find2(str, "search", 9, 30);
```

Return:

```
        21
```

First Character

Function

Returns the position of the first non-white-space character.

Syntax

```
int FristChar(
               char*        string
              );
```

Parameters

string
Source data string.

Sample

```
char str[80] = "       This is data string";
int pos = FirstChar(str);
```

Return:

```
        8
```

In Substitution Table

Function

Checks whether a string is in the substitution table. Returns 1 if the string is found in the substitution table, otherwise returns 0.

Syntax

```
int InSubst(
             char*        subst_tbl[][2],
             char*        search
            );
```

Parameters

sbst_tbl
Substitution table.

search
search string.

Sample

```
char *payment [][2] = { {"001", "Cash Payment"},
                        {"005", "Master Payment"},
                        {"003", "Visa Payment"},
                        {"007", "Check Payment"},
                        {"011", "GORO payment"},
                        {"\0", "\0"} };   // End of table

int intab = InSubst(payment, "002");
    intab = InSubst(payment, "007");
```

Return:

```
        0
        1
```

Insert String

Function

Inserts a character string after the specified position.

Syntax

```
char *Insert(
              char*       srcStr,
              ushort      pos,
              ushort      insertStr
             );
```

Parameters

string
Source data string.

pos
Character position where you insert a string after.

insertStr
The string to be inserted in the destination string.

Sample

```
char src[256] = "This is data string";
printf("%s", Insert(src,8,"inserted "));
```

Output:

```
       This is inserted data string
```

Is Empty

Function

Checks if a string is either of 0-byte length or contains white-space characters only. Returns 1 if it is empty otherwise returns 0.

Syntax

```
int IsEmpty(
             char*     string
            );
```

Parameters

string
data string.

Sample

```
char str1[80] = "This is data string";
char str2[80] = "          ";
char str3[80] = NULL;

int empty = IsEmpty(str1);

    empty = IsEmpty(str2);
    empty = IsEmpty(str3);
```

Return:

```
         0
         1
         1
```

Last Character

Function

Returns the position of the last non-white-space character.

Syntax

```
int LastChar(
              char*       string
             );
```

Parameters

string
Source data string.

Sample

```
char str[80] = "This is data string        ";
int pos = LastChar(str);
```

Return:

```
         19
```

Left Copy

Function

Copies characters from the left of the source string to destination string with null-terminated.

Syntax

```
char *Lcp(
            char*    dstStr,
            char*    srcStr,
            ushort   length
           );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

length
The number of characters to be copied.

Sample

```
char src[] = "This is a string testing";
char dst[30];
printf("Left Copied: %s", Lcp(dst, src, 16));
```

Output:

```
       Left Copied: This is a string
```

Left Copy and Pad

Function

Copies characters from the left of source string to destination string which may be padded with the pad character if the length of the source string is less than the specified length.

Syntax

```
char *LcpPad(
               char*         dstStr,
               char*         srcStr,
               ushort        length,
               char          pad
             );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

length
The length of the destination string.

pad
The character to be used to pad destination string.

Sample

```
char src[] = "This is padded string";
char dst[26];
printf("Padded String: %s", LcpPad(dst, src, 25, '.'));
```

Output:

```
Padded String: This is padded string....
```

Left Copy and Right Trim

Function

Copies characters from the left of the source string to the destination string. The white-space, carriage return, new line control codes on the right side of the destination will be trimmed before being terminated with NULL.

Syntax

```
char *LcpRtrim(
                char*      dstStr,
                char*      srcStr,
                ushort     length
               );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

length
The number of characters to be copied.

Sample

```
char src[] = "This is a string testing          The second string";
char dst[100];
printf("'Left Copied: %s'", LcpRtrim(dst, src, 28));
```

Output:

```
'Left Copied: This is a string testing'
```

Left Trim

Function

Trims white-space characters from the left side of the source string.

Syntax

```
char *Ltrim(
           char*  string
          );
```

Parameters

string
Data string to be left trim.

Sample

```
char str[] = "     This is a string testing";
printf("Left Trimmed: %s", Ltrim(str));
```

Output:

```
      Left Trimmed: This is a string testing
```

Left Trim for EBCDIC

Function

Trims EBCDIC white-space characters from the left side of the source EBCDIC string.

Syntax

```
char *E_Ltrim(
             char*     string
            );
```

Parameters

string
EBCDIC data string to be left trim.

Sample

Refer to the sample for the “LTrim” function.

Left Trim and Concatenate Strings

Function

Trims white-space characters from the left sides of multiple data strings before concatenating them.

Syntax

```
char *LtrimCat(
              char*          separator,
              char*          string,...
             );
```

Parameters

separator
characters to be inserted between concatenated strings.

string,...
variable-argument lists of multiple strings.

Sample

```
char str1[] = "      This is a string testing.";
char str2[] = "    The second string.";
printf("After Concatenation: %s", LtrimCat(" ", str1, str2));
```

Output: `After Concatenation: This is a string testing. The second string.`

Match String Comparing

Function

Recursively compares a string to a pattern, returning 1 if a match is found or 0 if not.

Syntax

```
int Match(
           char*    string,
           char*    pattern,
           bool     ignore_case
         );
```

Parameters

string
The NULL-terminated strings to compare.

pattern
The NULL-terminated pattern string to be used for the comparison. The general syntax of the pattern is:

`*'	Matches any sequence of characters (zero or more)
`?'	Matches any single character
[SET]	Matches any character in the specified set
[!SET] or [^SET]	Matches any character, not in the specified set

A set is composed of characters or ranges; a range looks like ``character hyphen character'' (as in 0-9 or A-Z). [0-9a-zA-Z_] is the minimal set of characters allowed in the [..] pattern construct.

To suppress the special syntactic significance of any of "[]*?!^-\", inside or outside a [..] construct, and match the character exactly, precede it with a "\" (backslash).

ignore_case
Specifies whether the upper and lower case is ignored.

Sample

```
char str1[] = "This is data string";
char str2[] = "TX 20890";
char str3[] = "Answer?";

int rc = Match(str1, "*.*", 0);
    rc = Match(str2, "[A-Z][A-Z] [0-9][0-9][0-9][0-9][0-9]", 0);
    rc = Match(str3, "*\?", 0);
```

Return: 0
1
1

Pattern Searching

Function

Searches a string for a set of characters that match a specified pattern, returns the characters

if

it is finding a match.

Syntax

```
Char *Pattern(
              char*       string,
              char*       pattern,
              int         start_pos
             );
```

Parameters

string
The NULL-terminated strings within which to search.

pattern
A NULL-terminated string of specification characters that identifies the pattern to seek. Valid characters are:

'@’	A single alphabetic character (*A* to *Z* or *a* to *z*)
'#'	A single numeric character (0 to 9)
'&'	A single alphabetic *or* numeric character
'+'	A single blank *or* numeric character
'='	A single blank or alphabetic character
'~'	A single non-blank character
'?'	Any single character

To suppress the special syntactic significance of any of “@#&+?~=", and match the character exactly, precede it with a "\" (backslash).

Start_pos
Specifies starting position in the string at which to begin the search.

Sample

```
char str[] = "Boulder, CO 12345-5768-88 USA";

Pattern(str, "#####", 1);
Pattern(str, "##-##", 15);
Pattern(str, "## USA", 23);
```

Return:

```
12345
45-57
88 USA
```

Remove String

Function

Removes a single instant of string or multiple instances of the string.

Syntax

Remove once:

```
char *Remove1(
              char*     srcStr,
              char*     rmStr
             );
```

Remove all:

```
char *Remove(
             char*      srcStr,
             char*      rmStr
            );
```

Parameters

srcStr
Source string.

rmStr
The string to be removed from the source string.

Sample

```
char str[] = "This is a string testing string.";
printf("After Removed once: %s\n", Remove1(str," string"));
printf("After Removed all: %s\n", Remove(str," string"));
```

Output:

```
After Removed once: This is a testing string.
After Removed all: This is a testing.
```

Replace String

Function

Replaces a single instant of string or multiple instances of the string.

Syntax

Replace once:

```
char *Replace1(
               char*     srcStr,
               char*     tgtStr,
               char*     newStr
              );
```

Replace all:

```
char *Replace(
              char*      srcStr,
              char*      tgtStr,
              char*      newStr
             );
```

Parameters

srcStr
Source string.

tgtStr
Target string to be replaced.

newStr
New string to be used to replace target string.

Sample

```
char str[] = "This is a string testing";
printf("After Replaced: %s", Replace(str,"string", "replaced"));
```

Output:

```
          After Replaced: This is a replaced testing
```

Reverse Find String

Function

Checks whether a substring is in the data string reversely(from right to left), returns its position if found, otherwise, return 0.

Syntax

```
int Rfind(
          char*        str,
          char*        search
         )
```

Or

```
int Rfind1(
          char*        str,
          char*        search,
          int          start_pos
         )
```

Or

```
int Rfind2(
          char*        str,
          char*        search,
          int          start_pos,
          int          stop_pos
         )
```

Parameters

str
Source data string.

search
Search string.

start_pos
The position to start the search.

stop_pos
The position to stop the search.

Sample

```
char str[80] = "This is data string search testing";
```

```
int pos = Rfind2(str, "search", 39, 15);
```

Return: 21

Right Copy

Function

Copy characters from a specified position until the end of the source string to the destination string.

Syntax

```
char *Rcp (
          char*      dstStr,
          char*      srcStr,
          ushort     from_pos
         );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to be copied from.

Sample

```
char str[] = "This is a string testing";
char dst[20];
printf("Right Copied: %s", Rcp(dst, str, 11));
```

Output:

```
       Right Copied: string testing
```

Right Copy and Left Trim

Function

Copies characters from the specified position until the end of the source string to the destination string where the white-space characters will be trimmed from the left side before being terminated with NULL.

Syntax

```
char *RcpLtrim(
               char*      dstStr,
               char*      srcStr,
               ushort     from_pos
              );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos

The starting position to be copied from.

Sample

```
char str[] = "This is a      string testing";
char dst[20];
printf("Result: %s", RcpLtrim(dst, str, 11));
```

Output:

```
          Result: string testing
```

Right Copy and Pad

Function

Copies the most right characters of a specified length from source string, and pad to the left of the destination string with the pad character if appropriate.

Syntax

```
char *RcpPad(
               char*      dstStr,
               char*      srcStr,
               ushort     length,
               char       pad
              );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

length
The length of the most right characters to be copied from the source string.

pad
The character to be padded to the left of the destination string if the length of the source string is less than the specified length.

Sample

```
char str[] = "The string testing";
char dst[25];
printf("Result: %s", RcpPad(dst, str, 25, '.'));
```

Output:

```
     Result:.......The string testing
```

Right Copy and Right Trim

Function

Copies characters from the specified position until the end of the source string to destination string where the white-space, carriage return, new line control codes would be trimmed from the right before being terminated with NULL.

Syntax

```
char *RcpRtrim(
               char*      dstStr,
               char*      srcStr,
               ushort     from_pos
              );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to be copied from.

Sample

```
char str[] = "This is a string testing       ";
char dst[20];

printf("Result: %s", RcpRtrim(dst, str, 11));
```

Output:

```
    Result: string testing
```

Right Trim

Function

Trims white-space, carriage return, new line control codes, from the right side of the source string.

Syntax

```
char *Rtrim(
            char*  string
           );
```

Parameters

string
Data string to be right trimmed.

Sample

```
char str[] = "This is a string testing      ";

printf("Right Trimmed: %s", Rtrim(str));
```

Output:

```
      Right Trimmed: This is a string testing
```

Right Trim and Concatenate Strings

Function

Trims white-space, carriage return, new line control codes from the right sides of multiple data strings before concatenating them.

Syntax

```
char *RtrimCat(
               char*     separator,
               char*     string,...
              );
```

Parameters

separator
characters to be inserted between concatenated strings.

string,...
variable-argument lists of multiple stringss.

Sample

```
char str1[] = "This is a string testing.      ";

char str2[] = "The second string.   ";

printf("After Concatenation: %s", RtrimCat(" ", str1, str2));
```

Output:

```
      After Concatenation: This is a string testing. The second string.
```

Right Trim for EBCDIC

Function

Trims EBCDIC white-space, carriage return, new line control codes, from the right side of the source EBCDIC string.

Syntax

```
char *E_Rtrim(
              char*      string
             );
```

Parameters

string
Data string to be right trimmed.

Sample

Refer to the sample for the Rtrim function.

Strings Concatenate

Function

Concatenates multiple strings into a string.

Syntax

```
char *Strcat(
            char*       string,...
            );
```

Parameters

string,...
Variable-argument lists of multiple strings.

