MakeAFP Formatter Reference

Version 4.0
This edition applies to the MakeAFP Formatter.

MakeAFP welcomes your comments and suggestions. You can send your comments and suggestions to:
support@makeafp.com

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

Chapter 1. MakeAFP Formatter Functions

Using the MakeAFP Formatter Function Calls to Build an AFP Document ..............................................1
Hierarchy of MakeAFP Formatter Calls ........................................................................................................2
Session Level calls ...............................................................................................................................................2
Document Level Calls .........................................................................................................................................2
Page-level Calls ..................................................................................................................................................3
Format of the Function Call Descriptions ........................................................................................................5
2D Aztec Barcode by Drawing ...........................................................................................................................6
2D DataMatrix Barcode by Drawing .....................................................................................................................8
2D MaxiCode Barcode by Drawing ......................................................................................................................10
2D MicroPDF417 Barcode by Drawing ................................................................................................................12
2D PDF417 Barcode by Drawing .........................................................................................................................14
2D PDF417 Truncated Barcode by Drawing ........................................................................................................16
2D QR Code Barcode by Drawing ......................................................................................................................18
Barchart by Shading ...........................................................................................................................................20
Barchart Element Label .....................................................................................................................................23
Barchart Legend ..................................................................................................................................................25
Barchart Shades ..................................................................................................................................................27
Barcode (Linear) by Drawing .............................................................................................................................28
BCOCA Barcode (Linear) ..................................................................................................................................31
BCOCA DataMatrix 2D Barcode ........................................................................................................................38
BCOCA MaxiCode 2D Barcode ..........................................................................................................................42
BCOCA PDF417 2D Barcode ...............................................................................................................................45
BCOCA QR Code 2D Barcode .............................................................................................................................48
Begin Box .............................................................................................................................................................52
Begin Horizontal Line ........................................................................................................................................53
Begin Index Group ............................................................................................................................................54
Begin Index Group With Encoding Conversion ...............................................................................................56
Begin Paragraph of 1-Byte Text ..........................................................................................................................58
Begin Paragraph of DBCS-HOST ........................................................................................................................60
Begin Paragraph of Japanese ..............................................................................................................................61
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic Fillet</td>
<td>156</td>
</tr>
<tr>
<td>Graphic Horizontal Boxes</td>
<td>157</td>
</tr>
<tr>
<td>Graphic Horizontal Line</td>
<td>158</td>
</tr>
<tr>
<td>Graphic Horizontal Lines</td>
<td>159</td>
</tr>
<tr>
<td>Graphic Lines</td>
<td>160</td>
</tr>
<tr>
<td>Graphic Linechart</td>
<td>162</td>
</tr>
<tr>
<td>Graphic Linechart Legend</td>
<td>166</td>
</tr>
<tr>
<td>Graphic Line Ending Style</td>
<td>168</td>
</tr>
<tr>
<td>Graphic Line Join Type</td>
<td>169</td>
</tr>
<tr>
<td>Graphic Line Type</td>
<td>170</td>
</tr>
<tr>
<td>Graphic Line Width</td>
<td>171</td>
</tr>
<tr>
<td>Graphic Marker</td>
<td>172</td>
</tr>
<tr>
<td>Graphic Partial Ellipse</td>
<td>173</td>
</tr>
<tr>
<td>Graphic Pattern</td>
<td>175</td>
</tr>
<tr>
<td>Graphic Piechart</td>
<td>176</td>
</tr>
<tr>
<td>Graphic Piechart Explode</td>
<td>178</td>
</tr>
<tr>
<td>Graphic Piechart Label</td>
<td>179</td>
</tr>
<tr>
<td>Graphic Piechart Legend</td>
<td>181</td>
</tr>
<tr>
<td>Graphic Polarchart</td>
<td>182</td>
</tr>
<tr>
<td>Graphic Polarchart Legend</td>
<td>186</td>
</tr>
<tr>
<td>Graphic Position (Query)</td>
<td>187</td>
</tr>
<tr>
<td>Graphic Position</td>
<td>188</td>
</tr>
<tr>
<td>Graphic Rendering Intent</td>
<td>189</td>
</tr>
<tr>
<td>Graphic Skew Line</td>
<td>191</td>
</tr>
<tr>
<td>Graphic Stacked Barchart</td>
<td>192</td>
</tr>
<tr>
<td>Graphic Vertical Boxes</td>
<td>196</td>
</tr>
<tr>
<td>Graphic Vertical Line</td>
<td>197</td>
</tr>
<tr>
<td>Graphic Vertical Lines</td>
<td>198</td>
</tr>
<tr>
<td>Horizontal Boxes</td>
<td>199</td>
</tr>
<tr>
<td>Horizontal Line</td>
<td>200</td>
</tr>
<tr>
<td>Horizontal Lines</td>
<td>201</td>
</tr>
<tr>
<td>Horizontal Shades</td>
<td>202</td>
</tr>
<tr>
<td>Inch Value</td>
<td>204</td>
</tr>
<tr>
<td>Import AFP Image Object</td>
<td>205</td>
</tr>
<tr>
<td>Import Data-Object Container</td>
<td>206</td>
</tr>
<tr>
<td>Import Data-Object Container from Memory Buffer</td>
<td>210</td>
</tr>
<tr>
<td>Import Data-Object Image from Multi-page TIFF</td>
<td>214</td>
</tr>
<tr>
<td>Import Image with Conversion to AFP IOCA FS11</td>
<td>218</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Paragraph of UTF-8 Text Converting from Wild-Char</td>
<td>279</td>
</tr>
<tr>
<td>Point Value</td>
<td>281</td>
</tr>
<tr>
<td>Position of Text</td>
<td>282</td>
</tr>
<tr>
<td>Print AFP File</td>
<td>283</td>
</tr>
<tr>
<td>Put Index Tag</td>
<td>285</td>
</tr>
<tr>
<td>Put Index Tag With Encoding Conversion</td>
<td>287</td>
</tr>
<tr>
<td>Put Paragraph Text of ASCII / EBCDIC</td>
<td>289</td>
</tr>
<tr>
<td>Put Paragraph Text of DBCS-HOST</td>
<td>291</td>
</tr>
<tr>
<td>Put Paragraph Text of Japanese</td>
<td>292</td>
</tr>
<tr>
<td>Put Paragraph Text of Korean</td>
<td>293</td>
</tr>
<tr>
<td>Put Paragraph Text of Simplified Chinese</td>
<td>294</td>
</tr>
<tr>
<td>Put Paragraph Text of Traditional Chinese</td>
<td>295</td>
</tr>
<tr>
<td>Put Paragraph Text of Wild-Char</td>
<td>296</td>
</tr>
<tr>
<td>Put Paragraph Text of Wild-Char Converting from Legacy String</td>
<td>298</td>
</tr>
<tr>
<td>Put Paragraph Text of UTF-8</td>
<td>300</td>
</tr>
<tr>
<td>Put Paragraph Text of UTF-8 Converting from Legacy String</td>
<td>302</td>
</tr>
<tr>
<td>Put Paragraph Text of UTF-8 Converting from Wild-Char</td>
<td>304</td>
</tr>
<tr>
<td>Rendering Color Intent</td>
<td>306</td>
</tr>
<tr>
<td>Right Align Single Byte Text</td>
<td>308</td>
</tr>
<tr>
<td>Right Align Japanese</td>
<td>309</td>
</tr>
<tr>
<td>Right Align Korean</td>
<td>310</td>
</tr>
<tr>
<td>Right Align Simplified Chinese</td>
<td>311</td>
</tr>
<tr>
<td>Right Align Traditional Chinese</td>
<td>312</td>
</tr>
<tr>
<td>Right Align SBCS-HOST/DBCS-HOST Text</td>
<td>313</td>
</tr>
<tr>
<td>Right Align Wild-Char Text</td>
<td>314</td>
</tr>
<tr>
<td>Right Align Wild-Char Text Converting from Legacy String</td>
<td>315</td>
</tr>
<tr>
<td>Right Align UTF-8 Text</td>
<td>316</td>
</tr>
<tr>
<td>Right Align UTF-8 Text Converting from Legacy String</td>
<td>317</td>
</tr>
<tr>
<td>Right Align UTF-8 Text Converting from Wild-Char</td>
<td>318</td>
</tr>
<tr>
<td>Set Default Unit</td>
<td>319</td>
</tr>
<tr>
<td>Shade</td>
<td>321</td>
</tr>
<tr>
<td>Simplified Chinese Width</td>
<td>323</td>
</tr>
<tr>
<td>Skip Lines</td>
<td>324</td>
</tr>
<tr>
<td>Start Session</td>
<td>325</td>
</tr>
<tr>
<td>Text Orientation</td>
<td>326</td>
</tr>
<tr>
<td>Text Width of ASCII/EBCDIC String</td>
<td>328</td>
</tr>
<tr>
<td>TIFF Pages</td>
<td>329</td>
</tr>
<tr>
<td>Traditional Chinese Width</td>
<td>330</td>
</tr>
</tbody>
</table>
Wild-Char String Width........................................................................................................331
UTF-8 String Width ................................................................................................................332
Vertical Boxes .........................................................................................................................333
Vertical Line ............................................................................................................................334
Vertical Lines ..........................................................................................................................335
Vertical Shades .......................................................................................................................336
View AFP File .........................................................................................................................337
Word Space ...............................................................................................................................338
X Absolute Position of Text ....................................................................................................339
X Current Position of Text (Query) .........................................................................................340
X Relative Move Position of Text ............................................................................................341
Y Absolute Position of Text ....................................................................................................342
Y Current Position of Text (Query) ..........................................................................................343
Y Relative Move Position of Text ............................................................................................344

Chapter 2. MakeAFP Formatter Parameters .............................................................................347

Conventions Used in This Chapter .................................................................................................347
CMR – Specifies a Colo Management Resource .........................................................................348
CPGID – Specifies a Code Page Identifier ..................................................................................348
FDEF – Specified a Form Definition ............................................................................................349
FDEFLIB – Specifies the Library Path of Form Definitions ..........................................................350
FONT – Specifies an AFP FOCA Raster or Outline Font ..............................................................351
FONT – Specifies an OpenType/TrueType Font ..........................................................................352
FONTLIB – Specifies the Library Path of Fonts .........................................................................353
INDEXOBJ – Specifies Generating of the AFP Index Object File ...............................................354
OBJT – Specifies an AFP Object or non-AFP Object ..................................................................355
OBJTLIB – Specifies the Library Path of Image and CMR Objects ............................................356
OVLY – Specifies an Overlay ......................................................................................................357
OVLYLIB – Specifies the Library Path of Overlays ....................................................................358
PRMODE – Specifies the Type of Input Data and Processing Option ........................................359
PSEG – Specifies a Page Segment .............................................................................................360
PSEGLIB – Specifies the Library Path of Page Segments ..........................................................361
RESLIB – Specifies the Library Path of AFP Resources and Objects ..........................................362
RESTYPE - Specifies the Types of Resources to be Retrieved ..................................................363

Chapter 3. MakeAFP Formatter Variables ..................................................................................364

FileInName – File Name of the Default Input File ......................................................................364
MaxPaging – Maximum Number of AFP Page Buffers ..............................................................365
Chapter 4. String Manipulation Functions .................................................................363

Comma Float .............................................................................................................363
Comma Integer .........................................................................................................364
Comma Digital String ...............................................................................................364
Delete Characters .....................................................................................................365
Extract Substrings ....................................................................................................366
Find String ................................................................................................................367
First Character .........................................................................................................368
In Substitution Table .................................................................................................368
Insert String ..............................................................................................................369
Is Empty ....................................................................................................................369
Last Character ..........................................................................................................370
Left Copy ..................................................................................................................370
Left Copy and Pad .....................................................................................................371
Left Copy and Right Trim .........................................................................................371
Left Trim ...................................................................................................................372
Left Trim for EBCDIC ...............................................................................................373
Match String Comparing .........................................................................................373
Pattern Searching .....................................................................................................374
Remove String .........................................................................................................375
Replace String .........................................................................................................375
Reverse Find String ..................................................................................................376
Right Copy ................................................................................................................377
Right Copy and Left Trim ..........................................................................................378
Right Copy and Pad ...................................................................................................378
Right Copy and Right Trim .......................................................................................379
Right Trim ................................................................................................................380
Right Trim for EBCDIC ..............................................................................................380
String Pad ..................................................................................................................380
Substitute String ......................................................................................................381
Substitute Change .....................................................................................................382
Substring ...................................................................................................................382
Substring Left Trim ..................................................................................................383
Substring Pad ............................................................................................................384
Substring Right Trim ...............................................................................................385
Substring Trim Both Sides .........................................................................................385
System Date and Time .............................................................................................................386
Title String .............................................................................................................................387
Thai Compose .........................................................................................................................388
Translate Digits to Simplified Chinese Figures .................................................................388
Translate Digits to Traditional Chinese Figures .................................................................389
Trim Both Sides .......................................................................................................................389
Trim Both Sides for EBCDIC .................................................................................................390
Vietnamese Alignment ...........................................................................................................390

Chapter 5. Conversion Functions .........................................................................................391
Codepage/Charset to Wild-Char Conversion ........................................................................391
Codepage/Charset TO UTF-8 Conversion ............................................................................392
Default Encoding Names .......................................................................................................392
Universal Conversion ............................................................................................................393
Wild-Char to Codepage/Charset Conversion ........................................................................394
Wild-Char to UTF-8 Conversion ............................................................................................394
UTF-8 to Wild-Char Conversion ............................................................................................395
UTF-8 to Codepage/Charset Conversion ...............................................................................395
Vietnamese Codepage/Charset Codepage/Charset Conversion ...............................................397

Appendix A. ASCII/EBCDIC AFP Code Pages and CPGID Summary ..................................398
Appendix B. SBCS/DBCS AFP Code Pages and CPGID Summary ........................................402
Appendix C. Combined SBCS/DBCS CPGID Summary .........................................................403
Appendix D. Encoding Names and Alias .............................................................................404
Appendix E. How to Specify a Locale ....................................................................................413
Appendix F. RGB Color Values ............................................................................................420
Appendix G. Visual Studio C++ Run-Time Routines ............................................................422
Buffer-Manipulation Routines ...............................................................................................422
Character-Classification Routines .........................................................................................422
Console and Port I/O Routines .............................................................................................423
Data-Conversion Routines .....................................................................................................424
Directory-Control Routines ...................................................................................................425
File-Handling Routines (File Descriptor) .............................................................................425
File-Handling Routines (Path or Filename) ..........................................................................425
File-Handling Routines (Open File) .....................................................................................426
Low-Level I/O Functions ........................................................................................................426
Stream I/O Routines ................................................................................................................427
String-Manipulation Routines ................................................................................................................428
Time Routines ...........................................................................................................................................430
Memory and Other Routines ..................................................................................................................431
Chapter 1. MakeAFP Formatter Functions

This chapter describes the functions for AFP provided by MakeAFP Formatter. To use these functions, you must obey certain structural rules which are very easy to understand, otherwise, MakeAFP Formatter 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 Formatter Function Calls to Build an AFP Document

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

1. Initialize a MakeAFP Formatter environment by calling the “Start” function. The “Start” session function must be called before using any other MakeAFP Formatter 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.
5. Put page group-level index tags if needed.
6. Call the “Open Page” function.
7. Specify the position of 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.
8. If needed, specify any changes to the attributes of the data by calling the “Set Font” and “Set Color” functions before placing the data.
9. Compose the page using the appropriate function call:
   - Place a line of ASCII/EBCDIC/DBCS/Unicode text string in left, right, or center alignment.
   - Place the fixed or dynamic text paragraph in left, right, center, and fully justify alignment.
   - Draws line or box at a fixed or dynamic position.
   - Place the barcode at a fixed or dynamic position.
   - Include external AFP Object, such as overlay or Page segment at a fixed or dynamic position.
   - Import a popular format of an image or embed an AFP image and graphic inline within an AFP document for high-speed dynamic image printing.
   - Plot the graphic box, line, color area, barchart, or piechart.
10. Repeat steps 7 through 9 until the page is finished.
11. Call the “Close Page” function.
12. Call the “End Page Group Index” if needed.
13. Repeat steps 4 through 12 until all pages are done.

Hierarchy of MakeAFP Formatter Calls

MakeAFP Formatter has three levels of function calls:

- **Session Level Calls**
  These calls start the MakeAFP Formatter 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 Formatter 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.

**End 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 Page
Opens an AFP page.

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

Begin Draw Box
Begins a dynamic size box.

End Box
Ends a box at the current location.

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

Import Graphic or Image Object
Imports the AFP image or graphic objects, data-object images (such as TIFF/JPEG/GIF, etc), then embeds them inline in the AFP document data stream.

Include Page Segment, Overlay, and Data-object
Includes the AFP overlays, page segments, and data-object images.

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

Put Fixed Paragraph
Puts a boxed text paragraph.

Begin Paragraph
Begins a text paragraph.

Put Text into the paragraph
Adds a string of text to the current paragraph.

End Paragraph
Ends the paragraph previously started with “Begin Paragraph”.

Measure Text String
Measures the width of the specified string in the given font and returns the width.

Draw Shade
Draws a fixed size shading.
Begin Shade
   Begins a dynamic size shading.

End Shade
   Ends shading at the current location.

Draw Horizontal Line
   Draws a fixed-length horizontal line.

Draw Vertical Line
   Draws a fixed-length vertical line.

Begin Line
   Begins drawing a dynamic length solid line in a horizontal or vertical direction.

End Line
   Ends the horizontal or vertical line previously started with "Begin 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.
Plot Graphic
Plots graphic line, box, patterned color area, circle, arc, ellipse, marker, etc.

Plot Barchart
Plots a graphic barchart.

Plot Piechart
Plots a graphic piechart.

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 Formatter function.
2D Aztec Barcode by Drawing

Function

Although MakeAFP Formatter 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 BC0CA object into other formats, like PDF, HTML, and XML, the same BC0CA objects may be printed in different dimensions on different vendors' IPDS printers.

This function generates Aztec 2D barcode in a 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

```c
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 as 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:

```c
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 Formatter supports the 2D barcodes defined in the latest IBM BOCOA standard for ultra-fast speed formatting with a very small AFP BOCOA data stream, most AFP viewers and transformers for AFP are not able to view or convert BOCOA object into other formats, like PDF, HTML, and XML, the same BOCOA 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

```c
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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEG0</td>
<td>The barcode image is not rotated.</td>
</tr>
<tr>
<td>DEG90</td>
<td>The barcode image is rotated 90 degrees clockwise</td>
</tr>
<tr>
<td>DEG180</td>
<td>The barcode image is rotated 180 degrees clockwise</td>
</tr>
<tr>
<td>DEG270</td>
<td>The barcode image is rotated 270 degrees clockwise</td>
</tr>
</tbody>
</table>

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:
```c
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:
```

![Sample QR Code Image]
2D MaxiCode Barcode by Drawing

Function

Although MakeAFP Formatter supports the 2D barcodes defined in the latest IBM BCODA standard for ultra-fast speed formatting with a very small AFP BCOCU data stream, most AFP viewers and transformers for AFP are not able to view or convert BCODA object into other formats, like PDF, HTML, and XML, the same BCODA 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

```c
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 left most element of the barcode symbol.

symbol_mode
Symbol mode, Valid mode values are:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Maximum Data Length for Capital Letters</th>
<th>Maximum Data Length for Numeric Digits</th>
<th>Number of Error Correction Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Structured Carrier Message for additional numeric postal code</td>
<td>84</td>
<td>126</td>
</tr>
<tr>
<td>3</td>
<td>Structured Carrier Message for additional alphanumeric postal code</td>
<td>84</td>
<td>126</td>
</tr>
<tr>
<td>4</td>
<td>Standard symbol (default value) for numeric and alphanumeric character</td>
<td>93</td>
<td>135</td>
</tr>
</tbody>
</table>
sequences (includes Standard Error Correction) | Full ECC, like MODE4 but with Enhanced Error Correction | 77 | 110 | 66
---|---|---|---|---
5 | 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:

<table>
<thead>
<tr>
<th>Characters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-9</td>
<td>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)</td>
</tr>
<tr>
<td>10-12</td>
<td>Three-digit country code according to ISO 3166</td>
</tr>
<tr>
<td>13-15</td>
<td>Three-digit service code. This depends on your parcel courier.</td>
</tr>
</tbody>
</table>

**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**
```c
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:
```

```c
```

- 11 -
2D MicroPDF417 Barcode by Drawing

Function

Although MakeAFP Formatter supports the 2D barcodes defined in the latest IBM BCOCAs standard for ultra-fast speed formatting with a very small AFP BCOCAs data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOCAs into other formats, like PDF, HTML, and XML, the same BCOCAs 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

```c
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.
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:

```c
char *data = "1234566787 MicroPDF417";
SetUnit(IN_U600);
OpenDoc();
  OpenPage(8.5,11);
    MDF417(data,1.2,1.5);
  ClosePage();
CloseDoc();
```

Print / display:

![Barcode Image]
2D PDF417 Barcode by Drawing

Function

Although MakeAFP Formatter supports the 2D barcodes defined in the latest IBM BCODCA standard for ultra-fast speed formatting with a very small AFP BCODCA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCODCA object into other formats, like PDF, HTML, and XML, the same BCODCA 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

```c
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:

```c
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:
```

---

![Sample PDF417 barcode image](image-url)
2D PDF417 Truncated Barcode by Drawing

Function

Although MakeAFP Formatter 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

```c
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 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. 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:

![Barcode Image]
2D QR Code Barcode by Drawing

Function

Although MakeAFP Formatter 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

```c
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,
);
```

```c
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:

- T0UTF8: Converts legacy encoding data to UTF-8 by MakeAFP Formatter, 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

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:

```c
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: 
```

![Sample QR Code](image-url)
Barchart by Shading

Function

Draws a barchart by legacy shading with its axis coordinates at the current presentation text position, and the current default font is used for its texts.

This function is developed for the old IPDS printer model which does not support GOCA graphics. It is strongly recommended that you use the Graphic Barchart function if your IPDS printer supports AFP GOCA graphics, with which MakeAFP provides an ultra-fast formatting speed with a much smaller AFP data stream, yet provides much better quality either in full color or in 256 grayscale emulation.

Syntax

```c
void Barchart(
    float      height,
    float      bar_width,
    float      gap_bar_set,
    uint       vertical_scale_unit,
    ushort    vertical_scales,
    ushort   number_bars,
    ushort   barsets,
    float*   values,
    char**     horizontal_labels,
    char**   legend_texts,
    encoding   encode = ASCII,
    bool         gridline = TRUE,
    color       ocaColor = TRUE,
    pattern   shading_pattern = LED
);
```

Parameters

**height**
The height of the barchart graphic.

**bar_width**
The width of each bar.

**gap_bar_set**
The gap of each bar set.

**vertical_scale_unit**
The unit size of each scale on the vertical axis.

**vertical_scales**
The number of scales on the vertical axis.

**number_bars**
The number of bars in each bar set.

**values**
The one dimension or two dimensions array of the data values.
horizontal_labels
The array of label texts for the horizontal axis. You can insert a new line control code \n or \x0a to split the text into multiple lines on output. This value is ignored if your number of bars value is 1.

Legend_texts
The array of texts for the legend.

code
The encoding of the output texts of barchart, the valid values are:

- ASCII: Text in ASCII, uses AFP ASCII font, this is the default value
- A2E: Converts text from ASCII to EBCDIC uses AFP EBCDIC font
- GBK: Text is GBK Chinese, uses AFP ASCII/GBK fonts
- GBK2DBCS: GBK will be converted to DBCS, uses AFP SBCS/DBCS fonts
- BIG5: Text is BIG5 Chinese, uses AFP ASCII/BIG5 fonts
- BIG52DBCS: BIG5 will be converted to DBCS, uses AFP SBCS/DBCS fonts
- SJIS: Text is SJIS Japanese, uses AFP ASCII/SJIS fonts
- SJIS2DBCS: SJIS will be converted to DBCS, uses AFP SBCS/DBCS fonts
- KSC: Text is KSC Korean, uses AFP ASCII/KSC fonts
- KSC2DBCS: KSC will be converted to DBCS, uses AFP SBCS/DBCS fonts
- DBCS: Text is SBCS/DBCS, uses AFP SBCS/DBCS fonts

gridline
Specifies whether gridlines are plotted on the barchart, default is TRUE.

ocaColor
Specifies whether shadings are drawn in AFP OCA monochrome colors predefined by MakeAFP, default is TRUE.

shading_pattern
You can define between three shading patterns depending on the printer type you are using. Valid values are LED (default value), SCREEN and STD.

Sample

```c
Float data1[7][3] = {{45, 55, 60},  // samples testing input data
{65, 35, 85}, {58, 95, 63}, {25, 35, 45},
{45, 25, 30}, {55, 65, 75}, {65, 85, 90}};
float data2[] = {45, 55, 60, 65, 35,  // samples testing input data
65, 35, 85, 95, 95};
char *lbls[] = {"Mon\nHoliday 1", "Tue\nHoliday 2", "Wed", "Thu", "Fri",
"Sat", "Sun"};
char *lbls2[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
char *legend[] = {"East", "West", "North"};

Start();                    // Start MakeAFP initiation
SetUnit(IN_U300);           // Set default units to in, 300 dpi
OpenDoc();
OpenPage(8.5,11);         // page size is 8.5" by 11"
LPI(8);                     // line space is 8 LPI
```
Pos(1, 2.5);  // Set position of the barchart

Font(1);  // select font 1 for barchart

Barchart(2,    // Max height of the barchart
    0.15,    // Width of each bar
    0.2,     // Gap between each bar sets
    10,      // unit size of v-axis scale
    10,      // number of v-axis scales
    3,       // number of bars per bar set
    7,       // number of bar set
    &data1[0][0],  // datAn array of input data
    lbls,    // label texts below H-axis
    legend);  // legend texts on right side

Pos(1, 5);  // Set position of the barchart

Barchart(2,    // Max height of the barchart
    0.2,    // Width of each bar
    0.2,    // Gap between each bar sets
    10,     // unit size of v-axis scale
    10,     // number of v-axis scales
    1,      // number of bars per bar set
    7,      // number of bar set
    data2,  // datAn array of input data
    lbls2,  // label texts below H-axis
    NULL);  // ignore if 1 bar per set

ClosePage();

CloseDoc();

Output:
**Barchart Element Label**

**Function**

Optional function for the shading barcharts, defines how to control the layout of the barchart element labels.

This function must be called before the Barchart() function.

**Syntax**

```c
void BarchartElement(
    position  element_label_position = NONE,
    char*     format_specification = "%0.1f",
    int       sbcs_fontid = DEFAULT
);
```

**Parameters**

- **element_label_position**
  Specifies where to place the bar element labels, Valid values are ABOVE (place labels on top of each bar), MIDDLE (place labels at the middle of each bar), and NONE (no element labels).

- **format_specification**
  Specifies the format-control string to format the bar element labels, refer to the MSDN library for more details about the format control string used in C standard function fprintf(). Default is "%0.1f", precision is 1.

- **sbcs_fontid**
  Specifies a 1-byte font id number to be used to present the bar element labels. Default is using the current ID of the font in 1-byte, UTF-8, or wild-char encoding.

**Sample**

```c
Float data1[7][3] = {{45, 55, 60},       // samples testing input data
                      {65, 35, 85}, {58, 95, 63}, {25, 35, 45},
                      {45, 25, 30}, {55, 65, 75}, {65, 85, 90}};
char *lbls[] = {"Mon;Holiday 1", "Tue\nHoliday 2", "Wed", "Thu", "Fri",
                "Sat", "Sun"};
char *legend[] = {"East", "West", "North"};
Start();                        // Start initiation, open definition file,
                               // AFP output file
SetUnit(IN_U600);              // Set default units to in, 600 dpi
OpenDoc();
OpenPage(8.5,11);              // page size is 8.5" by 11"
LPI(8);                        // line space is 8 LPI
Pos(1, 2.5);                   // Set position of the barchart
Font(1);                       // select font 1 for barchart
BarchartElement(ABOVE,         // place element label above bar
"%2.0f"   // format-specification of value
);

Barchart(2,              // Max height of the barchart
    0.15,           // Width of each bar
    0.2,            // Gap between each bar sets
    10,             // unit size of v-axis scale
    10,             // number of v-axis scales
    3,              // number of bars per bar set
    7,              // number of bar set
    &data1[0][0],   // dataAn array of input data
    lbls,           // label texts below H-axis
    legend,         // legend text
    ASCII,          // ASCII text
    FALSE);         // do not draw grid lines

ClosePage();

CloseDoc();

Output:

![Barchart Output](image-url)
Barchart Legend

Function

Optional function for the shading barcharts, defines how to position and format the legend labels of the chart.

This function must be called before the Barchart() function if you want to override the default settings of the legend.

Syntax

```c
void GBarchartLegend(
    float  x_rel_pos = DEFAULT,
    float  y_rel_pos  = DEFAULT,
    float      legend_box_width = DEFAULT,
    float      legend_box_height = DEFAULT,
    bool   vertical_layout = TRUE,
    fontid     fontID1 = DEFAULT,
    fontid     fontID2 = DEFAULT
);
```

Parameters

- **x_rel_pos**
  Specifies the X relative position of the highest label of the bar or stack chart legend, relative to the coordinate origin position of the bar or stack chart barchart. Default is using the X position auto-decided by the bar or stack chart function.

- **y_rel_pos**
  Specifies the Y relative position of the highest label of the barchart legend, relative to the coordinate origin position of the bar or stack chart. Default is using the Y position auto-decided by the bar or stack chart function.

- **legend_box_size**
  Specifies the box size of the legend label. Default is using the box size auto-decided by the bar or stack chart function.

- **vertical_layout**
  Specified whether placing the legend labels vertically (from top to bottom) or horizontally (from left to right). The default value is TRUE, vertical layout.

- **fontID1**
  Specifies a font ID number to be used to present the legend label texts in ASCII/EBCDIC/UTF-8/ wild-char encoding. Default is using the current ID of the font in 1-byte, UTF-8, or wild-char encoding.

- **fontID2**
  Specifies a font ID number to be used to present the legend label texts in DBCS-PC or DBCS-HOST encoding. Default is using the current ID of the 2-byte font.

Sample

```c
float data1[][3] = {{45, 55, 60},  // samples testing input data
    {65, 35, 85}, {58, 95, 63}, {25, 35, 45},
    {45, 25, 30}, {55, 65, 75}, {65, 85, 90}};
```
float data2[] = {45, 55, 60, 65, 35,  // samples testing input data
                65, 35, 85, 95, 95};
char *lbls[] = {"Mon;Holiday 1", "Tue;Holiday 2", "Wed", "Thu", "Fri", "Sat", "Sun"};
char *lbls2[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
char *legend[] = {"East", "West", "North"};

Start();                    // Start MakeAFP initiation
SetUnit(IN_U300);           // Set default units to in, 300 dpi
OpenDoc();

OpenPage(8.5,11);         // page size is 8.5" by 11"
LPI(8);                 // line space is 8 LPI
Pos(1, 2.5);            // Set position of the barchart
Font(1);                // select font 1 for barchart

GBarchartLegend(1.4,      // plus 2.5" to place legend X
                0.4,      // plus 0.4" to place legend Y
                0.1,      // legend box width is 0.1"
                0.1,      // legend box height is 0.1"
                FALSE);   // place legend horizontally

Barchart(2,             // Max height of the barchart
          0.15,         // Width of each bar
          0.2,          // Gap between each bar sets
          10,           // unit size of v-axis scale
          10,           // number of v-axis scales
          3,            // number of bars per bar set
          7,            // number of bar set
          &data1[0][0],  // datAn array of input data
          lbls,         // label texts below H-axis
          legend,       // legend texts
          ASCII,        // ASCII text
          FALSE);       // do not draw grid lines

ClosePage();
CloseDoc();

Output:
Barchart Shades

Function

Optional function for overriding the default shading percentage values predefined for the barchart by shading.

Syntax

void BarchartShades(
    int* shading_percent,
    int num_of_shades
);

Parameters

shading_percent
An array of values as the shading percentage, up to 10 values can be defined. Default shading values predefined by MakeAFP are {9, 18, 5, 14, 24, 11, 35, 55, 45, 65}.

num_of_shades
The number of shade values to be overridden.

Sample

Float data1[7][3] = {{45, 55, 60},   // samples testing input data
                     {65, 35, 85}, {58, 95, 63}, {25, 35, 45},
                     {45, 25, 30}, {55, 65, 75}, {65, 85, 90}};

char *lbs[] = {"Mon;Holiday 1", "Tue;Holiday 2", "Wed", "Thu", "Fri",
               "Sat", "Sun");

char *legend[] = { "East", "West", "North" };

shades[7] = {12,18,8,14,22,16,24};

Start();                    // Start MakeAFP initiation
SetUnit(IN_U300);           // Set default units to in, 300 dpi
BarchartShades(shades,7);
OpenDoc();
OpenPage(8.5,11);         // page size is 8.5" by 11"
LPI(8);                 // line space is 8 LPI
Pos(1, 2.5);            // Set position of the barchart
Font(1);                // select font 1 for barchart
Barchart(2,             // Max height of the barchart
    0.15,       // Width of each bar
    0.2,        // Gap between each bar sets
    10,         // unit size of v-axis scale
    10,         // number of v-axis scales
    3,          // number of bars per bar set
    7,          // number of bar set
    &data1[0][0],  // datAn array of input data
    lbs,        // label texts below H-axis
    legend);    // legend texts on right side
Barcode (Linear) by Drawing

Although MakeAFP Formatter supports the 1D barcodes defined in the latest IBM BCODA standard for ultra-fast speed formatting with a very small AFP BCODA data stream, most AFP viewers and transformers for AFP are not able to view or convert BCODA object into other formats, like PDF, HTML, and XML, the same BCODA 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