Sample

```
char str1[] = "This is a string testing. ";

char str2[] = "The second string.";

printf("After Concatenation: %s", Strcat(str1, str2));
```

Output:

```
After Concatenation: This is a string testing. The second string.
```

String Pad

Function

Pads a character to the right side of the string.

Syntax

```
char *StrPad(
              char     *srcStr,
              ushort   length,
              char     pad_char
             );
```

Parameters

srcStr
Source string.

length
Length of the new destination string to be returned.

pad_char

The character to be padded to the right of the destination string if the length of the source string is less than the specified length.

Sample

```
char str[] = "The string testing";
printf("Result: %s", StrPad(str, 25, '.'));
```

Output:

```
Result: The string testing.......
```

Substitute String

Function

Returns a string with a substitution found in the substitute table you defined, otherwise returns NULL.

Syntax

```
char *Subst(
             char*     subst_tbl[][2],
             char*     srtcStr
            );
```

Parameters

Subst_tbl
Substitution table.
srcStr
Source string.

Sample

```
char *tbl [][2] = { {"Jan", "January"},
                    {"Feb", "February"},
                    {"Mar", "March"},
                    {"Apr", "April"},
                    {"Jun", "June"},
                    {"Jul", "July"},
                    {"\0", "\0"}};        // end of initialization
printf("This Month is: %s", Subst(tbl,"Jun"));
```

Output:

```
       This Month is: June
```

Substitute Change

Function

Returns a string with all the substitutions found in the substitute table you defined.

Syntax

```
char *SubstChg(
                char*      subst_tbl[][2],
                char*      srcStr
               );
```

Parameters

Subst_tbl
Substitution table.

srcStr
Source string.

Sample

```
char *tbl [][2] = { {"001", "string 1"},
                    {"002", "string 2"},
                    {"003", "string 3"},
                    {"\0", "\0"}};        // end of initialization
char src[] = "  This is 001; This is 002, This is 003.";
printf("Result: %s", SubstChg(tbl,src));
```

Output:

```
       Result:   This is string 1; This is string 2, This is string 3.
```

Substring

Function

Gets the specified length of the substring from the specified position of the source string.

Syntax

```
char *SubStr(
             char*        dstStr,
             char*        srcStr,
             ushort       from_pos,
             ushort       length
            );
char *SubStr2(
              char*       srcStr,
              ushort      from_pos,
              ushort      length
             );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of substring.

Sample

```
char str[] = "This is a substring testing";
char dst[20];
printf("Result: %s", SubStr(dst,str,11,9));
```

Output:

```
Result: substring
```

Substring Left Trim

Function

Gets the specified length of the substring from the specified position of the source string, and trims white-space characters from the left of the destination string.

Syntax

```
char *SubStrLtrim(
                  char*   dstStr,
                  char*   srcStr,
                  ushort  from_pos,
                  ushort  length
                 );
```

```
char *SubStrLtrim2(
                  char*   srcStr,
                  ushort  from_pos,
                  ushort  length
                  );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of the substring.

Sample

```
char str[] = "This is a      substring testing";
char dst[20];
printf("Result: %s", SubStrLtrim(dst,str,11,14));
```

Output:

```
        Result: substring
```

Substring Pad

Function

Gets the specified length of the substring from the specified position of the source string, and pads with pad character to the right of the destination string if appropriate.

Syntax

```
char *SubStrPad(
                char*     dstStr,
                char*     srcStr,
                ushort    from_pos,
                ushort    length,
                char      pad_char
               );

char *SubStrPad2(
                char*     srcStr,
                ushort    from_pos,
                ushort    length,
                char      pad_char
               );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of the substring.

Pad_char
The character to use to pad destination string.

Sample

```
char str[] = "This is testing";
char dst[20];
SubStrPad(dst,str,9,12,'*');
printf("Result: %s", dst);
```

Output:

```
        Result: testing*****
```

Substring Right Trim

Function

Gets the specified length of the substring from the specified position of the source string, and trims white-space characters from the right of the destination string.

Syntax

```
char *SubStrRtrim(
                  char*   dstStr,
                  char*   srcStr,
                  ushort  from_pos,
                  ushort  length
                 );

char *SubStrRtrim2(
                  char*   srcStr,
                  ushort  from_pos,
                  ushort  length
                  );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of the substring.

Sample

```
char str1[] = "This is a substring         testing";
char dst1[20];
SubStrRtrim(dst1,str1, 11,16);
```

```
printf("Result: %s", dst1);
```

Output:

```
              Result: substring
```

Substring Trim Both Sides

Function

Gets the specified length of the substring from the specified position of the source string, and trims white-space, carriage return, new line control codes from both sides of the destination string.

Syntax

```
char *SubStrTrim(
                 char*    dstStr,
                 char*    srcStr,
                 ushort   from_pos,
                 ushort   length
                );

char *SubStrTrim2(
                 char*    srcStr,
                 ushort   from_pos,
                 ushort   length
                );
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of the substring.

Sample

```
char str1[] = "This is a      substring         testing";
char dst1[20];
printf("Result: %s", SubStrTrim(dst1,str1, 11,20));
```

Output:

```
        Result: substring
```

System Date and Time

Function

Returns a formatted time and date string.

Syntax

```
char *SysTime(
              char*   format
              );
```

Parameters

format
The format argument consists of one or more codes. The formatting codes are preceded by a percent sign (**%**). Characters that do not begin with **%** are copied unchanged.

The formatting codes for strftime are listed below:

%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for the locale
%d	Day of the month as decimal number (01 - 31)
%H	Hour in 24-hour format (00 - 23)
%I	Hour in 12-hour format (01 - 12)
%j	Day of year as a decimal number (001 - 366)
%m	Month as a decimal number (01 - 12)
%M	Minute as a decimal number (00 - 59)
%p	Current locale's A.M./P.M. indicator for 12-hour clock
%S	Second as a decimal number (00 - 59)
%U	Week of the year as a decimal number, with Sunday as the first day of the week (00 - 53)
%w	Weekday as a decimal number (0 - 6; Sunday is 0)
%W	Week of the year as a decimal number, with Monday as the first day of the week (00 - 53)
%x	Date representation for the current locale
%X	Time representation for the current locale
%y	Year without century, as a decimal number (00 - 99)
%Y	Year with century, as a decimal number
%z, %Z	Time-zone name or abbreviation; no characters if the time zone is unknown

Sample

```
printf("Current date is: ", SysTime("%Y-%m-%d"));
```

Output:

```
Current date is: 2008-10-22
```

Title String

Function

Returns a string with the first character of each word in uppercase.

Syntax

```
char *Title(
            char*   string
           );
```

Parameters

string
Data string to be processed.

Sample

```
char str[] = "This is a string testing";
printf("Title Text: %s", Title(str));
```

Output:

```
      Title Text: This Is A String Testing
```

Thai Compose

Function

Returns a composed Thai ASCII string, you need MakeAFP Thai AFP Font package to print Thai characters in Thai glyph standard layout.

Syntax

```
char *ThaiCompose(
                 char*   thai_string
                );
```

Parameters

Thai_string
The Thai ASCII data string to be used for composition.

Sample

No sample was provided.

Translate Digits to Simplified Chinese Figures

Function

Translate ASCII digits to Simplified Chinese figures in GB18030 encoding.

Syntax

```
char *DigitGBK(
              char*     dst,
              char*     src
             );
```

Parameters

dst
Destination of Simfilied Chinese figures string encoding in GD18030.

src
Source of digits string encoded in ASCII.

Sample

```
printf("Chinese figures: %s", DigitGBK("123.45"));
```

```
Output:

          Chinese figures: 壹佰 貳拾 叁元 肆角 伍分
```

Translate Digits to Traditional Chinese Figures

Function

Translate ASCII digits to Traditional Chinese figures in BIG5 encoding.

Syntax

```
char *DigitBIG5(
                char*     dst,
                char*     src
               );
```

Parameters

dst
Destination of Traditional Chinese figures string encoding in BIG5.

src
Source of digits string encoded in ASCII.

Sample

```
printf("Chinese figures: %s", DigitBIG5("123.45"));

Output:

          Chinese figures: 壹佰 貳拾 叁元 肆角 伍分
```

Trim Both Sides

Function

Trims white-space characters from both sides of the source string, as well as carriage return and new line control codes from the right side of the source string.

Syntax

```
char *Trim(
           char*    string
          );
```

Parameters

string
Data string to be trimmed.

Sample

```
char str[] = "      This is a string testing      ";
printf("Both Sides Trimmed: %s", Trim(str));
```

Output:

```
        Both Sides Trimmed: This is a string testing
```

Trim Both Sides for EBCDIC

Function

Trims EBCDIC white-space characters from both sides of the source string, as well as carriage return and new line control codes from the right side of the source EBCDIC string.

Syntax

```
char *E_Trim(
              char*     string
             );
```

Parameters

string
EBCDIC data string to be trimmed.

Sample

Refer to the sample for the Trim function.

Trim and Concatenate Strings

Function

Trims white-space characters from both sides and carriage-return and line-feed control codes

from the right side of the source strings before concatenating them into the destination string.

Syntax

```
char *TrimCat(
               char     *separators,
               char     *srcStr,...
              );
```

Parameters

separators
Characters to be inserted between concatenated strings.

srcStr,...
Variable-argument lists of multiple strings.

Sample

```
char str1[] = "      This is a string testing.    ";
char str2[] = "   This is 2nd string.    ";
```

```
printf("Result: %s", TrimCat(" ***** ", str1, str2));
```

Output:

```
  Result: This is a string testing. ***** This is 2nd string.
```

Chapter 5. Conversion Functions

When developing applications around legacy and Unicode characters, it is required to convert between legacy ASCII/DBCS-PC and EBCDIC/DBCS-HOST, between Unicode and legacy text data, or between Unicode encodings.

Codepage/Charset TO UTF-16 Conversion

Function

Converts from codepage/charset stream to Unicode UTF-16, and returns the length of the UTF-16 output.

Syntax

```
int32_t  ChartoU16(
                  UChar        *target,
                  int32_t      *targetCapacity,
                  char         *source,
                  int32_t      sourceLen = -1,
                  char         fromCode = NULL
                 );
```

Parameters

target
Point to the targeted UTF-16 output buffer.

targetCapacity
The maximum size of the targeted UTF-16 buffer.

source
Pointer to the input source buffer, in bytes.

sourceLen
Length of the input source, or default -1 for NULL-terminated input.

fromCode
The name of the source encoding. Default is NULL, uses the encoding name pre-defined and loaded by the “DefaultCode” function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

Codepage/Charset TO UTF-8 Conversion

Function

Converts from codepage/charset stream to Unicode UTF-8 and returns the length of the UTF-8 output.

Syntax

```
int32_t  ChartoU8(
                 UChar8         *target,
                 int32_t        *targetCapacity,
                 char           *source,
                 int32_t        sourceLen = -1,
                 char           fromCode = NULL
               );
```

Parameters

target
Point to the targeted UTF-8 output buffer.

targetCapacity
Maximum size the targeted UTF-8 buffer.

source
Pointer to the input source buffer, in bytes.

sourceLen
Length of the input source, or default -1 for NULL-terminated input.

fromCode
The name of the source encoding. Default is NULL, uses the encoding name pre-defined and loaded by the “DefaultCode” function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

Default Encoding Names

Function

Defines the current default input data encoding names.

Make sure you have defined a correct encoding name before calling data encoding conversion functions and paragraph functions.

Syntax

```
void     DefaultCode(
                    char        *codename = "windows-1252"
                  );
```

Parameters

codeName
The name of the default encoding, default is “windows-1252”. Refer to MakeAFP document *Encoding Names for* more details about the available names.

Universal Conversion

Function

Converts from one external charset to another, like conversion between legacy ASCII/DBCS-PC and EBCDIC/DBCS-HOST, between Unicode and legacy text data or between Unicode encodings. External string used as source or target for the conversion is always treated as a byte stream. It returns the length of the complete target output.

Syntax

```
int32_t  Convert(
                char            *tocode,
                char            *fromcode,
                char            *target,
                int32_t         targetCapacity,
                char            *source,
                int32_t         sourceLen = -1
               );
```

Parameters

toCode
The name of the destination encoding. Refer to MakeAFP document *Encoding Names for* more details about the available names.

fromCode
The name of the source encoding. Refer to MakeAFP document *Encoding Names* for more details about the available names.

target
Point to the target output buffer.

targetCapacity
The maximum size of the target buffer, in bytes.

source
Pointer to the input source buffer.

sourceLen
Length of the input source, in bytes, or default -1 for NULL-terminated input.

UTF-16 to Codepage/Charset Conversion

Function

Converts from Unicode UTF-16 to a codepage/charset stream and returns the length of the complete target output.

Syntax

```
int32_t  U16toChar(
                  char          *target,
                  int32_t       *targetCapacity,
                  UChar         *source,
                  int32_t       sourceLen = -1,
                  char          toCode = NULL
                 );
```

Parameters

target
Point to the target output buffer.

targetCapacity
The maximum size of the target buffer, in bytes.

source
Pointer to the UTF-16 input source buffer.

sourceLen
Length of the UTF-16 input source, or default -1 for NULL-terminated input.

toCode
The name of the target encoding. Default is NULL, uses the encoding names pre-defined and loaded by DefaultCode() function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

UTF-16 to UTF-8 Conversion

Function

Converts from Unicode UTF-16 to UTF-8 and returns the length of the complete UTF-8 target output.

Syntax

```
int32_t  U16toU8(
                UChar8         *target,
                int32_t        *targetCapacity,
                UChar          *source,
                int32_t        sourceLen = -1
               );
```

Parameters

target
Point to the target UTF-8 output buffer.

targetCapacity
The maximum size of the UTF-8 target buffer.

source
Pointer to the UTF-16 input source buffer.

sourceLen
Length of the UTF-16 input source, or default -1 for NULL-terminated input.

UTF-32 to UTF-16 Conversion

Function

Converts from Unicode UTF-32 to UTF-16 and returns the length of the complete UTF-16 target output.

Syntax

```
int32_t  U32toU16(
                 UChar         *target,
                 int32_t       *targetCapacity,
                 UChar32       *source,
                 int32_t       sourceLen = -1
                );
```

Parameters

target
Point to the target UTF-16 output buffer.

targetCapacity
The maximum size of the UTF-16 target buffer.

source
Pointer to the UTF-32 input source buffer.

sourceLen
Length of the UTF-32 input source, or default -1 for UTF-32 NULL-terminated input.

UTF-32 to UTF-8 Conversion

Function

Converts from Unicode UTF-32 to UTF-8 and returns the length of the complete UTF-8 target output.

Syntax

```
int32_t  U32toU8(
                UChar8         *target,
                int32_t        *targetCapacity,
                UChar32        *source,
                int32_t        sourceLen = -1
               );
```

Parameters

target
Point to the target UTF-8 output buffer.

targetCapacity
The maximum size of the UTF-8 target buffer.

source
Pointer to the UTF-32 input source buffer.

sourceLen
Length of the UTF-32 input source, or default -1 for UTF-32 NULL-terminated input.

UTF-8 to UTF-16 Conversion

Function

Converts from Unicode UTF-8 to UTF-16 and returns the length of the complete UTF-16 target output.

Syntax

```
int32_t  U8toU16(
                UChar          *target,
                int32_t        *targetCapacity,
                UChar8         *source,
                int32_t        sourceLen = -1
               );
```

Parameters

target
Point to the target UTF-16 output buffer.

targetCapacity
The maximum size of the UTF-16 target buffer.

source
Pointer to the UTF-8 input source buffer.

sourceLen
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

UTF-8 to Codepage/Charset Conversion

Function

Converts from Unicode UTF-8 to a codepage/charset stream returns the length of the complete target output.

Syntax

```
int32_t  U8toChar(
                 char           *target,
                 int32_t        *targetCapacity,
                 UCha8          *source,
                 int32_t        sourceLen = -1,
                 char           toCode = NULL
                );
```

Parameters

target
Point to the target output buffer.

targetCapacity
The maximum size of the target buffer, in bytes.

source
Pointer to the UTF-8 input source buffer.

sourceLen
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

toCode
The name of the target encoding. Default is NULL, uses the encoding names pre-defined and loaded by the “DefaultCode” function. Refer to MakeAFP document *Encoding Names for* more details about the available names.

Syntax

```
int32_t  U8toChar(
                 UChar          *target,
                 int32_t        *targetCapacity,
                 UChar8         *source,
                 int32_t        sourceLen = -1
                );
```

Parameters

target
Point to the target UTF-16 output buffer.

targetCapacity
The maximum size of the UTF-16 target buffer.

source
Pointer to the UTF-8 input source buffer.

sourceLen
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

Vietnamese Codepage/Charset Codepage/Charset Conversion

Function

Converts Vietnamese from one external charset to another, like conversion between legacy PC formats, between Unicode and legacy text data or between Unicode encodings. External string used as source or target for the conversion is always treated as a byte stream. It returns the length of the complete target output.

Syntax