```c
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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDB2OF7</td>
<td>AIM USS-Codabar, Codabar 2-of-7</td>
</tr>
<tr>
<td>CODE11</td>
<td>Code 11</td>
</tr>
<tr>
<td>CODE128</td>
<td>CODE 128, A, B, and C auto-switching mode</td>
</tr>
<tr>
<td>CODE128B</td>
<td>CODE 128, Set B, for suppress mode C in favor of mode B</td>
</tr>
<tr>
<td>CODE32</td>
<td>Code 32, up to 8 digits</td>
</tr>
<tr>
<td>CODE39E</td>
<td>Code 39 (3 of 9) Extended (full text)</td>
</tr>
<tr>
<td>CODE93</td>
<td>Code 93</td>
</tr>
<tr>
<td>DL2OF5</td>
<td>Data Logical Code 2 of 5</td>
</tr>
<tr>
<td>DPIDENT</td>
<td>Deutsche Post Identcode, 11 digits</td>
</tr>
<tr>
<td>DPLEIT</td>
<td>Deutsche Post Leitcode, 13 digits</td>
</tr>
<tr>
<td>EAN128</td>
<td>EAN 128</td>
</tr>
</tbody>
</table>
data
Either ASCII or EBCDIC input data. Make sure the PRMODE parameter is specified in your MakeAFP Formatter 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 the 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:

```c
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:
```

![Barcode Image]
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 Formatter 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

```c
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 ocaColor = 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
CB2OF7 AIM USS-Codabar, Codabar 2-of-7
CODE128 CODE 128, AIM USS-128
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
RM43C Royal Mail 4 State
UPCA UPC/CGPC Version A
UPCE UPC/CGPC Version E
UPC2SUPP UPC - two digit supplemental
UPCSSUPP 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 \001 through \008
AIM USS-39, Code 39 (3 of 9) \01 and \02
AIM USS-Codabar, Codabar 2-of-7 \01 and \02
Code 128, AIM USS-128, UCC/EAN128 \02 through \05
Code 93 \00
EAN-8 (includes JAN-short) \00
EAN-13 (includes JAN-standard) \00
EAN two-digit supplemental \00 and \01
EAN five-digit supplemental \00 and \01
Industrial 2-of-5 \01 and \02
Interleaved 2-of-5, AIM USS-I 2/5 \01 and \02
Japan postal \00 and \01
Matrix 2-of-5 \01 and \02
Modified Plessey \01 through \09
POSTNET, PLANET \00 through \04
Royal mail \00 and \01
UPC/CGPC Version A \00
UPC/CGPC Version E \00
UPC – 2 digit supplemental \00 through \02
UPC – 5 digit supplemental \00 through \02
USPS 4-State OneCode \00 through \03

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 Formatter 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**

```c
char *data = "1234567890";
OpenPage(8.5,11);  // Barcode data variable
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

<table>
<thead>
<tr>
<th>Bar Code Type</th>
<th>Valid Characters</th>
<th>Valid Data Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia Post Bar Modifier ‘\X01’</td>
<td>0123456789</td>
<td>Symbology: 8 digits&lt;br&gt;BCOCA range: 8 digits</td>
</tr>
<tr>
<td>Australia Post Bar Modifier ‘\X02’</td>
<td>0123456789</td>
<td>Symbology: 8–16 digits&lt;br&gt;BCOCA range: 8–16 digits</td>
</tr>
<tr>
<td>Australia Post Bar Modifier ‘\X03’</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgijklmnopqrstuvwxyz&lt;br&gt; (space), # (number sign)</td>
<td>Symbology: 8–13 characters&lt;br&gt;BCOCA range: 8–13 characters</td>
</tr>
<tr>
<td>Australia Post Bar Modifier ‘\X04’</td>
<td>0123456789 for sorting code 0–3 for customer information</td>
<td>Symbology: 8–24 digits&lt;br&gt;BCOCA range: 8–24 digits</td>
</tr>
<tr>
<td>Australia Post Bar Modifier ‘\X05’</td>
<td>0123456789</td>
<td>Symbology: 8–23 digits&lt;br&gt;BCOCA range: 8–23 digits</td>
</tr>
<tr>
<td>Australia Post Bar Modifier ‘\X06’</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgijklmnopqrstuvwxyz&lt;br&gt; (space), # (number sign)</td>
<td>Symbology: 8–18 digits&lt;br&gt;BCOCA range: 8–18 digits</td>
</tr>
<tr>
<td>Australia Post Bar Modifier X07’</td>
<td>0123456789 for sorting code 0–3 for customer information</td>
<td>Symbology: 8–39 digits&lt;br&gt;BCOCA range: 8–39 digits</td>
</tr>
<tr>
<td>Australia Post Bar Modifier ‘\X08’</td>
<td>0123456789</td>
<td>Symbology: 8 digits&lt;br&gt;BCOCA range: 8 digits</td>
</tr>
<tr>
<td>Code 128, AIM USS-128</td>
<td>All characters defined in the Code 128 code page</td>
<td>Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>Code 39 (3-of-9 Code), AIM USS-39</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ-.$/+% (space) A total of 43 valid characters</td>
<td>Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
</tbody>
</table>
| Code 93                                 | 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ-.$/+% 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. |
<p>| EAN-8 (includes)                        | 0123456789                                                                      | 7 characters                                                                    |</p>
<table>
<thead>
<tr>
<th>Symbology</th>
<th>JAN-code</th>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN-short)</td>
<td>0123456789</td>
<td>12 characters</td>
</tr>
<tr>
<td>EAN-13 (includes JAN-standard)</td>
<td>0123456789</td>
<td>12 characters</td>
</tr>
<tr>
<td>EAN Two-digit Supplemental</td>
<td>0123456789</td>
<td>12 characters</td>
</tr>
<tr>
<td>EAN Five-digit Supplemental</td>
<td>0123456789</td>
<td>12 characters</td>
</tr>
<tr>
<td>Industrial 2-of-5</td>
<td>0123456789</td>
<td>Unlimited. BCOD range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>Interleaved 2-of-5, AIM USS-I 2/5</td>
<td>0123456789</td>
<td>Unlimited. BCOD range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>Japan Postal Bar Code (Modifier X'00')</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ - (hyphen)</td>
<td>7 or more. BCOD range: 7 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>Japan Postal 4-State Bar Code (Modifier \X01)</td>
<td>0123456789 CC1,CC2,CC3,CC4,CC5,CC6,CC7,CC8 - (hyphen), start, stop</td>
<td>No length checking done.</td>
</tr>
<tr>
<td>Matrix 2-of-5</td>
<td>0123456789</td>
<td>Unlimited. BCOD range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>MSI (modified Plessey code)</td>
<td>0123456789</td>
<td>13 characters for Modifier \X01’ 12 characters for Modifier \X02’ 11 characters for all other modifiers</td>
</tr>
<tr>
<td>POSTNET</td>
<td>0123456789</td>
<td>13 characters for Modifier \X02’ 11 characters for all other modifiers</td>
</tr>
<tr>
<td>Royal Mail (RM4SCC, modifier X'00')</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ</td>
<td>Symbology: unlimited. BCOD range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>Royal Mail (Dutch KIX variation, modifier \X01)</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ</td>
<td>Symbology: unlimited. BCOD range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>UPC/CGPC Version A</td>
<td>0123456789</td>
<td>11 characters</td>
</tr>
<tr>
<td>UPC/CGPC Version E</td>
<td>0123456789</td>
<td>10 characters</td>
</tr>
<tr>
<td>UPC Two-digit Supplemental (Periodicals)</td>
<td>0123456789</td>
<td>12 characters for Modifier \X02’</td>
</tr>
</tbody>
</table>
| 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 DataMatrix 2D Barcode

Function

Generates data in BCOCA DataMatrix 2D barcode format, you should be familiar with the DataMatrix 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 Formatter 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 DataMatrix 2D barcode requires appropriate printer microcode support.

Syntax

```c
void BDataMatrix(
    char*      data,
    float      x_pos,
    float      y_pos,
    ushort     numrows = 0,
    ushort     rowsize = 0,
    ushort     specfunc = USERDEF,
    bool       NoESC = TRUE,
    ushort     degree = DEGO,
    ushort     ocaColor = BLACK,
    ushort     cmr_id = 0,
    ushort     process_mode = AUDIT,
    ushort     seqcount = 0,
    ushort     seqind = 0,
    ushort     fileid1 = 1,
    ushort     fileid2 = 1
);
```

Parameters

data
The null-terminated up to 3116 characters of ASCII or EBCDIC input data string.

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

**numrows**
The desired number of rows in the generated bar code symbol, including the finder pattern. The number of rows must be an even number from 8 to 144, but not all numbers are allowed. The supported sizes for Data Matrix bar codes vary depending on the number of rows, row size, and module size. See IBM Bar Code Object Content Architecture Reference for details.

Specify default value 0 as the number of rows to have the printer generate an appropriate row size based on the amount of symbol data.
rowsize
The number of modules in each row including the finder pattern. There must be an even number of modules per row and an even number of rows. There are square symbols with sizes from 10x10 to 144x144 and rectangular sizes from 8x18 to 16x48, not including quiet zones.

Specify default value 0 as the number of rows to have the printer generate an appropriate number of rows based on the amount of symbol data.

specfunc
This parameter is used to request special functions that can be used with a Data Matrix symbol. Valid values are:

- **FNC1UCC** UCC/EAN1 alternate data type identifier. A FNC1 is added in the first data position (or fifth position of a structured append symbol) to indicate that this bar code symbol conforms to the UCC/EAN application identifier standard format.
- **FNC1IND** Industry FNC1 alternate data type identifier. An FNC1 is added in the second data position (or sixth data position of a structured append symbol) to indicate that this bar code symbol conforms to a particular industry-standard format.
- **MAC5** This provides instructions to the bar code reader to insert an industry-specific header and trailer around the symbol data. the bar code symbol contains a 05 Macro codeword. The barcode symbol cannot be part of a structured append sequence.
- **MAC6** Same as MAC5 except the bar code symbol contains a 06 Macro codeword. The barcode symbol cannot be part of a structured append sequence.
- **RDRPROG** Use this when the symbol contains a message used to program the barcode reader. In this case, the barcode symbol cannot be part of a structured append sequence.
- **USERDEF** None of the above. This is a user-defined data symbol with no header or trailer instructions to the reader. This is the default value.

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 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
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 Formatter 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:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT</td>
<td>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).</td>
</tr>
<tr>
<td>INSTR</td>
<td>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.</td>
</tr>
</tbody>
</table>

The instruction 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).

seqcount and seqind
Multiple Data Matrix 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 16 Data Matrix 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). The Sequence Indicator must be 1 if the Sequence Count is 1.

If default 0 is specified, this symbol is not part of a structured append.
**fileid1 and fileid2**

These two parameters specify the high and low order bytes of a unique two-byte file identification for a set of structured append symbols. This helps to ensure that the symbols from two different structured appends are not linked together. You should specify the same File ID bytes for any symbols that are to be linked together. These parameters are ignored if the Sequence Count is 0 or 1.

**Sample**

```c
char *data = "1234567890";

OpenPage(8.5,11);

BDataMatrix(data,       // BCOCA barcode data
            1,         // Barcode x position to 1"  
            1);        // Barcode y position to 1"

ClosePage();
```
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 Formatter 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

```c
void BMaxiCode(
    char*    data,
    float    x_pos,
    float    y_pos,
    ushort  symbol_mode = MODE4,
    bool     zipper = FALSE,
    bool     NoESC = TRUE,
    ushort   degree = DEG0,
    ushort   ocaColor = 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 bar code. 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 Formatter 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**

```c
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 Formatter 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

```c
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        ocaColor = 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 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. 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

**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 Formatter 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**

```c
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();
```

- 47 -
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

```c
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 Formatter 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, the 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 recomposed 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**

```c
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 Box

Function

Begins a box of variable height with a specified width at the position you specified, the box is not drawn until the corresponding "End Box" function call is issued. The box must be ended before you end the page. You must ensure that the box you have specified fits on the page.

Syntax

```c
void BgnBox(
    float   x_pos,
    float   y_pos,
    float   width,
    float   thick,
    ushort  boxno = 1,
);
```

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.

- **width**
The width of the box.

- **thick**
The thickness of the box's lines.

- **boxno**
The ID number of the dynamic size box previously started, valid values are 1 through 24, the default value is 1.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
BgnBox(0.5, 4, 7.5, 0.02);     // Draw box from (0.5",4"),
                               // box width = 7.5", box lines
                               // thickness = 0.02", color is black
                               // by default value
Ypos(10);                    // Now Y position is at 10"
EndBox();                    // End box, so its height now is
                             // 10 - 4 = 6"
ClosePage();
CloseDoc();
```
Begin Horizontal Line

Function

Begin a horizontal line of variable length at the position you specified, the line is not drawn until the corresponding “End Horizontal Line” function is called. The line must be ended before you end the page. You must ensure that the line you specified fits on the page.

Syntax

```c
void BgnHline(
    float  x_pos,
    float  y_pos,
    float  line_thickness,
    ushort Hlineno = 1,
);
```

Parameters

- **x_pos**
The X starting position of the horizontal line.
- **y_pos**
The Y starting position of the horizontal line.
- **line_thickness**
The thickness of the horizontal line.
- **hlineno**
The ID number of the begin horizontal line to be started, Valid values are 1 through 24, the default value is 1.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Color(BLUE); // defines color for texts and legacy line
BgnHline(1,1,0.02); // begin 1st hline at (1",1"), its
                     // thickness is 0.02"
BgnHline(1,2,0.02,1); // begin 2nd hline at (1",2"), its
                      // thickness is 0.02"
:
:
Xpos(7.5); // now X position is at 7.5"
EndHline(); // End 1st hline, its length is 6.5
EndHline(2); // End 2nd hline, its length is 6.5
:
:
ClosePage();
CloseDoc();
```
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

```c
void BgnIdx(
    char*   groupname,
    ushort   docNo = 1
);
```

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, MakeAFP Formatter puts its characters string “as is” without any conversion, you may need to call one of 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 in ASCII. Make sure the CPGID parameter is defined properly in your MakeAFP definition file.

docNo

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

Sample

```c
// Now all input data of a client are formatted into the AFP page
// buffers, before write out all of the pages of a client, we can insert
// beginning of group index tag and index value tags
unsigned short numpages = PageNum;        // Keep total pages of a client
char tmp[35];
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
{
    PageNum = i + 1;                        // Switch to each page buffer
    sprintf(tmp, "Page %d of %d", PageNum, numpages);
    Pos(8.0,3.93);                        // Set position to (8", 3.93")
    Rtxt(tmp);                            // Right alignment of page number
    ClosePage();                           // on each page before close of each
                                            // page
```
EndIdx(); // End index page group, must be called after writing of the last page of each page group
Begin Index Group With Encoding Conversion

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

```c
void BgnIdx2(
    char*    groupname,
    ushort   docNo = 1
);
```

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. Make sure the CPGID parameter is defined properly in your MakeAFP definition file, and default input data encoding is defined properly by the functions of DefaultCode(), so that MakeAFP Formatter auto-converts your data encoding correctly.

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

Sample

```c
// Now all input data of a client are formatted into the AFP page
// buffers, before write out all of the pages of a client, we can insert
// beginning of group index tag and index value tags
unsigned short numpages = PageNum;       // Keep total pages of a client
char tmp[35];
sprintf(tmp, "%08d", ++groups);
BgnIdx2(tmp);                             // Begin index page group
PutIdx2("Customer Name", client_name);   // Put group-level index tags,
PutIdx2("Account Number", account_no);  // BgnIdx and PutIdx must be called
                                          // before writing of the first page
for (int i = 0; i < numpages; i++)       // of each page group
{
    PageNum = i + 1;                     // Switch to each page buffer
    sprintf(tmp, "Page %d of %d", PageNum, numpages);
    Pos(8.0, 3.93);                      // Set position to (8", 3.93")
    Rtxt(tmp);                           // Right alignment of page number
    // on each page before close of each
    ClosePage();                         // page
}                                        //
EndIdx();                          // End index page group, must be
called after writing of the last
page of each page group
Begin Paragraph of 1-Byte Text

Function

Begins a paragraph of the 1-byte text and specifies its alignment.

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 Formatter 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

```c
void BgnParTxt(
    float paragraph_width,
    alignment text_alignment
);
```

Parameters

- **paragraph_width**
  The width of the text paragraph.

- **text_alignment**
  Specifies how the texts in the 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
  ```

Sample

```c
char *msg1 = "Congratulations!";
char *msg2 = "You got a perfect AFP printing solution.";
SetUnit(MM_U600);
OpenDoc();
DefaultCode("ibm-437"); // Data from codepage 437, USAN ASCII
DefaultLocale("en_US"); // language locale is USA English
OpenPage(210,297);
Pos(20, 50);
LPI(4); // Set line spacing to 4 LPI
```
BgnParTxt(30, LEFT);  // Begin a paragraph, 30 mm width,
// left-aligned
Font(1);              // Font 1 is ASCII font
PutParTxt(msg1, PINK); // Put 1st ASCII text in pink color
Font(2);              // Font 2 is ASCII font
PutParTxt(msg2, BLUE); // Put 2nd text in blue color

EndParTxt();          // End paragraph and write into AFP output

ClosePage();
CloseDoc();
Begin Paragraph of DBCS-HOST

Function

Begins a paragraph of the SBCS-HOST/DBCS-HOST text and specifies its alignment.

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 EBCDIC encoded font, and the second one must be a DBCS-HOSt encoded font.

Make sure DefaultCode( ) and DefaultLocale( ) are defined properly according to your input data encoding and language locale, for example, with Japanese SBCS-HOST/DBCS data, you need to specify DefaultCode(“ibm-1390”) and DefaultCode(“ja_JP”).

Syntax

```c
void BgnParDbcs(
    float    paragraph_width,
    alignment text_alignment
);
```

Parameters

**paragraph_width**
The width of the SBCS-HOST/DBCS-HOST paragraph.

**text_alignment**
Specifies how the Japanese texts in the paragraph should be formatted. The valid values are:

- LEFT: DBCS-HOST texts are left-aligned
- RIGHT: DBCS-HOST Japanese texts are right-aligned
- CENTER: DBCS-HOST Japanese texts are center-aligned
- JUSTIFY: DBCS-HOST Japanese texts are justify-aligned

Sample

```c
char *msg1 = "ひらがな、カタカナ、漢字、数字の 123、";
char *msg2 = "Alphabet が混在した文章のサンプルです。";
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParDbcs(110, LEFT);   // Begin a Japanese paragraph, 110 mm width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is SJIS font
PutParDbcs(msg1, PINK);  // Put 1st Japanese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is SJIS font
PutParDbcs(msg2, BLUE);  // Put 2nd Japanese text in blue color
EndParDbcs(); // End Japanese paragraph and write into AFP output
ClosePage();
CloseDoc();
```

- 60 -
Begin Paragraph of Japanese

Function

Begins a paragraph of the Japanese ASCII/SJIS-PC text and specifies its alignment.

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 encoded font, and the second one must be an SJIS-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void BgnParJp(
    float paragraph_width,  // The width of the Japanese paragraph.
    alignment text_alignment);  // Specifies how the Japanese texts in the paragraph should be formatted. The valid values are:

    LEFT Japanese texts are left-aligned
    RIGHT Japanese texts are right-aligned
    CENTER Japanese texts are center-aligned
    JUSTIFY Japanese texts are justify-aligned

Sample
```

```c
char *msg1 = "ひらがな、カタカナ、漢字、数字の 123、"  // Set line spacing to 4 LPI
char *msg2 = "Alphabet が混在した文章のサンプルです。
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20,50);
LPI(4);
BgnParJp(110, LEFT);  // Begin a Japanese paragraph, 110 mm
// width, left-aligned
Font(1,2);  // Font 1 is ASCII font, font 2 is SJIS font
PutParJp(msg1, PINK);  // Put 1st Japanese text in pink color
Font(3,4);  // Font 3 is ASCII font, font 4 is SJIS font
PutParJp(msg2, BLUE);  // Put 2nd Japanese text in blue color
EndParJp();  // End Japanese paragraph and write into AFP output
ClosePage();
CloseDoc();
```
Begin Paragraph of Korean

Function

Begins a paragraph of the Korean ASCII/KSC-PC text and specifies its alignment.

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 encoded font, and the second one must be a KSC-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void BgnParKr(
    float        paragraph_width,
    alignment     text_alignment
);
```

Parameters

- **paragraph_width**
The width of the Korean paragraph.

- **text_alignment**
Specifies how the Korean texts in the paragraph should be formatted. The valid values are:

  - LEFT: Korean texts are left-aligned
  - RIGHT: Korean texts are right-aligned
  - CENTER: Korean texts are center-aligned
  - JUSTIFY: Korean texts are justify-aligned

Sample

```c
char *msg1 = "'함께해요~' 이벤트, 온 가족의 티셔츠가 내 품에!";
char *msg2 = "내 책 사면 <어린이 도서관>에 기증될 책이 하나 더!!";
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParKr(110, LEFT);  // Begin a Korean paragraph, 110 mm
                      // width, left-aligned
Font(1,2);           // Font 1 is ASCII font, font 2 is KSC font
PutParKr(msg1, PINK); // Put 1st Korean text in pink color
Font(3,4);           // Font 3 is ASCII font, font 4 is KSC font
PutParKr(msg2, BLUE); // Put 2nd Korean text in blue color
EndParKr();         // End Korean paragraph and write into AFP output
ClosePage();
CloseDoc();
```
Begin Paragraph of Simplified Chinese

Function

Begins a variable paragraph of the Simplified Chinese ASCII/GBK-PC text and specifies its alignment.

You can “Lines Per Inch” or “Line Spacing” functions 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 encoded font, and the second one must be a GBK-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode( ) is defined properly according to your input data encoding.

Syntax

```c
void BgnParSc(
      float   paragraph_width,
      alignment  text_alignment
);
```

Parameters

- **paragraph_width**
  The width of the Chinese paragraph.

- **text_alignment**
  Specifies how the Chinese texts in the paragraph should be formatted. The valid values are:

  - LEFT: Chinese texts are left-aligned
  - RIGHT: Chinese texts are right-aligned
  - CENTER: Chinese texts are center-aligned
  - JUSTIFY: Chinese texts are justify-aligned

Sample

```c
char *msg1 = "它允许用户为 XML 数据指定格式,";
char *msg2 = "从而缩短了第一页开始打印的时间.";

Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParSc(110, LEFT);   // Begin a Simplified Chinese paragraph, 110 mm
                      // width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is GBK font
PutParSc(msg1, PINK); // Put 1st Chinese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is GBK font
PutParSc(msg2, BLUE); // Put 2nd Chinese text in blue color
EndParSc();  // End Simplified Chinese paragraph and write
              // into AFP output
```
Begin Paragraph of Traditional Chinese

Function

Begins a paragraph of the Traditional Chinese ASCII/BIG5-PC text and specifies its alignment.

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 encoded font, and the second one must be a BIG5-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void BgnParTc(
    float paragraph_width,
    alignment text_alignment
);
```

Parameters

- `paragraph_width`
The width of the Chinese paragraph.
- `text_alignment`
  Specifies how the Chinese texts in the paragraph should be formatted. The valid values are:

  - **LEFT**: Chinese texts are left-aligned
  - **RIGHT**: Chinese texts are right-aligned
  - **CENTER**: Chinese texts are center-aligned
  - **JUSTIFY**: Chinese texts are justify-aligned

Sample

```c
char *msg1 = "它允许用户为 XML 数据指定格式,";
char *msg2 = "从而缩短了第一頁开始打印的时间.
```

```
P pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParTc(110, LEFT);   // Begin a Traditional Chinese paragraph, 110 mm
                      // width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is BIG5 font
PutParTc(msg1, PINK); // Put 1st Chinese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is BIG5 font
PutParTc(msg2, BLUE); // Put 2nd Chinese text in blue color
EndParTc();   // End Traditional Chinese paragraph and write
              // into AFP output
```
Begin Paragraph of Wild-Char

Function

Begins a paragraph of the wild-char text and specifies its alignment.

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 the data type UTF16BE by the FONT parameter in your MakeAFP definition file. Refer to Chapter 2 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

```c
void BgnParW(
    float  paragraph_width,
    alignment  text_alignment
);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paragraph_width</td>
<td>The width of the text paragraph.</td>
</tr>
<tr>
<td>text_alignment</td>
<td>Specifies how the texts in the paragraph should be formatted. The valid values are:</td>
</tr>
</tbody>
</table>

- **LEFT**  Texts are left-aligned
- **RIGHT** Texts are right-aligned
- **CENTER** Texts are center-aligned
- **JUSTIFY** Texts are justify-aligned

Sample

```c
wchar_t *msg1 = L"Congratulations!";
wchar_t *msg2 = L"You got a perfect AFP printing solution.";

DefaultCode("ibm-437");    // Data from codepage 437, USAN ASCII
DefaultLocale("en_US");    // language locale is USA English
LPI(4);  // Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);

BgnParW(110, LEFT);         // Begin a variable paragraph, 110 mm width, left-aligned
PutParW(msg1, BLUE);  // Put 1st wild-char text in blue color
PutParW(msg2, RED);  // Put 2nd wild-char text in red color
EndParW();              // End wild-char paragraph and write into AFP
```
Begin Paragraph of UTF-8

Function

Begins a paragraph of the UTF-8 text and specifies its alignment.

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 the data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to Chapter 2 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

```c
void BgnParU8(
    float   paragraph_width,
    alignment   text_alignment
);
```

Parameters

- **paragraph_width**
  The width of the text paragraph.

- **text_alignment**
  Specifies how the texts in the 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

Sample

```c
char *msg1 = "Congratulations!";
char *msg2 = "You got a perfect AFP printing solution.";

DefaultCode("ibm-437");          // Data from codepage 437, USAN ASCII
DefaultLocale("en_US");         // language locale is USA English

LPI(4);                           // Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);
BgnParU8(110, LEFT);             // Begin a variable paragraph, 110 mm
                                // width, left-aligned
PutParU8(msg1, BLUE);           // Put 1st UTF-8 text in blue color
PutParU8(msg2, RED);            // Put 2nd UTF-8 text in red color
EndParU8();                     // End UTF-8 paragraph and write into AFP
```
**Begin Shade**

**Function**

Begins a shaded area of variable height with a specified width from the location specified. The depth is decided by the vertical position when issuing the “End Shade” call. Shading is not started until the corresponding “End Shade” call is issued. You must ensure that the Shade fits on the page.

**Syntax**

```cpp
void BgnShade(
    float    x_pos,
    float    y_pos,
    float    shade_width,
    ushort   shade_percent,
    ushort   shadeno = 1,
    pattern  shading_pattern = LED,
    ocaColor color = BLACK
);
```

**Parameters**

- **x_pos**
  The X position of the top left corner of the shade.

- **y_pos**
  The Y position of the top left corner of the shade.

- **shade_width**
  The width of the shaded area.

- **shade_percent**
  The percentage of shading to be applied to the shaded area. The valid values are integers 0 to 100.

- **shadeno**
  The ID number of the “Begin Shade” session to be started. Valid values are 1 through 24, the default value is 1.

- **shading_pattern**
  You can define between three shading patterns depending on the printer type you are using. Valid values are LED (default value), SCREEN and STD.

- **ocaColor**
  Any of the defined AFP OCA color values: BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY, the default value is BLACK.

**Sample**

```cpp
OpenPage(8.5,11);

BgnShade(1,1,2,5,0)  // begin 1st shade at (1",1"), width 2", shading percentage 5%
```
BgnShade(5,1,3,10,1)  // begin 2nd shade at (5",1"), width 10"
                // shading 10 percent
 Ypos(9.5);      // now Y position is at 7.5"
EndShade();     // End shade no. 0, shade depth is 8.5"
EndShade(1)     // End shade no. 1, shade depth is 8.5"
ClosePage();
Begin Suppress

Function

Marks the beginning of a string of presentation text that may be suppressed from the visible output. It must be ended with the “End Suppress” function. Nesting of suppression control sequences is not allowed.

You can mark the suppression of a string of presentation text with the “Begin Suppress” and “End Suppress” functions, and turn the suppression on or off by calling of the different copy groups defined within an AFP form definition. A copy group is a portion of a form definition that defines a set of modifications that can be used when presenting a page. Modifications can include text suppression, overlays, and so forth. For certain types of documents, it is possible to print/view a page in multiple ways. For example, copy group one could be used to present a replica of a bill sent to the customer while copy group two could be used to present additional information attached to the bill, but not sent to the customer. With the AFP print server and the “Application Logical” views options of IBM DB2 Content Manager OnDemand, you can fully control the calling of copy groups defined within your form definition.

Syntax

```c
void BgnSuppress(
    int suppression_id,
);
```

Parameters

- **suppression_id**
  Specifies the identifier of suppression that is defined within a form definition.

Sample

**PPFA sample:**

```plaintext
FORMDEF PAYMNT
OFFSET 0.0 0.0 PRESENT PORTRAIT REPLACE YES;

SUPPRESSION SALARY;  /* first one, whose suppression id is 1 */
SUPPRESSION BONUS;     /* second one, whose suppression id is 2 */

COPYGROUP PAYMNT;
    SUBGROUP;
COPYGROUP SUPP1
    SUBGROUP SUPPRESSION SALARY;
COPYGROUP SUPP2
    SUBGROUP SUPPRESSION BONUS;
```

**MakeAFP Formatter sample:**

```plaintext
OpenPage(8.5,11);

BgnSuppress(1);
    Pos(1.5,2.5);
    Ltxt("This text string can be suppressed by copygroup SUPP1");
EndSuppress(1);

BgnSuppress(2);
```
Ltxt("This text string can be suppressed by copygroup SUPP2");
Ltxt("This text string can also be suppressed by copygroup SUPP2");
EndSuppress(2);:
ClosePage();
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();
Begin Vertical Line

Function

Begins a vertical line of variable length at the position you specified, the line is not drawn until the corresponding “End vertical Line” call is issued. The line must be ended before you end the page. You must ensure that the line you have specified fits on the page.

Syntax

```c
void BgnVline(
    float    x_pos,
    float    y_pos,
    float    line_thickness,
    ushort   vlineno = 1,
);
```

Parameters

- **x_pos**
The X starting position of the vertical line.

- **y_pos**
The Y starting position of the vertical line.

- **line_thickness**
The thickness of the vertical line.

- **vlineno**
The ID number of the “Begin Vertical Line” session to be started, Valid values are 1 through 24, the default value is 1.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Color(BLUE); // defines color for texts and legacy line
BgnVline(1,1,0.02); // begin 1st vline at (1",1"),its // thickness is 0.02"
BgnVline(5,1,0.02,1); // begin 2nd vline at (5",1"),its // thickness is 0.02"
:
:
Ypos(9.5); // now Y position is at 9.5"
EndVline(); // End 1st vline, its length is 8.5
EndVline(2); // End 2nd vline, its length is 8.5
:
:
ClosePage();
CloseDoc();
```
Box Drawing (Fixed Size)

Function

Draws a box at the specified position using the specified line thickness. Ensure that the box you have specified fits on the page. Do not issue this call within a paragraph.

Syntax

```c
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

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

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

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 Single-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 Formatter 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

```c
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 to which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
Font(3);  // assume font 3 is ASCII font
:
Pos(2,2);  // current position to (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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

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

Parameters

data
The NULL-terminated SJIS-PC 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 to which the next character would be placed.

Sample

```c
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 to (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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
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 to which the next character would be placed.

Sample

```c
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 to (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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
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 to which the next character would be placed.

Sample

```c
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 to (2",2")
Csc("实现 Win2000 与 Linux 的双引导"); // Center GBK text at (2",2") // ASCII/GBK text will be // converted into SBCS-HOST/ // DBCS-HOST encoding in AFP

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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
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 to which the next character would be placed.

Sample

```c
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 to (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 Text

Function

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

You need to define a pair of fonts with the “Font2” function, the first parameter must be an EBCDIC encoded font, and the second one must be a DBCS-HOST encoded 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 to 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 to (2",2")
Cdbcs("实现 Win2000 与 Linux 的双引导"); // Center DBCS text at (2",2")
:
:
ClosePage();
CloseDoc();
Center Align Wild-Char Text

Function

Center aligns a single-line of the wild-char string at the current position. Native wild-char string on Windows is in little-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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Cw(
    wchar_t* data,
    bool      same_pos = TRUE
);
```

Parameters

data  
The NULL-terminated wild-char 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

```c
/* wild-char string, "test" and CJK characters "测试" */
wchar_t    data[20] = {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 UTF16BE defined
Cw(data);                        // center put wild-char at (2",2")
:
:
ClosePage();
CloseDoc();
```
Center Align Wild-Char Text Converting from Legacy String

Function

Center aligns a single-line of the wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Cwc(
    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 wild-char. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to Appendix D. for more details about the available encoding names and alias.

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

```c
SetUnit(IN_U600);
OpenDoc();
DefaultCode("GB18030");        // set default codepage of input data
OpenPage(8.5,11);
    :
Pos(2,2);                     // set current position to (2",2")
Font(2);                      // Assume font 2 is a TrueType font
    // with data type UTF16BE defined
Cwc("test 测试");           // center put wild-char converting from
    // Chinese GB18030
    :
```
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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
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 to which the next character would be placed.

Sample

```c
/* UTF-8 string, "test" and CJK characters "测试" */
UChar8     data1[20] = "test\xe6\xb5\x8b\xe8\xa8\x95";

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
:

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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
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 Appendix D. for more details about the available encoding names and alias.