```
int   VietConv(
               char       *toCode,
               char       *fromCode,
               char       *target,
               int        targetCapacity,
               char       *source,
               int        sourceLen = -1
              );
```

Parameters

toCode
The name of the destination encoding, allowed values are BKHCM1, BKHCM2, ISC, NCR-DEC, NCR-HEX, TCVN3, UNI-COMP, UNICODE, UTF-8, UTF8, UVIQR, VIETWARE-F, VIETWARE-X, VIQR, VISCII,
VNI-MAC, VNI-WIN, VPS, CP1258.

fromCode
The name of the source encoding, allowed values are BKHCM1, BKHCM2, ISC, NCR-DEC, NCR-HEX, TCVN3, UNI-COMP, UNICODE, UTF-8, UTF8, UVIQR, VIETWARE-F, VIETWARE-X, VIQR, VISCII,
VNI-MAC, VNI-WIN, VPS, CP1258.

target
Point to the target output buffer.

targetCapacity
The maximum size of the target buffer, in bytes.

source
Pointer to the input source buffer.

sourceLen
Length of the input source, in bytes, or default -1 for NULL-terminated input.

Appendix A. ASCII/EBCDIC AFP Code Pages and CPGID Summary

Name	Description	CPGID	Encoding
T1000038	US-ASCII Character Set	38	EBCDIC
T1000259	Symbols, Set 7	259	EBCDIC
T1000260	Canadian French - 116	260	EBCDIC
T1000276	Canada (French) - 94	276	EBCDIC
T1000286	Austria/Germany F.R., Alt (3270)	286	EBCDIC
T1000287	Denmark/Norway, Alternate (3270)	287	EBCDIC
T1000288	Finland/Sweden, Alternate (3270)	288	EBCDIC
T1000289	Spain, Alternate (3270)	289	EBCDIC
T1000290	Japan (Katakana)	290	EBCDIC
T1000293	APL (USA)	293	EBCDIC
T1000310	Graphic Escape APL/TN	310	EBCDIC
T1000361	International Set 5	361	EBCDIC
T1000363	Symbols, Set 8	363	EBCDIC
T1000367	ASCII	367	ASCII
T1000382	Austria, Germany, Japan	382	EBCDIC
T1000383	Belgium	383	EBCDIC
T1000384	Brazil	384	EBCDIC
T1000385	Canada (French)	385	EBCDIC
T1000386	Denmark/Norway	386	EBCDIC
T1000387	Sweden/Finland	387	EBCDIC
T1000388	France, Japan	388	EBCDIC
T1000389	ITALY, Japan (Italian)	389	EBCDIC
T1000390	Japan (Latin)	390	EBCDIC
T1000391	Portugal	391	EBCDIC
T1000392	Spain/Philippines	392	EBCDIC
T1000393	Latin America (Spanish)	393	EBCDIC
T1000394	U.K., Austral., IRE., H.K., N.Z.	394	EBCDIC
T1000395	United States, Canada (English)	395	EBCDIC
T1000420	Arabic Bilingual	420	EBCDIC
T1000423	Greece - 183	423	EBCDIC
T1000424	Israel (Hebrew)	424	EBCDIC
T1000437	Personal Computer	437	ASCII
T1000803	Hebrew Character Set A	803	EBCDIC
T1000808	PC, Cyrillic, Russian with euro	808	ASCII
T1000813	Greece - ISO/ASCII 8-Bit	813	ASCII
T1000819	Latin1 ISO/ANSI 8-BIT	819	ASCII
T1000829	Math Symbols	829	EBCDIC
T1000836	Peoples Republic of China (PRC)	836	EBCDIC
T1000838	Thai - EBCDIC	838	EBCDIC
T1000848	PC, Cyrillic, Ukraine with Euro	848	ASCII
T1000849	PC, Cyrillic, Belo Russian Euro	849	ASCII
T1000850	PC Multilingual	850	ASCII
T1000851	Greek - Personal Computer	851	ASCII
T1000852	Latin2 Multilingual PC	852	ASCII
T1000853	Latin3 Personal Computer	853	ASCII

T1000855	Cyrillic - Personal Computer	855	ASCII
T1000856	Hebrew - Personal Computer	856	ASCII
T1000857	Latin5 PC	857	ASCII
T1000858	PC - Multilingual with euro	858	ASCII
T1000860	Portugal - Personal Computer	860	ASCII
T1000861	Iceland - Personal Computer	861	ASCII
T1000862	Hebrew - Personal Computer	862	ASCII
T1000863	Canadian French - PC	863	ASCII
T1000864	Arabic - Personal Computer	864	ASCII
T1000865	Nordic - Personal Computer	865	ASCII
T1000866	Cyrillic #2 - Personal Computer	866	ASCII
T1000867	Israel - Personal Computer	867	ASCII
T1000869	Greece - Personal Computer	869	ASCII
T1000870	Latin2 Multilingual	870	EBCDIC
T1000872	Cyrillic PC with Euro	872	ASCII
T1000874	Thai - Personal Computer	874	ASCII
T1000875	Greece	875	EBCDIC
T1000876	OCR-AN ASCII	876	ASCII
T1000877	OCR-B ASCII	877	ASCII
T1000880	Cyrillic Multilingual	880	EBCDIC
T1000889	Thailand	889	EBCDIC
T1000892	OCR - A	892	EBCDIC
T1000893	OCR - B	893	EBCDIC
T1000897	Japan PC #1	897	ASCII
T1000899	Symbols, Set 7 ASCII	899	ASCII
T1000901	PC, Baltic - Multilingual w Euro	901	ASCII
T1000902	8-bit Estonia with euro	902	ASCII
T1000903	Peoples Republic of China - PC	903	ASCII
T1000904	Republic of China (ROC) - PC	904	ASCII
T1000905	Latin3 Multilingual	905	EBCDIC
T1000910	APL ASCII	910	ASCII
T1000912	Latin2 ISO/ANSI 8-BIT	912	ASCII
T1000913	Latin 3, ISO/ASCII	913	ASCII
T1000914	Latin4 ISO/ANSI 8-BIT	914	ASCII
T1000915	Cyrillic ISO/ASCII 8-Bit	915	ASCII
T1000916	Hebrew ISO/ASCII 8-Bit	916	ASCII
T1000920	Latin5 ISO/ANSI 8-BIT	920	ASCII
T1000921	PC, Baltic - Multilingual	921	ASCII
T1000922	Estonia PC	922	ASCII
T1000923	Latin 9	923	ASCII
T1000924	Latin 9 EBCDIC	924	EBCDIC
T1001002	DCF REL 2 Compatibility	1002	EBCDIC
T1001003	U.S. Text Subset	1003	EBCDIC
T1001004	IBM PC Desktop Publishing	1004	ASCII
T1001008	Arabic ISO/ASCII 8-Bit	1008	ASCII
T1001025	Cyrillic Multilingual	1025	EBCDIC
T1001026	Latin5	1026	EBCDIC
T1001027	Japanese (Latin) Extended	1027	EBCDIC
T1001028	Hebrew Publishing	1028	EBCDIC
T1001029	Arabic Extended ISO/ACSII 8-Bit	1029	ASCII
T1001032	MICR, E13-B Combined	1032	EBCDIC
T1001033	MICR, CMC-7 Combined	1033	EBCDIC
T1001038	Symbols, Adobe ASCII	1038	ASCII
T1001039	GML List Symbols	1039	EBCDIC
T1001041	Japanese Extended - PC	1041	ASCII
T1001042	Simplified Chinese Extended - PC	1042	ASCII
T1001043	Traditional Chinese Extended PC	1043	ASCII
T1001046	Arabic Extended ISO/ASCII 8-Bit	1046	ASCII

T1001068	Text With Numeric Spacing	1068	EBCDIC
T1001069	Latin4 EBCDIC	1069	EBCDIC
T1001087	Symbols, Adobe	1087	EBCDIC
T1001091	Symbol Set 7, Modified	1091	EBCDIC
T1001092	Symbol Set 7, Modified - PC	1092	ASCII
T1001093	IBM LOGO	1093	EBCDIC
T1001110	Latin2 Multilingual	1110	EBCDIC
T1001111	Latin2 ISO/ANSI 8-BIT	1111	ASCII
T1001112	Baltic - Multilingual, EBCDIC	1112	EBCDIC
T1001122	Estonia, EBCDIC	1122	EBCDIC
T1001123	Cyrillic, Ukraine EBCDIC	1123	EBCDIC
T1001124	Cyrillic, Ukraine ISO-8	1124	ASCII
T1001125	PC, Cyrillic Ukrainian	1125	ASCII
T1001129	Vietnamese ISO-8	1129	ASCII
T1001130	Vietnamese EBCDIC	1130	EBCDIC
T1001131	PC, Cyrillic, Belo Russian	1131	ASCII
T1001132	Lao EBCDIC	1132	EBCDIC
T1001133	Lao ISO-8	1133	ASCII
T1001139	Japan Alphanumeric Katakana	1139	ASCII
T1001140	USA, Canada ECECP	1140	EBCDIC
T1001141	Austria, Germany ECECP	1141	EBCDIC
T1001142	Denmark, Norway ECECP	1142	EBCDIC
T1001143	Finland, Sweden ECECP	1143	EBCDIC
T1001144	Italy ECECP	1144	EBCDIC
T1001145	Spain, Latin America ECECP	1145	EBCDIC
T1001146	UK ECECP	1146	EBCDIC
T1001147	France ECECP	1147	EBCDIC
T1001148	International ECECP	1148	EBCDIC
T1001149	Iceland ECECP	1149	EBCDIC
T1001153	Latin2 Multilingual with Euro	1153	EBCDIC
T1001254	Windows Turkish	1254	ASCII
T1001257	Windows Baltic Rim	1257	ASCII
T1001258	Windows Vietnamese	1258	ASCII
T1001275	Apple Latin 1	1275	ASCII
T1001276	Adobe PS Standard	1276	ASCII
T1001277	Adobe PS ISO Latin 1	1277	ASCII
T1001280	Apple Greece	1280	ASCII
T1001281	Apple Turkey	1281	ASCII
T1001282	Apple Central Europe	1282	ASCII
T1001283	Apple Cyrillic	1283	ASCII
T1001300	GENERIC BAR CODE/OCR-B	1300	EBCDIC
T1005346	Latin 2 – Windows	1250	ASCII
T1005347	Cyrillic – Windows	1251	ASCII
T1005348	Latin 1 – Windows	1252	ASCII
T1005349	Greece – Windows	1253	ASCII
T1005350	Turkey – Windows	1254	ASCII
T1005351	Israel – Windows	1255	ASCII
T1005352	Arabic – Windows	1256	ASCII
T1005353	Latin 4 – Windows	1257	ASCII
T1005354	Vietnamese – Windows	1258	ASCII
T1V10037	USA/Canada - CECP	37	EBCDIC
T1V10273	Germany F.R./Austria- CECP	273	EBCDIC
T1V10274	Belgium - CECP	274	EBCDIC
T1V10275	Brazil - CECP	275	EBCDIC
T1V10277	Denmark/Norway - CECP	277	EBCDIC
T1V10278	Finlandd/Sweden- CECP	278	EBCDIC
T1V10280	ITALY- CECP	280	EBCDIC

T1V10281	Japan (Latin) - CECP	281	EBCDIC
T1V10282	Portugal - CECP	282	EBCDIC
T1V10284	Spain/Latin America - CECP	284	EBCDIC
T1V10285	UNITED KINGDOM - CECP	285	EBCDIC
T1V10290	Japan (Katakana)	290	EBCDIC
T1V10297	France - CECP	297	EBCDIC
T1V10500	International #5	500	EBCDIC
T1V10871	Iceland - CECP	871	EBCDIC

Appendix B. SBCS/DBCS AFP Code Pages and CPGID Summary

Name	Description	CPGID	Encoding
T1H00037	Traditional Chinese EBCDIC	037	EBCDIC
T1H00290	Japanese Katakana Extended	290	EBCDIC
T1H00833	Korean EBCDIC	833	EBCDIC
T1H00836	Simplified Chinese EBCDIC	836	EBCDIC
T1H01002	Japanese DCF Rel 2 Compatibility	1002	EBCDIC
T1H01027	Japanese Latin Extended	1027	EBCDIC
T1H01030	Japanese Katakana Extended with Box Characters	1030	EBCDIC
T1H01031	Japanese Latin Extended with Box Characters	1031	EBCDIC
T1H01041	Japanese PC Extended	1041	ASCII
T1H01043	Traditional Chinese PC	1043	ASCII
T1H01114	Traditional Chinese PC BIG5 with Euro	1114	ASCII
T1H01115	Simplified Chinese PC (GB)	1115	ASCII
T1H01126	Korean PC	1126	ASCII
T1H01150	Korean EBCDIC with Box Characters	1150	EBCDIC
T1H01151	Simplified Chinese EBCDIC with Box Characters	1151	EBCDIC
T1H01152	Traditional Chinese EBCDIC with Box Characters	1152	EBCDIC
T1H01159	Traditional Chinese EBCDIC with Euro	1159	EBCDIC
T1H01252	Simplified Chinese PC (GB18030)	1252	ASCII
T1HK0037	Japanese English	037	EBCDIC
T1HK0290	Japanese Katakana	290	EBCDIC
T10300, T1I300, T1J300, T1K300	Japanese DBCS-HOST	300	DBCS-HOST
T10834	Korean DBCS-HOST (Small Set)	834	DBCS-HOST
T10835	Traditional Chinese DBCS-HOST	835	DBCS-HOST
T10837	Simplified Chinese DBCS-HOST (GB2312)	837	DBCS-HOST
T10941	Japanese SJIS-PC	941	DBCS-PC
T10947	Traditional Chinese BIG5-PC	947	DBCS-PC
T10951	Korean KSC-PC (Small Set)	951	DBCS-PC
T11362	Korean KSC-PC (Big Set)	1362	DBCS-PC
T11374	Traditional Chinese HKSCS-PC	1374	DBCS-PC
T11376	Traditional Chinese HKSCS-HOST	1376	DBCS-HOST
T11380	Simplified Chinese GB2312-PC (Small Set)	1380	DBCS-PC
T11385	Simplified Chinese GBK-PC (Big Set)	1385	DBCS-PC
T1K834	Korean DBCS-HOST (Big Set)	837	DBCS-HOST
T1K837	Simplified Chinese DBCS-HOST (GB18030)	837	DBCS-HOST

Appendix C. Combined SBCS/DBCS CPGID Summary

CPGID	Description	Encoding
937	Combination of Traditional Chinese SBCS-HOST and DBCS-HOST	EBCDIC/DBCS-HOST (CP37/CP835)
939	Combination of Japanese SBCS-HOST and DBCS-HOST	EBCDIC/DBCS-HOST (CP1027/CP300)
943	Combination of Japanese ASCII and SJIS-PC for open systems	ASCII/SJIS-PC (CP897/CP941)
950	Combination of Traditional Chinese ASCII and BIG5-PC for open systems	ASCII/BIG5-PC (CP1114/CP947)
1363	Combination of Korean ASCII and KSC-PC for open systems	ASCII/KSC-PC (CP1126/CP1362)
1364	Combination of Korean SBCS-HOST and DBCS-HOST	EBCDIC/DBCS-HOST (CP833/CP834)
1375	Combination of Traditional Chinese ASCII and HKSCS-PC for open systems	ASCII/HKSCS-PC (CP1114/CP1374)
1377	Combination of Traditional Chinese SBCS-HOST and DBCS-HOST for HKSCS	ASCII/GBK-PC (CP37/CP1376)
1386	Combination of Simplified Chinese ASCII and GBK-PC for open systems	ASCII/GBK-PC (CP1114/CP1385)
1388	Combination of Simplified Chinese SBCS-HOST and DBCS-HOST for GBK	EBCDIC/DBCS-HOST (CP836/CP837)
1392	Combination of Simplified Chinese ASCII and GB18030-PC for open systems	ASCII/GB18030
13767	Combination of Simplified Chinese SBCS-HOST and DBCS-HOST for GB18030	EBCDIC/DBCS-HOST (CP836/CP837)

Appendix D. Encoding Names and Alias

Encoding Name and Alias	Description	CPGID
BOCU-1, csBOCU-1, ibm-1214, ibm-1215	Binary Ordered Compression for Unicode, it combines the wide applicability of UTF-8 with the compactness of Standard Compression Scheme for Unicode	
CESU-8, ibm-9400	CESU-8 is a Compatibility Encoding Scheme for UTF-16 (CESU) that serializes a Unicode code point as a sequence of one, two, three or six bytes	
ebcdic-xml-us	XML in EBCDIC-US	
HZ, HZ-GB-2312	Simplified Chinese, International and national Standard	
ibm-37, IBM037, ibm-037, ebcdic-cp-us, ebcdic-cp-ca, ebcdic-cp-wt, ebcdic-cp-nl, csIBM037, cp037, 037, cpibm37, cp37, T1V10037	USA/Canada – CECP, EBCDIC	037
Ibm-259, IBM-Symbols, csIBMSymbols	Symbols, Set 7, EBCDIC	259
ibm-273, IBM273, CP273, csIBM273, ebcdic-de, cpibm273, 273, T1V10273	Germany F.R./Austria- CECP, EBCDIC	273
ibm-277, IBM277, cp277, EBCDIC-CP-DK, EBCDIC-CP-NO, csIBM277, ebcdic-dk, cpibm277, 277, T1V10277	Denmark/Norway - CECP, EBCDIC	277
ibm-278, IBM278, cp278, ebcdic-cp-fi, ebcdic-cp-se, csIBM278, ebcdic-sv, cpibm278, 278, T1V10278	Finland/Sweden- CECP, EBCDIC	278
ibm-280, IBM280, CP280, ebcdic-cp-it, csIBM280, cpibm280, 280, T1V10280	ITALY- CECP, EBCDIC	280
ibm-284, IBM284, CP284, ebcdic-cp-es, csIBM284, cpibm284, 284, T1V10284	Spain/Latin America - CECP, EBCDIC	284
ibm-285, IBM285, CP285, ebcdic-cp-gb, csIBM285, ebcdic-gb, cpibm285, 285, T1V10285	United Kingdom - CECP, EBCDIC	285
Ibm-286, EBCDIC-AT-DE-A, csEBCDICATDEA, T1000286	Austria/Germany F.R., Alt (3270), EBCDIC	286
ibm-290, IBM290, cp290, EBCDIC-JP-kana, csIBM290, T1V10290	Japan (Katakana), EBCDIC	290
Ibm-293, T1000293	APL EBCDIC	293
ibm-297, IBM297, cp297, ebcdic-cp-fr, csIBM297, cpibm297, 297,	France – CECP, EBCDIC	297

T1V10297		
Ibm-367, US-ASCII, ASCII, ANSI_X3.4-1968, ANSI_X3.4-1986, ISO_646.irv:1991, iso_646.irv:1983, ISO646-US, us, csASCII, iso-ir-6, cp367, ascii7, windows-20127, ibm367, T1000367	ASCII	367
ibm-420, IBM420, cp420, ebcdic-cp-ar1, csIBM420, 420, T1000420	Arabic Bilingual, EBCDIC	420
ibm-424, IBM424, cp424, ebcdic-cp-he, csIBM424, 424, T1000424	Israel (Hebrew), EBCDIC	424
ibm-437, IBM437, cp437, 437, csPC8CodePage437, windows-437, T1000437	US Personal Computer, ASCII	437
ibm-500, IBM500, CP500, ebcdic-cp-be, csIBM500, ebcdic-cp-ch, cpibm500, 500, T1V10500	International #5, EBCDIC	500
ibm-720, windows-720, DOS-720	PC Arabic, ASCII	720
Ibm-737, IBM737, cp737, windows-737, 737	PC Greek, ASCII	737
Ibm-775, IBM775, cp775, csPC775Baltic	PC Baltic, ASCII	775
ibm-803, cp803, T1000803	Hebrew Character Set A, EBCDIC	803
Ibm-808, T1000808	Cyrillic, Russian with euro, ASCII	808
ibm-813, iso-8859-7, greek, greek8, ELOT_928, ECMA-118, csISOLatinGreek, iso-ir-126, ISO_8859-7:1987, 8859_7, cp813, 813, T1000813	Greece - ISO/ASCII 8-Bit, ASCII	813
ibm-819, IBM819, cp819, latin1, 8859_1, csISOLatin1, iso-ir-100, ISO_8859, ISO_8859-1:1987, l1, 819, T1000819	Latin-1, West European, ASCII	819
ibm-838, IBM-Thai, csIBMThai, cp838, 838, ibm-9030, ibm838, T1000838	Thai EBCDIC	838
Ibm-848, T1000848	Cyrillic, Ukraine with Euro, ASCII	848
Ibm-849, T1000849	Cyrillic, Belarus Russian Euro, ASCII	849
ibm-850, IBM850, cp850, 850, csPC850Multilingual, windows-850, T1000850	PC Multilingual, ASCII	850
ibm-851, IBM851, cp851, 851, csPC851, T1000851	Greek - Personal Computer, ASCII	851
ibm-852, IBM852, cp852, 852, csPCp852, windows-852, T1000852	Latin2 Multilingual PC, ASCII	852
ibm-855, IBM855, cp855, 855, csIBM855, csPCp855, windows-855, T1000855	Cyrillic - Personal Computer, ASCII	855
ibm-856, cp856, 856, T1000856	Hebrew (old) - Personal Computer, ASCII	856
ibm-857, IBM857, cp857, 857, csIBM857, windows-857, T1000857	Latin5 PC, ASCII	857
ibm-858, IBM00858, CCSID00858, CP00858, PC-Multilingual-850+euro, cp858, windows-858, T1000858	PC - Multilingual with euro, ASCII	858
ibm-860, IBM860, cp860, 860, csIBM860, T1000860	Portugal - Personal Computer, ASCII	860

ibm-861, IBM861, cp861, 861, cp-is, csIBM861, windows-861, T1000861	Iceland - Personal Computer, ASCII	861
ibm-862, IBM862, cp862, 862, DOS-862, csPC862LatinHebrew, windows-862, T1000862	Hebrew - Personal Computer, ASCII	862
ibm-863, IBM863, cp863, 863, csIBM863, T1000863	Canadian French - PC, ASCII	863
ibm-864, IBM864, cp864, csIBM864, T1000864	Arabic - Personal Computer, ASCII	864
ibm-865, IBM865, cp865, 865, csIBM865, T1000865	Nordic - Personal Computer, ASCII	865
ibm-866, IBM866, cp866, 866, csIBM866, windows-866, T1000866	Cyrillic #2 - Personal Computer, ASCII	866
ibm-867, cp867, T1000867	Israel - Personal Computer, ASCII with Euro sign	867
ibm-868, IBM868, CP868, 868, csIBM868, cp-ar, T1000868	Urdu, ASCII	868
ibm-869, IBM869, cp869, 869, cp-gr, csIBM869, windows-869, T1000869	Greece - Personal Computer, ASCII	869
ibm-870, IBM870, CP870, ebcdic-cp-roece, ebcdic-cp-yu, csIBM870, T1000870	Latin2 Multilingual, EBCDIC	870
ibm-871, IBM871, ebcdic-cp-is, csIBM871, CP871, ebcdic-is, cpibm871, 871, T1V10871	Iceland – CECP, EBCDIC	871
Ibm-872, T1000872	Cyrillic PC with Euro, ASCII	872
ibm-874, ibm-9066, cp874, TIS-620, tis620.2533, eucTH, cp9066, ibm874, MS874, windows-874, T1000874	Thai - Personal Computer, ASCII	874
ibm-875, IBM875, cp875, 875, T1000875	Greece, EBCDIC	875
ibm-878, KOI8-R, koi8, csKOI8R, cp878, windows-20866, T1000878	Russian internet, ASCII	878
Ibm-880, IBM880, cp880,EBCDIC-Cyrillic,csIBM880, windows-20880, T1000880	Cyrillic Multilingual, ASCII	880
Ibm-896, T1000896	Japanese Katakana, ASCII	896
ibm-897, JIS_X0201, X0201, csHalfWidthKatakana, T1000897	Japanese, Half Width Katakana, ASCII	897
ibm-901, T1000901	Baltic – PC Multilingual, ASCII with Euro sign	901
ibm-902, T1000902	Estonia, ASCII with Euro sign	902
Ibm-905, IBM905, CP905, ebcdic-cp-tr, csIBM905, windows-20905, T1000905	Latin3 Multilingual, EBCDIC	905
ibm-912, iso-8859-2, ISO_8859-2:1987, latin2, csISOLatin2, iso-ir-101, l2, 8859_2, cp912, 912, windows-28592, T1000912	Latin2, Central European, ISO/ANSI 8-BIT, ASCII	912
ibm-913, iso-8859-3, ISO_8859-3:1988, latin3, csISOLatin3, iso-ir-109, l3, 8859_3, cp913, 913, windows-28593, T1000913	Latin 3, Maltese Esperanto, ISO/ASCII	913
ibm-914, iso-8859-4, latin4, csISOLatin4, iso-ir-110, ISO_8859-	Latin4, Baltic, ISO/ANSI 8-BIT, ASCII	914

4:1988, l4, 8859_4, cp914, 914, windows-28594, T1000914		
ibm-915, iso-8859-5, cyrillic, csISOLatinCyrillic, iso-ir-144, ISO_8859-5:1988, 8859_5, cp915, 915, windows-28595, T1000915	Latin5, Cyrillic ISO/ASCII 8-Bit, ASCII	915
ibm-916, cp916, 916, T1000916	Hebrew ISO/ASCII 8-Bit, ASCII	916
ibm-918, IBM918, CP918, ebcdic-cp-ar2, csIBM918, ebcdic-cp-ar2, T1000918	Urdu, EBCDIC	918
ibm-920, iso-8859-9, latin5, csISOLatin5, iso-ir-148, ISO_8859-9:1989, l5, 8859_9, cp920, 920, windows-28599, ECMA-128, Turkish, turkish8, T1000920	Latin5 ISO/ANSI 8-BIT, ASCII	920
ibm-921, iso-8859-13, 8859_13, cp921, 921, windows-28603, T1000921	Baltic – PC Multilingual, ASCII	921
ibm-922, IBM922, cp922, 922, T1000922	Estonian, ASCII	922
ibm-923, iso-8859-15, Latin-9, l9, 8859_15, latin0, csisolatin0, csisolatin9, iso8859_15_fdis, cp923, 923, windows-28605, T1000923	Latin 9, ASCII	923
Ibm-924, IBM00924, CCSID00924, CP00924, ebcdic-Latin9—euro, T1000924	Latin 9 EBCDIC	924
Ibm-930, Ibm930, ibm-5026, cp930, cpibm930, 930	Japanese, mixed EBCDIC/DBCS-HOST	930
ibm-931, ibm-5035, cp939, 939	Japanese, EBCDIC	931
ibm-932, ibm-942, cp932, shift_jis78, sjis78, ibm-942_VSUB_VPUA, ibm-932_VSUB_VPUA	Japanese, SJIS without MS/IBM extensions	932
Ibm-933, ibm933, cp933, cpibm933, 933	Korean, mixed EBCDIC/DBCS-HOST	933
Ibm-935, ibm935, cp935, cpibm935, 935	Chinese (simplified), mixed EBCDIC/DBCS-HOST	935
Ibm-936, windows-936-2000, GBK, CP936, MS936, windows-936	Windows Chinese(simplified), GBK	936
Ibm-937, cp937, cpibm937, 937	Chinese (traditional), mixed EBCDIC/DBCS-HOST	937
ibm-943, Shift_JIS, MS_Kanji, csShiftJIS, windows-31j, csWindows31J, x-sjis, x-ms-cp932, cp932, windows-932, cp943c, IBM-943C, ms932, pck, sjis, ibm-943_VSUB_VPUA	Japanese, Shift-JIS standard with extension	943
ibm-949, cp949, 949, ibm-949_VASCII_VSUB_VPUA, cp949c, ibm-949_VSUB_VPUA	Windows Korean, (old) encoding	949
Ibm-950, windows-950-2000, Big5, csBig5, windows-950, x-big5, cp950, 950	Windows Chinese(Traditional), ASCII BIG5	950
ibm-954, EUC-JP, csEUCPkdFmtJapanese, X-EUC-JP, eucjis, ujis	Extended Unix Code Packed Format for Japanese	954

ibm-964, EUC-TW, ibm-eucTW, cns11643, cp964, 964, ibm-964_VPUA	Extended Unix Code Packed Format for Chinese (Traditional)	964
ibm-970, EUC-KR, KSC_5601-1987, windows-51949, csEUCKR, ibm-eucKR, KSC_5601, 5601, ibm-970_VPUA, cp970, 970	Extended Unix Code Packed Format for Korean	970
ibm-971, ibm-971_VPUA	Extended Unix Code Packed Format for Korean	971
Ibm-1004, T1001004	IBM PC Desktop Publishing, ASCII	1004
ibm-1006, cp1006, 1006	Urdu, 8-bit EBCDIC	1006
Ibm-1008, cp1008, T1001008	Arabic ISO/ASCII 8-Bit, ASCII	1008
Ibm-1025, cp1025, 1025, T1001025	Cyrillic Multilingual, EBCDIC	1025
ibm-1026, IBM1026, CP1026, csIBM1026, 1026, T1001026	Latin5, Turkey, EBCDIC	1026
ibm-1046, T1001046	Arabic Extended ISO/ASCII 8-Bit, ASCII	1046
ibm-1047, IBM1047, cpibm1047	Open systems Latin1, EBCDIC	1047