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

```c
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
 Cu8c("test 测试","GB18030"); // center put UTF-8 converting from 
    // Chinese GB18030
    :
    :
ClosePage();
CloseDoc();
```
Center Align UTF-8 Text Converting from Wild-Char

Function

Center aligns a single-line of the UTF-8 string converting from the wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Cu8w(
    wchar_t*     data,
    bool     same_pos = TRUE
);
```

Parameters

data
The NULL-terminated wild-char 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

```c
/* wild-char string, "test" and CJK characters "测试" */
wchar_t    data1[] = {0x0074, 0x0065, 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
    Cu8w(data); // center put UTF-8 converting from
                 // wild-char
    ClosePage();
CloseDoc();
```
Centimeter Value

**Function**

Specifies a value in centimeters.

**Syntax**

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

**Parameters**

- **value**
  The value in centimeters.

**Sample**

```c
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();
```
Character Rotation

Function
Sets the character’s rotation relative to the inline direction of the text line.

Syntax

```c
void CharRotate(
    ushort degree
);
```

Parameters

degree
The character rotation. The valid value is 0°, 90°, 180°, or 270.

The following figure illustrates changes in character rotation with different text orientations:

```
<table>
<thead>
<tr>
<th>Inline Direction</th>
<th>Baseline Direction</th>
<th>0</th>
<th>90</th>
<th>180</th>
<th>270</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90 or 270</td>
<td>ABC</td>
<td>➔</td>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>90</td>
<td>180 or 0</td>
<td>ABC</td>
<td>➔</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>180</td>
<td>270 or 90</td>
<td>ABC</td>
<td>➔</td>
<td>C</td>
<td>➔</td>
</tr>
<tr>
<td>270</td>
<td>0 or 180</td>
<td>ABC</td>
<td>➔</td>
<td>➔</td>
<td>➔</td>
</tr>
</tbody>
</table>
```

Sample

Font(1);

CharRotate(270);

TextOrient(I90B180);

Pos(1, 1);

Lu16c("基本月租费");

Output: 基本月租费
Character Space

Function

Sets the inter-character spacing.

Syntax

```cpp
void CharSpace(
    float intercharacter_spacing = 0
);
```

Parameters

`intercharacter_spacing`  
The amount of extra space inserted between characters. Default value 0 indicates that the inter-character spacing is determined by the font without any extra space inserted.

Sample

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

Pos(5,5); // current position to (5,5) mm
CharSpace(2); // 2 MM extra space between chars.
Ctxt("Extra Intercharacter spacing");
CharSpace(); // character space back to default

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.

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);
BgnBox(0.5, 4, 7.5, 0.02);    // Draw box from (0.5",4"),
    // box width = 7.5", box lines
    // thickness = 0.02, color is black
    // by default value

Ypos(10);       // Now Y position is at 10"
EndBox();                      // End box, so its height now is
    // 10 - 4 = 6"
ClosePage();
CloseDoc();
Close Overlay

Function

Closes an AFP overlay previously opened with an “Open Overlay” call, once you have completed the design of your overlay, you must close it with the “Close Overlay” function.

The Open and Close Overlay functions are specially developed for you to use all the powerful formatting functions offered by MakeAFP to create your colorful graphic AFP overlay.

You can include AFP page segments or other overlays as the inline AFP resources for AFP viewing during your development just for your convenience by defining parameter “restype=pseg,ovly,inline” in your MakeAFP definition file, however, you must not include inline AFP resources within your overlay if you would like to generate it as the final AFP overlay to be released for production use or to be used by other MakeAFP formatting programs, make sure parameter “restype=none” is defined before generating your final release version of the AFP overlay.

Syntax

```c
void CloseOvly();
```

Parameters

No parameter to be specified.

Sample

```c
Start();

SetUnit(IN_U600);
OpenOvly(8.5,11); // Overlay size 8.5" x 11"

GLineWidth(5); // Set line width to about 0.05"
GBox(1.3, 1.8, 3, 1.2, 0.15); // Plot a rounded box

GColor(BLACK); // set OCA black color
GBgnFill(); // start filling of a box area
GBox(0.5, 4, 7.5, 0.26); // plot a box
GEndFill(); // end filling of box area
GColor(255,255,255); // set RGB white color
GVline(1.5, 4, 0.25); // plot vertical lines
GVline(9, 4, 0.25);
GVline(6, 4, 0.25);
GVline(7, 4, 0.25);

CloseOvly();
```
Close Page

Function

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

With MakeAFP, you can open multiple pages by the “Open 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 the MaxPaging variable or “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 Formatter, you can open multiple pages by the “Open Page” functions, and then process different pages in an interleaved manner once each page is initialized, all 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 the PageNum 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

```c
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

```c
void main(void)
{
    Start();
    SetUnit(CM_U300);
    OpenDoc();           // Open first AFP document
    OpenDoc(2)           // Open second AFP document
    ;
    PageNum = 3;         // indicate to open page 3
    OpenPage(21, 29.7);  // A4 paper size, 21cm x 29.7cm
}
ClosePage(2);    // Close AFP page, write to second
                 // AFP document file
CloseDoc(2);     // Close second AFP document
:
CloseDoc();      // Close first AFP document
}
Color for Text

Function

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

Syntax

For OCA color:

\[
\text{Color}(
\text{ocaColor} \quad \text{ocaColor} = \text{BLACK}
\);
\]

For RGB color:

\[
\text{ColorRGB}(
\text{UCHAR} \quad \text{red\_color},
\text{UCHAR} \quad \text{green\_color},
\text{UCHAR} \quad \text{blue\_color}
\);
\]

For CYMK color:

\[
\text{ColorCMYK}(
\text{UCHAR} \quad \text{cyan\_color\_percentage},
\text{UCHAR} \quad \text{magenta\_color\_percentage},
\text{UCHAR} \quad \text{yellow\_color\_percentage},
\text{UCHAR} \quad \text{black\_color\_percentage}
\);
\]

Parameters

\text{ocaColor}

Any of the defined AFP 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, default value is BLACK.

\text{RGB values}

Valid RGB intensity range values for each component are 0 through 255 (hex value x’FF’).

\text{CYMK color percentage values}

Valid CYMK percentage range values for each component are 0 through 100 (hex value x’64’).

Sample

\begin{verbatim}
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(5,5); // current position to (5,5) mm
\text{Color}(255,0,0); // RGB red color
Ltxt("RGB Red Color Text");
Pos(5,10.);
\text{Color}(0,0,0,100); // CYMK black color
Ltxt("CYMK Black Color Text");
\end{verbatim}
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 Formatter.

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, required by the AFP system to process a color 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.

Refer to the latest Infoprint PPFA User’s Guide and Infprint Manager Procedures books for more about AFP color management.

Syntax

Invokes CMR Association:

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

Revolves CMR Association:

```c
void RevokeCMR( );
```

Parameters

**cmr_id**
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Formatter 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:

```c
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

```plaintext
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();
```
Comment

Function

Puts up to 250 characters of a comment in the AFP data stream for use with postprocessors or other applications, such as IBM ACIF Indexing for DB2 Content Manager OnDemand and Infoprint Manager.

Syntax

```c
void Comment(
    char*   comment_text
);
```

Parameters

**comment_text**
The string of up to 250 characters to be placed in the AFP output as a comment.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Comment("This is my comment string");
:
:
ClosePage();
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

```c
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

```c
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 Input Data Encoding

Function

Defines the default encoding names for your input data.

Make sure you have defined a correct encoding name once at the global level before calling an AFP text function that requires internal conversion from ASCII/DBCS-PC to EBCDIC, Unicode.

Syntax

```c
void DefaultCode(
    char       *codename = "windows-1252"
);```

Parameters

codeName
The name of the default encoding, default is “windows-1252”. Refer to Appendix D. for more details about the available encoding names and alias.
Default Language Locale

Function

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

Make sure you have defined a correct locale name once at the global level before calling of paragraph functions.

Syntax

```c
void     DefaultLocale(
            char        *localeName = "en_US"

);
```

Parameters

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

Refer to Appendix E. for more details about how to specify a locale name.
DBCS Width

Function

Measures and then returns the width of mixed SBCS-HOST/DBCS-HOST string in the given fonts. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” functions before you call this function.

Syntax

```c
float DbcsWidth(
    char* host_data,
    ushort font1 = DEFAULT,
    ushort font2 = DEFAULT
);
```

Parameters

- **host_data**
  The string of mixed SBCS-HOST/DBCS-HOST characters to measure.

- **font1**
  The ID number of the SBCS-HOST font you defined in your MakeAFP definition file. Default is using your current SBCS font ID.

- **font2**
  The ID number of the DBCS-HOST font you defined in your MakeAFP definition file. Default is using your current DBCS font ID.

Sample

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

Font((4,5); // font 4 is 1-byte font
     // font 5 is 2-byte font

float w = DbcsWidth("实现 Win2000 与 Linux 的双引导");

ClosePage();
CloseDoc();
```
End Box

Function

Ends a box of variable height at the current position. The box must be previously started with a “Begin Box” call and must be ended before you end the page.

Syntax

```c
void EndBox(
    int boxno = 1
);
```

Parameters

- **boxno**
  The ID number of the box started with the “Begin Box” call, valid values are 1 through 24, the default value is 1.

Sample

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

BgnBox(0.5, 4, 7.5, 0.02);      // Draw box from (0.5", 4"),
                                // box width = 7.5", box lines
                                // thickness = 0.02, color is black
                                // by default value

Ypos(10);                       // Now Y position is at 10"
EndBox();                       // End box, so its height now is
                                // 10 - 4 = 6"

ClosePage();
CloseDoc();
```
End Horizontal Line

Function

Ends a horizontal line of variable length at the current position. The line must be previously started with a “Begin Horizontal Line” call and must be ended before you end the page.

Syntax

```c
void EndHline(
    ushort  hlineno = 1
);
```

Parameters

- **hlineno**
  
The ID number of the horizontal line started with the “Begin Horizontal Line” call, valid values are 1 through 24, the default value is 1.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
BgnHline(1,1,0.02)     // begin 1st hline at (1",1"),its
                      // thickness is 0.02"
BgnHline(1,2,0.02,1)   // begin 2nd hline at (1",2"),its
                      // thickness is 0.02"
:
:
Xpos(7.5);             // now X position is at 7.5"
EndHline();            // End 1st hline, its length is 6.5
EndHline(2)            // End 2nd hline, its length is 6.5
:
:
ClosePage();
CloseDoc();
```
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

```c
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

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

unsigned short numpages = PageNum;        // Keep total pages of a client
char tmp[35];
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
{
    PageNum = i + 1;                     // Switch to each page buffer
    sprintf(tmp, "Page %d of %d", PageNum, numpages);
    Pos(8.0,3.93);                        // Set position to (8", 3.93")
    Rtxt(tmp);                            // Right alignment of page number
    // on each page before end of each page
    ClosePage();                          // page
}
EndIdx();                        // End index page group, must be called after writing of the last page of each page group

PageNum = 1;                      // Reset AFP page buffer number to 1 for the next customer statement
```
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 Paragraph of Single-Byte Text

Function

Ends a paragraph of ASCII or EBCDIC text previously started with the function “Begin Paragraph of Text”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

void EndParTxt();

Parameters

No parameter to be specified.

Sample

char *msg1 = "Congratulations!";
char *msg2 = "You got a perfect AFP printing solution."

SetUnit(MM_U600);
OpenDoc();

DefaultCode("ibm-437");  // Data from codepage 437, USAN ASCII
DefaultLocale("en_US");  // language locale is USA English

OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI

BgnParTxt(30, LEFT);  // Begin a paragraph, 30 mm width,  
                        // left-aligned
Font(1);   // Font 1 is ASCII font
PutParTxt(msg1, PINK);  // Put 1st ASCII text in pink color

Font(2);   // Font 2 is ASCII font
PutParTxt(msg2, BLUE);  // Put 2nd text in blue color

EndParTxt();  // End paragraph and write into AFP output

ClosePage();
CloseDoc();
End Paragraph of DBCS-HOST

Function

Ends a paragraph of SBCS-HOST/DBCS-HOST previously started with the function “Begin Paragraph of DBCS”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

void EndParDbcs();

Parameters

No parameter to be specified.

Sample

char *msg1 = "ひらがな、カタカナ、漢字、数字の 123、"
char *msg2 = "Alphabet が混在した文章のサンプルです。";

SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParDbcs(110, LEFT);   // Begin a Japanese paragraph, 110 mm
                         // width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is SJIS font
PutParDbcs(msg1, PINK); // Put 1st Japanese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is SJIS font
PutParDbcs(msg2, BLUE); // Put 2nd Japanese text in blue color

EndParDbcs();       // End Japanese paragraph and write into AFP output
ClosePage();
CloseDoc();
End Paragraph of Japanese

Function

Ends a paragraph of Japanese previously started with the function “Begin Paragraph of Japanese”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

```c
void EndParJp();
```

Parameters

No parameter to be specified.

Sample

```c
char *msg1 = "ひらがな、カタカナ、漢字、数字の 123",";
char *msg2 = "Alphabetが混在した文章のサンプルです。";

SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParJp(110, LEFT);   // Begin a Japanese paragraph, 110 mm
                      // width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is SJIS font
PutParJp(msg1, PINK);  // Put 1st Japanese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is SJIS font
PutParJp(msg2, BLUE);  // Put 2nd Japanese text in blue color

EndParJp();        // End Japanese paragraph and write into AFP output

ClosePage();
CloseDoc();
```
End Paragraph of Korean

Function

Ends a paragraph of Korean previously started with the function “Begin Paragraph of Korean”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

void EndParKr();

Parameters

No parameter to be specified.

Sample

char *msg1 = "'함께해요~!' 이벤트, 은 가족의 티셔츠가 내 품에!";  
char *msg2 = "내 책 사면 <어린이 도서관>에 기증될 책이 하나 더!";
SetUnit(MM_U600);  
OpenDoc();  
OpenPage(210,297);  
Pos(20, 50);  
LPI(4);   // Set line spacing to 4 LPI  
BgnParKr(110, LEFT);   // Begin a Korean paragraph, 110 mm  
// width, left-aligned  
Font(1,2);   // Font 1 is ASCII font, font 2 is KSC font  
PutParKr(msg1, PINK);   // Put 1st Korean text in pink color  
Font(3,4);   // Font 3 is ASCII font, font 4 is KSC font  
PutParKr(msg2, BLUE);   // Put 2nd Korean text in blue color  
EndParKr();   // End Korean paragraph and write into AFP output  
ClosePage();  
CloseDoc();
End Paragraph of Simplified Chinese

Function

Ends a paragraph of Simplified Chinese previously started with the function “Begin Paragraph of Simplified Chinese”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

```c
void EndParSc();
```

Parameters

No parameter to be specified.

Sample

```c
char *msg1 = "它允许用户为 XML 数据指定格式."
char *msg2 = "从而缩短了第一页开始打印的时间."
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParSc(110, LEFT);   // Begin a Simplified Chinese paragraph, 110 mm width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is GBK font
PutParSc(msg1, PINK);   // Put 1st Chinese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is GBK font
PutParSc(msg2, BLUE);   // Put 2nd Chinese text in blue color

EndParSc();       // End Simplified Chinese paragraph and write into AFP output
ClosePage();
CloseDoc();
```
End Paragraph of Traditional Chinese

Function

Ends a paragraph of Traditional Chinese previously started with the function “Begin Paragraph of Traditional Chinese”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

```c
void EndParTc();
```

Parameters

No parameter to be specified.

Sample

```c
char *msg1 = "它允许用户为 XML 数据指定格式,"
char *msg2 = "从而缩短了第一页开始打印的时间.";

SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParTc(110, LEFT);   // Begin a Traditional Chinese paragraph, 110 mm
                      // width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is BIG5 font
PutParTc(msg1, PINK); // Put 1st Chinese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is BIG5 font
PutParTc(msg2, BLUE); // Put 2nd Chinese text in blue color

EndParTc();       // End Traditional Chinese paragraph and write
                  // into AFP output

ClosePage();
CloseDoc();
```
End Paragraph of Wild-Char

Function

Ends a paragraph of wild-char text previously started with the function “Begin Paragraph of Wild-Char”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

void EndParW();

Parameters

No parameter to be specified.

Sample

wchar_t *msg1 = L"Congratulations!";
wchar_t *msg2 = L"You got a perfect AFP printing solution.";

DefaultLocale("en_US"); // language locale is USA English
LPI(4); // Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);
BgnParW(110, LEFT); // Begin a variable paragraph, 110 mm width, left-aligned
PutParW(msg1, BLUE); // Put 1st wild-char text in blue color
PutParW(msg2, RED); // Put 2nd wild-char text in red color
EndParW(); // End wild-char paragraph and write into AFP
End Paragraph of UTF-8

Function

Ends a paragraph of UTF-8 text previously started with the function “Begin Paragraph of UTF-8”, and writes its AFP data stream into AFP output file.

The paragraph must be ended before you end the page. You must ensure that the paragraph you have specified fits on the page.

Syntax

void EndParU8();

Parameters

No parameter to be specified.

Sample

char *msg1 = "Congratulations!";
char *msg2 = "You got a perfect AFP printing solution.";

DefaultCode("ibm-437");   // Data from codepage 437, USAN ASCII
DefaultLocale("en_US");   // language locale is USA English

LPI(4);                      // Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);
BgnParU8(110, LEFT);         // Begin a variable paragraph, 110 mm
                            // width, left-aligned
PutParU8(msg1, BLUE);       // Put 1st UTF-8 text in blue color
PutParU8(msg2, RED);        // Put 2nd UTF-8 text in red color
EndParU8();                 // End UTF-8 paragraph and write into AFP
End Shade

Function

Ends a shaded area at the current location. The shaded area must be previously started with a “Begin Shade” function call and it must be ended before you end the page.

You must ensure that the shade you have specified fits on the page.

Syntax

```c
void EndShade(
    ushort   shadeno = 1
);```

Parameters

- `shadeno`
  The ID number of the dynamic shade previously started with the “Begin Shade” function request, valid values are 1 through 24, the default value is 1.

Sample

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

:  
   BgnShade(1,1,2,5,0,RED)       // begin 1st shade at (1",1"), its
   BgnShade(5,1,3,10,1,BLUE)     // begin 2nd shade at (5",1"), 10

:  
   Ypos(9.5);                   // now Y position is at 7.5"
   EndShade();                  // End shade no. 0, shade depth is 8.5"
   EndShade(1);                 // End shade no. 1, shade depth is 8.5"

:  
   ClosePage();
   CloseDoc();
```
End Suppress

Function

Marks the ending of a string of presentation text that may be suppressed from the visible output. It must be started with the “Begin Suppress” function. Nesting of suppression control sequences is not allowed.

You can mark the suppression of a string of presentation text with the “Begin Suppress” and “End Suppress” functions, and turn the suppression on or off by calling of the different copy groups defined within an AFP form definition. A copy group is a portion of a form definition that defines a set of modifications that can be used when presenting a page. Modifications can include text suppression, overlays, and so forth. For certain types of documents, it is possible to print/view a page in multiple ways. For example, copy group one could be used to present a replica of a bill sent to the customer while copy group two could be used to present additional information attached to the bill, but not sent to the customer. With the AFP print server and the “Application Logical” views options of IBM DB2 Content Manager OnDemand, you can fully control the calling of copy groups defined within your form definition.

Syntax

```c
void EndSuppress(  
    UCHAR suppression_id,  
);  
```

Parameters

- **suppression_id**
  Specifies the identifier of suppression that is defined within the AFP form definition.

Sample

**PPFA sample:**

```c
FORMDEF PAYMNT  
    OFFSET 0.0 0.0 PRESENT PORTRAIT REPLACE YES;  
    SUPPRESSION SALARY; /* first one, whose suppression id is 1 */  
    SUPPRESSION BONUS;  /* second one, whose suppression id is 2 */  
COPYGROUP PAYMNT;  
    SUBGROUP;  
COPYGROUP SUPP1  
    SUBGROUP SUPPRESSION SALARY;  
COPYGROUP SUPP2  
    SUBGROUP SUPPRESSION BONUS;  
```

**MakeAFP Formatter sample:**

```c
OpenPage(8.5,11);  
    BgnSuppress(1);  
    Pos(1.5,2.5);  
    Ltxt("This text string can be suppressed by copygroup SUPP1");  
    EndSuppress(1);  
    BgnSuppress(2);  
```
Ltxt("This text string can be suppressed by copygroup SUPP2");
Ltxt("This text string can also be suppressed by copygroup SUPP2");

EndSuppress(2);

ClosePage();
End Underscore

Function

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

Syntax

void EndUscore();

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();
End Vertical Line

**Function**

Ends a vertical line previously started with a “Begin Vertical Line” call at the current position.

The line must be ended before you end the page. You must ensure that the line you have specified fits on the page.

**Syntax**

```cpp
void EndVline(
    ushort vlineno = 1
);
```

**Parameters**

- **vlineno**
  The ID number of the begin vertical line started, valid values are 1 through 24, the default value is 1.

**Sample**

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

::

  BgnVline(1,1,0.02)       // begin 1st vline at (1",1"),its
  BgnVline(5,1,0.02,1)     // thickness is 0.02"

  Ypos(9.5);              // now Y position is at 9.5"
  EndVline();            // End 1st vline, its length is 8.5
  EndVline(2)             // End 2nd vline, its length is 8.5

ClosePage();
CloseDoc();
```
File Name of the Default Input File (Query)

Function

Queries the filename of the default input data file, specified by the flag parameter "-i" of your command-line.

This function is mainly developed for calling from other programming languages. with C/C++, you can use FileInName variable directly.

Syntax

Char * FileInName();

Sample

None.
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 Formatter 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-16BE:

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

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

```c
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 Formatter definition file with the FONT parameter.

`SBCS_fontid`

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

`DBCS_fontid`

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

Sample

**AFP fonts defined in the MakeAFP definition file:**

```makefile
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 Formatter program:**

```c
c
const int 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 14
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 Query

Function

Queries the current font ID number.

Syntax

ushort FontID(); Returns the current SBCS or UTF-8/UTF16BE font ID
ushort 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();
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 Formatter PageNum variable.

Syntax

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

Sample

```c
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 3 for open page 3
  OpenPage(8.5,11);
  ClosePage();
  :  GotoPage(15); // switch to page 15 for open page 15
  OpenPage(8.5,11);
  ClosePage();
```
Graphic Arc

Function

Plots a GOCA graphic arc (partial circle). The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

At specify position:
void GArc(
    float x_pos,
    float y_pos
    float radius,
    float startang,
    float sweeping,
    bool background = TRUE
);

At current position:
void GArc2(
    float radius,
    float startang,
    float sweeping,
    bool background = TRUE
);

Parameters

x_pos
The graphic X position of the arc’s center.

y_pos
The graphic Y position of the arc’s center.

radius
The radius of the arc.

startang
The start angle, measured clockwise, between the X-axis and the start point of the arc.

sweepang
The angle, measured clockwise, between the two radii plotted from the center of the circle to the start and end points of the arc.

background
Specifies whether GOCA graphic is to be drawn as the background.

Sample

SetUnit(MM_U600);
OpenPage(220,297);
GPos(125,65); // set GOCA position for arc
GArc2(15, 150, 125); // set arc radius to 15 MM, start angle is 150 degree and sweep angle is 125 degree
GEndFill(); // End of fill GOCA area
ClosePage();
Graphic Barchart

Function

Plots a GOCA graphic barchart with its axis coordinates at the current graphic position, and the current default font is used for its texts.

With the “Barchart Element” and “Barchart Legend” functions, you can control how to place the legend labels and indicate whether to present the element labels or not.

With the “Graphic Chart Colors” function, you can override the default RGB color values predefined for the graphic barchart and piechart.

GBarchartW() function is also provided for the barchart texts encoded in wild-char Unicode.

Syntax

```c
void GBarchart(
    float          height,
    float          bar_width,
    float          gap_barset,
    uint      vertical_scale_unit,
    ushort     vertical_scales,
    ushort     number_bars,
    ushort     bar_sets,
    float*     values,
    char**     horizontal_labels,
    char**     legend_texts,
    bool       3D_barchart = TRUE,
    encoding   encode = ASCII,
    bool       gridline = TRUE,
    bool       background = TRUE
);```

Parameters

- **height**
  The height of the barchart graphic.

- **bar_width**
  The width of each bar.

- **gap_barset**
  The gap of each bar set.

- **vertical_scale_unit**
  The unit size of each scale on the vertical axis.

- **vertical_scales**
  The number of scales on the vertical axis.

- **number_bars**
  The number of bar elements in each bar set.
Bar_sets
The number of bar-sets.

values
The one dimension or two dimensions array of the data values.

horizontal_labels
The array of label texts for the horizontal axis. You can insert a new line control code \n or \x0a to split the text into multiple lines on output. This value is ignored if your number of bars is 1.

3D_barchart
Specify whether it is a 3D barchart or not, the default is TRUE, plots the 3D barchart.

legend_texts
The array of texts for the legend.

encode
The encoding of the output texts of barchart, the valid values are:

- ASCII Text is ASCII, uses ASCII font, this is the default value
- A2E Converts text from ASCII to EBCDIC, uses EBCDIC font
- GBK Text is GBK Chinese, uses ASCII/GBK-PC fonts
- GBK2DBCS GBK will be converted to DBCS, uses SBCS-DBCS-HOST/DBCS-HOST fonts
- BIG5 Text is BIG5 Chinese, uses ASCII/BIG5-PC fonts
- BIG52DBCS BIG5 will be converted to DBCS, uses SBCS-DBCS-HOST/DBCS-HOST fonts
- SJIS Text is SJIS Japanese, uses ASCII/SJIS-PC fonts
- SJIS2DBCS SJIS will be converted to DBCS, uses SBCS-DBCS-HOST/DBCS-HOST fonts
- KSC Text is KSC Korean, uses ASCII/KSC-PC fonts
- KSC2DBCS KSC will be converted to DBCS, uses SBCS-DBCS-HOST/DBCS-HOST fonts
- DBCS Text is SBCS/DBCS, uses SBCS-DBCS-HOST/DBCS-HOST fonts
- TOUTF16BE Converts legacy encoding data to wild-char big-endian, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly
- TOUTF8 Converts legacy encoding data to UTF-8, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly
- UTF8 Text is UTF-8, only uses OpenType/ TrueType font

gridline
Specifies whether gridlines are plotted on the barchart, default is TRUE.

background
Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
Float data1[4][3] = {{45, 55, 60}, // samples input data 1
                      {65, 35, 85},
                      {58, 95, 63},
                      {25, 35, 45}};
float data2[] = {45, 55, 60, 85}; // samples input data 2
char *lblr1[] = {"Mon\nHoliday 1", "Tue\nHoliday 2", "Wed", "Thu"};
char *lblr2[] = {"Mon", "Tue", "Wed", "Thu"};
char *legend[] = { "East", "West", "North" };
SetUnit(IN_U600);
OpenDoc();
```
OpenPage(8.5,11);
LPI(8);                      // line space is 8 LPI
Font(1);                      // select font 1 for barchart text
GPos(1, 2.5);                 // Set position of the barchart
GBarchart(2,                   // height of the barchart, 2"
  0.15,
  0.2,                 // Width of each bar, 0.15"
  10,
  10,                 // Gap between each bar sets, 0.2"
  3,                 // number of v-axis scales
  4,                  // 4 bar sets
  &data1[0][0],      // 2D datAn array of input data
  lbls1,            // label texts for H-axis
  legend);          // legend texts on right side
GPos(1, 5);                   // Set position of the barchart
GBarchart(2,                   // Max height of the barchart, 2"
  0.2,
  0.2,                 // Width of each bar, 0.15"
  10,
  10,                 // Gap between each bar set, 0.2"
  1,                 // number of v-axis scales
  4,                  // 4 bar sets
  data2,            // 1D datAn array of input data
  lbls2,            // label texts below H-axis
  NULL);            // no legend labels
ClosePage();
CloseDoc();

Output:
Graphic Barchart Element Label

Function

Optional function for the graphic barchart and stack chart, defines how to control the layout of the barchart element labels.

This function must be called before the GBarchart() function.

Syntax

```c
void GBarchartElement(
    position   element_label_position = NONE,
    char*   format-specification = "%0.1f",
    int    sbcs_fontid = DEFAULT
);
```

Parameters

element_label_position
Specifies where to place the bar element labels, Valid values are ABOVE (place labels on top of each bar), MIDDLE (place labels at the middle of each bar), and NONE (no element labels).

format-specification
Specifies the format-control string to format the bar element labels, refer to the MSDN library for more details about the format control string used in C standard function fprintf(). Default is "%0.1f", precision is 1.

sbcs_fontid
Specifies a 1-byte font id number to be used to present the bar element labels. Default is using the current ID of the font in 1-byte, UTF-8, or wild-char encoding.

Sample

```c
float data1[14][3] = {{45, 55, 60},    // samples testing input data
                      {65, 35, 85}, {58, 95, 63}, {25, 35, 45},
                      {45, 25, 30}, {55, 65, 75}, {65, 85, 90}};
char *lbls[]  = {
    "Mon;Holiday 1", "Tue\nHoliday 2", "Wed", "Thu", "Fri",
    "Sat", "Sun"};
char *legend[] = { "East", "West", "North" };
Start();                        // Start initiation
SetUnit(IN_U600);               // Set default units to in, 600 dpi
OpenDoc(); // page size is 8.5" by 11"  
OpenPage(8.5,11);               // line space is 8 LPI
LPI(8);                        // Set position of the barchart
GPos(1, 2.5);                  // select font 1 for barchart
Font(1);                      // place element label above bar
GBarchartElement(ABOVE,       // format-specification of value
    "%2.0f" );
```


```c
GBarchart(2, // Max height of the barchart
0.15,        // Width of each bar
0.2,         // Gap between each bar sets
10,          // unit size per v-axis scale
10,          // number of v-axis scales
3,           // number of bars per bar set
7,           // number of bar set
&data1[0][0], // datAn array of input data
lbls,        // label texts below H-axis
tlegend,     // legend text
FALSE,       // not 3D barchart
ASCII,       // ASCII text
FALSE);      // do not plot grid lines

ClosePage();
CloseDoc();

Output:
```

![Bar Chart Output](image_url)
Graphic Barchart Legend

Function

The optional function for the graphic barchart and stack chart, defines how to position and format the legend labels of the chart.

This function must be called before the GBarchart( ) or GStackchart( ) function if you want to override the default settings of the legend label.

Syntax

```c
void GBarchartLegend(
    float  x_rel_pos = DEFAULT,
    float  y_rel_pos  = DEFAULT,
    float      legend_box_width = DEFAULT,
    float      legend_box_height = DEFAULT,
    bool   vertical_layout = TRUE,
    fontid     fontID1 = DEFAULT,
    fontid     fontID2 = DEFAULT
);
```

Parameters

- **x_rel_pos**
  Specifies the X relative position of the highest label of the barchart legend, relative to the coordinate origin position of the barchart. Default is using the X position auto-decided by the Barchart function.

- **y_rel_pos**
  Specifies the Y relative position of the highest label of the barchart legend, relative to the coordinate origin position of the barchart. Default is using the Y position auto-decided by the Barchart function.

- **Legend_box_size**
  Specifies the box size of the legend label. Default is using the box size auto-decided by the Barchart function.

- **vertical_layout**
  Specified whether placing the legend labels vertically (from top to bottom) or horizontally (from left to right). The default value is TRUE, vertical layout.

- **fontID1**
  Specifies a font ID number to be used to present the legend label texts in ASCII/EBCDIC/UTF-8/ wild-char encodings. Default is using the current ID of the font in 1-byte, UTF-8, or wild-char encoding.

- **fontID2**
  Specifies a font ID number to be used to present the legend label texts in DBCS-PC or DBCS-HOST encoding. Default is using the current ID of the 2-byte font.

Sample

```c
float data1[14][3] = {
    {45, 55, 60},        // samples testing input data
    {65, 35, 85},
    {58, 95, 63},
    {25, 35, 45},
    {45, 25, 30},
    {55, 65, 75},
    {65, 85, 90}];//
```

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

Start();                     // Start initiation
SetUnit(IN_U600);            // Set default units to in, 600 dpi

OpenDoc();
  OpenPage(8.5,11);         // page size is 8.5" by 11"
LPI(8);                   // line space is 8 LPI
GPos(1, 2.5);             // Set position of the barchart
Font(1);                  // select font 1 for barchart
GBarchartLegend(1.4,         // plus 2.5" to place legend X
               0.4,         // plus 0.4" to place legend Y
               0.1,         // legend box width is 0.1"  
               0.1,         // legend box height is 0.1"  
               FALSE);      // place legend horizontally

GBarchart(2,                // Max height of the barchart
          0.15,             // Width of each bar
          0.2,              // Gap between each bar sets
          10,               // unit size per v-axis scale
          10,               // number of v-axis scales
          3,                // number of bars per bar set
          7,                // number of bar set
          &data1[0][0],     // datAn array of input data
          lbls,             // label texts below H-axis
          legend,           // legend texts
          ASCII,            // text is ASCII
          FALSE);            // do not plot grid lines

ClosePage();
CloseDoc();

Output:
Graphic Begin Area

Function

Begins a GOCA area inside a GOCA object. Areas are two-dimensional primitives that are filled with a pattern defined by the “Graphic Pattern” function request.