ibm-1089, ISO-8859-6, Arabic, iso-ir-127, 8859_6, csISOLatinArabic, ISO_8859-6:1987, ECMA-114, ASMO-708, cp1089, 1089, windows-28596, ISO-8859-6-I, ISO-8859-6-E, T1001089	Arabic ISO-PC, ASCII	1089
ibm-1097, cp1097, 1097	Farsi, EBCDIC	1097
ibm-1098, cp1098, 1098	Farsi, ASCII	1098
Ibm-1112, cp1112, 1112, T1001112	Baltic – Multilingual, EBCDIC	1112
Ibm-1122, cp1122, 1122, T1001122	Estonia, EBCDIC	1122
Ibm-1123, cp1123, 1123, cpibm1123, T1001123	Cyrillic, Ukraine EBCDIC	1123
ibm-1124, cp1124, 1124, T1001124	ISO Cyrillic Ukraine, ASCII	1124
ibm-1125, cp1125, T1001125	Cyrillic Ukraine ASCII	1125
ibm-1129, T1001129	ISO Vietnamese, ASCII	1129
ibm-1130, T1001130	Vietnamese EBCDIC	1130
ibm-1131, cp1131, T1001131	Cyrillic Belarus EBCDIC	1131
ibm-1132, T1001132	Lao EBCDIC	1132
ibm-1133, T1001133	ISO Lao, ASCII	1133
ibm-1137	Devanagari EBCDIC	1137
ibm-1140, IBM01140, CCSID01140, CP01140, cp1140, cpibm1140, ebcdic-us-37+euro, T1001140	USA, EBCDIC with the Euro sign	1140
ibm-1141, IBM01141, CCSID01141, CP01141, cp1141, cpibm1141, ebcdic-de-273+euro, T1001141	Austria, Germany ECECP, EBCDIC with the Euro sign	1141
ibm-1142, IBM01142, CCSID01142, CP01142, cp1142, cpibm1142, ebcdic-dk-277+euro, ebcdic-no-277+euro, T1001142	Denmark, Norway ECECP, EBCDIC with the Euro sign	1142
ibm-1143, IBM01143, CCSID01143 CP01143, cp1143, cpibm1143, ebcdic-fi-278+euro, ebcdic-se-278+euro, T1001143	Finland, Sweden ECECP, EBCDIC with the Euro sign	1143
ibm-1144, IBM01144, CCSID01144, CP01144, cp1144, cpibm1144, ebcdic-it-280+euro, T1001144	Italy ECECP, EBCDIC with the Euro sign	1144
ibm-1145, IBM01145, CCSID01145, CP01145, cp1145, cpibm1145,	Spain, Latin America ECECP, EBCDIC with the Euro sign	1145

ebcdic-es-284+euro, T1001145		
ibm-1146, IBM01146, CCSID01146, CP01146, cp1146, cpibm1146, ebcdic-gb-285+euro, T1001146	UK ECECP, EBCDIC with the Euro sign	1146
ibm-1147, IBM01147, CCSID01147, CP01147, cp1147, cpibm1147, ebcdic-fr-297+euro, T1001147	France ECECP, EBCDIC with the Euro sign	1147
ibm-1148, IBM01148, CCSID01148, CP01148, cp1148, cpibm1148, ebcdic-international-500+euro, T1001148	International Latin 1, EBCDIC with the Euro sign	1148
ibm-1149, IBM01149, CCSID01149, CP01149, cp1149, cpibm1149, ebcdic-is-871+euro, T1001149	Iceland, EBCDIC with the Euro sign	1149
ibm-1153, cpibm1153, T1001153	Latin2 Multilingual with Euro, EBCDIC	1153
ibm-1154, cpibm1154, T1001154	Cyrillic Multilingual with Euro, EBCDIC	1154
ibm-1155, cpibm1155, T1001155	EBCDIC Turkey with Euro, EBCDIC	1155
ibm-1156, cpibm1156, T1001156	EBCDIC Baltic - Multi with Euro, EBCDIC	1156
ibm-1157, cpibm1157, T1001157	EBCDIC Estonia with Euro, EBCDIC	1157
ibm-1158, cpibm1158, T1001158	EBCDIC Cyrillic Ukraine with Euro, EBCDIC	1158
ibm-1160, cpibm1160, T1001160	Thailand EBCDIC with Euro, EBCDIC	1160
Ibm-1161, windows-874-2000, TIS-620, windows-874, MS874, T1001161	Thai, with the Euro sign	1161
ibm-1162, TIS-620, Windows-874, T1001162	Windows Thai, ASCII, with the Euro sign	1162
ibm-1164, cpibm1164, T1001164	Vietnamese, EBCDIC with the Euro sign	1164
Ibm-1200, UTF-16BE, x-utf-16be, UTF16BE, ibm-1201, ibm-13488, ibm-13489, ibm-17584, ibm-17585, ibm-21680, ibm-21681, ibm-25776, ibm-25777, ibm-29872, ibm-29873, ibm-61955, ibm-61956, windows-1201, cp1200, cp1201, UTF16_BigEndian, T11200	Unicode UTF-16 Big Endian, a Unicode character set with two or 4 bytes encoding units	1200
Ibm-1202, UTF-16LE, x-utf-16le, ibm-1203, ibm-13490, ibm-13491, ibm-17586, ibm-17587, ibm-21682, ibm-21683, ibm-25778, ibm-25779, ibm-29874, ibm-29875, UTF16_LittleEndian, windows-1200, UTF16LE	Unicode UTF-16 Little Endian. a Unicode character set with two or 4 bytes encoding units	1202
ibm-1204, UTF-16, ISO-10646-UCS-2, unicode, ibm-1205, csUnicode, ucs-2, UTF16	Unicode UTF-16, a Unicode character set with two or 4 bytes encoding units, a byte order mark can be used to indicate big-endian or little-endian.	1204
ibm-1208, UTF-8, UTF8, ibm-1209, ibm-5304, ibm-5305, ibm-13496, ibm-13497, ibm-17592, ibm-17593, windows-65001, cp1208	Unicode UTF-8, a Unicode character set with multibyte characters	1208
ibm-1232, UTF-32BE, UTF32_BigEndian, ibm-1233, ibm-9424	Unicode UTF-32 Big Endian, a Unicode character set with four-byte encoding units	1232
ibm-1234, UTF-32LE, UTF32_LittleEndian, ibm-1235	Unicode UTF-32 Little Endian, a Unicode character set with four-byte encoding units	1234

Ibm-1236, UTF-32, ISO-10646-UCS-4, ibm-1237, csUCS4, ucs-4	Unicode UTF-32, a Unicode character set with four-byte encoding units, a byte order mark can be used to indicate big-endian or little-endian.	1236
ibm-1250, ibm-5346, windows-1250, cp1250, T1001250	Windows, Latin 2, ASCII with Euro sign	1250
ibm-1251, ibm-5347, windows-1251, cp1251, T1001251	Windows Cyrillic, ASCII with Euro sign	1251
ibm1252, ibm-5348, windows-1252, cp1252, T1001252	Windows, Latin 1, ASCII with Euro sign	1252
ibm-1253, ibm-5349, windows-1253, cp1253, T1001253	Windows Greek, ASCII with Euro sign	1253
ibm-1254, ibm-5350, windows-1254, cp1254, T1001254	Windows Turkish, ASCII with Euro sign	1254
Ibm-1255, ibm-9447, windows-1255, cp1255	Windows Hebrew, with the Euro sign	1255
Ibm-1256, windows-1256-2000, windows-1256, cp1256	Windows Arabic, with the Euro sign	1256
ibm-1257, windows-1257, cp1257, ibm-9449, ibm-9449, ibm-5353, T1001257	Windows Baltic Rim, ASCII with Euro sign	1257
ibm-1258, ibm-5354, windows-1258, cp1258, T1001258	Windows Vietnamese, ASCII with Euro sign	1258
ibm-1276, Adobe-Standard-Encoding, csAdobeStandardEncoding, T1001276	Adobe Standard Encoding, ASCII	1276
ibm-1277, Adobe-Latin1-Encoding, T1001277	Adobe Latin1 Encoding, ASCII	1277
ibm-1363, KS_C_5601-1987, KS_C_5601-1989, KSC_5601, csKSC56011987, korean, iso-ir-149, cp1363, 5601, ksc, windows-949, ibm-1363_VSUB_VPUA, ibm-1363_VASCII_VSUB_VPUA, KS_C_5601-1987, KS_C_5601-1989, KSC_5601, ms949	Korean standard KSC for open systems	1363
ibm-1364, cp1364	Korean, mixed EBCDIC/DBCS-HOST with Euro sign	1364
ibm-1371, cpibm1371	Chinese (Traditional), EBCDIC/DBCS-HOST with the Euro sign	1371
ibm-1373, windows-950	Chinese (traditional), Windows BIG5	1373
ibm-1375, Big5-HKSCS, big5hk, HKSCS-BIG5, Big5HKSCS	Chinese (traditional) BIG5 with Hong Kong Supplementary Character Set (HKSCS)	1375
ibm-1377, T11376, HKSCS-HOST	Chinese (traditional) BIG5 with Hong Kong Supplementary Character Set (HKSCS) for IBM hots	1377
ibm-1381, cp1381, 1381	Chinese (Simplified) GB standard	1381
ibm-1383, GB2312, csGB2312, EUC-CN, ibm-eucCN, hp15CN, cp1383, 1383, ibm-1383_VPUA	Extended Unix Code Packed Format for Chinese (Simplified)	1383
ibm-1386, cp1386, windows-936, ibm-1386_VSUB_VPUA	Chinese (Simplified) GBK PC standard	1386
ibm-1388, ibm-9580	Chinese EBCDIC/DBCS-HOST for GBK with the Euro sign	1388

ibm-1390, cpibm1390	Japanese, mixed EBCDIC/DBCS-HOST with the Euro sign	1390
ibm-1392, windows-54936, gb18030, GB18030	Chinese GB18030 PC standard	1392
ibm-1399	Japan, EBCDIC, host MBCS (Latin-Kanji), with the Euro sign	1399
ibm-4899, cpibm4899	Old EBCDIC Hebrew, with the Euro sign	4899
ibm-4909	Greek ISO, ASCII with Euro sign	4909
ibm-4971, cpibm4971	EBCDIC Greek, with the Euro sign	4971
ibm-5012, ISO-8859-8, hebrew, csISOLatinHebrew, iso-ir-138, ISO_8859-8:1988, ISO-8859-8-I, ISO-8859-8-E, 8859_8, windows-28598, hebrew8	Hebrew, ASCII	5012
ibm-5471, MS950_HKSCS, hkbig5, big5-hkscs:unicode3.0	Chinese (traditional) Big5-HKSCS-2001 Hong Kong with Unicode 3.0 mappings	5471
ibm-5478, GB_2312-80, chinese, iso-ir-58, csISO58GB231280, gb2312-1980, GB2312.1980-0	Extended Unix Code Packed Format for Chinese (Simplified)	5478
ibm-9005, ISO-8859-7, greek, greek8, ELOT_928, ECMA-118, csISOLatinGreek, iso-ir-126, ISO_8859-7:1987, windows-28597, sun_eu_greek	ISO Greek (with euro update), ASCII	9005
ibm-12712, cpibm12712, ebcdic-he	EBCDIC Hebrew (new sheqel, control characters update), with the Euro sign	12712
ibm-16684	Japanese, Jis + Roman Jis Host	939
ibm-16804, cpibm16804, ebcdic-ar	Arabic, EBCDIC with the Euro sign	16804
ibm-33722, ibm-5050, EUC-JP, cp33722, 33722Extended_UNIX_Code_Packed_Format_for_Japanese, csEUCPkdFmtJapanese, X-EUC-JP, eucjis, windows-51932, ibm-33722_VPUA, IBM-eucJP, 33722_VASCII_VPUA	Extended Unix Code Packed Format for Japanese	33722
ISO-8859-10, iso-ir-157, l6, ISO_8859-10:1992, csISOLatin6, latin6	Nordic	
ISO-8859-14, iso-ir-199, ISO_8859-14:1998, latin8, iso-celtic, l8	Celtic	
x-iscii-de, windows-57002, iscii-dev	Windows Devanagari	
x-iscii-be, windows-57003, iscii-bng, windows-57006, x-iscii-as	Windows Bengali	
x-iscii-pa, windows-57011, iscii-gur	Windows Punjabi	
x-iscii-gu, windows-57010, iscii-guj	Windows Gujarati	
x-iscii-or, windows-57007, iscii-ori	Windows Oriya	
x-iscii-ta, windows-57004, iscii-tml	Windows Tamil	
x-iscii-te, windows-57005, iscii-tlg	Windows Telugu	
x-iscii-ka, windows-57008, iscii-knd	Windows Kannada	
x-iscii-ma, windows-57009, iscii-mlm	Windows Malayalam	
macos-0_2-10.2, macintosh, mac, csMacintosh, windows-10000	Macintosh, ASCII	

macos-2566-10.2, Big5-HKSCS, big5hk, HKSCS-BIG5	Chinese (traditional) BIG5 with Hong Kong Supplementary Character Set (HKSCS)	
macos-29-10.2, x-mac-centraleurroman, windows-10029, x-mac-ce, macce	Macintosh, Central Euro, ASCII	
macos-35-10.2, x-mac-turkish, windows-10081, mactr	Macintosh, Turkish, ASCII	
macos-6-10.2, x-mac-greek, windows-10006, macgr	Macintosh, Greek, ASCII	
macos-7_3-10.2, x-mac-cyrillic, windows-10007, maccy	Macintosh, Cyrillic, ASCII	
UTF-7, windows-65000	Unicode UTF-7, a Unicode character set with 7-bit characters; used primarily for email headers	

Appendix E. How to Specify a Locale

A Locale represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called *locale-sensitive* and uses the Locale to tailor information for the user. For example, word or line breaking in Unicode is a locale-sensitive operation, it should be based on the customs/conventions of the user's native country, region, or culture.

You create a Locale with one of the options listed below. Each of the component is separated by '_' in the locale string. For example, locale “en_US” is for USA English, ”zh_CN” is for Chinese used in China.

```
Language
Language_Country
```

The first option is a valid ISO Language Code. These codes are the lower-case two-letter codes as defined by ISO-639.

The second option includes an additional ISO Country Code. These codes are the upper-case two-letter codes as defined by ISO-3166.

ISO 639 - Code for the representation of names of languages

Code	Language	Code	Language
aa	Afar	cy	Welsh
ab	Abkhazian	da	Danish
af	Afrikaans	de	German
am	Amharic	dz	Bhutani
ar	Arabic	el	Greek
as	Assamese	en	English
ay	Aymara	eo	Esperanto
az	Azerbaijani	es	Spanish
ba	Bashkir	et	Estonian
be	Byelorussian	eu	Basque
bg	Bulgarian	fa	Persian
bh	Bihari	fo	Faroese
bi	Bislama	fr	French
bn	Bengali; Bangla	fy	Frisian
bo	Tibetan	ga	Irish
br	Breton	gd	Scots Gaelic
ca	Catalan	gl	Galician
co	Corsican	gn	Guarani
cs	Czech	gu	Gujarati
ha	Hausa	rn	Kirundi

he	Hebrew	ro	Romanian
hi	Hindi	ru	Russian
hr	Croatian	rw	Kinyarwanda
hu	Hungarian	sa	Sanskrit
hy	Armenian	sd	Sindhi
ia	Interlingua	sg	Sangho
id	Indonesian	sh	Serbo-Croatian
ie	Interlingue	si	Sinhalese
ik	Inupiak	sk	Slovak
is	Icelandic	sl	Slovenian
it	Italian	sm	Samoan
iu	Inuktitut	sn	Shona
ja	Japanese	so	Somali
jw	Javanese	sq	Albanian
ka	Georgian	sr	Serbian
kk	Kazakh	ss	Siswati
kl	Greenlandic	st	Sesotho
km	Cambodian	su	Sundanese
kn	Kannada	sv	Swedish
ko	Korean	sw	Swahili
ks	Kashmiri	ta	Tamil
ku	Kurdish	te	Telugu
ky	Kirghiz	tg	Tajik
la	Latin	th	Thai
ln	Lingala	ti	Tigrinya
lo	Laothian	tk	Turkmen
lt	Lithuanian	tl	Tagalog
lv	Latvian, Lettish	tn	Setswana
mg	Malagasy	to	Tonga
mi	Maori	tr	Turkish
mk	Macedonian	ts	Tsonga
ml	Malayalam	tt	Tatar
mn	Mongolian	tw	Taiwan
mo	Moldavian	ug	Uighur
mr	Marathi	uk	Ukrainian
ms	Malay	ur	Urdu
mt	Maltese	uz	Uzbek
my	Burmese	vi	Vietnamese
na	Nauru	vo	Volapuk
ne	Nepali	wo	Wolof
nl	Dutch	xh	Xhosa
no	Norwegian	yi	Yiddish
oc	Occitan	yo	Yoruba
om	(Afan) Oromo	za	Zhuang
or	Oriya	zh	Chinese
pa	Punjabi	zu	Zulu
pl	Polish		
ps	Pashto, Pushto		
pt	Portuguese		
qu	Quechua		
rm	Rhaeto-Romance		

ISO 3166 – Country/Area Codes

Code	Country / Area
AF	AFGHANISTAN
AX	ALAND ISLANDS
AL	ALBANIA
DZ	ALGERIA
AS	AMERICAN SAMOA
AD	ANDORRA
AO	ANGOLA
AI	ANGUILLA
AQ	ANTARCTICA
AG	ANTIGUA AND BARBUDA
AR	ARGENTINA
AM	ARMENIA
AW	ARUBA
AU	AUSTRALIA
AT	AUSTRIA
AZ	AZERBAIJAN
BS	BAHAMAS
BH	BAHRAIN
BD	BANGLADESH
BB	BARBADOS
BY	BELARUS
BE	BELGIUM
BZ	BELIZE
BJ	BENIN
BM	BERMUDA
BT	BHUTAN
BO	BOLIVIA
BA	BOSNIA AND HERZEGOVINA
BW	BOTSWANA
BV	BOUVET ISLAND
BR	BRAZIL
IO	BRITISH INDIAN OCEAN TERRITORY
BN	BRUNEI DARUSSALAM
BG	BULGARIA
BF	BURKINA FASO
BI	BURUNDI
KH	CAMBODIA
CM	CAMEROON
CA	CANADA
CV	CAPE VERDE
KY	CAYMAN ISLANDS
CF	CENTRAL AFRICAN REPUBLIC
TD	CHAD
CL	CHILE
CN	CHINA
CX	CHRISTMAS ISLAND
CC	COCOS (KEELING) ISLANDS
CO	COLOMBIA
KM	COMOROS

CG	CONGO
CD	CONGO, THE DEMOCRATIC REPUBLIC OF THE
CK	COOK ISLANDS
CR	COSTA RICA
CI	COTE D'IVOIRE
HR	CROATIA
CU	CUBA
CY	CYPRUS
CZ	CZECH REPUBLIC
DK	DENMARK
DJ	DJIBOUTI
DM	DOMINICA
DO	DOMINICAN REPUBLIC
EC	ECUADOR
EG	EGYPT
SV	EL SALVADOR
GQ	EQUATORIAL GUINEA
ER	ERITREA
EE	ESTONIA
ET	ETHIOPIA
FK	FALKLAND ISLANDS (MALVINAS)
FO	FAROE ISLANDS
FJ	FIJI
FI	FINLAND
FR	FRANCE
GF	FRENCH GUIANA
PF	FRENCH POLYNESIA
TF	FRENCH SOUTHERN TERRITORIES
GA	GABON
GM	GAMBIA
GE	GEORGIA
DE	GERMANY
GH	GHANA
GI	GIBRALTAR
GR	GREECE
GL	GREENLAND
GD	GRENADA
GP	GUADELOUPE
GU	GUAM
GT	GUATEMALA
GG	GUERNSEY
GN	GUINEA
GW	GUINEA-BISSAU
GY	GUYANA
HT	HAITI
HM	HEARD ISLAND AND MCDONALD ISLANDS
VA	HOLY SEE (VATICAN CITY STATE)
HN	HONDURAS
HK	HONG KONG
HU	HUNGARY
IS	ICELAND
IN	INDIA

ID	INDONESIA
IR	IRAN (ISLAMIC REPUBLIC OF)
IQ	IRAQ
IE	IRELAND
IM	ISLE OF MAN
IL	ISRAEL
IT	ITALY
JM	JAMAICA
JP	JAPAN
JE	JERSEY
JO	JORDAN
KZ	KAZAKHSTAN
KE	KENYA
KI	KIRIBATI
KP	KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF
KR	KOREA, REPUBLIC OF
KW	KUWAIT
KG	KYRGYZSTAN
LA	LAO PEOPLE'S DEMOCRATIC REPUBLIC
LV	LATVIA
LB	LEBANON
LS	LESOTHO
LR	LIBERIA
LY	LIBYAN ARAB JAMAHIRIYA
LI	LIECHTENSTEIN
LT	LITHUANIA
LU	LUXEMBOURG
MO	MACAO
MK	MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF
MG	MADAGASCAR
MW	MALAWI
MY	MALAYSIA
MV	MALDIVES
ML	MALI
MT	MALTA
MH	MARSHALL ISLANDS
MQ	MARTINIQUE
MR	MAURITANIA
MU	MAURITIUS
YT	MAYOTTE
MX	MEXICO
FM	MICRONESIA, FEDERATED STATES OF
MD	MOLDOVA, REPUBLIC OF
MC	MONACO
MN	MONGOLIA
ME	MONTENEGRO
MS	MONTSERRAT
MA	MOROCCO
MZ	MOZAMBIQUE
MM	MYANMAR
NA	NAMIBIA
NR	NAURU

NP	NEPAL
NL	NETHERLANDS
AN	NETHERLANDS ANTILLES
NC	NEW CALEDONIA
NZ	NEW ZEALAND
NI	NICARAGUA
NE	NIGER
NG	NIGERIA
NU	NIUE
NF	NORFOLK ISLAND
MP	NORTHERN MARIANA ISLANDS
NO	NORWAY
OM	OMAN
PK	PAKISTAN
PW	PALAU
PS	PALESTINIAN TERRITORY, OCCUPIED
PA	PANAMA
PG	PAPUA NEW GUINEA
PY	PARAGUAY
PE	PERU
PH	PHILIPPINES
PN	PITCAIRN
PL	POLAND
PT	PORTUGAL
PR	PUERTO RICO
QA	QATAR
RE	REUNION
RO	ROMANIA
RU	RUSSIAN FEDERATION
RW	RWANDA
SH	SAINT HELENA
KN	SAINT KITTS AND NEVIS
LC	SAINT LUCIA
PM	SAINT PIERRE AND MIQUELON
VC	SAINT VINCENT AND THE GRENADINES
WS	SAMOA
SM	SAN MARINO
ST	SAO TOME AND PRINCIPE
SA	SAUDI ARABIA
SN	SENEGAL
RS	SERBIA
SC	SEYCHELLES
SL	SIERRA LEONE
SG	SINGAPORE
SK	SLOVAKIA
SI	SLOVENIA
SB	SOLOMON ISLANDS
SO	SOMALIA
ZA	SOUTH AFRICA
GS	SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS
ES	SPAIN
LK	SRI LANKA

SD	SUDAN
SR	SURINAME
SJ	SVALBARD AND JAN MAYEN
SH	ST. HELENA
PM	ST. PIERRE AND MIQUELON
SZ	SWAZILAND
SE	SWEDEN
CH	SWITZERLAND
SY	SYRIAN ARAB REPUBLIC
TW	TAIWAN
TJ	TAJIKISTAN
TZ	TANZANIA, UNITED REPUBLIC OF
TH	THAILAND
TL	TIMOR-LESTE
TG	TOGO
TK	TOKELAU
TO	TONGA
TT	TRINIDAD AND TOBAGO
TN	TUNISIA
TR	TURKEY
TM	TURKMENISTAN
TC	TURKS AND CAICOS ISLANDS
TV	TUVALU
UG	UGANDA
UA	UKRAINE
AE	UNITED ARAB EMIRATES
GB	UNITED KINGDOM
US	UNITED STATES
UM	UNITED STATES MINOR OUTLYING ISLANDS
UY	URUGUAY
UZ	UZBEKISTAN
VU	VANUATU
VA	VATICAN CITY STATE (HOLY SEE)
VE	VENEZUELA
VN	VIET NAM
VG	VIRGIN ISLANDS (BRITISH)
VI	VIRGIN ISLANDS (U.S.)
WF	WALLIS AND FUTUNA
EH	WESTERN SAHARA
YE	YEMEN
YU	YUGOSLAVIA
ZM	ZAMBIA
ZW	ZIMBABWE

Appendix F. Visual Studio C++ Run-Time Routines

For reference convenience, some Visual Studio C++ routines you may use are listed here by the categories. Refer to Microsoft MSDN Library for more details.

Buffer-Manipulation Routines

Routine	Use
_memcpy	Copy characters from one buffer to another until given character or given number of characters has been copied
memchr	Return pointer to first occurrence, within specified number of characters, of given character in buffer
memcmp	Compare specified number of characters from two buffers
memcpy	Copy specified number of characters from one buffer to another
_memicmp	Compare specified number of characters from two buffers without regard to case
memmove	Copy specified number of characters from one buffer to another
memset	Use given character to initialize specified number of bytes in the buffer
_swab	Swap bytes of data and store them at specified location

Character-Classification Routines

Routine	Character test condition
isalnum, iswalnum, _ismbcalnum	True if alphanumeric
isalpha, iswalpha, _ismbcalpha	True if alphabetic
_isascii, iswascii	True if ASCII
iscntrl, iswcntrl	True if control character
_iscsym	True if letter, underscore, or digit
_iscsymf	True if letter or underscore
isdigit, iswdigit, _ismbcdigit	True if decimal digit
isgraph, iswgraph, _ismbcgraph	True if printable other than space
islower, iswlower, _ismbclower	True if lowercase
_ismbchira	True if Hiragana

_ismbckata	True if Katakana
_ismbclegal	True if Legal multibyte character
_ismbcl0	True if Japan-level 0 multibyte character
_ismbcl1	True if Japan-level 1 multibyte character
_ismbcl2	True if Japan-level 2 multibyte character
_ismbcsymbol	True if Nonalphanumeric multibyte character
isprint, iswprint, _ismbcprint	True if printable character
ispunct, iswpunct, _ismbcpunct	True if punctuation
isspace, iswspace, _ismbcspace	True if white-space
isupper, iswupper, _ismbcupper	True if uppercase
iswctype	Property specified by *desc* argument
isxdigit, iswxdigit	True if hexadecimal digit
mblen	Return length of valid multibyte character; result depends on **LC_CTYPE** category setting of current

Console and Port I/O Routines

Routine	Use
_cgets, _cgetws	Read string from console
_cprintf, _cwprintf	Write formatted data to console
_cputs	Write string to console
_cscanf, _cwscanf	Read formatted data from console
_getch, _getwch	Read character from console
_getche, _getwche	Read character from console and echo it
_inp	Read one byte from specified I/O port
_inpd	Read float word from specified I/O port
_inpw	Read 2-byte word from specified I/O port
_kbhit	Check for keystroke at console; use before attempting to read from console
_outp	Write one byte to specified I/O port
_outpd	Write float word to specified I/O port
_outpw	Write word to specified I/O port
_putch, _putwch	Write character to console
_ungetch, _ungetwch	"Unget" last character read from console so it becomes next character read

Data-Conversion Routines

Routine	Use
abs	Find absolute value of integer
atof	Convert string to **float**
atoi, _atoi64	Convert string to **int**
atol	Convert string to **long**
_ecvt	Convert **float** to string of specified length
_fcvt	Convert **float** to string with specified number of digits following decimal point
_gcvt	Convert **float** number to string; store string in buffer
_itoa, _i64toa, _itow, _i64tow	Convert **int** to string
labs	Find absolute value of **long** integer
_ltoa, _ltow	Convert **long** to string
_mbbtombc	Convert 1-byte multibyte character to corresponding 2-byte multibyte character
_mbcjistojms	Convert Japan Industry Standard (JIS) character to Japan Microsoft (JMS) character
_mbcjmstojis	Convert JMS character to JIS character
_mbctohira	Convert multibyte character to 1-byte hiragana code
_mbctokata	Convert multibyte character to 1-byte katakana code
_mbctombb	Convert 2-byte multibyte character to corresponding 1-byte multibyte character
mbstowcs	Convert sequence of multibyte characters to corresponding sequence of wide characters
mbtowc	Convert multibyte character to corresponding wide character
strtod, wcstod	Convert string to **float**
strtol, wcstol	Convert string to **long** integer
strtoul, wcstoul	Convert string to **unsigned long** integer
strxfrm, wcsxfrm	Transform string into collated form based on locale-specific information
_toascii	Convert character to ASCII code
tolower, towlower, _mbctolower	Test character and convert to lowercase if currently uppercase
_tolower	Convert character to lowercase unconditionally
toupper, towupper, _mbctoupper	Test character and convert to uppercase if currently lowercase
_toupper	Convert character to uppercase unconditionally
_ultoa, _ultow	Convert **unsigned long** to string
wcstombs	Convert sequence of wide characters to corresponding sequence of multibyte characters

wctomb	Convert wide character to corresponding multibyte character
_wtof	Convert wide-character string to a **float**
_wtoi, _wtoi64	Convert wide-character string to **int** or **_int64**
_wtol	Convert wide-character string to **long**

Directory-Control Routines

Routine	Use
_chdir, _wchdir	Change current working directory
_chdrive	Change current drive
_getcwd, _wgetcwd	Get current working directory for default drive
_getdcwd, _wgetdcwd	Get current working directory for specified drive
_getdiskfree	Populates a **_diskfree_t** structure with information about a disk drive.
_getdrive	Get current (default) drive
_getdrives	Returns a bitmask representing the currently available disk drives.