Syntax

```c
void GBgnArea(
    bool  plot_boundary = TRUE,
    bool   background = TRUE
);
```

Parameters

- **plot_boundary**
  Defines whether the graphic area boundary is plotted, default is TRUE.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2);   // set graphic line width
    // (about 0.02") for box boundary
GPattern(DOT8);  // define DOT8 pattern to be used
GBgnArea();      // for the begin GOCA area
GBox(1,1,4, 7.5);  // draw 4 x 7.5" box at (1", 1")
    // box width = 7.5",
GEndArea();      // end GOCA area
ClosePage();
CloseDoc();
```
Graphic Begin Box

Function

Begins a graphic box of variable height from the position you specified, the box is not drawn until the corresponding “Graphic End Box” call is issued. The box must be ended before you end the page. You must ensure that the box you have specified fits on the page.

The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

```c
void GBgnBox(
    float  x_pos,
    float  y_pos,
    float  width,
    ushort boxno = 1,
    float  rounded_corner = 0
);
```

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.

- **width**
The width of the box.

- **boxno**
The ID number of this graphic box. Valid values are 1 through 24, the default value is 1.

- **rounded_corner**
The radius of the rounded box corners.

Sample

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

GLineWidth(2);   // set line width (about 0.02")
// for box border
GBgnBox(0.5, 4, 7.5);   // draw box from (0.5",4"),
// box width = 7.5",
:
:

GPos(0.5,10);   // now graphic position is at
// (0.5",10")
GEndBox();   // End of box, so its height now is
// 10 - 4 = 6"
ClosePage();
CloseDoc();
```
Graphic Begin Filling Pattern

Function

Begins filling a graphic area with a pattern.

Syntax

```c
void GBgnFill(
    type    pattern = SOLID,
    bool    plot_boundary = TRUE,
    bool    background = TRUE
);
```

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pattern</strong></td>
<td>Type of fill pattern, valid values are:</td>
</tr>
<tr>
<td>SOLID</td>
<td>Solid fill, default value</td>
</tr>
<tr>
<td>DOT1 through DOT8</td>
<td>Dotted patterns of decreasing density</td>
</tr>
<tr>
<td>VERTLN</td>
<td>Vertical lines</td>
</tr>
<tr>
<td>HORZLN</td>
<td>Horizontal lines</td>
</tr>
<tr>
<td>BLTR1, BLTR2</td>
<td>Diagonal lines from bottom-left to top-right</td>
</tr>
<tr>
<td>TLBR1, TLBR2</td>
<td>Diagonal lines from top-left to bottom-right</td>
</tr>
<tr>
<td>NOFILL</td>
<td>No fill pattern</td>
</tr>
</tbody>
</table>

| **plot_boundary** | Defines whether the graphic area boundary is plotted, default is TRUE. |
| **background**    | Specifies whether GOCA graphic is to be drawn as the background. |

Sample

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

GLineWidth(2);      // set line width (about 0.02")
// for box boundary
GBgnFill(DOT8);     // begin fill area with DOT8 pattern
GBox(1,1,4, 7.5);   // draw 4x7.5" box at (1",1")
// box width = 7.5"
GEndFill();
ClosePage();
CloseDoc();
```
Graphic Begin Horizontal Line

Function

Begins a horizontal graphic line of variable length from the position you specified, the line is not drawn until the corresponding “Graphic End Horizontal Line” call is issued. The line must be ended before you end the page. You must ensure that the line you have specified fits on the page.

The thickness and type of line are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

```c
void GBgnHline(
    float x_pos,
    float y_pos,
    ushort hlineno = 1
);
```

Parameters

- **x_pos**
The X starting position of the horizontal line.

- **y_pos**
The Y starting position of the horizontal line.

- **hlineno**
The ID number of this horizontal graphic line. Valid values are 1 through 24, the default value is 1.

Sample

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

GBgnHline(1,1); // begin 1st ghline at (1",1")
GBgnHline(1,2,1); // begin 2nd ghline at (1",2")
:

GPos(7.5,2); // now graphic position is at (7.5",2")
GEndHline(); // End 1st ghline, its length is 6.5
GEndHline(2); // End 2nd ghline, its length is 6.5
:

ClosePage();
CloseDoc();
```
Graphic Begin Vertical Line

Function

Begins a vertical graphic line of variable height from the position you specified, the line is not drawn until the corresponding “Graphic End Vertical Line” call is issued. The line must be ended before you end the page. You must ensure that the line you have specified fits on the page.

The thickness and type of line are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

```c
void GBgnVline(
    float x_pos,
    float y_pos,
    ushort vlineno = 1
);
```

Parameters

- `x_pos`
  The X starting position of the vertical line.

- `y_pos`
  The Y starting position of the vertical line.

- `vlineno`
  The ID number of this vertical graphic line, Valid values are 1 through 24, the default value is 1.

Sample

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

GBgnVline(1,1); // begin 1st gvline at (1",1")
GBgnVline(2,1,1); // begin 2nd gvline at (2",1")

GPos(1,7.5); // now graphic position is at (1",7.5")
GEndVline(); // End 1st gvline, its length is 6.5
GEndVline(2); // End 2nd gvline, its length is 6.5

ClosePage();
CloseDoc();
```
Graphic Box

Function

Plots a box at either the current graphic position or the position you specified. The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

At specify position:

```c
void GBox(
    float  x_pos,
    float  y_pos,
    float  width,
    float  height,
    float  corner_radius = 0,
    bool   background = TRUE
);
```

At current position:

```c
void GBox2(
    float  width,
    float  height,
    float  corner_radius = 0,
    bool   background = TRUE
);
```

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.

- **width**
  The width of the box.

- **height**
  The height of the box.

- **corner_radius**
  The radius of the rounded box corners.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
SetUnit(IN_U600);
OpenPage(8.5,11);
GLineWidth(2);    // set line width to about 0.02"
// for box border
Box(0.5, 4, 7.5, 1, 0.1);    // box size 7.5" x 1", at (0.5",4"),
```
// with small rounded corners

ClosePage();
Graphic Box Rounded

Function

Plots a rounded box. The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

```c
void GRbox(
    float  x_pos,
    float  y_pos,
    float  width,
    float  height,
    float  top_left_corner,
    float  top_right_corner,
    float  bottom_left_corner,
    float  bottom_right_corner,
    bool   background = TRUE
);
```

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.
- **width**
  The width of the box.
- **height**
  The height of the box.
- **top_left_corner, top_right_corner, bottom_left_corner, bottom_right_corner**
  The radius values to be used to round the corners.
- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2);        // set line width (about 0.02")
// for box border
GRbox(0.5,4,7.5,1,0.15,0,0.15,0);  // box size 7.5"x1", at (0.5",4"),
                                        // with rounded corners at top-  // left and bottom-left
                                           :
                                           :
ClosePage();
CloseDoc();
```
Graphic Circle

Function

Plots a circle at either the current graphic position or the position you specified. It does not change the current graphic position.

The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

At specify position:
void GCircle(
    float  x_pos,
    float  y_pos,
    float  radius,
    bool   background = TRUE
);

At current position:
void Gcircle2(
    float  radius,
    bool   background = TRUE
);

Parameters

x_pos
The center position of the circle.

y_pos
The center position of the circle.

radius
The length of the radius.

background
Specifies whether GOCA graphic is to be drawn as the background.

Sample

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
GLineWidth(2);       // set line width (about 0.02")
// for circle
GCircle(0.5, 4, 2);  // draw a circle at (0.5",4"), its // radius is 2"
  :
  :
ClosePage();
CloseDoc();
Graphic Chart Colors

Function

Optional function for overriding the default color values predefined for the graphic bar chart, pie chart, and stack chart.

Syntax

```c
void GChartColors(
    uint * colors,
    short num_of_colors,
    short color_mode = GOCA_RGB,
);
```

Parameters

colors
An array of RGB or CMYK colors in hex values predefined for graphic barchart and piechart, up to 30 values can be defined.

RGB color values must be defined by format of 0x[red][green][blue], valid RGB intensity range values for each component are 0 through 255 (hex value x'FF').

CMYK color values must be defined by format of 0x[cyan][magenta][yellow][black], valid CMYK percentage range values for each component are 0 through 100 (hex value x'64').

The default color values predefined by MakeAFP Formatter are RGB colors:

```c
{ 0x00ff00, /* GREEN */ 0x8effff, /* SKY_BLUE */
  0x00c0c0, /* TURQUOISE */ 0xffc66b, /* TAN */
  0x9f9fff, /* SKY_GRAY */ 0xc1ffc1, /* SKY_GREEN */
  0x0060ff, /* LIGHT_BLUE */ 0xff00ff, /* HOT_PINK */
  0xff995b, /* MEDIUM_BROWN */ 0xbfbfff, /* LIGHT_GRAY */
  0x9900cc, /* PURPLE */ 0x00ffff, /* CYAN */
  0xff9933, /* ORANGE */ 0xff90ff, /* SKY_PINK */
  0xc7b1ff, /* LAVENDER */ 0x00ff80, /* LIGHT_GREEN */
  0x80ff80, /* LIGHT_GREEN */ 0x6060ff, /* LIGHT_BLUE */
  0x8000ff, /* DARK_PURPLE */ 0xff9933, /* ORANGE */
  0x8000ff, /* DARK_PURPLE */ 0x208020, /* DARK_GREEN */
  0x208020, /* DARK_GREEN */ 0x80ff80, /* LIGHT_GREEN */
  0x00ff00, /* YELLOW */ 0xc00000, /* DARK_RED */
  0x00c000, /* FOREST_GREEN */ 0x80c0c0, /* GRAY */ }
```

num_of_colors
The number of RGB color values to be overridden.

color_mode
Indicates the mode of color, whether is GOCA_RGB or GOCA_CMYK. Valid values are GOCA_RGB and GOCA_CMYK.

Sample

```c
float data[] = {45, 65, 35, 70, 40, 50};
```
char *label[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
USHORT colors[] = {0x6565ff, 0xc000c0, 0xff996c, 0xbfbfff, 0x8000ff, 0x00c0c0};

SetUnit(IN_U600);
GChartColors(colors, 6); // override default predefined RGB colors
OpenDoc();
OpenPage(8.5, 11);
Font(1); // select font 1 for piechart
LPI(8); // Set line space to 8 LPI for label texts
GPos(2, 2); // Set current GOCA position
GPiechart(1.6, // piechart width
1, // piechart height
0.13, // 2D piechart shadow depth
6, // sectors of piechart
data2, // array of input data
label); // array of label text
Graphic Chart Patterns

Function

An optional function specifies whether draw GOCA patterns on the 1D bar chart, pie chart, and stack chart.

Syntax

```c
void GChartPatterns(
    bool       draw_goca_patterns,
    ushort *   goca_patterns = NULL,
    ushort     num_of_patterns = 0
);
```

Parameters

**draw_goca_patterns**
Specifies whether use GOCA patterns to draw the 1D bar chart, pie chart, or stack chart.

**goca_patterns**
Specifies an array of GOCA filling patterns, to override the default patterns predefined by the MakeAFP Formatter.

Valid values are:

- **DIACROSS1, DIACROSS2** - Cross lines – by the diagonal lines
- **CROSS** - Cross lines – by the horizontal & vertical lines
- **DOT1 through DOT8** - Dotted patterns of decreasing density
- **BLTR1, BLTR2** - Diagonal lines from bottom-left to top-right
- **HORZLN** - Horizontal lines
- **NOFILL** - No filling pattern
- **SOLID** - Solid filling, default value
- **TLBR1, TLBR2** - Diagonal lines from top-left to bottom-right
- **VERTLN** - Vertical lines

The default GOCA pattern values predefined by MakeAFP Formatter are:

```c
{ SOLID, DIACROSS1, BLTR2, VERTLN, TLBR2, HORZLN,
  DIACROSS2, BLTR1, DOT6, CROSS, TLBR1, DOT3 };
```

**num_of_patterns**
The number of GOCA patterns to be overridden.

Sample

```c
float data[] = {45, 65, 35, 70, 40, 50};
char *label[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
SetUnit(IN_U600);
GChartPatterns(true);    // draw GOCA patterns with 1D piechart
OpenDoc();
OpenPage(8.5,11);
```
Font(1); // select font 1 for piechart
LPI(8); // set line space to 8 LPI for label texts
GPos(2, 2); // set current GOCA position
GPiechart(1.6, // piechart width
1, // piechart height
0, // 1D piechart, no shadow depth
6, // sectors of piechart
data2, // array of input data
label); // array of label text
Graphic Color

Function

Specifies the color for the consequent graphics.

Syntax

For OCA color:

\[ \text{GColor}(\]
\[ \quad \text{ocaColor} \quad \text{ocaColor} = \text{BLACK},\]
\[ \quad \text{bool} \quad \text{background} = \text{TRUE}\]
\[ );\]

For RGB color:

\[ \text{GColorRGB}(\]
\[ \quad \text{UCHAR} \quad \text{red\_color},\]
\[ \quad \text{UCHAR} \quad \text{green\_color},\]
\[ \quad \text{UCHAR} \quad \text{blue\_color},\]
\[ \quad \text{bool} \quad \text{background} = \text{TRUE}\]
\[ );\]

For CYMK color:

\[ \text{GColorCMYK}(\]
\[ \quad \text{UCHAR} \quad \text{cyan\_color\_percentage},\]
\[ \quad \text{UCHAR} \quad \text{magenta\_color\_percentage},\]
\[ \quad \text{UCHAR} \quad \text{yellow\_color\_percentage},\]
\[ \quad \text{UCHAR} \quad \text{black\_color\_percentage},\]
\[ \quad \text{bool} \quad \text{background} = \text{TRUE}\]
\[ );\]

Parameters

\textbf{ocaColor}

Any of the defined AFP 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.

\textbf{RGB values}

Valid RGB intensity range values for each component are 0 through 255 (hex value x'FF').

\textbf{CYMK color percentage values}

Valid CYMK percentage range values for each component are 0 through 100 (hex value x'64').

\textbf{background}

Specifies whether GOCA graphic is to be drawn as the background.

Sample

\begin{verbatim}
SetUnit(MM_U600);
:
GColorRGB(255,0,0);  // RGB red color
GHline(1,2,4);       // red color line
:
\end{verbatim}
GColorCMYK(0,0,0,100); // CYMK black color
GHline(1,3,4) // black color line

GColor(CYAN); // AFP OCA CYAN color
GHline(1,4,4) // Cyan color line
Graphic Color Management Resource Association

Function

Associates a CMR (Color Management Resource) with the GOCA vector graphics at the current page.

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, required by the AFP system to process a color 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.

Refer to the latest Infoprint PPFA User's Guide and Infprint Manager Procedures books for more about AFP color management.

Syntax

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

Parameters

**cmr_id**

The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Formatter 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).

- **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();
  :
OpenPage(210,297);
  :
GCMR(1, INSTR);  // Invoke a CMR association for the graphics at the current page, ID 1 of CMR was predefined in the MakeAFP Formatter definition file with parameter CMR1
GLineWidth(5);  // Set line width to about 0.05"
GBox(1.3, 1.8, 3, 1.2, 0.15);  // Plot a rounded box
GColor(BLACK);  // set OCA black color
GBgnFill();  // start filling of a box area
GBox(0.5, 4, 7.5, 0.26, 0.02);  // plot a box
GEndFill();  // end filling of box area
GColor(255,255,255);  // set RGB white color
GVline(1.5, 4, 0.25);  // plot vertical lines
GVline(5, 4, 0.25);
GVline(6, 4, 0.25);
GVline(7, 4, 0.25);
  :
ClosePage();
  :
CloseDoc();
Graphic Ellipse

Function

Plots an ellipse at either the current graphic position or the position you specified. It does not change the current graphic position.

The thickness and type of border are determined from the "Graphic Line Width" and "Graphic Line Type" function calls.

Syntax

At specify position:

```c
void GEllipse(
    float x_pos,
    float y_pos,
    float xmaj,
    float ymaj,
    float xmin,
    float ymin,
    bool  background = TRUE
);
```

At current position:

```c
void Gellipse2(
    float xmaj,
    float ymaj,
    float xmin,
    float ymin,
    bool  background = TRUE
);
```

Parameters

- **x_pos**
  The center position of the ellipse.

- **y_pos**
  The center position of the ellipse.

- **xmaj, ymaj**
  The coordinates of the endpoint of the major axis of the ellipse. In the above figure, P is the X major coordinate and S is the Y major coordinate.

- **Xmin, ymin**
  The coordinates of the endpoint of the minor axis of the ellipse. In the above figure, R is the X minor coordinate and Q is the Y minor coordinate.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
GColor(YELLOW);
GBgnFill();
```
GColor(BLUE);
GPos(25,250);
GEllipse2(15,5,5,15);
GEndFill();
Graphic End Area Pattern

Function

Ends a GOCA area started with a “Graphic Begin Area” function call. The GOCA area must be ended before you end the page.

This function does not change the current graphic position.

Syntax

```c
void GEndArea(
    bool     background = TRUE
);
```

Parameters

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2);      // set line width (about 0.02")
// for box boundary
GPattern(DOT3);      // set graphic patter to DOT3
GBgnArea();         // begin GOCA area
GBox(1,1,4, 7.5);   // draw 4x7.5" box at (1",1")
// box width = 7.5",
GEndArea();         // end GOCA area
ClosePage();
CloseDoc();
```
Graphic End Box

Function

Ends a graphic box of variable height at the current location. The box must be previously started with the “Graphic Begin Box” function call and must be ended before you end the page.

Syntax

```c
void GEndBox(
    USHORT boxno = 1,
    bool background = TRUE
);
```

Parameters

**boxno**
The ID number of the graphic box started with the “Graphic Begin Box” function call, Valid values are 1 through 24, the default value is 1.

**background**
Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2); // set line width (about 0.02")
    // for box border
GBgnBox(0.5, 4, 7.5); // draw box from (0.5",4"),
    // box width = 7.5",
    :
    :

GPos(0.5,10); // now graphic position is at
    // (0.5",10")
GEndBox(); // End of box, so its height now is
    // 10 - 4 = 6"
ClosePage();
CloseDoc();
```
Graphic End Filling Pattern

Function

Ends filling a graphic area started with the “Graphic Begin Filling Pattern” call and must be ended before you end the page.

This function does not change the current graphic position.

Syntax

```c
void GEndFill(
    bool      background = TRUE
);
```

Parameters

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2); // set line width (about 0.02")
// for box boundary
GBgnFill(DOT8); // begin fill area with DOT8 pattern
GBox(1,1,4,7.5); // draw 4x7.5" box at (1",1")
// box width = 7.5",
GEndFill(); // end fill area
ClosePage();
CloseDoc();
```
Graphic End Horizontal Line

Function

Ends a horizontal graphic line started with the “Graphic Begin Horizontal Line” call at the current position.

The line must be ended before you end the page. You must ensure that the line you have specified fits on the page.

Syntax

```c
void GEndHline(
    ushort   lineno = 1,
    bool    background = TRUE
);
```

Parameters

- `lineno`
  The ID number of the graphic horizontal line started, Valid values are 1 through 24, the default value is 1.

- `background`
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GBeginHline(1,1);  // begin 1st ghline at (1",1")
GBeginHline(1,2,1);  // begin 2nd ghline at (1",2")

GPos(7.5,2);          // now graphic position is at (7.5",2")
GEndHline();          // End 1st ghline, its length is 6.5
GEndHline(2);         // End 2nd ghline, its length is 6.5

ClosePage();
CloseDoc();
```
Graphic End Vertical Line

Function

Ends a vertical graphic line started with the “Graphic Begin Vertical Line” call at the current position.

The line must be ended before you end the page. You must ensure that the line you have specified fits on the page.

Syntax

```c
void GEndVline(
    ushort   lineno = 1,
    bool    background = TRUE
);
```

Parameters

**lineno**

The ID number of the graphic vertical line started, Valid values are 1 through 24, the default value is 1.

**background**

Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
GBgnVline(1,1);     // begin 1st gvline at (1",1")
GBgnVline(2,1,1);   // begin 2nd gvline at (2",1")
:
:
GPos(1,7.5);        // now graphic position is at (1",7.5")
GEndVline();        // End 1st gvline, its length is 6.5
GEndVline(2);       // End 2nd gvline, its length is 6.5
:
:
ClosePage();
CloseDoc();
```
Graphic Fillet

Function

Plots a fillet from the current position. The positions for the objects are supplied as two arrays of floats consisting of the X and Y graphic positions.

The thickness and type of line are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

The current graphic position is set to the last of the (X, Y) coordinate pairs that are specified after this call.

Refer to IBM Graphics Object Content Architecture Reference for more detailed information about fillets.

Syntax

```c
void GFillet(
    float* x_array,
    float* y_array,
    ushort counter,
    bool background = TRUE
);
```

Parameters

- **x_array, y_array**
  The address of the arrays containing the X and Y coordinates of the GOCA graphic where the fillet is to be drawn.

- **counter**
  The number of coordinates that are supplied in the X and Y arrays.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
float x_array1[] = { 150, 159, 165, 176, 180, 200};
float y_array1[] = { 270, 235, 266, 285, 235, 275};

SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
GColor(RED);
GPos(145, 270);
GFillet(x_array1, y_array1, 6);
:
ClosePage();
CloseDoc();
```

Output:
Graphic Horizontal Boxes

**Function**

Repeat plotting of graphic boxes horizontally from the position you specified.

The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

**Syntax**

```c
void GHboxes(
  float x_pos,
  float y_pos,
  float width,
  float height,
  ushort repeat
  float space,
  float corner_radius = 0,
  bool background = TRUE
);
```

**Parameters**

- **x_pos**
  The X position of the top left corner of the first box.

- **y_pos**
  The Y position of the top left corner of the first box.

- **width**
  The width of the box.

- **height**
  The height of the box.

- **repeat**
  The number of additional boxes to be repeated.

- **space**
  The gap space between the boxes.

- **corner_radius**
  The radius of rounded box corners.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

**Sample**

```c
SetUnit(MM_U600);
OpenPage(220,297);

GHboxes(10,10,20,5,9,2);  // repeat 10 boxes horizontally from
// (10,10)mm, box width is 20mm, height
// is 5mm, 2mm space between boxes

ClosePage();
```
Graphic Horizontal Line

Function

Plots a graphic horizontal line from either the current graphic position or the graphic position you specified.

The thickness and type of line are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

From specify position:

```c
void GHline(
    float  x_pos,
    float  y_pos,
    float  length,
    bool  background = TRUE
);
```

From current position:

```c
void GHline2(
    float  length,
    bool  background = TRUE
);
```

Parameters

- **x_pos**
The X starting position of the line.

- **y_pos**
The Y starting position of the line.

- **length**
The length of the line.

- **background**
Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:    :
GHline(10,10,80);    // plot a horizontal line from (10,10) mm,  // 80 mm length
    :
:    :
ClosePage();
CloseDoc();
```
Graphic Horizontal Lines

Function

Repeat plotting of graphic horizontal lines from the graphic position you specified.

The thickness and type of lines are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

```c
void GHlines(
    float x_pos,
    float y_pos,
    float length,
    ushort repeat,
    float space
    ushort direction = DOWN,
    bool background = TRUE
);
```

Parameters

- `x_pos`  
The X starting position of the line.

- `y_pos`  
The Y starting position of the line.

- `length`  
The length 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.

- `background`  
Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
:
GHlines(10,10,80,9,5); // plot 10 horizontal lines from (10,10) // length is 80 mm, 5 mm space between // lines :
:
ClosePage();
CloseDoc();
```
Graphic Lines

Function

Plots a series of connected straight lines from the given position or the current position.

The thickness and type of line are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

The current graphic position is set to the last of the (X, Y) coordinate pairs that are specified after this call.

Syntax

From specified position:

```c
void GLines(
    float    x_pos,
    float    y_pos,
    float*   x_array,
    float*   y_array,
    ushort   counter,
    bool     background = TRUE
);
```

From current position:

```c
void GLines2(
    float*   x_array,
    float*   y_array,
    ushort   counter,
    bool     background = TRUE
);
```

Parameters

- **x_pos**
  The X starting position.

- **y_pos**
  The Y starting position.

- **x_array**
  The address of an array containing the X coordinates of the GOCA graphic of where the fillet is to be drawn.

- **y_array**
  The address of an array containing the Y coordinates of the GOCA graphic of where the fillet is to be drawn.

- **counter**
  The number of coordinates that are supplied in the X and Y arrays.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.
Sample

float x_array[] = { 100, 107, 115, 128, 139};
float y_array[] = { 280, 235, 245, 255, 275};

GLines(75, 275, x_array, y_array, 5);
Graphic Linechart

Function

Plots a GOCA graphic linechart with its axis coordinates at the current graphic position, and the current default font is used for its label texts.

GLinechartW() function is also provided for the linechart texts encoded in wild-char Unicode.

Syntax

```c
void GLinechart(
    float      width,
    float  height,
    uint       vertical_scale_unit,
    ushort      vertical_scales,
    float*      values,
    ushort  numData_perLine,
    char**  label_texts,
    ushort        numLabels,
    char**  legend_texts,
    ushort      lines = 1,
    float       line_weight = 1.5,
    encode      encoding = ASCII,
    bool       grid_line = true,
    bool      symbol_node = false,
    bool      line_label = false,
    uint*  line_colors = NULL,
    ushort  color_mode = GOCA_OCA,
    ushort*  line_types = NULL,
    bool      background = true
);
```

Parameters

- **width**
The width of the linechart graphic.

- **height**
The height of the linechart graphic.

- **vertical_scale_unit**
The unit size of each scale on the vertical axis.

- **vertical_scales**
The number of scales on the vertical axis.

- **values**
The one dimension or multi-dimensions array of the data values.

- **numData_perLine**
The number of data values per line.

- **label_texts**
The array of label texts under the horizontal axis.
The line_colors
Specifies line_color
The symbol_node
Specifies symbol
Specifies grid_line
The grid_line
Specifies whether gridlines are plotted on the linechart, default is TRUE.

The array of texts for the legend.

The number of lines per linechart, the default value is 1, draws only one line.

Defines line weight as a multiplier of the normal AFP OCA line weight (which is approximately 0.01 inches). The default value is 1.5, which is about 0.015 inches.

The encoding of the output label texts, the valid values are:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>Text is ASCII, uses ASCII font, this is the default value</td>
</tr>
<tr>
<td>A2E</td>
<td>Converts text from ASCII to EBCDIC, uses EBCDIC font</td>
</tr>
<tr>
<td>GBK</td>
<td>Text is GBK Chinese, uses ASCII/GBK-PC fonts</td>
</tr>
<tr>
<td>GBK2DBCS</td>
<td>GBK will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>BIG5</td>
<td>Text is BIG5 Chinese, uses ASCII/BIG5-PC fonts</td>
</tr>
<tr>
<td>BIG52DBCS</td>
<td>BIG5 will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>SJIS</td>
<td>Text is SJIS Japanese, uses ASCII/SJIS-PC fonts</td>
</tr>
<tr>
<td>SJIS2DBCS</td>
<td>SJIS will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>KSC</td>
<td>Text is KSC Korean, uses ASCII/KSC-PC fonts</td>
</tr>
<tr>
<td>KSC2DBCS</td>
<td>KSC will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>DBCS</td>
<td>Text is SBCS/DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>TOUTF16BE</td>
<td>Converts legacy encoding data to wild-char big-endian, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly</td>
</tr>
<tr>
<td>TOUTF8</td>
<td>Converts legacy encoding data to UTF-8, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly</td>
</tr>
<tr>
<td>UTF8</td>
<td>Text is UTF-8, only uses OpenType/ TrueType font</td>
</tr>
</tbody>
</table>

The one dimension array defining with AFP OCA, RGB, or CMYK colors, if you want to override the MakeAFP default OCA color values for the lines.

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, GRAY.

RGB color values must be defined by format of 0x[red][green][blue], valid RGB intensity range values for each component are 0 through 255 (hex value x’FF’).

CMYK color values must be defined by format of 0x[cyan][magenta][yellow][black], valid CMYK percentage range values for each component are 0 through 100 (hex value x’64’).
**color_mode**
Indicates the mode of color. Valid values are GOCA_OCA, GOCA_RGB, and GOCA_CMYK.

**line_types**
The one dimension array defining with GOCA line patterns. Valid values are:

- **SOLID**  Solid line.
- **DOTTED** Dotted line.
- **SHORTDASH** Short dashed line.
- **DSHLINE** Dash-dot line.
- **DBLDOT** Double-dotted line.
- **LONGDASH** Long dashed line.
- **DSHDBLDOT** Dash-double-dot line.

**background**
Specifies whether GOCA graphic is to be drawn as the background.

**Sample**

```plaintext
float data[] = { 30, 36, 28, 26, 34, 42, 40, 35, 31, 37, 30, 35, 32, 38, 78, 80, 80, 79, 70, 75, 71, 62, 56, 58, 62, 70, 58, 50, 53 }; char *lbs[] = { "02.2", "02.4", "02.6", "02.8", "02.10", "02.12", "02.14", "02.16", "02.18", "02.20", "02.22", "02.24", "02.26", "02.28", "02.30" }; char *lgdText[] = { "Power", "Water" }; UINT lineColors[] = { RED, DARKBLUE }; Start();
```

```
SetUnit(IN_U600); // set default units to inch, 600 dpi
OpenDoc();
OpenPage(8.27, 11.67);
Font(1); // define the default for Linechart
GPos(1.75,3); // set position of linechart
GLinechartLegend(3.8, -0.4, 0.3, true, 2); // relative X-position of legend
GLinechartLegend(3.8, 0.3, true, 2); // font ID of legend texts
GLinechart(3.5, 2, 10, 10, data, 15, lbs, 15, lgdText, 2, 2, // linechart width
```
ASCII, // Text encoding is ASCII,
true,  // draw grid lines
false, // no symbol node
false, // no line label
lineColors); // Override the default colors of lines

ClosePage();

CloseDoc();

Output:
Graphic Linechart Legend

Function

The optional function for the graphic line chart, defines how to position and format the legend labels of the chart.

This function must be called before the GLinechart() function if you want to control the settings of the legend label.

Syntax

```c
void GLinechartLegend(
    float  x_rel_pos,
    float  y_rel_pos,
    float  line_length,
    bool   vertical_layout = TRUE,
    fontid fontID1 = DEFAULT,
    fontid fontID2 = DEFAULT
);
```

Parameters

- **x_rel_pos**
  Specifies the X relative position of the highest label of the line chart legend, relative to the coordinate origin position of the barchart. MakeAFP Formatter does not present the legend of the line chart if this value is 0.

- **y_rel_pos**
  Specifies the Y relative position of the highest label of the line chart legend, relative to the coordinate origin position of the barchart. MakeAFP Formatter does not present the legend of the line chart if this value is 0.

- **line_length**
  Specifies the length of the legend line.

- **fontID1**
  Specifies a font ID number to be used to present the legend label texts in ASCII/EBCDIC/UTF-8/ wild-char encoding. Default is using the current ID of the font in 1-byte, UTF-8, or wild-char encoding.

- **fontID2**
  Specifies a font ID number to be used to present the legend label texts in DBCS-PC or DBCS-HOST encoding. Default is using the current ID of the 2-byte font.

Sample

```c
float data[] = {  78, 80, 80, 79, 70, 75, 71, 62, 56, 58,
    62, 70, 58, 50, 53,
    30, 36, 28, 26, 34, 42, 40, 35, 31, 37,
    30, 35, 34, 32, 38 }; 

char *lbls[] = { "02.2", "02.4", "02.6", "02.8", "02.10",
    "02.12", "02.14", "02.16", "02.18", "02.20",
    "02.22", "02.24", "02.26", "02.28", "02.30" }; 
```
char *lgdText[] = { "Power", "Water" };
USHORT lineColors[] = { RED, DARKBLUE };

Start();
SetUnit(IN_U600); // set default units to inch, 600 dpi
OpenDoc();
OpenPage(8.27, 11.67);
Font(1); // define the default for Linechart
Pos(1.75,3); // set position of linechart

GLinechartLegend(1, // relative X-position of legend
    -2,  // relative Y-position of legend
    0.3, // line length of legend
    false, // draw horizontal legend
    2);  // font ID of legend texts

GLinechart(3.5, // linechart width
    2,  // linechart height
    10, // V-axis unit
    10, // V-axis scales
    data, // data values by 1D-array
    15, // Number of values per line
    lbls, // H-label texts
    15, // Number of h-labels
    lgdText, // Legend texts
    2, // Draw two lines
    1, // line weight, about 0.01"
    ASCII, // Text encoding is ASCII,
    false, // no grid lines
    true, // draw symbol nodes
    true, // draw line label along with line
    lineColors); // Override the default colors of lines

ClosePage();
CloseDoc();

Output:
Graphic Line Ending Style

Function

Sets the graphic line ending style that specifies how the endpoint of a line will be drawn.

*Note: This GOCA feature needs the latest IPDS controller code, check with your IPDS printer vendors whether they have supported it.

Syntax

```c
void GLineEndStyle(
    ushort   line_end_style
    bool    background = TRUE
);
```

Parameters

- **line_end_style**
  
  Valid line ending styles are:

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLAT</td>
<td>The stroke is squared off at the endpoint of the path. There is no projection beyond the end of the path.</td>
</tr>
<tr>
<td>ROUND</td>
<td>A semicircular arc with a diameter equal to the line width is drawn around the endpoint and filled in.</td>
</tr>
<tr>
<td>SQUARE</td>
<td>The stroke continues beyond the endpoint of the path for a distance equal to half the line width and is squared off.</td>
</tr>
</tbody>
</table>

- **background**
  
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
GLineWidth(2);   // set line width (about 0.02")
                // for box border
GLineEndStyle(FLAT);          // set line ending style to FLAT
GHline(0.5, 4, 7.5);
    :
    :
ClosePage();
CloseDoc();
```
Graphic Line Join Type

Function

Sets the graphic line join style which specifies how two line segments are connected.

*Note: This GOCA feature needs the latest IPDS controller code, check with your IPDS printer vendors whether they have supported it.

Syntax

```cpp
void GLineJoinStyle(  
    ushort  ine_join_style,  
    bool   background = TRUE  
);  
```

Parameters

- **line_join_style**
  
  Valid line join styles are:

  ![MITER](image1.png)
  ![ROUND](image2.png)
  ![BEVEL](image3.png)

  The outer edges of the strokes for the two segments are extended until they meet at an angle, as in a picture frame. If the segments meet at too sharp an angle, a bevel join is used instead.

  An arc of a circle with a diameter equal to the line width is drawn around the point where the two segments meet, connecting the outer edges of the strokes for the two segments. This pieslice-shaped figure is filled in, producing a rounded corner.

  The two segments are finished with FLAT ending (see GLineEndStyle( ) function) and the resulting notch beyond the ends of the segments is filled with a triangle.

- **background**
  
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2);  // set line width (about 0.02")  
GLineJoinStyle(MITER);  // for box border

GBox(0.5, 4, 7.5, 1);  
:

ClosePage();  
CloseDoc();
```
Graphic Line Type

Function

Sets the type of line to be plotted with subsequent “Graphic Box”, “Graphic Circle”, “Graphic Ellipse”, “Graphic Arc”, “Graphic Piechart”, “Graphic Barchart”, “Graphic Fillet” and “Graphic Line” functions’ calls.

Syntax

```c
void GLineType(
    linetype    line_type,
    bool   background = TRUE
);
```

Parameters

- `line_type`:
  The type of line patterns to be plotted. Valid values are:
  - SOLID: Solid line.
  - DEFAULT: Solid line.
  - DOTTED: Dotted line.
  - SHORTDSH: Short dashed line.
  - DSHDOT: Dash-dot line.
  - DBLDOT: Double-dotted line.
  - LONGDASH: Long dashed line.
  - DSHDBLDOT: Dash-double-dot line.
  - INVSBLE: Invisible line.

- `background`:
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2);       // set line width (about 0.02")
                      // for box border
GLineType(DOTTED);   // set line type to dotted
                      // for box border
GBox(0.5, 4, 7.5,1, 0.1); // size 7.5"x1", at (0.5",4"),
                           // small rounded corners
                           :

ClosePage();
CloseDoc();
```
Graphic Line Width

Function

Sets the width of the line to be plotted with subsequent “Graphic Box”, “Graphic Circle”, “Graphic Ellipse”, “Graphic Arc”, “Graphic Piechart”, “Graphic Barchart”, “Graphic Fillet” and “Graphic Line” functions’ calls.

Syntax

```c
void GLineWidth(
    float   width,
    bool    background = TRUE
);
```

Parameters

- **line_width**
  Defines line width as a multiplier of the normal line width (which is approximately 0.01 inches). Supported values by IPDS printers may up to 13, which is about 0.13 inches.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2);       // set line width (about 0.02")
                     // for box border
GBox(0.5, 4, 7.5,1, 0.1); // size 7.5"x1", at (0.5",4"),
                          // small rounded corners

ClosePage();
CloseDoc();
```
Graphic Marker

Function

Plots a graphic marker with GOCA default size at the specified position.

Syntax

```c
void GMarker(
    float  x_pos,
    float  y_pos,
    marker  marker_type = DEFAULT,
    bool   background = TRUE
);
```

Parameters

- **x_pos, y_pos**
The X and Y positions of the marker.

- **Marker_type**
The type of marker symbol, valid values are:
  - DEFAULT: Same as a cross
  - CROSS: Cross
  - PLUS: Plus sign
  - DIAMOND: Hollow diamond
  - SQUARE: Hollow square
  - STAR6: 6-point star
  - STAR8: 8-point star
  - FILLDMND: Filled diamond
  - FILLSQR: Filled square
  - DOT: Solid dot
  - CIRCLE: Circle

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

: GMarker(2,3.5,PLUS); // plot a plus sign at (2",3.5")
: ClosePage();
CloseDoc();
```
Graphic Partial Ellipse

Function

Plots part of an ellipse at either the current graphic position or the position you specified. It does not change the current graphic position.

The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

At specify position:

```c
void GPartEllipse(
    float   x_pos,
    float   y_pos,
    float   xmaj,
    float   ymaj,
    float   xmin,
    float   ymin,
    float   startang,
    float   sweeping,
    bool    background = TRUE
);
```

At current position:

```c
void GpartEllipse2(
    float   xmaj,
    float   ymaj,
    float   xmin,
    float   ymin,
    float   startang,
    float   sweeping,
    bool    background = TRUE
);
```

Parameters

**x_pos**
The center position of the ellipse.

**y_pos**
The center position of the ellipse.

**Xmaj, ymaj**
The coordinates of the endpoint of the major axis of the ellipse. See the figure for the “Graphic Ellipse” function, P is the X major coordinate and S is the Y major coordinate.

**Xmin, ymin**
The coordinates of the endpoint of the minor axis of the ellipse. See the figure for the “Graphic Ellipse” function, R is the X minor coordinate and Q is the Y minor coordinate.

**startang**
The angle, measured clockwise, between the X-axis and the starting point of the ellipse.
**sweepang**
The angle, measured clockwise, between the two radii plotted from the center of the circle to the starting and ending points of the arc.

**background**
Specifies whether GOCA graphic is to be drawn as the background.
Graphic Pattern

Function

Specifies the type of filling pattern to be used for circles, arcs, ellipses, boxes, and a graphic area formed by paths.

Syntax

```cpp
void GPattern(
    type  pattern = SOLID,
    bool  background = TRUE
);
```

Parameters

- **pattern**
  Type of filling pattern, valid values are:
  - SOLID: Solid filling, default value
  - DOT1 through DOT8: Dotted patterns of decreasing density
  - VERTLN: Vertical lines
  - HORZLN: Horizontal lines
  - BLTR1, BLTR2: Diagonal lines from bottom-left to top-right
  - TLBR1, TLBR2: Diagonal lines from top-left to bottom-right
  - NOFILL: No filling pattern

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

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

GLineWidth(2);      // set line width (about 0.02")
// for box boundary
GPattern(DOT8);      // begin fill DOT8 pattern
GBgnArea();         // begin GOCA area
GBox(1,1,4, 7.5);   // draw 4x7.5" box at (1",1")
// box width = 7.5",
GEndArea();         // end GOCA area
ClosePage();
CloseDoc();
```
Graphic Piechart

Function

Plots a GOCA graphic piechart with its center of the pie at the current position, and the current default fonts are used for its label texts.

With the "Graphic Chart Colors" function, you can override the default RGB color values predefined for the graphic barchart and piechart.

GPiechartW() function is also provided for the piechart texts encoded in wild-char Unicode.

Syntax

```c
void GPiechart(
    float  width,
    float  height,
    float  shadow_depth,
    ushort  sectors,
    float*  values,
    char**  label_texts,
    float  start_angle = 0,
    label  label_layout = CIRCLE,
    encoding  encode = ASCII,
    bool   plot_boundary = FALSE,
    bool   background = TRUE
);
```

Parameters

width
The width of the piechart.

height
The height of the piechart.

shadow_depth
The shadow depth of the 3D piechart, specify value 0 for the 1D piechart.

sectors
The number of sectors of the piechart.

values
The array of the data values.

label_texts
The array of the label texts.

start_angle
The starting angle of piechart in degree, default is 0.

label_layout
Defines the layout of the piechart labels. Default is CIRCLE, labels are plotted around the piechart. Valid values are:

- CIRCLE Plots the labels around the piechart
- LEGEND Plots the percentages with label texts by the legend

encode
The encoding of the output texts of piechart, the valid values are:

- **ASCII**
  - Text is ASCII, uses ASCII font, this is the default value
- **A2E**
  - Converts text from ASCII to EBCDIC, uses EBCDIC font
- **GBK**
  - Text is GBK Chinese, uses ASCII/GBK-PC fonts
- **GBK2DBCS**
  - GBK will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **BIG5**
  - Text is BIG5 Chinese, uses ASCII/BIG5-PC fonts
- **BIG52DBCS**
  - BIG5 will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **SJIS**
  - Text is SJIS Japanese, uses ASCII/SJIS-PC fonts
- **SJIS2DBCS**
  - SJIS will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **KSC**
  - Text is KSC Korean, uses ASCII/KSC-PC fonts
- **KSC2DBCS**
  - KSC will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **DBCS**
  - Text is SBCS/DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **TOUTF16BE**
  - Converts legacy encoding data to wild-char big-endian, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly
- **TOUTF8**
  - Converts legacy encoding data to UTF-8, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly
- **UTF8**
  - Text is UTF-8, only uses OpenType/ TrueType font

**plot_boundary**
Defines whether the boundaries of the piechart are plotted, default is FALSE, does not plot boundary.

**background**
Specifies whether GOCA graphic is to be drawn as the background.

**Sample**

```c
float data[] = {45, 65, 35, 70, 40, 50};
char *label[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday");

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
Font(1); // select font 1 for piechart
LPI(8); // Set line space to 8 LPI for label texts
PiechartExplode(1, 0.1); // Explode sector 1, with distant 0.1"
PiechartLabel(0.10, "{percent}\%\n{label}"); // change defaults of label
GPos(2, 2); // Set current GOCA position
GPiechart(1.6, // Piechart width
1, // Piechart height
0.13, // 2D piechart shadow depth
6, // sectors of the piechart
data2, // Input datAn array
label, // Label text array
0, // start angle
PIELBL, // plots labels around piechart
ASCII, // ASCII text label
TRUE); // plots boundary

ClosePage();
CloseDoc();
```

**Output:**

![Pie chart sample](image_url)
Graphic Piechart Explode

Function

Optional function for the graphic piechart, defines which sector to be exploded from the pie and by how much.

This function must be called before the “Graphic Piechart” function, and you can call it multiple times if more than 1 sector needs to be explored. Label distance needs to be reset to 0 if that sector is not required to be exploded from the piechart anymore.

Syntax

```c
void GPiechartExplode(
    ushort sectorNo,
    float distance = DEFAULT
);
```

Parameters

- `sectorNo`
The sector number of the sector to be exploded from the piechart, starting from 1.
- `distance`
The explosion distance from the perimeter of the piechart. The default value is DEFAULT, the distance is determined automatically.

Sample

See sample for the Graphic Piechart function.
Graphic Piechart Label

Function

Optional functions for the graphic piechart, define the position and label format of the circular labels.

They must be called before the “Graphic Piechart” function.

Syntax

```c
void GPiechartLabel(
    float  lbl_distance,
    char  *  lbl_fmt = "\{label\}\n\{percent\}%",
    bool   draw_label_line = FALSE
);
```

```c
void GPiechartLabel2(
    float  *lbl_distances,
    ushort num_of_lbl_distances,
    char  *  lbl_fmt = "\{label\}\n\{percent\}%",
    bool   draw_label_line = FALSE
);
```

Parameters

**label_distance**
The distance between the sector perimeter and its label. A negative value means labels are inside the pie.

**label_distances**
Specifies an array in which keeps the variable distances between each sector perimeter and its label. A negative value means the label is inside the pie.

**num_of_lbl_distances**
Indicates the number of label distances.

**label_format**
Defines the format of the labels, MakeAFP Formatter uses parameter substitution to allow you to configure the information contained in the piechart labels and their format precisely. MakeAFP Formatter supports the following keyword fields:

```
{label}       The text label of the sector.
{value}       The data value of the sector.
{percent}     The percentage value of the sector.
\n         The new line control code.
```

You can change the label format by changing the format string, for instance, if you change the format to "\{label\}: US\{value\}K (\{percent\}%)", then the piechart label text will become something like "ABCD: US$123.23K (35.56%)".

**draw_label_line**
Defines whether draw the join lines to connect the piechart labels to the pie sectors, default is FALSE.
int main(void)
{
    float data[] = {10, 35, 40, 25, 30, 45, 55};
    float label_dist_array[7] = {0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1};
    char *label[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

    Start();                    // Start initiation
    SetUnit(IN_U600);           // Set default units to Inch, 600 dpi
    OpenDoc();
    OpenPage(8.5,11);
    Font(1);
    GPiechartLabel(label_dist_array, 6, "{label}\n\{percent}%", TRUE);
    LPI(8);                     // Set line space to 8 LPI for label texts
    GPos(5.7, 3.5);             // Set current GOCA position
    GPiechart(1.6, 1.6, 0, 6, data, label);       // Piechart width
                                                // Piechart height
                                                // No piechart shadow depth, so it is 1D piechart
                                                // Slices of piechart
                                                // Input datAn array
                                                // Label text array

    ClosePage();
    CloseDoc();

    Output:

    ![Pie Chart Output]
Graphic Piechart Legend

Function

The optional function for the graphic piechart defines how to position and format the legend labels of the graphic piechart.

This function must be called before the “Graphic Barchart” function is called if you want to override the default settings of the legend label.

Syntax

```c
void GPiechartLegend(
    float  x_rel_pos = DEFAULT,
    float  y_rel_pos  = DEFAULT,
    float      legend_box_size = DEFAULT
);
```

Parameters

- **x_rel_pos**
  Specifies the X relative position of the highest label of the piechart legend, relative to the coordinate origin position of the piechart. Default is using the X position determined by the Piechart function automatically.

- **y_rel_pos**
  Specifies the Y relative position of the highest label of the Piechart legend, relative to the coordinate origin position of the Piechart. Default is using the Y position determined by the Piechart function automatically.

- **Legend_box_size**
  Specifies the box size of the legend label. Default is using the box size determined by the Piechart function automatically.

Sample

None.
Graphic Polarchart

Function

Plots a GOCA graphic polar chart with its center of the circle at the current graphic position, and the current default font (if output encoding is ASCII, EBCDIC, UTF-8, or UTF-16) or fonts (if output encodings are mixed ASCII/DCBS-PC, or EBCDIC/DBCS-HOST) will be used for the label texts around the outer circle as well as the legend texts.

GPolarchartW() function is also provided for the polar chart texts encoded in wild-char Unicode.

Syntax

```c
void GPolarchart(
    float  radius,
    float  scale_unit,
    ushort  scales,
    float*  values,
    char**  label_texts,
    ushort  num_labels,
    ushort  axis_labels_fontID,
    encode encoding = ASCII,
    char*  axis_label_format = "%0.0f",
    ushort  num_groups = 1,
    float  line_weight = 2.0,
    uint*  line_colors = NULL,
    uint*  node_colors = NULL,
    uint   background_color = CYAN,
    ushort  color_mode = GOCA_OCA,
    char**  legend_texts = NULL,
    bool   background = true
);
```

Parameters

**height**
The radius of the outer circle.

**scale_unit**
The unit size of each scale on the radial axis.

**vertical_scales**
The number of scales on the radial axis.

**values**
The one dimension or multi-dimensions array of the data values.

**label_texts**
The array of label texts around the outer circle.

**num_labels**
The number of labels around the outer circle.
axis_labels_fontID
The font ID to be used to format the radial axis label texts.

encode
The encoding of the output label and legend texts, the valid values are:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>Text is ASCII, uses ASCII font, this is the default value</td>
</tr>
<tr>
<td>A2E</td>
<td>Converts text from ASCII to EBCDIC, uses EBCDIC font</td>
</tr>
<tr>
<td>GBK</td>
<td>Text is GBK Chinese, uses ASCII/GBK-PC fonts</td>
</tr>
<tr>
<td>GBK2DBC5</td>
<td>GBK will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>BIG5</td>
<td>Text is BIG5 Chinese, uses ASCII/BIG5-PC fonts</td>
</tr>
<tr>
<td>BIG52DBC5</td>
<td>BIG5 will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>SJIS</td>
<td>Text is SJIS Japanese, uses ASCII/SJIS-PC fonts</td>
</tr>
<tr>
<td>SJIS2DBC5</td>
<td>SJIS will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>KSC</td>
<td>Text is KSC Korean, uses ASCII/KSC-PC fonts</td>
</tr>
<tr>
<td>KSC2DBC5</td>
<td>KSC will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>DBCS</td>
<td>Text is SBCS/DBC5, uses SBCS-HOST/DBCS-HOST fonts</td>
</tr>
<tr>
<td>TUTF16BE</td>
<td>Converts legacy encoding data to wild-char big-endian, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly</td>
</tr>
<tr>
<td>TUTF8</td>
<td>Converts legacy encoding data to UTF-8, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly</td>
</tr>
<tr>
<td>UTF8</td>
<td>Text is UTF-8, only uses OpenType/ TrueType font</td>
</tr>
</tbody>
</table>

axis_label_format
Specifies the format-control string to format the axis labels, refer to the MSDN library for more details about the format control string used in C standard function fprintf().
Default is "%.0f".

num_groups
The number of the line groups.

line_weight
Defines line weight as a multiplier of the normal AFP GOCA line weight (which is approximately 0.01 inches). The default value is 2.0, which is about 0.02 inches.

line_colors
The one dimension array defining with color values, if you want to override the MakeAFP default color values in AFP OCA colors BLUE, RED, GREEN, CYAN.

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, GRAY.

RGB color values must be defined by format of 0x[red][green][blue], valid RGB intensity range values for each component are 0 through 255 (hex value x’FF’).

CMYK color values must be defined by format of 0x[cyan][magenta][yellow][black], valid CMYK percentage range values for each component are 0 through 100 (hex value x’64’).

node_colors
The one dimension array defining with color values of OCA, RGB, or CMYK.

RGB color values must be defined by format of 0x[red][green][blue], valid RGB intensity range values for each component are 0 through 255 (hex value x’FF’).
CMYK color values must be defined by format of 0x[cyan][magenta][yellow][black], valid CYMK percentage range values for each component are 0 through 100 (hex value x’64’).

**background_color**
Defines the circular background color. This value is being ignored if its value is NULL.

RGB color values must be defined by format of 0x[red][green][blue], valid RGB intensity range values for each component are 0 through 255 (hex value x’FF’).

CMYK color values must be defined by format of 0x[cyan][magenta][yellow][black], valid CYMK percentage range values for each component are 0 through 100 (hex value x’64’).

**color_mode**
Indicates the mode of color. Valid values are GOCA_OCA, GOCA_RGB, and GOCA_CMYK.

**Legend_texts**
The array of texts for the legend.

**background**
Specifies whether GOCA graphic is to be drawn as the background.

### Sample

```c
int main()
{
    float data2[2][6] = { {78, 80, 90, 80, 70, 75},
                          {35, 40, 55, 60, 40, 55} };
    char *lbls[] = {"Speed", "Reliability", "Comfort", "Safety",
                    "Efficiency", "Economy"};
    char *legend[] = { "Car-1", "Car-2" };
    Start();
    SetUnit(IN_U600);                  // set default units to inch, 600 dpi
    OpenDoc();
    OpenPage(8.27, 11.67);
    Font(2);                          // Set default font the legend and labels
                                      // around outer circle
    GPos(4.13, 1.7);                  // Set position of polar chart
    GPolarChartLegend(0.9, -1.1, 0.15);
    GPolarChart(1,                     // radius of polar chart
                20,                      // size per unit of radial axis
                5,                      // 5 units
               &data2[0][0],           // data by a 2D array
                lbls,                   // labels' texts
                6,                      // 6 labels
                1,                      // font ID for the radial axis texts
               ASCII,                   // texts are in ASCII encoding
               %0.0f",                  // format of axis label texts
                2,                      // draw 2 groups of lines
                2.0,                    // line width
                NULL,                   // by default colors for lines
                NULL,                   // by default colors for nodes
               CYAN,                    // circular background color
               GOCA_OCA,                // in RGB mode
```
legend);       // legend texts

ClosePage();
CloseDoc();
}

**Output:**

![Radar chart](image-url)
Graphic Polarchart Legend

Function

The optional function for the graphic polar chart defines how to position and format the legend labels of the polar chart.

This function must be called before the GPolarchart() function if you want to define the settings for the legend label.

Syntax

```c
void GPolarchartLegend(
    float       x_rel_pos = DEFAULT,
    float       y_rel_pos  = DEFAULT,
    float       line_length = DEFAULT,
    bool        vertical_layout = TRUE,
);```

Parameters

- **x_rel_pos**
  Specifies the X relative position of the highest label of the polar chart legend, relative to the coordinate origin position of the polar chart. MakeAFP Formatter does not present the legend of the polar chart if this value is 0.

- **y_rel_pos**
  Specifies the Y relative position of the highest label of the polar chart legend, relative to the coordinate origin position of the polar chart. MakeAFP Formatter does not present the legend of the polar chart if this value is 0.

- **line_length**
  Specifies the length of the legend line.

- **vertical_layout**
  Specified whether placing the legend labels vertically (from top to bottom) or horizontally (from left to right). The default value is TRUE, vertical layout.

Sample

None.
Graphic Position (Query)

Function

Queries the current graphic position on the page.

Syntax

float GXpos( );  \hspace{1cm} \text{Returns current graphic X position}

float GYpos( );  \hspace{1cm} \text{Returns current graphic Y position}

Sample

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.27, 11.67);
 :
 :
if ( GYpos() > 5.5 )  \hspace{1cm} // if current graphic Y position
\hspace{1cm} // more than 5.5"
{
  :
  :
} else
{
  :
  :
}
ClosePage();
CloseDoc();
Graphic Position

Function

Sets the current X and Y coordinates of the GOCA graphic.

Syntax

```cpp
void GPos(
    float x_pos,
    float y_pos,
    bool background = TRUE
);
```

Parameters

- **x_pos**
  The X coordinate of the graphic.

- **y_pos**
  The Y coordinate of the graphic.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

```cpp
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
GLineWidth(2); // set line width (about 0.02")
GPOS(3,4.5);  // set graphic position to (3",4.5")
GHline(2);    // then plot 2" horizontal line
: :
ClosePage();
CloseDoc();
```
Graphic Rendering Intent

Function

Specifies the color rendering intent for the GOCA vector graphics on the current page.

Refer to the latest Infoprint PPFA User’s Guide and Infprint Manager Procedures books for more about AFP color management.

Syntax

```c
void GRender(
    render_type     render_intent
);
```

Parameters

- **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.
    - **RELCM**
      - 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

```c
SetUnit(IN_U600);

OpenPage(8.5,11);
```
GRnder(SATUR);  // saturation rendering to GOCA
// graphics
GBox(1.3, 1.8, 3, 1.2, 0.15);  // Plot a graphic rounded box

ClosePage();
Graphic Skew Line

Function

Plots a skew line from either the current graphic position or the graphic position you specified.

The thickness and type of line are determined from the “Graphic Line Width” and “graphic Line Type” function calls.

Syntax

From specify position:
void GSline(
    float  from_x,
    float  from_y,
    float  to_x,
    float  to_y,
    bool   background = TRUE
);

From current position:
void GSline2(
    float  to_x,
    float  to_y,
    bool   background = TRUE
);

Parameters

from_x
The X starting position of the slant line.

from_y
The Y starting position of the slant line.

to_x
The X ending position of the slant line.

To_y
The Y ending position of the slant line.

background
Specifies whether GOCA graphic is to be drawn as the background.

Sample

SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
    GLineWidth(2)                   // line width is about 0.02"
    GSslant(10,10,30,40);  // plot slant line from (10,10) mm to
                          // (30,40) mm
    ClosePage();
    CloseDoc();
Graphic Stacked Barchart

Function

Plots a GOCA graphic stacked barchart with its axis coordinates at the current graphic position, and the current default font is used for its texts.

With the “Graphic Barchart Element” and “graphic Barchart Legend” functions, you can control how to place the legend labels and indicate whether to present the element labels or not.

With the “Graphic Chart Colors” function, you can override the default RGB color values predefined for the graphic barchart and piechart.

GStackchartW() function is also provided for the stack chart texts encoded in wild-char Unicode.

Syntax

```c
void GStackchart(
    float  height,
    float  bar_width,
    float  gap_barset,
    uint   vertical_scale_unit,
    ushort vertical_scales_positive,
    ushort vertical_scales_negative,
    ushort number_bars,
    ushort bar_sets,
    float* values,
    char** horizontal_labels,
    char** legend_texts,
    encoding encode = ASCII,
    bool   gridline = TRUE,
    bool   showValues = TRUE,
    bool   font_id = DEFAULT,
    bool   background = TRUE
);
```

Parameters

- **height**
  The height of the barchart graphic.

- **bar_width**
  The width of each bar.

- **gap_barset**
  The gap of each bar set.

- **vertical_scale_unit**
  The unit size of each scale on the vertical axis.

- **vertical_scales_positive**
  The number of scales on the vertical axis for positive values.
**vertical_scales_negative**
The number of scales on the vertical axis for negative values.

**number_bars**
The number of bar elements in each bar set.

**Bar_sets**
The number of bar-sets.

**values**
The one dimension or two dimensions array of the data values. With each data-set, allot positive values first and then negative values according to their top-down presenting orders.

**horizontal_labels**
The array of label texts for the horizontal axis. You can insert a new line control code \n or \x0a to split the text into multiple lines on output. This value is ignored if your number of bars is 1.

**legend_texts**
The array of texts for the legend.

**encode**
The encoding of the output texts, the valid values are:

- **ASCII** Text is ASCII, uses ASCII font, this is the default value
- **A2E** Converts text from ASCII to EBCDIC, uses EBCDIC font
- **GBK** Text is GBK Chinese, uses ASCII/GBK-PC fonts
- **GBK2DBC** GBK will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **BIG5** Text is BIG5 Chinese, uses ASCII/BIG5-PC fonts
- **BIG52DBC** BIG5 will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **SJIS** Text is SJIS Japanese, uses ASCII/SJIS-PC fonts
- **SJIS2DBC** SJIS will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **KSC** Text is KSC Korean, uses ASCII/KSC-PC fonts
- **KSC2DBC** KSC will be converted to DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **DBCS** Text is SBCS/DBCS, uses SBCS-HOST/DBCS-HOST fonts
- **TOUTF16BE** Converts legacy encoding data to wild-char big-endian, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly
- **TOUTF8** Converts legacy encoding data to UTF-8, only uses OpenType/ TrueType font, make sure function DefaultCode() is called properly
- **UTF8** Text is UTF-8, only uses OpenType/ TrueType font

**gridline**
Specifies whether plot the grid lines on the stack chart, default is TRUE.

**showValues**
Specifies whether show value figures on the stack chart, default is TRUE.

**font_id**
Specifies a font ID number to be used to present the total value of each stacked bar. Default is using the current ID of the font in 1-byte, UTF-8, or wild-char encoding.

**background**
Specifies whether GOCA graphic is to be drawn as the background.
Sample 1

```c
int main()
{
    float data[] = {45.00, 55.45, -27.73,
                    45.00, 67.34, -33.67,
                    45.00, 50.68, -25.34};

    char *lbls2[] = {"Sept", "Oct", "Nov");
    char *legend[] = {"Fix Charges", "Call Charges", "Discount");

    Start();
    SetUnit(IN_U600);
    OpenDoc();

    OpenPage(8.5,11); // page size is 8.5" by 11"
    GPos(1, 2.3); // Set position of barchart
    Font(1); // select font 1 for barchart
    BarchartElement(ABOVE, "%.2f"); // Defines for element

    GBarchartLegend(0.3, // plus relative 1.3" for X
                    0.4, // plus relative 0.4" for Y
                    0.10, // legend box width 0.1" 
                    0.10, // legend box height 0.1" 
                    false, // place legend horizontally
                    2); // show legend text by 2nd font

    GPos(1, 3); // set position of barchart

    GStackchart(1.8, // height of barchart
               0.7, // width of each bar
               0.5, // gap between each bar
               20, // unit size of v-axis scale
               7, // number of v-axis scales for positive values
               2, // number of v-axis scales for negative values
               3, // number of stacks per bar
               3, // number of bars
               data, // data array of input data
               lbls, // label texts below H-axis
               legend2, // legend texts
               ASCII, // ASCII text
               true, // plot grid lines
               true, // show value figures
               2); // font ID for the total value of bar

    ClosePage();
    CloseDoc();

    #ifdef _DEBUG
    ViewAFP(); // View AFP file only in debug mode
    #endif

    return 0;
}
```
**Graphic Vertical Boxes**

**Function**

Repeat plotting of graphic boxes vertically from the position you specified. The thickness and type of border are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

**Syntax**

```c
void GVboxes(
    float  x_pos,
    float  y_pos,
    float  width,
    float  height,
    ushort repeat,
    float  space,
    ushort corner_radius = 0,
    bool   background = TRUE
);
```

**Parameters**

- **x_pos**
  The X position of the top left corner of the 1st box.
- **y_pos**
  The Y position of the top left corner of the 1st box.
- **width**
  The width of the box.
- **height**
  The height of the box.
- **repeat**
  The number of additional boxes to be repeated.
- **space**
  The gap space between the boxes.
- **corner_radius**
  The radius of rounded box corners.
- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

**Sample**

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

GVboxes(10,10,20,5,9,2); // repeat 10 boxes vertically from
// (10,10)mm, box width is 20mm, height
// is 5mm, 2mm space between boxes

ClosePage();
CloseDoc();
```
Graphic Vertical Line

Function

Plots a graphic vertical line from either the current graphic position or the graphic position you specified.

The thickness and type of line are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

From specify position:

```c
void GVline(
    float  x_pos,
    float  y_pos,
    float  length,
    bool   background = TRUE
);
```

From current position:

```c
void GVline2(
    float  length,
    bool   background = TRUE
);
```

Parameters

- **x_pos**
  The X starting position of the line.

- **y_pos**
  The Y starting position of the line.

- **length**
  The length of the line.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
:
GVline(10,10,80); // plot a vertical line from (10,10) mm, // 80 mm length
:
:
ClosePage();
CloseDoc();
```
Graphic Vertical Lines

Function

Repeat plotting of graphic vertical lines from the graphic position you specified.

The thickness and type of line are determined from the “Graphic Line Width” and “Graphic Line Type” function calls.

Syntax

```c
void GVlines(
    float  x_pos,
    float  y_pos,
    float  length,
    ushort repeat,
    float  space
    ushort direction = ACROSS,  
    bool   background = TRUE
);```

Parameters

- **x_pos**
  The X starting position of the line.

- **y_pos**
  The Y starting position of the line.

- **length**
  The length 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.

- **background**
  Specifies whether GOCA graphic is to be drawn as the background.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
:
GVlines(10,10,80,9,5); // plot 10 vertical lines from (10,10)
// length is 80 mm, 5 mm space between
// lines
:
:
ClosePage();
CloseDoc();
```
Horizontal Boxes

Function

Repeat drawing boxes horizontally.

Syntax

```c
void Hboxes(
    float x_pos,
    float y_pos,
    float width,
    float height,
    float thickness,
    ushort repeat,
    float space,
);
```

Parameters

- **x_pos**
  - The X position of the top left corner of the first box, specify CP if you want to use the current position.

- **y_pos**
  - The Y position of the top left corner of the first box, specify CP if you want to use the current position.

- **width**
  - The width of the box.

- **height**
  - The height of the box.

- **thickness**
  - The thickness of the box border.

- **repeat**
  - Number of boxes to be repeated if you want to draw more than one box of the same size and border thickness.

- **space**
  - The gap space between the boxes.

Sample

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

Color(RED); // defines color for texts and legacy boxes
Hboxes(10,10,20,5,1,9,2); // repeat 10 boxes horizontally from (10,10)mm, box width is 20mm, height is 5mm, thickness is 1mm, 2mm space between boxes, red color border

ClosePage();
CloseDoc();
```
Horizontal Line

Function

Draws a horizontal line.

Syntax

```cpp
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

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

Color(BLUE);               // defines color for texts and legacy line
Hline(10, 10, 100, 1);     // draw 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

```c
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

```c
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();
```
Horizontal Shades

Function

Repeat drawing of shades horizontally.

Syntax

```c
void Hshades(
  float  x_pos,
  float  y_pos,
  float  width,
  float  height,
  float  shading_percentage,
  ushort  repeat,
  float  space,
  pattern  shading_pattern = LED,
  ocaColor  color = BLACK
);
```

Parameters

- **x_pos**
  The X position of the top left corner of the first shade.

- **y_pos**
  The Y position of the top left corner of the first shade.

- **width**
  The width of the shade.

- **height**
  The height of the shade.

- **Shading_percentage**
  The percentage of shading.

- **repeat**
  The number of additional shades to be repeated.

- **space**
  The amount of space you want between the shades.

- **Shading_pattern**
  You can define between three shading patterns depending on the printer type you are using. Valid values are STD, SCREEN, and LED (default value).

- **color**
  Any of the defined AFP OCA color values: BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY, the default value is BLACK.

Sample

```c
SetUnit(MM_U600);
   :
OpenPage(220,297);
   :
```
Hshades(10,10,20,5,15,9,2);  // repeat 10 shades horizontally from
// (10,10)mm, shades width is 20mm, height
// is 5mm, shading percentage is 15,
// 2mm space between shades

ClosePage();
Inch Value

Function

Specifies a value in inches.

Syntax

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

Parameters

value

The value in inches.

Sample

```c
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();
```
Import AFP Image Object

Function

Imports a previously created AFP page segment and then embed it inline within the AFP as a part of the document data stream at the specified position. The input filename must be fully qualified with the pathname and must be accessible with appropriate read permission.

Syntax

```c
void ImportAFPimg(
    char*  in_file,
    float  x_pos,
    float  y_pos,
    degree  degree = DEG0,
    ocaColor  color = BLACK
);
```

Parameters

- **in_file**
  The name of the input file, fully qualified with the pathname. It must be available to MakeAFP Formatter at the time of formatting.