_mkdir, _wmkdir	Make new directory
_rmdir, _wrmdir	Remove directory
_searchenv, _wsearchenv	Search for given file on specified paths

File-Handling Routines (File Descriptor)

Routine	Use
_chsize	Change file size
_filelength	Get file length
_fstat, _fstat64, _fstati64	Get file-status information on descriptor
_isatty	Check for character device
_locking	Lock areas of file
_setmode	Set file-translation mode

File-Handling Routines (Path or Filename)

Routine	Use
_access, _waccess	Check file-permission setting
_chmod, _wchmod	Change file-permission setting
_fullpath, _wfullpath	Expand a relative path to its absolute path name
_get_osfhandle	Return operating-system file handle associated with existing stream **FILE** pointer
_makepath, _wmakepath	Merge path components into single, full path

_mktemp, _wmktemp	Create unique filename
_open_osfhandle	Associate C run-time file descriptor with existing operating-system file handle
remove, _wremove	Delete file
rename, _wrename	Rename file
_splitpath, _wsplitpath	Parse path into components
_stat, _stat64, _stati64, _wstat, _wstat64, _wstati64	Get file-status information on named file
_umask	Set default permission mask for new files created by program
_unlink, _wunlink	Delete file

File-Handling Routines (Open File)

Routine	Use
fopen	Opens a file and returns a pointer to the open file.
_fsopen	Open a stream with file sharing and returns a pointer to the open file.
_open	Opens a file and returns a file descriptor to the opened file.
_sopen	Open a file with file sharing and returns a file descriptor to the open file.
_fdopen	Associates a stream with a file that was previously opened for low-level I/O and returns a pointer to the open stream.
_fileno	Gets the file descriptor associated with a stream.
_open_osfhandle	Associates C run-time file descriptor with an existing operating-system file handle.
_pipe	Creates a pipe for reading and writing.
freopen	Reassign a file pointer.

Low-Level I/O Functions

Function	Use
_close	Close file
_commit	Flush file to disk
_creat, _wcreat	Create file
_dup	Return next available file descriptor for given file
_dup2	Create second descriptor for given file
_eof	Test for end of file
_lseek, _lseeki64	Reposition file pointer to given location
_open, _wopen	Open file
_read	Read data from file
_sopen, _wsopen	Open file for file sharing

_tell, _telli64	Get current file-pointer position
_umask	Set file-permission mask
_write	Write data to file

Stream I/O Routines

Routine	Use
clearerr	Clear error indicator for stream
fclose	Close stream
_fcloseall	Close all open streams except **stdin**, **stdout**, and **stderr**
_fdopen, wfdopen	Associate stream with file descriptor of open file
feof	Test for end of file on stream
ferror	Test for error on stream
fflush	Flush stream to buffer or storage device
fgetc, fgetwc	Read character from stream (function versions of **getc** and **getwc**)
_fgetchar, _fgetwchar	Read character from **stdin** (function versions of **getchar** and **getwchar**)
fgetpos	Get position indicator of stream
fgets, fgetws	Read string from stream
_fileno	Get file descriptor associated with stream
_flushall	Flush all streams to buffer or storage device
fopen, _wfopen	Open stream
fprintf, fwprintf	Write formatted data to stream
fputc, fputwc	Write a character to a stream (function versions of **putc** and **putwc**)
_fputchar, _fputwchar	Write character to **stdout** (function versions of **putchar** and **putwchar**)
fputs, fputws	Write string to stream
fread	Read unformatted data from stream
freopen, _wfreopen	Reassign **FILE** stream pointer to new file or device
fscanf, fwscanf	Read formatted data from stream
fseek	Move file position to given location
fsetpos	Set position indicator of stream
_fsopen, _wfsopen	Open stream with file sharing
ftell	Get current file position
fwrite	Write unformatted data items to stream
getc, getwc	Read character from stream (macro versions of **fgetc** and **fgetwc**)
getchar, getwchar	Read character from **stdin** (macro versions of **fgetchar** and **fgetwchar**)
_getmaxstdio	Returns the number of simultaneously open files permitted at the stream I/O level.
gets, getws	Read line from **stdin**

_getw	Read binary **int** from stream
perror	Displays a string version of the current error to **STDERR**
printf, wprintf	Write formatted data to **stdout**
putc, putwc	Write character to a stream (macro versions of **fputc** and **fputwc**)
putchar, putwchar	Write character to **stdout** (macro versions of **fputchar** and **fputwchar**)
puts, _putws	Write line to stream
_putw	Write binary **int** to stream
remove	Erase a file
rename	Rename a file
rewind	Move file position to beginning of stream
_rmtmp	Remove temporary files created by **tmpfile**
scanf, wscanf	Read formatted data from **stdin**
setbuf	Control stream buffering
_setmaxstdio	Set a maximum for the number of simultaneously open files at the stream I/O level.
setvbuf	Control stream buffering and buffer size
_snprintf, _snwprintf	Write formatted data of specified length to string
_snscanf, _snwscanf	Read formatted data of a specified length from the standard input stream.
sprintf, swprintf	Write formatted data to string
sscanf, swscanf	Read formatted data from string
_tempnam, _wtempnam	Generate temporary filename in given directory
tmpfile	Create temporary file
tmpnam, _wtmpnam	Generate temporary filename
ungetc, ungetwc	Push character back onto stream
vfprintf, vfwprintf	Write formatted data to stream
vprintf, vwprintf	Write formatted data to **stdout**
_vsnprintf, _vsnwprintf	Write formatted data of specified length to buffer
vsprintf, vswprintf	Write formatted data to buffer

String-Manipulation Routines

Routine	Use
_mbscoll, _mbsicoll, _mbsncoll, _mbsnicoll	Compare two multibyte-character strings using multibyte code page information (**_mbsicoll** and **_mbsnicoll** are case-insensitive)
_mbsdec, _strdec, _wcsdec	Move string pointer back one character
_mbsinc, _strinc, _wcsinc	Advance string pointer by one character

_mbslen	Get number of multibyte characters in multibyte-character string; dependent upon OEM code page
_mbsnbcat	Append, at most, first *n* bytes of one multibyte-character string to another
_mbsnbcmp	Compare first *n* bytes of two multibyte-character strings
_mbsnbcnt	Return number of multibyte-character bytes within supplied character count
_mbsnbcpy	Copy *n* bytes of string
_mbsnbicmp	Compare *n* bytes of two multibyte-character strings, ignoring case
_mbsnbset	Set first *n* bytes of multibyte-character string to specified character
_mbsnccnt	Return number of multibyte characters within supplied byte count
_mbsnextc, _strnextc, _wcsnextc	Find next character in string
_mbsninc. _strninc, _wcsninc	Advance string pointer by *n* characters
_mbsspnp, _strspnp, _wcsspnp	Return pointer to first character in given string that is not in another given string
_mbstrlen	Get number of multibyte characters in multibyte-character string; locale-dependent
_scprintf, _scwprintf	Return the number of characters in a formatted string
_snscanf, _snwscanf	Read formatted data of a specified length from the standard input stream.
sprintf, _stprintf	Write formatted data to a string
strcat, wcscat, _mbscat	Append one string to another
strchr, wcschr, _mbschr	Find first occurrence of specified character in string
strcmp, wcscmp, _mbscmp	Compare two strings
strcoll, wcscoll, _stricoll, _wcsicoll, _strncoll, _wcsncoll, _strnicoll, _wcsnicoll	Compare two strings using current locale code page information (**_stricoll**, **_wcsicoll**, **_strnicoll**, and **_wcsnicoll** are case-insensitive)
strcpy, wcscpy, _mbscpy	Copy one string to another
strcspn, wcscspn, _mbscspn,	Find first occurrence of character from specified character set in string
_strdup, _wcsdup, _mbsdup	Duplicate string
strerror, _wcserror	Map error number to message string
_strerror, _wcserror	Map user-defined error message to string
strftime, wcsftime	Format date-and-time string
_stricmp, _wcsicmp, _mbsicmp	Compare two strings without regard to case
strlen, wcslen, _mbslen, _mbstrlen	Find length of string
_strlwr, _wcslwr, _mbslwr	Convert string to lowercase

strncat, wcsncat, _mbsncat	Append characters of string
strncmp, wcsncmp, _mbsncmp	Compare characters of two strings
strncpy, wcsncpy, _mbsncpy	Copy characters of one string to another
_strnicmp, _wcsnicmp, _mbsnicmp	Compare characters of two strings without regard to case
_strnset, _wcsnset, _mbsnset	Set first *n* characters of string to specified character
strpbrk, wcspbrk, _mbspbrk	Find first occurrence of character from one string in another string
strrchr, wcsrchr,_mbsrchr	Find last occurrence of given character in string
_strrev, _wcsrev,_mbsrev	Reverse string
_strset, _wcsset, _mbsset	Set all characters of string to specified character
strspn, wcsspn, _mbsspn	Find first substring from one string in another string
strstr, wcsstr, _mbsstr	Find first occurrence of specified string in another string
strtok, wcstok, _mbstok	Find next token in string
_strupr, _wcsupr, _mbsupr	Convert string to uppercase
strxfrm, wcsxfrm	Transform string into collated form based on locale-specific information
vsprintf, _vstprint	Write formatted output using a pointer to a list of arguments

Time Routines

Function	Use
asctime, _wasctime	Convert time from type **struct tm** to character string
clock	Return elapsed CPU time for process
ctime, _ctime64, _wctime, _wctime64	Convert time from type **time_t** or **_time64_t** to character string
difftime	Compute difference between two times
_ftime, _ftime64	Store current system time in variable of type **struct _timeb** or type **struct _timeb64**
_futime, _futime64	Set modification time on open file
gmtime, _gmtime64	Convert time from type **time_t** to **struct tm** or from type **_time64_t** to **struct tm**
localtime, _localtime64	Convert time from type **time_t** to **struct tm** or from type **_time64_t** to **struct tm** with local correction
mktime, _mktime64	Convert time to calendar value
_strdate, _wstrdate	Return current system date as string
strftime, wcsftime	Format date-and-time string for international use
_strtime, _wstrtime	Return current system time as string

time, _time64	Get current system time as type **time_t** or as type **_time64_t**
_tzset	Set external time variables from environment time variable **TZ**
_utime, _utime64, _wutime, _wutime64	Set modification time for specified file using either current time or time value stored

Memory and Other Routines

Function	Use
abort	Stops the program
assert	Stops the program if an expression isn't true
atexit	Sets a function to be called when the program exits
bsearch	Perform a binary search
calloc	Allocates a two-dimensional chunk of memory
exit	Stop the program
free	Frees memory available for future allocation
getenv	Get enviornment information about a variable
longjmp	Start execution at a certain point in the program
qsort	Perform a quicksort
malloc	Allocates memory
raise	Send a signal to the program
rand	Returns a pseudorandom number
realloc	Changes the size of previously allocated memory
setjmp	Set execution to start at a certain point
signal	Register a function as a signal handler
srand	Initialize the random number generator
system	Perform a system call
va_arg	Use variable length parameter lists

MakeAFP