- **x_pos**
  The X position of the AFP image.

- **y_pos**
  The Y position of the AFP image.

- **degree**
  The rotation for the image. 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

- **ocaColor**
  Any of the defined AFP OCA color values: 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.

With this color parameter, you can change the monochrome color of the IOCA image.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
:
ImportAFPimg("d:\afpimg\S1TEST01",10,10);  // Import AFP image from a
// page segment file, place
// at (10,10) mm
:
```
Import Data-Object Container

Function

Imports a data-object container (such as EPS, GIF, JPEG, PDF, TIFF, etc) and embeds it inline directly within the AFP as a part of the document data stream at the specified position. The data-object filename must be fully qualified with the pathname and must be accessible with appropriate read permission.

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

Syntax

```c
void ImportObjt(
    char*     image_file,
    float     x_pos,
    float     y_pos,
    float     width = DEFAULT,
    float     height = DEFAULT,
    mapping   object_mapping = FIT,
    ushort    cmr_id = 0,
    cmr_mode  process_mode = AUDIT,
    render_type rendering_intent = NONE,
    degree    degree = DEG0
);
```

Parameters

**image_file**
The name of the image file with image type file extension, fully qualified with the pathname in your hard disk. It must be available to MakeAFP Formatter at the time of formatting.

The following image file extensions are supported:

<table>
<thead>
<tr>
<th>Description</th>
<th>File Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encapsulated Postscript</td>
<td>*.EPS</td>
</tr>
<tr>
<td>Graphics Interchange Format</td>
<td>*.GIF</td>
</tr>
<tr>
<td>JPEG file Interchange Format</td>
<td>*.JPG</td>
</tr>
<tr>
<td>PDF Single Page Object</td>
<td>*.PDF</td>
</tr>
<tr>
<td>Tagged Image File Format</td>
<td>*.TIF</td>
</tr>
</tbody>
</table>

**x_pos**
The X position of the image.

**y_pos**
The Y position of the image.

**width**
The width of the object placement area, the DEFAULT is the width specified in the data-object image.
height
The height of the object placement area, the DEFAULT is the height specified in the data-object image.

object_mapping
The mapping of the object to the object placement area, DEFAULT is the mapping option within the object is used. The default value is FIT, valid options are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CENTER</td>
<td>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.</td>
</tr>
<tr>
<td>LEFT</td>
<td>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 &amp; object_height parameters is not trimmed and could cause an exception condition by the IPDS printer.</td>
</tr>
<tr>
<td>FILL</td>
<td>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 totally 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.</td>
</tr>
<tr>
<td>FIT</td>
<td>Specifies scale to fit. The object is to be scaled to fit within the object placement area, as defined by the object_width &amp; 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.</td>
</tr>
</tbody>
</table>

cmr_id
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Formatter 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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT</td>
<td>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).</td>
</tr>
</tbody>
</table>
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.

- **RELCM**  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.

**degree**

The rotation for the image. 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
Sample

SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
:  
ImportObjt("d:\\images\\TEST01.jpg",10,10);  // Import JPEG image and  
// place at(10mm,10mm)
  
ClosePage();
CloseDoc();
Import Data-Object Container from Memory Buffer

Function

Imports a data-object container (such as EPS, GIF, JPEG, PDF, TIFF, etc) from a memory buffer and then put it inline directly within the AFP as a part of the document data stream at the specified position.

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

Syntax

```c
void ImportObjtBuf(
    char*        object_memory_buffer,
    float        object_stream_size,
    float        x_pos,
    float        y_pos,
    float        width = DEFAULT,
    float        height = DEFAULT,
    mapping    object_mapping = FIT,
    ushort     cmr_id = 0,
    cmr_mode    process_mode = AUDIT,
    render_type    rendering_intent = NONE,
    degree        degree = DEG0
);
```

Parameters

**object_memory_buffer**
The address of the memory buffer of the data-object container. The following formats are supported:

<table>
<thead>
<tr>
<th>Description</th>
<th>File Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encapsulated Postscript</td>
<td>*.EPS</td>
</tr>
<tr>
<td>Graphics Interchange Format</td>
<td>*.GIF</td>
</tr>
<tr>
<td>JPEG file Interchange Format</td>
<td>*.JPG</td>
</tr>
<tr>
<td>PDF Single Page Object</td>
<td>*.PDF</td>
</tr>
<tr>
<td>Tagged Image File Format</td>
<td>*.TIF</td>
</tr>
</tbody>
</table>

**object_stream_size**
The size of the data-object stream is stored in the memory buffer.

**x_pos**
The X position of the image.

**y_pos**
The Y position of the image.

**width**
The width of the object placement area, the DEFAULT is the width specified in the data-object image.
**height**
The height of the object placement area, the DEFAULT is the height specified in the data-object image.

**object_mapping**
The mapping of the object to the object placement area, DEFAULT is the mapping option within the object is used. 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.

**cmr_id**
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Formatter 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).
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.

**RELCM**
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.

**degree**
The rotation for the image. 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

```c
struct stat buf, buf2;
int result;
char *mem_buffer,
SetUnit(IN_U600);
OpenDoc();

OpenPage(220, 297);

if ((result = stat("d:\test\image1.jpg", &buf) == 0)) {
    if (mem_buffer = (char*)malloc(buf.st_size * sizeof(char))) {
        if ((FILE *stream = fopen("image1.eps", "rb")) != 0) {
            fread(mem_buffer, sizeof(BYTE), buf.st_size, stream);
            fclose(stream);
        }
    }
}

ImportObjtBuf(mem_buffer, buf.st_size, 1.2, 1.4);

ClosePage();
CloseDoc();
```
Import Data-Object Image from Multi-page TIFF

Function

Imports a TIFF image from the previously created multiple-page TIFF image and embeds it inline directly within the AFP document data stream at the specified position. The TIFF image filename must be fully qualified with the pathname and must be accessible with appropriate read permission.

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

Syntax

```c
void ImportTiffObjt(
    char*        tiff_file,
    ushort     page_index,
    float        x_pos,
    float        y_pos,
    float        width = DEFAULT,
    float        height = DEFAULT,
    mapping    object_mapping = FIT,
    ushort     cmr_id = 0,
    cmr_mode    process_mode = AUDIT,
    render_type    rendering_intent = NONE,
    degree        degree = DEG0
);```

Parameters

tiff_file
The name of the TIFF image, fully qualified with the pathname in your hard disk. It must be available to MakeAFP Formatter at the time of formatting.

page_index
The index number of an image starts from 0 which is referred to as the first available image. With the TiffPages function, you can count the number of images in a multiple-page TIFF file.

x_pos
The X position of the image.

y_pos
The Y position of the image.

width
The width of the object placement area, the DEFAULT is the width specified in the data-object image.

height
The height of the object placement area, the DEFAULT is the height specified in the data-object image.

object_mapping
The mapping of the object to the object placement area, DEFAULT is the mapping option within the object is used. The default value is FIT, valid options are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CENTER</td>
<td>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.</td>
</tr>
<tr>
<td>LEFT</td>
<td>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 &amp; object_height parameters is not trimmed and could cause an exception condition by the IPDS printer.</td>
</tr>
<tr>
<td>FILL</td>
<td>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.</td>
</tr>
<tr>
<td>FIT</td>
<td>Specifies scale to fit. The object is to be scaled to fit within the object placement area, as defined by the object_width &amp; 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.</td>
</tr>
</tbody>
</table>

**cmr_id**

The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Formatter 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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT</td>
<td>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).</td>
</tr>
</tbody>
</table>

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).

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

RELCM  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.

define degree

The rotation for the image. 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

```c
SetUnit(MM_U600);
OpenDoc();
:

int pages = TiffPages("pages.tif");
```
for (int i=0; i< pages; i++)
{
    OpenPage(8.27,11.67);    // A4 page size
    ImportTiffObjt("pages.tif", // TIFF image file name
                    i,       // TIFF page index No.
                    0,       // X position
                    0);      // Y position
    ClosePage();             // End of a AFP page
}

CloseDoc();
Import Image with Conversion to AFP IOCA FS11

Function

Imports a previously created image and embed it inline within the AFP document data stream at the specified position as an AFP IOCA FS11 image object. The image file name must be fully qualified with the pathname and must be accessible with appropriate read permission.

This function is developed for your legacy IPDS printer. Although this function does import over 20 of the most popular raster image file formats from open systems, to achieve the best formatting performance, pre-conversion from image format to AFP page segment by MakeAFP img2psg utility is recommended. You can use the “Include Page Segment” function to reference an AFP page segment on a page or use the “Import AFP Image” function to embed the AFP IOCA image inline within your AFP file.

Syntax

```c
void ImportImg(
    char*    image_file,
    float    x_pos,
    float    y_pos,
    ushort   dpi_resolution = DEFAULT,
    float    scale = 100,
    halftone halftone_type = 0,
    float    brightness = 0,
    float    contrast = 0,
    threshold threshold_value = 0,
    bool     invert = FALSE,
    degree   degree = DEG0,
    ocaclolor color = BLACK
);
```

Parameters

`image_file`

The name of the image, fully qualified with the pathname in your hard disk. It must be available to MakeAFP Formatter at the time of formatting. The following bitmap image formats are supported:

<table>
<thead>
<tr>
<th>Description</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows or OS/2 Bitmap File</td>
<td>*.BMP</td>
</tr>
<tr>
<td>Graphics Interchange Format</td>
<td>*.GIF</td>
</tr>
<tr>
<td>JPEG Network Graphics</td>
<td>*.JNG</td>
</tr>
<tr>
<td>Independent JPEG Group</td>
<td>*.JPG</td>
</tr>
<tr>
<td>Commodore 64 Koala format</td>
<td>*.KOA</td>
</tr>
<tr>
<td>Multiple Network Graphics</td>
<td>*.MNG</td>
</tr>
<tr>
<td>Kodak PhotoCD</td>
<td>*.PCD</td>
</tr>
<tr>
<td>PCX bitmap format</td>
<td>*.PCX</td>
</tr>
<tr>
<td>Portable Network Graphics</td>
<td>*.PNG</td>
</tr>
</tbody>
</table>
Photoshop *.PSD
Silicon Graphics SGI image *.SGI
Targa files *.TGA
Tagged Image File Format *.TIFF

### x_pos
The X position of the converted IOCA image.

### y_pos
The Y position of the converted IOCA image.

### dpi_resolution
The resolution of your output image.

### scale
Magnification relative to the original image size by percentage. Numbers below 100 mean reductions, while numbers above 100 mean enlargement, default is 100 meaning original dimension is kept. For instance, if your image is 300 dpi grayscale and printer resolution is 600 dpi, for better printing quality and to keep the same image dimension, you have to scale it up 200 percent before it can be converted into the bi-level AFP raster image.

### brightness
The amount of percentage to change the brightness. The brightness ranges from -100 to 100. A positive value increases (or lightens) the brightness of the bitmap image. A negative value decreases (or darkens) the brightness of the bitmap image. Default is 0.

### contrast
the amount of percentage to change the contrast. The contrast ranges from -100 to 100. A positive value increases the contrast of the bitmap image. A negative value decreases the contrast of the bitmap image. Default is 0.

### halftone
Indicates whether the image will be converted into a halftone raster image. For printing of photographic images, halftone printing may provide smoother printout quality with a smaller AFP image data stream size. The halftone ranges from 0 to 3, the default is 0 without halftone.

### threshold
Indicates whether the image will be converted into a 1-bit monochrome bitmap by using a threshold value between 1 to 255. Any brightness level that is less than the value is set to zero, otherwise to 1. This option can be used with the text type of image.

### invert
Specifies whether the AFP IOCA monochrome bitmap image is inverted.

### degree
The rotation for the converted IOCA image. 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
color
Any of the defined AFP OCA color values: 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
SetUnit(MM_U600);
OpenPage(220,297);

  ImportImg("d:\jpegimg\test02.jpg", // 300 dpi JPEG image input,
            10, 10,             // place image at (10mm, 10mm)
            600, 200);          // with 600 dpi, scale size 200%

ClosePage();
Import Image from Memory Buffer with Conversion to AFP IOCA FS11

Function

Imports an image from a memory buffer and then put it inline within the AFP document data stream at the specified position as an AFP IOCA FS11 image object.

Syntax

```c
void ImportImgBuf(
    char*     imgage_memory_buffer,
    uint*     imgage_stream_size,
    float     x_pos,
    float     y_pos,
    ushort    dpi_resolution = DEFAULT,
    float     scale = 100,
    halftone  halftone_type = 0,
    float     brightness = 0,
    float     contrast = 0,
    threshold threshold_value = 0,
    bool      invert = FALSE,
    degree    degree = DEG0,
    ocaclolor color = BLACK
);
```

Parameters

- **image_memory_buffer**
  The address of the image’s memory buffer. The following image formats are supported:

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows or OS/2 Bitmap</td>
</tr>
<tr>
<td>Graphics Interchange Format</td>
</tr>
<tr>
<td>JPEG Network Graphics</td>
</tr>
<tr>
<td>Independent JPEG Group</td>
</tr>
<tr>
<td>JPEG-2000 codestream</td>
</tr>
<tr>
<td>JPEG-2000 Format</td>
</tr>
<tr>
<td>Commodore 64 Koala format</td>
</tr>
<tr>
<td>Multiple Network Graphics</td>
</tr>
<tr>
<td>Kodak PhotoCD</td>
</tr>
<tr>
<td>PCX bitmap format</td>
</tr>
<tr>
<td>Portable Network Graphics</td>
</tr>
<tr>
<td>Photoshop</td>
</tr>
<tr>
<td>Silicon Graphics SGI image</td>
</tr>
<tr>
<td>Targa files</td>
</tr>
<tr>
<td>Tagged Image File Format</td>
</tr>
</tbody>
</table>

- **image_stream_size**
  The size of the image stream is stored in the memory buffer.

- **x_pos**
  The X position of the converted IOCA image.
**y_pos**  
The Y position of the converted IOCA image.

**dpi_resolution**  
The resolution of your output image.

**scale**  
Magnification relative to the original image size by percentage. Numbers below 100 mean reductions, while numbers above 100 mean enlargement, default is 100 meaning original dimension is kept. For instance, if your image is 300 dpi grayscale and printer resolution is 600 dpi, for better printing quality and to keep the same image dimension, you have to scale it up 200 percent before it can be converted into the bi-level AFP raster image.

**brightness**  
The amount of percentage to change the brightness. The brightness ranges from -100 to 100. A positive value increases (or lightens) the brightness of the bitmap image. A negative value decreases (or darkens) the brightness of the bitmap image. Default is 0.

**contrast**  
The amount of percentage to change the contrast. The contrast ranges from -100 to 100. A positive value increases the contrast of the bitmap image. A negative value decreases the contrast of the bitmap image. Default is 0.

**halftone**  
Indicates whether the image will be converted into a halftone raster image. For printing of photographic images, halftone printing may provide smoother printout quality with a smaller AFP image data stream size. The halftone ranges from 0 to 3, the default is 0 without halftone.

**threshold**  
Indicates whether the image will be converted into a 1-bit monochrome bitmap by using a threshold value between 1 and 255. Any brightness level that is less than the value is set to zero, otherwise to 1. This option can be used with the text type of image.

**invert**  
Specifies whether the AFP IOCA monochrome bitmap image is inverted.

**degree**  
The rotation for the converted IOCA image. 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

**ocaColor**  
Any of the defined AFP OCA color values: 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**

```c
struct stat buf, buf2;
int result;
char *mem_buffer,
```
if ((result = stat("d:\test\image1.jpg", &buf) == 0)) {
    if (mem_buffer = (char*)malloc(buf.st_size * sizeof(char))) {
        if ((FILE *stream = fopen("image1.eps", "rb") != 0) {
            fread(mem_buffer, sizeof(BYTE), buf.st_size, stream);
            fclose(stream);
        }
    }
}

ImportImgBuf(mem_buffer, buf.st_size, 1.2, 1.4);
Import Image from Multi-page TIFF with Conversion to AFP IOCA FS11

Function

Imports a TIFF image from the previously created multiple-page TIFF image and embed it inline within the AFP document data stream at the specified position as an AFP IOCA image object. The TIFF image file name must be fully qualified with the pathname and must be accessible with appropriate read permission.

Syntax

```c
void ImportTiff(
    char*    tiff_name,
    ushort   page_index,
    float    x_pos,
    float    y_pos,
    ushort   dpi_resolution,
    float    scale = 100,
    halftone_type halftone_type = 0,
    float    brightness = 0,
    float    contrast = 0,
    threshold threshold_value = 0,
    bool     invert = FALSE,
    degree   degree = DEG0,
    ocaclolor color = BLACK
);
```

Parameters

tiff_name
The name of the TIFF image, fully qualified with the pathname in your hard disk. It must be available to MakeAFP Formatter at the time of formatting.

page_index
The index number of an image starts from 0 that is refers to the first available image. With the TiffPages function, you can count the number of images in a multiple-page TIFF file.

x_pos
The X position of the converted IOCA image.

y_pos
The Y position of the converted IOCA image.

dpi_resolution
The resolution of your output image.

scale
Magnification relative to the original image size by percentage. Numbers below 100 mean reductions, while numbers above 100 mean enlargement, default is 100 meaning original dimension is kept. For instance, if your image is 300 dpi grayscale and printer resolution is 600 dpi, for better printing quality and to keep the same image dimension, you have to scale it up 200 percent before it can be converted into the bi-level AFP raster image.
brightness
The amount of percentage to change the brightness. The brightness ranges from -100 to 100. A positive value increases (or lightens) the brightness of the bitmap image. A negative value decreases (or darkens) the brightness of the bitmap image. Default is 0.

contrast
the amount of percentage to change the contrast. The contrast ranges from -100 to 100. A positive value increases the contrast of the bitmap image. A negative value decreases the contrast of the bitmap image. Default is 0.

halftone
Indicates whether the image will be converted into a halftone raster image. For printing of photographic images, halftone printing may provide smoother printout quality with a smaller AFP image data stream size. The halftone ranges from 0 to 3, the default is 0 without halftone.

threshold
Indicates whether the image will be converted into a 1-bit monochrome bitmap by using a threshold value between 1 to 255. Any brightness level that is less than the value is set to zero, otherwise to 1. This option can be used with the text type of image.

invert
Specifies whether the AFP IOCA monochrome bitmap image is inverted.

degree
The rotation for the converted IOCA image. 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

color
Any of the defined AFP OCA color values: 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

```c
Start();       // Start initiation
OpenDoc();                 // Open AFP document */
SetUnit(IN_U300);          // Set default units to inch, 300 dpi

int pages = TiffPages("pages.tif");

for (int i=0; i< pages; i++)
{
    OpenPage(8.27,11.67);   // A4 page size

    ImportTiff("pages.tif", // TIFF image file name
               i,        // TIFF page index No.
               0,        // X position
               0,        // Y position
               300);     // As 300 dpi inline IOCA image
```
ClosePage(); // End of a AFP page
}

CloseDoc(); // End of AFP document
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

```c
void Inc1Objt(
    char*   object_name,
    float   object_xpos,
    float   object_ypos,
    float   object_width = DEFAULT,
    float   object_height = 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

**object _name**
The name of the data-object previously defined in your MakeAFP Formatter 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 Formatter finds more than one data-object image with the same file base-name in the same resource 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. 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.

**object_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.

**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 totally 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 BCBCA barcode object.

cmr_id
The ID number of a CMR (Color Management Resource) is defined in your MakeAFP Formatter 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.

- **RELCM**
  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 colors, 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**

```java
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 definition file

InclObjt("FLOWER02");  // Include a TIFF image at current
// position, image type TIFF is defined
// in the MakeAFP Formatter definition
// file

ClosePage();
CloseDoc();
Include Overlay

Function

Includes a reference to an overlay at the specified position or current position. You can include up to 256 unique page overlays on a page.

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

Syntax

```c
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

```c
SetUnit(MM_U600);

OpenDoc();
OpenPage(220,297);

InclOvly("O1TEST01",10,10);  // Include an overlay at (10,10)
```


InclOvly("01TEST02"); // 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 256 unique page segments on a page.

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

Syntax

```c
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

```c
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();
```
Japanese String Width

Function

Measures and then returns the width of the Japanese SJIS string in the given fonts. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” functions before you call this function.

Syntax

```c
float JpWidth(
    char*   sjis_data,
    ushort  font1 = DEFAULT,
    ushort  font2 = DEFAULT
);```

Parameters

sjis_data
The string of Japanese SJIS characters to measure.

font1
The ID number of ASCII or SBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the font in 1-byte, UTF-8, or wild-char encoding.

font2
The ID number of SJIS or DBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the 2-byte font.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Font((4,5); // font 4 is 1-byte font
    // font 5 is 2-byte font
    :
    float w;
    w = JpWidth("Alphabet が混在した文章のサンプルです");
    :
    :
ClosePage();
CloseDoc();```
Korean String Width

Function

Measures and then returns the width of the Korean KSC string in the given fonts. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” functions before you call this function.

Syntax

```c
float KrWidth(
    char*   ksc_data,
    ushort  font1 = DEFAULT,  
    ushort  font2 = DEFAULT
);
```

Parameters

- **ksc_data**
The string of Japanese KSC characters to measure.
- **font1**
The ID number of ASCII or SBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the font in 1-byte, UTF-8, or wild-char encoding.
- **font2**
The ID number of KSC or DBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the 2-byte font.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
  :
  :
  
Font((4,5);        // font 4 is 1-byte font
  :
     // font 5 is 2-byte font
  :
float w;

w = KrWidth("함께해요~' 이벤트, 온 가족의 티셔츠가 내 품에!");    // measure KSC width by
      // current fonts

ClosePage();
CloseDoc();
```
Left Align Single-Byte 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 Formatter 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

```c
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 to which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
    Pos(2,2); // current position to (2",2")
    Font(3); // assume font 3 is ASCII font
    Ltxt("texts are left-aligned"); // left put text at (2",2")
: ClosePage();
CloseDoc();
```
Left Align Japanese

Function

Left aligns a single-line of the Japanese ASCII/SJIS-PC 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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
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 to which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);   // position to (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 ASCII/KSC-PC 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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

**Syntax**

```cpp
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 to which the next character would be placed.

**Sample**

```cpp
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);   // position to (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 ASCII/GBK-PC 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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
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 to which the next character would be placed.

Sample

```c
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 to (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 ASCII/BIG5-PC 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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
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 to which the next character would be placed.

Sample

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

Pos(2,2);   // current position to (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 Text

Function

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

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

Syntax

```c
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 to which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);                     // current position to (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 Wild-Char Text

Function

Left aligns a single-line of the wild-char string at the current position. Native wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Lw(
    wchar_t* data,
    bool same_pos = TRUE
);
```

Parameters

data
The NULL-terminated wild-char 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

```c
/* wild-char string, "test" and CJK characters "测试" */
wchar_t data1[20] = {0x0074, 0x0065, 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 UTF16BE defined
Lw(data1);                        // left put wild-char at (2",2")
    :
    :
ClosePage();
CloseDoc();
```
Left Align Wild_char Text Converting from Legacy String

Function

Left aligns a single-line of the Wild-Char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Lwc(
    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 wild-char. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to Appendix D. for more details about the available encoding names and alias.

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

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

DefaultCode("GB18030");   // set default codepage of input data
OpenPage(8.5,11);

Pos(2,2);                 // set current position to (2",2")
Font(2);                           // Assume font 2 is a TrueType font
// with data type UTF16BE defined
Lwc("test 测试");          // left put wild-char converting from
// Chinese GB18030

ClosePage();
CloseDoc();
```
Left Align UTF-8 Text

Function

Left 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
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 to which the next character would be placed.

Sample

```c
/* UTF-8 string, "test" and CJK characters "测试" */
UChar8     data1[20] = "test\xe6\xb5\xb5\xe8\xb8\xe8\x9af\x95";

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
Lu8(data1);                      // left put UTF-8 at (2",2")
:
:
ClosePage();
CloseDoc();
```
Left Align UTF-8 Text Converting from Legacy String

Function

Left 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
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 Appendix D. for more details about the available encoding names and alias.

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

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

DefaultCode("GB18030");       // set default codepage of input data
OpenPage(8.5,11);

Pos(2,2);                      // set current position to (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 of the UTF-8 string converting from the UTF16-LE text, at the current position.

Before calling of 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

**Syntax**

```c
void Lu8w(
    wchar_t* data,
    bool same_pos = TRUE
);
```

**Parameters**

- **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**

```c
/* wild-char string, "test" and CJK characters "测试" */
UChar data1[] = {0x0074, 0x0065, 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
Lu8w(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

```c
void LPI(
    float     lines
);
```

Parameters

lines
The lines per inch to set up the default line spacing for the subsequent text.

Sample

```c
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 current measurement unit for the subsequent texts.

Syntax

```c
void LineSpace(
    float  increment
);
```

Parameters

increment
The baseline increment in terms of the measurement unit for the subsequent text.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
:
LineSpace(4); // subsequent baseline spacing will be 4 mm
:
:
LineSpace(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 texts to be positioned with the “Next Line” and “Skip Lines” function.

Syntax

```c
void Margin(
    float   margin
);
```

Parameters

*margin*

The left inline margin for the text in terms of the current measurement unit.

Sample

```c
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();
```
**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 Formatter, 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**

```c
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**

```c
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();
Next Line

Function

Starts a new text line from the left inline margin defined by the “Margin” function, 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 Default Input Data File

Function

Opens the default input data file specified by the flag parameter “‐i input_data_file” of the command-line.

It returns the file pointer pointed to the default input data file opened.

It is able to read the input data from the standard‐in (STDIN) if the default input data file is not specified with the flag –i parameter. For instance, with the following example you can read the compressed data on the fly by the system pipe while decompressing the data stream:

```bash
unzip -p test1.zip | makeafp_program -d test1.def.def -o test1.afp
```

Syntax

```c
FILE *OpenDataFile( );
```

Parameters

None.

Sample

```c
FILE *fdin;

void main(void)
{
    Start();           /* Start initiation, open definition file &      */
    /* AFP output file, retrieve AFP resources,       */
    /* allocate memory required                        */

    fdin = OpenDataFile();  /* Open default input data file specified */
    /* by the command-line flag parameter -i */

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

MakeAFP Formatter transfers AFP resources into each AFP output document file if the AFP resource inline is specified by the MakeAFP definition file.

**Syntax**

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

    OpenDoc();       // Open default AFP document, the first
                     // AFP document and open its AFP file

    OpenDoc(2);      // Open second AFP document and open its
                     // AFP file
    :
    :

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

    CloseDoc(2);     // Close second AFP document and its file
}
```
Open Overlay

Function

Opens an AFP overlay. Once you completed the design of your overlay, you must close it with the “Close Overlay” function. This function is specially developed for you to use all the powerful formatting functions offered by MakeAFP Formatter to create your colorful graphic AFP overlay.

You can include AFP page segments as the inline AFP resources for AFP viewing during your development just for your convenience, by defining the parameter "restype=pseg,inline" in your MakeAFP definition file, however you must not include inline AFP resource within your overlay if you would like to generate it as the final AFP overlay to be released for production use or to be used by other MakeAFP formatting programs, make sure parameter “restype= none” is defined before generating your final release version of the AFP overlay.

Syntax

```c
void OpenOvly(
    float  overlay_width,
    float  overlay_height
);
```

Parameters

- `overlay_width`
  Width of the overlay.

- `overlay_length`
  Length of the overlay.

Sample

```c
Start();
SetUnit(IN_U600);
OpenOvly(8.5,11); // Overlay size 8.5" x 11"

GLineWidth(5); // Set line width to about 0.05"
GBox(1.3, 1.8, 3, 1.2, 0.2); // Plot a rounded box

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

CloseOvly();
```
Open Page

Function

Opens an AFP page or reset the page buffer previously opened. Once the page formatting is completed, you must 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 Form Definition).

With the “Maximum Pagination” function or MaxPaging variable, 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 Formatter, you can open multiple pages by the “Open Page” functions, and then process different pages in an interleaved manner once each page is initialized, all 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 PageNum 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

```c
void OpenPage(
    float  page_width,
    float  page_height
);
```

Parameters

- page_width
  Width of the page.
- page_length
  Length of the page.

Sample

```c
SetUnit(IN_U600);
MaxPaging = 1000;        // Maximum page buffers are 1000, must be
                        // defined before Start() function call
Start();                // Start a MakeAFP Formatter session
OpenDoc();              
PageNum = 3;            // switch to page 3 for open page 3
OpenPage(8.5,11);
ClosePage();            
PageNum = 15;           // switch to page 15 for open page 15
```

- 257 -
OpenPage(8.5,11);
ClosePage();
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 Formatter 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

```c
void ParTxt(
    char*  text,
    float  paragraph_width,
    alignmode  alignment = LEFT,
    bool   same_pos = FALSE
);
```

Parameters

text
The ASCII text to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘0a’x) is allowed to split your text line, and the following escape formatting 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:

<table>
<thead>
<tr>
<th>Color</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUE</td>
<td>&quot;01&quot;</td>
</tr>
<tr>
<td>PINK</td>
<td>&quot;03&quot;</td>
</tr>
<tr>
<td>GREEN</td>
<td>&quot;04&quot;</td>
</tr>
<tr>
<td>TURQ</td>
<td>&quot;05&quot;</td>
</tr>
<tr>
<td>BLACK</td>
<td>&quot;08&quot;</td>
</tr>
<tr>
<td>BROWN</td>
<td>&quot;10&quot;</td>
</tr>
<tr>
<td>PURPLE</td>
<td>&quot;0B&quot;</td>
</tr>
<tr>
<td>DARKCYAN</td>
<td>&quot;0D&quot;</td>
</tr>
</tbody>
</table>
MUSTARD "OE" GRAY "OF"

\[C=rrggbb\] \(rr, gg, bb\) 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**

```c
char *msg = "The paragraph of text will be right-aligned nicely";

DefaultCode("ibm-437"); // Data from codepage 437, US ASCII
DefaultLocale("en_US"); // language locale is USA English

OpenPage(8.5,11);
: LineSpace(0.25); // Line spacing is 0.25", 4 LPI
ParTxt(msg,3,RIGHT); // texts are right-aligned into 3" width
// paragraph
: ClosePage();
```
Paragraph of DBCS-HOST

Function

Formats a line of the SBCS-HOST/DBCS-HOST 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 EBCDIC encoded font, and the second one must be a DBCS-HOST encoded font.

Make sure DefaultCode( ) and DefaultLocale( ) are defined properly according to your input data encoding and language locale, for example, with Japanese SBCS-HOST/DBCS data, you need to specify DefaultCode(“ibm-1390”) and DefaultCode(“ja_JP”).

Syntax

```c
void ParDbcs(
    char* text,
    float paragraph_width,
    alignmode alignment = LEFT,
    bool same_pos = FALSE
);
```

Parameters

text
The SBCS-HOST/DBCS-HOST data to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘0a’x) is allowed to split your text line, and the following escape formatting control codes can be inserted into your SBCS-HOST/DBCS-HOST 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”.
```
\[
C = \{ cc, mm, yy, kk \}
\]
c, 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 DBCS-HOST paragraph.

**alignment**
Specifies how the Japanese text in the fixed paragraph should be formatted. The valid values are:

- **LEFT**
  DBCS texts are left-aligned
- **RIGHT**
  DBCS texts are right-aligned
- **CENTER**
  DBCS texts are center-aligned
- **JUSTIFY**
  DBCS 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**

```c
char *msg = "The Japanese ひらがな、漢字、数字 will be center-aligned";
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
  
  LineSpace(0.25); // Line spacing is 0.25", 4 LPI

  Font2(1,2);

  ParDbcs(msg,3,CENTER); // DBCS-HOST is center-aligned into 3" width
                      // paragraph

  ClosePage();
CloseDoc();
```


Paragraph of Japanese

Function

Formats a line of the Japanese ASCII/SJIS-PC 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 encoded font, and the second one must be an SJIS-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void ParJp (
    char*  text,
    float  paragraph_width,
    alignmode  alignment = LEFT,
    bool    same_pos = FALSE
);
```

Parameters

text

The Japanese data to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘0a’x) is allowed to split your text line, and the following escape formatting control codes can be inserted into your ASCII/SJIS-PC 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:

<table>
<thead>
<tr>
<th>Color</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUE</td>
<td>&quot;01&quot;</td>
</tr>
<tr>
<td>PINK</td>
<td>&quot;03&quot;</td>
</tr>
<tr>
<td>GREEN</td>
<td>&quot;04&quot;</td>
</tr>
<tr>
<td>TURQ</td>
<td>&quot;05&quot;</td>
</tr>
<tr>
<td>BLACK</td>
<td>&quot;08&quot;</td>
</tr>
<tr>
<td>BROWN</td>
<td>&quot;10&quot;</td>
</tr>
<tr>
<td>PURPLE</td>
<td>&quot;0B&quot;</td>
</tr>
<tr>
<td>DARKCYAN</td>
<td>&quot;0D&quot;</td>
</tr>
<tr>
<td>MUSTARD</td>
<td>&quot;0E&quot;</td>
</tr>
</tbody>
</table>

- `[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”.

- 263 -
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 texts are left-aligned
  RIGHT   Japanese texts are right-aligned
  CENTER  Japanese texts are center-aligned
  JUSTIFY Japanese 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 Japanese ひらがな、漢字、数字  will be center-aligned";
    SetUnit(IN_U600);
    OpenDoc( );
    OpenPage(8.5,11);
    LineSpace(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 ASCII/KSC-PC 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 encoded font, and the second one must be a KSC-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode( ) is defined properly according to your input data encoding.

Syntax

```c
void ParKr(
    char*  text,
    float  paragraph_width,
    alignmode  alignment = LEFT,
    bool   same_pos = FALSE
);
```

Parameters

text
The Korean data to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘0a’x) is allowed to split your text line, and the following escape formatting control codes can be inserted into your ASCII/KSC-PC 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=rrggb] rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
[C=ccmmykk] 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 Korean paragraph.

alignment
Specifies how the Korean text in the fixed paragraph should be formatted. The valid values are:

- LEFT: Korean texts are left-aligned
- RIGHT: Korean texts are right-aligned
- CENTER: Korean texts are center-aligned
- JUSTIFY: Korean 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

```c
char *msg = "The Korean 온 가족의 티셔츠가 내 폴에 will be center-aligned";
OpenPage(8.5,11);
  LineSpace(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();
```
Function

Formats a line of the Simplified Chinese ASCII/GBK-PC 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 encoded font, and the second one must be a GBK-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void ParSc(
    char*  text,
    float  paragraph_width,
    alignmode alignment = LEFT,
    bool   same_pos = FALSE,
);
```

Parameters

text
The Simplified Chinese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘0a’x) is allowed to split your text line, and the following escape formatting control codes can be inserted into your ASCII/GBK-PC 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=ccmyykk}  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 texts are left-aligned
- RIGHT Chinese texts are right-aligned
- CENTER Chinese texts are center-aligned
- JUSTIFY Chinese 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

```c
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";
OpenPage(8.5,11);
  LineSpace(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 ASCII/BIG5-PC 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 encoded font, and the second one must be a BIG5-PC or DBCS-HOST encoded font. MakeAFP Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void ParTc(
    char*  text,
    float  paragraph_width,
    alignmode  alignment = LEFT,
    bool   same_pos = FALSE
);
```

Parameters

text

The Traditional Chinese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘0a’x) is allowed to split your text line, and the following escape formatting control codes can be inserted into your ASCII/BIG5-PC 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"
  - DarksCYAN "0D" DarksTURQ "0D"
  - MUSTARD "0E" GRAY "0F"
- **[C=rrggb]** rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

\[C=cc\text{mmyykk}\] 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 texts are left-aligned
RIGHT      Chinese texts are right-aligned
CENTER     Chinese texts are center-aligned
JUSTIFY    Chinese 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

```c
char *msg = "The Chinese 越來越多的電腦用戶 will be center-aligned";
OpenPage(8.5,11);
LineSpace(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 Wild-Char Text

Function

Formats a line of the wild-char text into a fixed-width paragraph. Native wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your default language locale is defined properly by the functions DefaultLocale( ) before calling this function, otherwise the default locale is “en_US”.

Syntax

```c
void ParU16(
    wchar_t* text,
    float paragraph_width,
    alignmode alignment = LEFT,
    bool same_pos = FALSE
);
```

Parameters

text

The wild-char text to be aligned into a fixed-width paragraph. The following escape formatting control codes in wild-char 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”.
```
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**

```c
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";
wchar_t[128];    // Defines a buffer of wchar_t

DefaultCode(\"gb18030\") ;  // Text is in Chinese GB18030
DefaultLocale(\"zh_Cn\") ;  // language locale is Simplified Chinese

ChartoW(msg16, 128, msg);   // Converts GB18030 to wild-char

OpenPage(8.5,11);
:
LPI(4);                   // Line spacing is 4 lines per inch
   // for paragraph

ParW(msg16, 3, CENTER);   // Chinese texts are right-aligned
   // into 3" width paragraph
:
ClosePage();
```
Paragraph of Wild-Char Text Converting from Legacy String

Function

Formats a line of the wild-char 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 \texttt{UTF16BE} by the FONT parameter in your MakeAFP definition file. Refer to Chapter 2 for more details about how to define OpenType/TrueType fonts in AFP.

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

\begin{verbatim}
void ParWc(
    char*     text,
    float     paragraph_width,
    alignmode    alignment = LEFT,
    bool         same_pos = FALSE
);
\end{verbatim}

Parameters
text

The legacy codepage string to be converted to wild-char and aligned into a fixed-width paragraph. The following escape formatting control codes can be inserted into your ASCII/DBCS-PC text data for dynamic control of font, color, and underscore switching:

\begin{itemize}
  \item \texttt{[F=xx]} \begin{itemize}
    \item xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
  \end{itemize}
  \item \texttt{[U=01]} Turns on underscore
  \item \texttt{[U=00]} Turns off underscore
  \item \texttt{[C=xx]} \begin{itemize}
    \item xx are the color ID in two characters of hex code value:
      \begin{tabular}{ll}
        \texttt{BLUE} & "01" \texttt{RED} & "02" \\
        \texttt{PINK} & "03" \texttt{MAGENTA} & "03" \\
        \texttt{GREEN} & "04" \texttt{CYAN} & "05" \\
        \texttt{TURQ} & "05" \texttt{YELLOW} & "06" \\
        \texttt{BLACK} & "08" \texttt{DARKBLUE} & "09" \\
        \texttt{BROWN} & "10" \texttt{ORANGE} & "0A" \\
        \texttt{PURPLE} & "0B" \texttt{DARKGREEN} & "0C" \\
        \texttt{DARKCYAN} & "0D" \texttt{DARKTURQ} & "0D" \\
        \texttt{MUSTARD} & "0E" \texttt{GRAY} & "0F"
      \end{tabular}
  \end{itemize}
  \item \texttt{[C=rrggbb]} \begin{itemize}
    \item rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
  \end{itemize}
\end{itemize}
\[C=ccmmyykk\] \(cc\), \(mm\), \(yy\), \(kk\) 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

```c
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"); // language locale is Simplified Chinese

OpenPage(8.5,11);
:

LPI(4); // Line spacing is 4 lines per inch
// for paragraph

ParWC(msg, 3, CENTER); // Chinese texts are 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 2 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

```c
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 formatting 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT</td>
<td>Texts are left-aligned</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Texts are right-aligned</td>
</tr>
<tr>
<td>CENTER</td>
<td>Texts are center-aligned</td>
</tr>
<tr>
<td>JUSTIFY</td>
<td>Texts are justify-aligned</td>
</tr>
</tbody>
</table>

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

```c
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"); // language locale is Simplified Chinese

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 texts are 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

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

```cpp
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 formatting control codes can be inserted into your ASCII/DBCS-PC 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”.

- 277 -
[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**

```c
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"); // language locale is Simplified
                      // Chinese

OpenPage(8.5,11);
:

LPI(4);              // Line spacing is 4 lines per inch
                     // for paragraph
ParU8c(msg, 3, CENTER); // Chinese texts are right-aligned
                       // into 3" width paragraph
:
ClosePage();
```
Paragraph of UTF-8 Text Converting from Wild-Char

Function

Formats a line of the Unicode UTF-8 text converting from the wild-char 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 2 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

```c
void ParU8w(
    wchar_t*     text,
    float        paragraph_width,
    alignmode    alignment = LEFT,
    bool         same_pos = FALSE
);
```

Parameters

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" |
| 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”.

\texttt{\textcolor{magenta}{[C=ccmmyykk}}} \textcolor{red}{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:

- \texttt{LEFT} \textcolor{red}{Texts are left-aligned}
- \texttt{RIGHT} \textcolor{red}{Texts are right-aligned}
- \texttt{CENTER} \textcolor{red}{Texts are center-aligned}
- \texttt{JUSTIFY} \textcolor{red}{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**

```c
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

ParU8w(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(INCH);
    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 points and Y position to 3.5"
    :
    :
    ClosePage();
    CloseDoc();
Position of Text

Function

Sets the absolute horizontal X position and absolute vertical Y position for the output text on the page. The origin position on the page is at (0, 0).

Syntax

```c
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 invalid.

**y_position**
The value of the absolute vertical position from the page origin. Negative values are invalid.

Sample

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

Pos(5,10);  // Set X position to 5 mm and y position to 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

```c
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 or Postscript printer, default is print to your Windows default printer if it is not specified.

Sample

```c
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,   
    int     docNo = 1   
);

Parameters

MakeAFP Formatter 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 Formatter 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.

Sample

// Now all input data of a client are formatted into the AFP page
// buffers, before write out all of the pages of a client, we can insert
// beginning of group index tag and index value tags
unsigned short numpages = PageNum;        // Keep total pages of a
client
char tmp[25];
char tmp[35];
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
{
    PageNum = i + 1;                      // Switch to each page buffer
    sprintf(tmp, "Page %d of %d", PageNum, numpages);
    Pos(8.0,3.93);                        // Set position to (8", 3.93")
    Rtxt(tmp);                            // Right alignment of page number
ClosePage(); // on each page before end of each page
} // End index page group, must be called after writing of the last page of each page group

EndIdx();

PageNum = 1; // Reset AFP page buffer number to 1 for the next customer statement
Put Index Tag With Encoding Conversion

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

```c
void PutIdx2(
    char*   index_name,
    char*   index_value,
    int     docNo = 1
);
```

Parameters

Make sure the CPGID parameter is defined properly in your MakeAFP definition file, and default input data encoding is defined properly by the functions of DefaultCode(), so that MakeAFP Formatter auto-converts your index name and value encoding correctly.

**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.

Sample

```c
// Now all input data of a client are formatted into the AFP page buffers, before write out all of the pages of a client, we can insert
// beginning of group index tag and index value tags
unsigned short numpages = PageNum;        // Keep total pages of a client
char tmp[25];
char tmp[35];
sprintf(tmp, "%08d", ++groups);
BgnIdx2(tmp);                     // Begin index page group
PutIdx2("Customer Name", client_name);  // Put group-level index tags,
PutIdx2("Account Number", account_no);  // BgnIdx and PutIdx must be called
// before writing of the first page
for (int i = 0; i < numpages; i++) // of each page group
{
    PageNum = i + 1;          // Switch to each page buffer
    sprintf(tmp, "Page %d of %d", PageNum, numpages);
    Pos(8.0,3.93);            // Set position to (8", 3.93")
    Rttx(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

PageNum = 1;                           // Reset AFP page buffer number to 1
                                        // for the next customer statement
Put Paragraph Text of ASCII / EBCDIC

Function

Adds a string of the 1-byte text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

You need to define an ASCII or EBCDIC encoded font with the “Font” function. MakeAFP Formatter 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

```c
void PutParTxt(
    char* data,
    ocaColor text_color = BLACK,
    Bool underscore = FALSE
);
```

Parameters

data
The string of ASCII or EBCDIC text to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "Congratulations!";
char *msg2 = "You got a perfect AFP printing solution.";
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParTxt(30, LEFT);   // Begin a paragraph, 30 mm width,
                       // left-aligned
Font(1);   // Font 1 is ASCII font
```
PutParTxt(msg1, PINK); // Put 1st ASCII text in pink color
Font(2); // Font 2 is ASCII font
PutParTxt(msg2, BLUE); // Put 2nd text in blue color
EndParTxt(); // End paragraph and write into AFP output
ClosePage();
Put Paragraph Text of DBCS-HOST

Function

Adds a string of the SBCS-HOST/DBCS-HOST text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

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

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```cpp
void PutParDbcs(
    char*   data,
    ocaColor  text_color = BLACK,
    bool  underscore = FALSE
);
```

Parameters

data
The string of SBCS-HOST/DBCS-HOST text to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```cpp
char *msg1 = "ひらがな、カタカナ、漢字、数字の 123、";
char *msg2 = "Alphabetが混在した文章のサンプルです。";
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParDbcs(110, LEFT);   // Begin a Japanese paragraph, 110 mm
    // width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is SJIS font
PutParDbcs(msg1, PINK);   // Put 1st Japanese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is SJIS font
PutParDbcs(msg2, BLUE);   // Put 2nd Japanese text in blue color
EndParDbcs();   // End Japanese paragraph and write
    // into AFP output
```

- 291 -
Put Paragraph Text of Japanese

Function

Adds a string of the Japanese ASCII/SJIS-PC text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void PutParJp(  
  char*  data,  
  ocaColor text_color = BLACK,  
  bool  underscore = FALSE  
);
```

Parameters

data
The string of Japanese SJIS text to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "ひらがな、カタカナ、漢字、数字の123、";
char *msg2 = "アルファベットが混在した文章のサンプルです。";
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParJp(110, LEFT);   // Begin a Japanese paragraph, 110 mm width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is SJIS font
PutParJp(msg1, PINK);  // Put 1st Japanese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is SJIS font
PutParJp(msg2, BLUE);  // Put 2nd Japanese text in blue color
EndParJp();   // End Japanese paragraph
```
Put Paragraph Text of Korean

Function

Adds a string of the Korean ASCII/KSC-PC text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode( ) is defined properly according to your input data encoding.

Syntax

```c
void PutParKr(
    char*  data,
    ocaColor text_color = BLACK,
    bool  underscore = FALSE
);
```

Parameters

data
The string of Korean KSC text to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "’합께해요~’ 이벤트, 음 가족의 티셔츠가 내 품에!";
char *msg2 = "내 책 사면 <어린이 도서관>에 기증될 책이 하나 더!"
;
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParKr(110, LEFT);   // Begin a Korean paragraph, 110 mm
// width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is KSC font
PutParKr(msg1, PINK); // Put 1st Korean text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is KSC font
PutParKr(msg2, BLUE); // Put 2nd Korean text in blue color
EndParKr();   // End Korean paragraph and write into AFP output
```
Put Paragraph Text of Simplified Chinese

Function

Adds a string of the Simplified Chinese ASCII/GBK-PC text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void PutParSc(
    char*  data,
    ocaColor text_color = BLACK,
    bool     underscore = FALSE
);
```

Parameters

data
The string of Simplified Chinese text to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "Windows 10 兼具友好界面，";
char *msg2 = "越来越多的电脑用户都安装了 Windows 10.";
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParSc(110, LEFT);   // Begin a Simplified Chinese paragraph, 110 mm
Font(1,2);   // Font 1 is ASCII font, font 2 is GBK font
PutParSc(msg1, PINK); // Put 1st Chinese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is GBK font
PutParSc(msg2, BLUE); // Put 2nd Chinese text in blue color
EndParSc();  // End Simplified Chinese paragraph
```
Put Paragraph Text of Traditional Chinese

Function

Adds a string of the Traditional Chinese ASCII/BIG5-PC text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Make sure DefaultCode() is defined properly according to your input data encoding.

Syntax

```c
void PutParTc(
    char*    data,
    ocaColor   text_color = BLACK,
    bool        underscore = FALSE
);
```

Parameters

data
The string of Traditional Chinese text to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "Windows 10 兼具友好界面,";
char *msg2 = "越来越多的電脑用户都安装了 Windows 10.";
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
Pos(20, 50);
LPI(4);   // Set line spacing to 4 LPI
BgnParTc(110, LEFT);   // Begin a Traditional Chinese paragraph, 110 mm width, left-aligned
Font(1,2);   // Font 1 is ASCII font, font 2 is BIG5 font
PutParTc(msg1, PINK); // Put 1st Chinese text in pink color
Font(3,4);   // Font 3 is ASCII font, font 4 is BIG5 font
PutParTc(msg2, BLUE); // Put 2nd Chinese text in blue color
EndParTc();  // End Traditional Chinese paragraph
```
Put Paragraph Text of Wild-Char

Function

Adds a string of the wild-char text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical page.

Native wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your language locale is defined properly by the functions of DefaultLocale() before calling this function.

Syntax

```c
void PutParW(
    wchar_t*      data,
    ocaColor   text_color = BLACK,
    Bool       underscore = FALSE
);
```

Parameters

data
The string of wild-char text to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
wchar_t *msg1 = L"Congratulations!";
wchar_t *msg2 = L"You got a perfect AFP printing solution.";
UChar msgU16[80];

DefaultLocale("en_US"); // language locale is USA English
LPI(4); // Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);
BgnParW(110, LEFT); // Begin a variable paragraph, 110 mm
```
ChartoW(msgU16, 80, msg1);  // Converts ASCII CP437 to UTF-16LE
PutParW(msgU16, BLUE);      // Put 1st wild-char text in blue color
ChartoW(msgU16, 80, msg2);  // Converts ASCII CP437 to UTF-16LE
PutParW(msgU16, RED);       // Put 2nd wild-char text in red color
EndParW();                  // End wild-char paragraph and write into AFP
Put Paragraph Text of Wild-Char Converting from Legacy String

Function

Adds a string of the wild-char text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

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 default input data encoding is “Windows-1252” and locale is “en_US”.

Syntax

```c
void PutParWc(
    char* data,
    ocaColor text_color = BLACK,
    Bool underscore = FALSE
);
```

Parameters

data
The legacy string to be converted to wild-char and added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "Congratulations!";
char *msg2 = "You got a perfect AFP printing solution.";

DefaultCode("ibm-437");       // Data from codepage 437, USAN ASCII
DefaultLocale("en_US");       // language locale is USA English
:
LPI(4);                        // Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);
BgnParW(110, LEFT);            // Begin a variable paragraph, 110 mm
// width, left-aligned
```
\textbf{PutParWc}(msg1, BLUE);  \hspace{1cm} // Put 1st wild-char text in blue color
\textbf{PutParWc}(msg2, RED);  \hspace{1cm} // Put 2nd wild-char text in red color
EndParW(); \hspace{1cm} // End wild-char paragraph and write into AFP
Put Paragraph Text of UTF-8

Function

Adds a string of the UTF-8 text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Make sure your language locale is defined properly by the functions of DefaultLocale( ) before calling this function.

Syntax

```c
void PutParU8(  
    UChar8*    data,  
    ocaColor   text_color = BLACK,  
    Bool      underscore = FALSE  
    );
```

Parameters

data
The string of UTF-8 to be added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "Congratulations!";  
char *msg2 = "You got a perfect AFP printing solution.";
UChar8 msgU8[256];

DefaultCode("ibm-437");  
// Data from codepage 437, USAN ASCII
DefaultLocale("en_US");  
// language locale is USA English

LPI(4);  
// Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);

BgnParU8(110, LEFT);  
// Begin a variable paragraph, 110 mm  
// width, left-aligned

ChartoU8(msgU8, 80, msg1);  
// Converts ASCII CP437 to UTF-8
```
PutParU8(msgU16, BLUE);  // Put 1st wild-char text in blue color
ChartoU8(msgU16, 80, msg2); // Converts ASCII CP437 to UTF-8
PutParU8(msgU16, RED);    // Put 2nd wild-char text in red color
EndParU16();              // End wild-char paragraph and write into AFP
Put Paragraph Text of UTF-8 Converting from Legacy String

Function

Adds a string of the UTF-8 text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

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 default input data encoding is “Windows-1252” and locale is “en_US”.

Syntax

```c
void PutParU8c(
    char*    data,
    ocaColor text_color = BLACK,
    Bool     underscore = FALSE
);
```

Parameters

data
The legacy string to be converted to UTF-8 and added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
cchar *msg1 = "Congratulations!";
cchar *msg2 = "You got a perfect AFP printing solution.";
UChar msgU16[80];

DefaultCode("ibm-437"); // Data from codepage 437, USAN ASCII
DefaultLocale("en_US"); // language locale is USA English
:
LPI(4); // Set default line spacing to 4 LPI
Font(1);
Pos(50, 140);
BgnParU16(110, LEFT); // Begin a variable paragraph, 110 mm
```
ChartoU16(msgU16, 80, msg1);  // Converts ASCII CP437 to UTF-16LE
PutParU16(msgU16, BLUE);     // Put 1st wild-char text in blue color
ChartoU16(msgU16, 80, msg2);  // Converts ASCII CP437 to UTF-16LE
PutParU16(msgU16, RED);       // Put 2nd wild-char text in red color
EndParU16();                  // End wild-char paragraph and write into AFP
Put Paragraph Text of UTF-8 Converting from Wild-Char

Function

Adds a string of the UTF-8 text with formatting characteristics to the current paragraph. The paragraph must be ended before you end the page. Ensure that the text you have specified fits on the logical 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

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 default input data encoding is “Windows-1252” and locale is “en_US”.

Syntax

```c
void PutParU8w(
    wchar_t*    data,
    ocaColor    text_color = BLACK,
    Bool        underscore = FALSE
);
```

Parameters

data
The UTF16_LE string to be converted to UTF-8 and added in the current paragraph.

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

underscore
Specifies whether the text is underscored. The default value is FALSE, text is not underscored.

Sample

```c
char *msg1 = "Congratulations!";
char *msg2 = "You got a perfect AFP printing solution.";
UChar msgU16[80];
DefaultCode("ibm-437"); // Data from codepage 437, USAN ASCII
DefaultLocale("en_US"); // language locale is USA English
    LPI(4);               // Set default line spacing to 4 LPI
Font(1);
    Pos(50, 140);
BgnParU8(110, LEFT);      // Begin a variable paragraph, 110 mm
                               // width, left-aligned
```
ChartoW(msgU16, 80, msg1); // Converts ASCII CP437 to UTF-16LE
PutParU8w(msgU16, BLUE);  // Put 1st text in blue color
ChartoW(msgU16, 80, msg2); // Converts ASCII CP437 to UTF-16LE
PutParU8w(msgU16, RED);   // Put 2nd text in red color
EndParU8();                // End UTF-8 paragraph and write into AFP
Rendering Color Intent

Function

Specifies the color rendering intent for the subsequent AFP pages or an overlay created by MakeAFP Formatter, to modify the final appearance of the color object.

This function can be repeated to define the rendering intents for all object types.

Refer to the latest Infoprint PPFA User’s Guide and Infprint Manager Procedures books for more about AFP color management.

Syntax

Invokes Color Rendering:

```c
void Render(
    objt_type     object_type,  
    render_type     render_intent  
);
```

Revokes Color Rendering:

```c
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.
- RELCM: 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**

```plaintext
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 Single 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 Formatter 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 to (2",2")
        Rtxt("texts are 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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
void Rjp(  
    char*    data,  
);  
```

Parameters

data
The NULL-terminated SJIS data string.

Sample

```c
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 to (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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
void Rkr(
    char* data,
);
```

Parameters

- **data**
  The NULL-terminated KSC data string.

Sample

```c
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 to (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 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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

Syntax

```c
void Rsc(
    char* data,
);
```

Parameters

data
The NULL-terminated GBK data string.

Sample

```c
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 to (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 Formatter converts data encoding internally, based on the encodings of AFP fonts defined.

**Syntax**

```c
void Rtc(char* data);
```

**Parameters**

- **data**
  - The NULL-terminated BIG5 data string.

**Sample**

```c
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 to (2",2")
Rtc("實現 Win2000 與 Linux 的雙引導"); // right align BIG5 text at (2",2")

ClosePage();
CloseDoc();
```
Right Align SBCS-HOST/DBCS-HOST Text

Function

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

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

Syntax

```c
void Rdbcs(
    char* data,
);
```

Parameters

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

Sample

```c
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 to (2",2")
Rdbcs("实现 Win2000 与 Linux 的双引导"); // right align DBCS text at (2",2")

: ClosePage();
CloseDoc();
```
**Right Align Wild-Char Text**

**Function**

Right aligns a single-line of the wild-char string at the current position. Native wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

**Syntax**

```c
void Rw(
    UChar* data,
);   
```

**Parameters**

data
The NULL-terminated wild-char string.

**Sample**

```c
/* wild-char string, "test" and CJK characters "测试" */
Wchar_t data1[20] = \{0x0074, 0x0065, 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 UTF16BE defined
Rw(data1);                        // right put wild-char at (2",2")
    :
    :
ClosePage();
CloseDoc();
```
Right Align Wild-Char Text Converting from Legacy String

Function

Right aligns a single-line of the wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Rwc(
    char* data,
    char* fromcode = NULL,
);
```

Parameters

data
The NULL-terminated legacy codepage string.

fromcode
The encoding name of the source string to be converted into wild-char. Default is NULL, using default encoding name predefined by the DefaultCode() function. Refer to Appendix D. for more details about the available encoding names and alias.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
DefaultCode("GB18030");   // set default converter for input data
OpenPage(8.5,11);
Pos(2,2);                 // set current position to (2",2")
Font(2);                           // Assume font 2 is a TrueType font
// with data type UTF16BE defined
Rwc("test 测试");       // right put wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Ru8(
    UChar8* data,
); 
```

Parameters

data
The NULL-terminated UTF-8 string.

Sample

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

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
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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

void Ru8c(
    char* data,
    char* fromcode = NULL,
);

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 Appendix D. for more details about the available encoding names and alias.

Sample

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    :
    :
    DefaultCode("GB18030");            // set default converter for input data
    Pos(2,2);                          // current position to (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 Wild-Char

Function

Right aligns a single-line of the UTF-8 string converting from the wild-char 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 2 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax

```c
void Ru8w(
    wchar_t*     u16_data,
);
```

Parameters

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

Sample

```c
/* wild-char string, "test" and CJK characters "测试" */
UChar    data1[] = {0x0074, 0x0065, 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

Ru8w(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 Formatter default is IN_U600 if you do not call this function.

Syntax

```c
void SetUnit(
    unit    makeafp_unit
);
```

Parameters

**makeafp_unit**
Valid values are:

- 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

void main(void)
{
    Start();

    SetUnit(MM_U600);  // Set default units to MM, 600 pixels per inch
    OpenDoc();

    OpenPage(215.9, 279.4);  // LETTER page size, 215.9 mm x 279.4 mm
        :
        :

    CloseDoc();
}
Shade

Function

Draws a shaded area.

Syntax

```c
void Shade(
    float   x_pos,
    float   y_pos,
    float   width,
    float   height,
    float   shading_percentage,
    pattern shading_pattern = LED,
    ocaColor color = BLACK
);
```

Parameters

- **x_pos**
The X position of the top left corner of the shade.
- **y_pos**
The Y position of the top left corner of the shade.
- **width**
The width of the shade.
- **height**
The height of the shade.
- **shading_percentage**
The percentage of shading.
- **shading_pattern**
You can choose between three shading patterns depending on the printer type you are using. Valid values are LED (default value), STD, SCREEN.
- **color**
Any of the defined AFP OCA color values: BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY, the default value is BLACK.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220,297);
:
:
Shade(10,10,20,5,15,LED,RED);    // Draw a red shade from (10,10)mm
    // shade area width is 20mm, height
    // is 5mm, shading percentage is 15,
    // LED pattern
:
:
```
Simplified Chinese Width

Function

Measures and then returns the width of the Simplified Chinese GBK string in the given fonts. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” function before you call this function.

Syntax

```c
float ScWidth(
    char*     gbk_data,
    ushort   font1 = DEFAULT,
    ushort    font2 = DEFAULT
);
```

Parameters

- **gbk_data**
  The string of Simplified Chinese GBK characters to measure.

- **font1**
  The ID number of the ASCII or SBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the font in 1-byte, UTF-8, or wild-char encoding.

- **font2**
  The ID number of the GBK or DBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the 2-byte font.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Font(4,5);                        // font 4 is 1-byte font
    // font 5 is 2-byte font
:
ScWidth("实现 Win2000 与 Linux 的双引导");    // measure GBK width by
    // current fonts
:
:
ClosePage();
CloseDoc();
```
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

```c
void Skip(
    float lines
);
```

Parameters

- **lines**
  The number of lines to skip.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
LineSpace(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 Formatter session before calling any other MakeAFP Formatter functions.

This function starts and establishes initiation of a MakeAFP Formatter session, allocate required memory, parses the parameters defined in the MakeAFP Formatter definition file, and merges all the AFP resources and OpenType/TrueType fonts required by your program either by generating an external AFP resource file or embedding 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

void Start(
    char* command_line_arguments = NULL
);  
void StartW(
    UChar* command_line_arguments_utf16le = 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 the command.

Refer to Chapter 3. Running MakeAFP Formatter in Batch Mode, MakeAFP Formatter Users’ Guide, for more details about command-line flag-arguments supported by MakeAFP Formatter.

Sample

void main(void)
{
    Start();  // Start initiation, open definition file and AFP output file, retrieve resources and font Information, allocate memory required
    :
    :
    :
}
Text Orientation

Function

Sets the combination of inline and baseline orientations in which the subsequent text will be presented.

Syntax

```c
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

The following figure illustrates changes in orientation with no change in character rotation:
The following figure illustrates changes in character rotation with different text orientations:

<table>
<thead>
<tr>
<th>Inline Direction</th>
<th>Baseline Direction</th>
<th>0</th>
<th>90</th>
<th>180</th>
<th>270</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90 or 270</td>
<td>ABC</td>
<td>A</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>90</td>
<td>180 or 0</td>
<td>ABC</td>
<td></td>
<td></td>
<td>A B</td>
</tr>
<tr>
<td>180</td>
<td>270 or 90</td>
<td>ABC</td>
<td></td>
<td></td>
<td>C B A</td>
</tr>
<tr>
<td>270</td>
<td>0 or 180</td>
<td>ABC</td>
<td>C B A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sample

Font(1);

CharRotate(270);

TextOrient(l90b180);

Pos(1, 1);

Lu16c("基本月租费");

Output:

基本月租费
Text Width of ASCII/EBCDIC String

Function

Measures and then returns the width of the ASCII or EBCDIC string in the given font. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” function before you call this function.

Syntax

```c
float TxtWidth(
    char*  text_data,
    ushort font1 = DEFAULT
);
```

Parameters

text_data
The string of ASCII or EBCDIC characters to measure.

font1
The ID number of the ASCII or EBCDIC font you defined in your MakeAFP definition file. Default is using your current ASCII or EBCDIC font ID.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Font(4); // font 4 is 1-byte font
:
:
float w;
w = TxtWidth("Get a text string width"); // measure text width by // current font
:
:
ClosePage();
CloseDoc();
```
TIFF Pages

Function

Counts the number of images in a multiple-page TIFF file.

Syntax

```c
void ImportTiff(
    char*   tiff_name,

);```

Parameters

tiff_name
The name of the TIFF image, fully qualified with the pathname in your hard disk. It must be available to MakeAFP Formatter at the time of formatting.

Sample

```c
Start();       // Start initiation
OpenDoc();                 // Open AFP document
SetUnit(IN_U300);          // Set default units to inch, 300 dpi
int pages = TiffPages("pages.tif");
for (int i=0; i< pages; i++)
{
    OpenPage(8.27,11.67);   // A4 page size
    ImportTiff("pages.tif", // TIFF image file name
        i,        // TIFF page index No.
        0,         // X position
        0,         // Y position
        300);       // As 300 dpi inline IOCA image
    ClosePage();            // End of a AFP page
}
CloseDoc();              // End of AFP document```
Traditional Chinese Width

Function

Measures and then returns the width of the Traditional Chinese BIG5 string in the given fonts. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” function before you call this function.

Syntax

```c
float TcWidth(
    char*   big5_data,
    ushort font1 = DEFAULT,
    ushort  font2 = DEFAULT
);
```

Parameters

- **big5_data**
The string of Traditional Chinese BIG5 characters to measure.

- **font1**
The ID number of the ASCII or SBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the font in 1-byte, UTF-8, or wild-char encoding.

- **font2**
The ID number of the BIG5 or DBCS-HOST font you defined in your MakeAFP definition file. Default is using your current ID of the 2-byte font.

Sample

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Font((4,5); // font 4 is 1-byte font  
// font 5 is 2-byte font
:

float w;
w = TcWidth("實現 Win2000 與 Linux 的雙引導"); // measure ASCII/BIG5 string
// width by current fonts
:
:
ClosePage();
CloseDoc();
```
Wild-Char String Width

Function

Measures and then returns the width of the native wild-char string in the given font. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” function before you call this function.

Syntax

```cpp
void WWidth(
    wchar_t   data,
    ushort    font1 = DEFAULT
);```

Parameters

data
The string of native wild-char characters to measure.

font1
The ID number of the OpenType/TrueType or TrueType Collection font you defined in your MakeAFP definition file. Default is using your current font ID.

Sample

```cpp
// wild-char string, "test" and CJK characters "测试"
wchar_t  data[10] = {0x0074, 0x0065, 0x0074, 0x6d4b, 0x8bd5};
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Font(4); // assume font 4 is a TrueType font
:
float w = WWidth(data); // measure wild-char string width by // font 4
:
:
ClosePage();
CloseDoc();```
UTF-8 String Width

Function

Measures and then returns the width of the UTF-8 string in the given font. The return value is in the default unit you defined with the “Set Default Unit” function.

You must reset your word and character spacing back to font default by the “Word Space” and “Character Space” function before you call this function.

Syntax

```c
void U8Width(
    UChar8     utf8_data,
    ushort    font1 = DEFAULT
);
```

Parameters

- **Utf8_data**: The string of UTF-8 characters to measure.
- **font1**: The ID number of the OpenType/TrueType or TrueType Collection font you defined in your MakeAFP definition file. Default is using your current font ID.

Sample

```c
// UTF-8 string, "test" and CJK characters "测试"
UChar8           utf8data [20] = "test\xe6\xb5\x8b\xe8\xa8\x95";
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
    :
    :
Font(4);                   // assume font 4 is a TrueType font
    :
    :
U8Width(utf8data);         // measure UTF-8 string width by
                           // font 4
    :
    :
ClosePage();
CloseDoc();
```
Vertical Boxes

Function

Repeat drawing boxes vertically.

Syntax

```c
void Vboxes(
    float  x_pos,
    float  y_pos,
    float  width,
    float  height,
    float  thickness,
    ushort repeat,
    float  space,
);
```

Parameters

- **x_pos**
  The X position of the top left corner of the first box, specify CP if you want to use the current position.

- **y_pos**
  The Y position of the top left corner of the first box, specify CP if you want to use the current position.

- **width**
  The width of the box.

- **height**
  The height of the box.

- **thickness**
  The thickness of the box border.

- **repeat**
  The number of additional boxes to be repeated.

- **space**
  The gap space between the boxes.

Sample

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

Color(RED); // defines color for texts and legacy line
Vboxes(10,10,20,5,1,9,2,RED); // repeat 10 boxes vertically from
// (10,10)mm, box width is 20mm, height
// is 5mm, thickness is 1mm, 2mm space
// between boxes, red color border

ClosePage();
CloseDoc();
```
Vertical Line

Function

Draws a vertical line.

Syntax

```c
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

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

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

Vline(10,10,100,1,BLUE); // 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**

```c
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**

```c
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();
```
Vertical Shades

Function

Repeat drawing shades vertically.

Syntax

```c
void Vshades(
    float    x_pos,
    float    y_pos,
    float    width,
    float    height,
    float    shading_percentage,
    ushort   repeat,
    float    space,
    pattern  shading_pattern = LED,
    ocaColor color = BLACK
);
```

Parameters

- **x_pos**
The X position of the top left corner of the first shade.

- **y_pos**
The Y position of the top left corner of the first shade.

- **width**
The width of the shade.

- **height**
The height of the shade.

- **shading_percentage**
The percentage of the shading.

- **repeat**
The number of additional shades to repeated.

- **space**
The amount of space you want between the shades.

- **shading_pattern**
You can define between three shading patterns depending on the printer type you are using. Valid values are LED (default), SCREEN and STD.

- **color**
Any of the defined AFP OCA color values: BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY, the default value is BLACK.

Sample

```c
SetUnit(MM_U600);
;
OpenPage(220,297);
```
Vshades(10,10,20,5,15,9,2);  // repeat 10 shades vertically from
               // (10,10)mm, shades width is 20mm, height
               // is 5mm, shading percentage is 15,
               // 2mm space between shades

ClosePage();
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 Formatter 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 Formatter calls the AFP viewer automatically if an error message has taken place during your development or production, or once the 200 pages limitation is reached if it is running in the demo mode without a software license key or hardware license 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 the pathname 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
## Word Space

### Function

Sets the word spacing.

### Syntax

```cpp
void WordSpace(
    float word_spacing = 0
);
```

### Parameters

**word_spacing**
The amount of space between words. Default value 0 indicates that the word spacing is determined by the font default.

### Sample

```cpp
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Pos(5,5); // current position to (5,5) mm
WordSpace(5); // Word space is 5 mm
Ctxt("Extra Intercharacter spacing");
WordSpace(); // reset word space back to font
    // default
:
:
ClosePage();
CloseDoc();
```
X Absolute Position of Text

Function

Sets the new horizontal X absolute position 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 text absolute horizontal position from the page origin. Negative values are invalid.

Sample

SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
    :
    :
Xpos(5);          // Set X position to 5 mm
    :
    :
ClosePage();
CloseDoc();
X Current Position of Text (Query)

Function

Queries the current text horizontal position on the page, returns a value is in the current measurement unit.

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 Relative Move Position of Text

Function

Moves the text horizontal X position relative to the current text horizontal coordinate position.

Syntax

```c
void Xmove(
    float x_move
);
```

Parameters

- **x_move**
  The value of the text horizontal movement relative to the current text horizontal position (X). A positive value moves the position to the right; a negative value moves the position to the left.

Sample

```c
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 of Text

Function

Sets the new vertical absolute Y position for the output text on the page. The origin position on the page is at (0, 0).

Syntax

```cpp
void Ypos(
    float y_position
);
```

Parameters

- **y_position**
The value of the vertical position absolute from the page origin. Negative values are invalid.

Sample

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

Ypos(15); // Set Y position to 15 mm

ClosePage();
CloseDoc();
```
Y Current Position of Text (Query)

**Function**

Queries the current vertical Y position on the page, returns a value is in the current measurement unit.

**Syntax**

```c
float GetYpos();
```

**Sample**

```c
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 Relative Move Position of Text

Function

Moves the text vertical Y position relative to the current text vertical coordinate position.

Syntax

```c
void Ymove(
    float   y_move
);
```

Parameters

`y_move`
The value of the text vertical movement relative to the current text vertical position (Y).
Positive values move the position down; negative values move the position up.

Sample

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

Ymove(25);       // Move 25 mm down

Ymove(-10);      // Move 10 mm up

ClosePage();
CloseDoc();
```
Chapter 2. MakeAFP Formatter Parameters

This chapter describes the MakeAFP Formatter 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:

  \[ \text{FONTLIB} = \text{pathname} \]

  means that you enter \text{FONTLIB} as shown and then replace the variable \text{pathname} with a value that represents any valid pathname.

- Do not enter the following symbols as part of the command:

  \[ \text{Vertical bar} \quad | \quad \text{Braces} \quad \{ \} \quad \text{Brackets} \quad [ ] \quad \text{Underscore} \quad _{ } \quad \]

  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:

    \[ \text{PRMODE} = \{ \text{EBCDIC} \mid \text{SOSI1} \mid \text{SOSI2} \} \]

    means that when you enter \text{PRMODE}, 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:

\[
\text{FONT1} = \{ \text{CDF} \mid \text{CHS, CDP} \} [\text{height\_point}] [\text{scale\_ratio}]
\]

means that height\_point and scale\_ratio 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:

\[
\text{RESTYPE} = \{ \text{NONE} \mid \text{ALL} \mid \text{...} \}
\]

means that if the RESTYPE parameter is not entered, MakeAFP Formatter 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**

\[
\text{CMR}n = \text{cmr\_file}
\]

**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.

\[
\text{cmr\_file}
\]

Specifies the file name of an AFP CMR with or without file name extension of *.cmr.

**Sample**

<table>
<thead>
<tr>
<th>objtlib</th>
<th>Specify the object library path where CMRs are stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmr1 = EUROISOCC001000.cmr</td>
<td>Specify CMR of Europe ISO Coated</td>
</tr>
<tr>
<td>cmr2 = JPSTD2CC001000.cmr</td>
<td>Specify CMR of Japan Standard V2</td>
</tr>
</tbody>
</table>

---

**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 Formatter indexing functions.
The Code Page Global Identifier is used by an AFP viewer or AFP archiving system, which must display indexing information. The AFP software uses this identifier with code page translation tables to represent the index attribute and value data.

For more information about IBM code pages, see Appendix A to Appendix C, or refer to IBM document AFP Fonts: Technical Reference for Code Pages, S544-3802, or IBM webpage http://www-01.ibm.com/software/globalization/cp/cp_cpgid.jsp.

Syntax

\[ \text{CPGID} = \text{codepageID} \]

Parameter

\[ \text{codepageID} \]

Any valid code page ID, which is a two through the four-character decimal value that defines an IBM-registered code page ID.

If this parameter is not specified, MakeAFP Formatter uses code page ID 850 (Personal Computer - Multilingual Page ASCII) as the default.

Sample

\[ \text{cpgid} = 437 \]  
Code page ID for US English ASCII

FDEF – Specifies a Form Definition

Function

Specifies the form definition to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Formatter.

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

\[ \text{FDEF} = \text{fdefname} \]

Parameter

\[ \text{fdefname} \]

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

\[ \text{fdef} = \text{F1TEST01} \]
FDEFLIB – Specifies the Library Path of Form Definitions

Function

Specifies the directories in which form definitions are stored.

Syntax

FDEFLIB = pathlist

Parameter

pathlist
Any valid search path. Multiple paths must be separated by the semicolon (;). MakeAFP Formatter searches the paths in the order you specified.

When MakeAFP Formatter finds more than one form definition with the same file base-name 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

Note: 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 FOCA raster or outline font to be used in the MakeAFP formatting.

A separate FONT parameter is required for each font, up to a maximum of 127 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 alphanumerical 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 alphanumerical 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 alphanumerical 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

tfont1 = x0gt10 Specify an AFP raster font by coded font name
tfont2 = c0d0gt12,t1d0base Specify an AFP raster font by character set and coded page name
tfont3 = xzhe00,12.5 Specify an AFP outline font by coded font name and size
tfont4 = 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

\[
\text{FONTn} = \{ \text{ttf_filename} \mid (\text{ttf_filename}, \text{font_index}) \}, \text{encoding}, \text{height_point} \\
\text{[scale_ratio]}
\]
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.

\[ \text{ttf\_filename} \]
Any valid filename of the OpenType/TrueType font or TrueType font collection with file extensions of .ttf, .otf and .ttc.

\[ \text{ttf\_index} \]
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.

\[ \text{encoding} \]
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 other 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 controller or AFP viewer.

MakeAFP Formatter also provides comprehensive encoding conversion functions from other encodings to UTF8 or UTF16BE for using OpenType/TrueType fonts.

The following encoding types are allowed in AFP standard:

- **T1xxxxxx**
  ASCII, EBCDIC, DBCS-HOST codepage name defined by IBM codepage standard, refer to Appendix A to B for more details about IBM codepage name.

- **UTF8**
  AFP text data is encoded in Unicode UTF-8. Simple code point conversions from UTF8 to UTF-16BE is performed quickly in the presentation device for AFP.

- **UTF16BE**
  AFP text data is encoded in UTF16BE, it is the native encoding being used by AFP IPDS printer controllers for using OpenType/TrueType fonts.

\[ \text{height\_point} \]
Specifies the height of the outline font in points (Each point is equal to 1/72 of one inch).

\[ \text{scale\_ratio} \]
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 old 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 list. Multiple paths can be separated by the semicolon (;). MakeAFP Formatter searches the paths in the order in which they are specified.

MakeAFP Formatter auto-searches TrueType/OpenType font from your Windows system font path if it cannot be found from the font libraries specified.

When MakeAFP Formatter finds more than one AFP font with the same file base-name 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 AFP font 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 Formatter 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  |  NO  }

**Parameter**

**YES**

Specifies that the AFP index object file is generated to be used by an AFP archiving system or AFP Viewer. MakeAFP Formatter generates the AFP index object file with the file name extension .ind.

Make sure the CPGID parameter is defined in your MakeAFP Formatter 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 AFP or non-AFP object to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Formatter.

A separate OBJT parameter is required for each object, and a maximum of 256 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 Formatter 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 double-quoted object name can be 1 to 125 alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

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.

type
Indicates type of the object:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCOCA</td>
<td>AFP BCOCA barcode object</td>
</tr>
<tr>
<td>GOCA</td>
<td>AFP GOCA graphic object</td>
</tr>
<tr>
<td>IOCA</td>
<td>AFP IOCA image object</td>
</tr>
<tr>
<td>PSEG</td>
<td>AFP Page Segment image object</td>
</tr>
<tr>
<td>BMP</td>
<td>Windows Device Dependent Bit Map</td>
</tr>
<tr>
<td>EPS</td>
<td>Encapsulated Postscript</td>
</tr>
<tr>
<td>EPSTR</td>
<td>EPS with Transparency</td>
</tr>
<tr>
<td>GIF</td>
<td>Graphics Interchange Format</td>
</tr>
<tr>
<td>JPEG</td>
<td>JPEG file Interchange Format</td>
</tr>
<tr>
<td>JPEG2</td>
<td>JPEG2000 file Interchange Format</td>
</tr>
<tr>
<td>PCX</td>
<td>Paintbrush Picture File Format</td>
</tr>
<tr>
<td>PDF</td>
<td>PDF Single Page Object</td>
</tr>
<tr>
<td>PDFSPOTR</td>
<td>PDF Single Page Object with Transparency</td>
</tr>
<tr>
<td>TIFF</td>
<td>Tag Image File Format</td>
</tr>
</tbody>
</table>

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. Multiple paths must be separated by the semicolon (;). MakeAFP Formatter searches the paths in the order in which they are specified.

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

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

Note: Some data-object image 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 overlay to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Formatter.

A separate OVLY parameter is required for each overlay, and a maximum of 256 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 Formatter 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 = 01CDP01
ovly = 01CDP02

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. Multiple paths must be separated by the semicolon (;). MakeAFP Formatter searches the paths in the order in which they are specified.

When MakeAFP Formatter finds more than one overlay with the same file base-name in the same directory, it selects the matching overlay resource 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
Note: Some AFP overlay 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 Formatter must perform optional processing on that data or not.

MakeAFP Formatter default is ASCII input data if you do not specify this parameter.

**Syntax**

```
PRMODE={EBCDIC | SOSI1 | SOSI2}
```

**Parameter**

**EBCDIC**

Specifies that input data is in EBCDIC encoding from IBM mainframes.

**SOSI1**

Specifies that input data is in SBCS-HOST/DBCS-HOST encoding, replaces each SO(shift-out) and SI(shift-in) code with a blank character.

**SOSI2**

Specifies that input data is in SBCS-HOST/DBCS-HOST encoding, removes SO(shift-out) and SI(shift-in) codes.

**SOSI3**

Specifies that input data is in SBCS-HOST/DBCS-HOST encoding, removes the SO(shift-out) code, and replaces the SI(shift-in) code with two blank characters.

Sample

```
prmode = sosi1
```

---

**PSEG – Specifies a Page Segment**

**Function**

Specifies the page segment to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Formatter.

A separate PSEG parameter is required for each page segment, and a maximum of 256 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
Formatter builds 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. Multiple paths must be separated by the semicolon (;). MakeAFP Formatter searches the paths in the order in which they are specified.

When MakeAFP Formatter finds more than one page-segment with the same file base-name in the same directory, it selects the matching page segment resource 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

**Note:** Some page segment 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. Multiple paths must be separated by the semicolon (;). MakeAFP Formatter searches the paths in the order you specified. MakeAFP Formatter auto-searches TrueType/OpenType font from your Windows system font path if it cannot be found from the resources libraries specified.

Sample

reslib = c:\makeafp\reslib;d:\ipmwin\reslib

RESTYPE - Specifies the Types of Resources to be Retrieved

Function

Specifies the types of resources that should be retrieved from the resource directories, and whether the retrieved resources are being embedded inline within the AFP document file or as a separated AFP resource file.

Syntax

RESTYPE = \{ NONE | ALL | [CMR] [FDEF] [FONT] [OBJT] [OVLY] [PSEG] \} [INLINE]

Parameter

MakeAFP Formatter supports the specification of the parameters in any combination.

NONE

Specifies that no AFP resources file is being created or AFP resources are written inline within the AFP document file. This is the default, in this case, you have to 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 document file.

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 document file.

FONT
Specifies that all AFP fonts and OpenType/TrueType fonts are embedded in the resource file or inline within the AFP document file.

OBJT
Specifies that all AFP objects or non-AFP objects are embedded in the resource file or inline within the AFP document file.

OVLY
Specifies that all overlays are embedded in the resource file or inline within the AFP document file.

PSEG
Specifies that all page segments are embedded in the resource file or inline within the AFP document file.

INLINE
Specifies that resources are embedded inline within the AFP document file, otherwise a separate AFP resource file with 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 form definition, overlays, non-AFP object and page segments inline for viewing by IBM AFP viewer and IBM DB2 Content Manager OnDemand:

\[ \text{restype} = \text{fdef,ovly,pseg,OBJT,inline} \]
Chapter 3. MakeAFP Formatter Variables

In addition to the MakeAFP Formatter functions described in Chapter 1, there are several variables maintained by MakeAFP Formatter for internal use or exchanging of information during the data formatting. The MakeAFP Formatter variables described below are accessible from your program.

FileInName – File Name of the Default Input File

Contains the name of default input data file specified by the flag parameter “‐i infile_filename” of your command in batch mode.

This variable gives you a choice to open and access the input data file by your programming completely.

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 Formatter 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 Formatter default value is 1, you can override its value before you start a MakeAFP Formatter session by calling the function of “Start”. MakeAFP Formatter reports an error message if its value is not enough for your job.

PageNum – Current AFP Page Buffer Number

Defines the current AFP page buffer number. With the PageNum 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 Formatter 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

```c
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

```c
_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

```c
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

```c
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

```c
char *DeleteString(
    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

```c
char data[30] = "This is a string testing";
```
printf("After Deleted: %s", DeleteString (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

```c
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 substring 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 substring 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, and returns its position if found, otherwise returns 0.

**Syntax**

```c
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 FirstChar(
    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

```c
char *payment [] [2] = { {"001", "Cash Payment"},
{"005", "Master Payment"},
{"003", "Visa Payment"},
{"007", "Check Payment"},
{"011", "GORO payment"},
{"\0", "\0"} }; // End of table
```

```c
int intab = InSubst(payment, "002");
intab = InSubst(payment, "007");
```

Return:

0
1

---

**Insert String**

**Function**

Inserts a character string after the specified position.

**Syntax**

```c
char *InsertString(
    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 into the destination string.

**Sample**

```c
char src[256] = "This is data string";
printf("%s", InsertString(src,8,"inserted"));
```

Output:

This is inserted data string

---

**IsEmpty**

**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**

```c
int IsEmptyString(char* string);
```
Parameters

```
string
data string.
```

Sample

```
char str1[80] = "This is data string";
char str2[80] = "
    ";
char str3[80] = NULL;
int empty = isEmptyString(str1);
   empty = isEmptyString(str2);
   empty = isEmptyString(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**
Number of characters to be copied.

Sample

```c
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 the 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

```c
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

```c
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, newline 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
```c
char *E_Ltrim(char* string);
```

Parameters
- `string` EBCDIC data string to be left trim.

Sample
Refer to the sample for the “LTrim” function.

Match String Comparing

Function
Recursively compares a string to a pattern, returning 1 if a match is found or 0 if not.

Syntax
```c
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

```c
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 found a match.

**Syntax**

```c
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**

```c
char str[] = "Boulder, CO 12345-5768-88 USA";

Pattern(str, "#####", 1);
Pattern(str, "##-##", 15);
Pattern(str, "## USA", 23);
```
Remove String

Function

Removes a single instant of string or multiple instances of the string.

Syntax

Remove once:

```c
char *RemoveString1(
    char* srcStr,
    char* rmStr
);
```

Remove all:

```c
char *RemoveString(
    char* srcStr,
    char* rmStr
);
```

Parameters

srcStr
Source string.

rmStr
The string to be removed from the source string.

Sample

```c
char str[] = "This is string testing string."
printf("After Removed once: %s\n", RemoveString1(str," string"));
printf("After Removed all: %s\n", RemoveString (str," string"));
```

Output:

After Removed once: This is testing string.
After Removed all: This is testing.

Replace String

Function

Replaces a single instant of string or multiple instances of the string.

Syntax

Replace once:

```c
char *Replace1(
    char* srcStr,
    char* tgtStr,
    char* newStr
);
```
Replace all:

```c
char *Replace(
    char* srcStr,
    char* tgtStr,
    char* newStr
);
```

Parameters

- **srcStr**: Source string.
- **tgtStr**: Target string to be replaced.
- **newStr**: The new string to be used to replace the target string.

Sample

```c
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

```c
int Rfind(
    char* str,
    char* search
)
```

Or

```c
int Rfind1(
    char* str,
    char* search,
    int start_pos
)
```

Or

```c
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

```c
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

```c
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 the source string, and pad to the left of the destination string with the pad character if appropriate.

Syntax

```c
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

```c
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 the destination string where the white-space, carriage return, newline control codes would be trimmed from the right before being terminated with NULL.

**Syntax**

```c
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**

```c
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, newline 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 for EBCDIC

Function

Trims EBCDIC white-space, carriage return, newline 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.

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
```c
char *SubstChg(
    char* subst_tbl[][2],
    char* srcStr
);
```

Parameters
- **subst_tbl**: Substitution table.
- **srcStr**: Source string.

Sample
```c
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
```c
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 the substring.

Sample

```c
char str[] = "This is a substring testing";
printf("Result: %s", SubStr2(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

```c
char *SubStrLtrim(
    char*   dstStr,
    char*   srcStr,
    ushort   from_pos,
    ushort   length
);
```

```c
char *SubStrLtrim(
    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";
printf("Result: %s", SubStrLtrim2(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
);

cchar *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";
printf("Result: %s", SubStrPad2(str,9,12,'*'));

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

```c
char *SubStrRtrim(
    char* dstStr,
    char* srcStr,
    ushort from_pos,
    ushort length
);
```

```c
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

```c
char str1[] = "This is a substring testing";
printf("Result: %s", SubStrRtrim2(str1, 11, 16));
```

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, newline control codes from both sides of the destination string.

Syntax

```c
char *SubStrTrim(
    char* dstStr,
    char* srcStr,
    ushort from_pos,
    ushort length
);
```
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";

printf("Result: %s", SubStrTrim2(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

c 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

```c
char *ThaiCompose(
    char* thai_string,
    bool align_field = false
);
```

```c
char *ThaiCompose2(
    char* thai_dst_string,
    char* thai_src_string,
    bool align_field = false
);
```

Parameters

- `thai_string`, `thai_src_string`
  The Thai ASCII data string to be used for composition.

- `thai_dst_string`
  The destination Thai string.

- `align_fields`
  Specifies whether aligns Thai data fields in byte-level.

Sample

No sample was provided.

Translate Digits to Simplified Chinese Figures

Function

Translate ASCII digits to Simplified Chinese figures in GB18030 encoding.

Syntax

```c
char *DigitGBK(
    char* dst,
    char* src
);
```

Parameters

- `dst`
  Destination of Simplified 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));
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

```c
char *E_Trim(char* string);
```

Parameters

- **string**
  EBCDIC data string to be trimmed.

Sample

Refer to the sample for the Trim function.

Vietnamese Alignment

Function

Aligns Vietnamese data fields in byte-level.

Syntax

```c
char *VietAlign(char *str);
char *VietAlign2(char *dst char *str);
```

Parameters

- **str**
  The source data string.

- **dst**
  The destination string.

Sample

None.
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 Wild-Char Conversion

Function

Converts from codepage/charset stream to wild-char returns the length of the wild-char output.

Syntax

```c
int32_t ChartoW (
    wchar_t *target,
    int32_t *targetCapacity,
    char *source,
    int32_t sourceLen = -1,
    char fromCode = NULL
);
```

Parameters

- **target**
  Point to the targeted wild-char output buffer.

- **targetCapacity**
  The maximum size of the targeted wild-char 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 Appendix D. for more details about the available encoding names and alias.
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

```c
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`
  The maximum size of 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 Appendix D. for more details about the available encoding names and alias.

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

```c
void DefaultCode(
    char       *codename = "windows-1252"
);
```
Parameters

codeName
The name of the default encoding, default is “windows-1252”. Refer to Appendix D. for more details about the available encoding names and alias.

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

```c
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 Appendix D. for more details about the available encoding names and alias.

fromCode
The name of the source encoding. Refer to Appendix D. for more details about the available encoding names and alias.

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.
Wild-Char to Codepage/Charset Conversion

Function

Converts from wild-char to a codepage/charset stream and returns the length of the complete target output.

Syntax

```c
int32_t WtoChar(
    char *target,
    int32_t *targetCapacity,
    wchar_t *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 wild-char input source buffer.

sourceLen
Length of the wild-char 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 Appendix D. for more details about the available encoding names and alias.

Wild-Char to UTF-8 Conversion

Function

Converts from wild-char to UTF-8 and returns the length of the complete UTF-8 target output.

Syntax

```c
int32_t WtoU8(
    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 wild-char input source buffer.

**sourceLen**
Length of the wild-char input source, or default -1 for NULL-terminated input.

---

**UTF-8 to Wild-Char Conversion**

**Function**

Converts from UTF-8 to wild-char and returns the length of the complete wild-char target output.

**Syntax**

```c
int32_t U8toW(
    wchar_t *target,
    int32_t *targetCapacity,
    UChar8 *source,
    int32_t sourceLen = -1
);
```

**Parameters**

**target**
Point to the target wild-char output buffer.

**targetCapacity**
The maximum size of the wild-char 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, and returns the length of the complete target output.
Syntax

```c
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 Appendix D for more details about the available encoding names and alias.

Syntax

```c
int32_t U8toChar(
    UChar *target,
    int32_t *targetCapacity,
    UCha8 *source,
    int32_t sourceLen = -1
);
```

**Parameters**

**target**
Point to the target wild-char output buffer.

**targetCapacity**
The maximum size of the wild-char 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

```c
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>CPGID</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1000038</td>
<td>US-ASCII Character Set</td>
<td>38</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000259</td>
<td>Symbols, Set 7</td>
<td>259</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000260</td>
<td>Canadian French - 116</td>
<td>260</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000276</td>
<td>Canada (French) - 94</td>
<td>276</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000286</td>
<td>Austria/Germany F.R., Alt (3270)</td>
<td>286</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000287</td>
<td>Denmark/Norway, Alternate (3270)</td>
<td>287</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000288</td>
<td>Finland/Sweden, Alternate (3270)</td>
<td>288</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000289</td>
<td>Spain, Alternate (3270)</td>
<td>289</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000290</td>
<td>Japan (Katakana)</td>
<td>290</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000293</td>
<td>APL (USA)</td>
<td>293</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000310</td>
<td>Graphic Escape APL/TN</td>
<td>310</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000361</td>
<td>International Set 5</td>
<td>361</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000363</td>
<td>Symbols, Set 8</td>
<td>363</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000367</td>
<td>ASCII</td>
<td>367</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000382</td>
<td>Austria, Germany, Japan</td>
<td>382</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000383</td>
<td>Belgium</td>
<td>383</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000384</td>
<td>Brazil</td>
<td>384</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000385</td>
<td>Canada (French)</td>
<td>385</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000386</td>
<td>Denmark/Norway</td>
<td>386</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000387</td>
<td>Sweden/Finland</td>
<td>387</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000388</td>
<td>France, Japan</td>
<td>388</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000389</td>
<td>ITALY, Japan (Italian)</td>
<td>389</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000390</td>
<td>Japan (Latin)</td>
<td>390</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000391</td>
<td>Portugal</td>
<td>391</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000392</td>
<td>Spain/Philippines</td>
<td>392</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000393</td>
<td>Latin America (Spanish)</td>
<td>393</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000394</td>
<td>U.K., Austral., IRE., H.K., N.Z.</td>
<td>394</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000395</td>
<td>United States, Canada (English)</td>
<td>395</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000420</td>
<td>Arabic Bilingual</td>
<td>420</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000423</td>
<td>Greece - 183</td>
<td>423</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000424</td>
<td>Israel (Hebrew)</td>
<td>424</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000437</td>
<td>Personal Computer</td>
<td>437</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000803</td>
<td>Hebrew Character Set A</td>
<td>803</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000808</td>
<td>PC, Cyrillic, Russian with euro</td>
<td>808</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000813</td>
<td>Greece - ISO/ASCII 8-Bit</td>
<td>813</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000819</td>
<td>Latin1 ISO/ANSI 8-BIT</td>
<td>819</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000829</td>
<td>Math Symbols</td>
<td>829</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000836</td>
<td>Peoples Republic of China (PRC)</td>
<td>836</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000838</td>
<td>Thai - EBCDIC</td>
<td>838</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000848</td>
<td>PC, Cyrillic, Ukraine with Euro</td>
<td>848</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000849</td>
<td>PC, Cyrillic, Belo Russian Euro</td>
<td>849</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000850</td>
<td>PC Multilingual</td>
<td>850</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000851</td>
<td>Greek - Personal Computer</td>
<td>851</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000852</td>
<td>Latin2 Multilingual PC</td>
<td>852</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000853</td>
<td>Latin3 Personal Computer</td>
<td>853</td>
<td>ASCII</td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td>Code</td>
<td>Encoding</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>T1000855</td>
<td>Cyrillic - Personal Computer</td>
<td>855</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000856</td>
<td>Hebrew - Personal Computer</td>
<td>856</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000857</td>
<td>Latin5 PC</td>
<td>857</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000858</td>
<td>PC - Multilingual with euro</td>
<td>858</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000860</td>
<td>Portugal - Personal Computer</td>
<td>860</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000861</td>
<td>Iceland - Personal Computer</td>
<td>861</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000862</td>
<td>Hebrew - Personal Computer</td>
<td>862</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000863</td>
<td>Canadian French - PC</td>
<td>863</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000864</td>
<td>Arabic - Personal Computer</td>
<td>864</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000865</td>
<td>Nordic - Personal Computer</td>
<td>865</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000866</td>
<td>Cyrillic #2 - Personal Computer</td>
<td>866</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000867</td>
<td>Israel - Personal Computer</td>
<td>867</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000869</td>
<td>Greece - Personal Computer</td>
<td>869</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000870</td>
<td>Latin2 Multilingual</td>
<td>870</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000872</td>
<td>Cyrillic PC with Euro</td>
<td>872</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000874</td>
<td>Thai - Personal Computer</td>
<td>874</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000875</td>
<td>Iceland - Personal Computer</td>
<td>875</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000876</td>
<td>OCR-AN ASCII</td>
<td>876</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000877</td>
<td>OCR-B ASCII</td>
<td>877</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000880</td>
<td>Cyrillic Multilingual</td>
<td>880</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000889</td>
<td>Thailand</td>
<td>889</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000892</td>
<td>OCR - A</td>
<td>892</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000893</td>
<td>OCR - B</td>
<td>893</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000897</td>
<td>Japan PC #1</td>
<td>897</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000899</td>
<td>Symbols, Set 7 ASCII</td>
<td>899</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000901</td>
<td>PC, Baltic - Multilingual w Euro</td>
<td>901</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000902</td>
<td>8-bit Estonia with euro</td>
<td>902</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000903</td>
<td>Peoples Republic of China - PC</td>
<td>903</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000904</td>
<td>Republic of China (ROC) - PC</td>
<td>904</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000905</td>
<td>Latin3 Multilingual</td>
<td>905</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1000910</td>
<td>APL ASCII</td>
<td>910</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000912</td>
<td>Latin2 ISO/ANSI 8-BIT</td>
<td>912</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000913</td>
<td>Latin 3, ISO/ASCII</td>
<td>913</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000914</td>
<td>Latin4 ISO/ANSI 8-BIT</td>
<td>914</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000915</td>
<td>Cyrillic ISO/ASCII 8-Bit</td>
<td>915</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000916</td>
<td>Hebrew ISO/ASCII 8-Bit</td>
<td>916</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000920</td>
<td>Latin5 ISO/ANSI 8-BIT</td>
<td>920</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000921</td>
<td>PC, Baltic - Multilingual</td>
<td>921</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000922</td>
<td>Estonia PC</td>
<td>922</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000923</td>
<td>Latin 9</td>
<td>923</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1000924</td>
<td>Latin 9 EBCDIC</td>
<td>924</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001002</td>
<td>DCF REL 2 Compatibility</td>
<td>1002</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001003</td>
<td>U.S. Text Subset</td>
<td>1003</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001004</td>
<td>IBM PC Desktop Publishing</td>
<td>1004</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001008</td>
<td>Arabic ISO/ASCII 8-Bit</td>
<td>1008</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001025</td>
<td>Cyrillic Multilingual</td>
<td>1025</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001026</td>
<td>Latin5</td>
<td>1026</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001027</td>
<td>Japanese (Latin) Extended</td>
<td>1027</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001028</td>
<td>Hebrew Publishing</td>
<td>1028</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001029</td>
<td>Arabic Extended ISO/ASCII 8-Bit</td>
<td>1029</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001032</td>
<td>MICR, E13-B Combined</td>
<td>1032</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001033</td>
<td>MICR, CMC-7 Combined</td>
<td>1033</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001038</td>
<td>Symbols, Adobe ASCII</td>
<td>1038</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001039</td>
<td>GML List Symbols</td>
<td>1039</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001041</td>
<td>Japanese Extended - PC</td>
<td>1041</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001042</td>
<td>Simplified Chinese Extended - PC</td>
<td>1042</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001043</td>
<td>Traditional Chinese Extended PC</td>
<td>1043</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001046</td>
<td>Arabic Extended ISO/ASCII 8-Bit</td>
<td>1046</td>
<td>ASCII</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Encoding</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>T1001068</td>
<td>Text With Numeric Spacing</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001069</td>
<td>Latin4 EBCDIC</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001087</td>
<td>Symbols, Adobe</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001091</td>
<td>Symbol Set 7, Modified</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001092</td>
<td>Symbol Set 7, Modified - PC</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001093</td>
<td>IBM LOGO</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001110</td>
<td>Latin2 Multilingual</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001111</td>
<td>Latin2 ISO/ANSI 8-BIT</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001112</td>
<td>Baltic - Multilingual, EBCDIC</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001122</td>
<td>Estonia, EBCDIC</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001123</td>
<td>Cyrillic, Ukraine EBCDIC</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001124</td>
<td>Cyrillic, Ukraine ISO-8</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001125</td>
<td>PC, Cyrillic Ukrainian</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001129</td>
<td>Vietnamese ISO-8</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001130</td>
<td>Vietnamese EBCDIC</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001131</td>
<td>PC, Cyrillic, Belo Russian</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001132</td>
<td>Lao EBCDIC</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001133</td>
<td>Lao ISO-8</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001139</td>
<td>Japan Alphanumeric Katakana</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001140</td>
<td>USA, Canada ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001141</td>
<td>Austria, Germany ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001142</td>
<td>Denmark, Norway ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001143</td>
<td>Finland, Sweden ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001144</td>
<td>Italy ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001145</td>
<td>Spain, Latin America ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001146</td>
<td>UK ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001147</td>
<td>France ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001148</td>
<td>International ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001149</td>
<td>Iceland ECECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001153</td>
<td>Latin2 Multilingual with Euro</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1001254</td>
<td>Windows Turkish</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001257</td>
<td>Windows Baltic Rim</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001258</td>
<td>Windows Vietnamese</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001275</td>
<td>Apple Latin 1</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001276</td>
<td>Adobe PS Standard</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001277</td>
<td>Adobe PS ISO Latin 1</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001280</td>
<td>Apple Greece</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001281</td>
<td>Apple Turkey</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001282</td>
<td>Apple Central Europe</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001283</td>
<td>Apple Cyrillic</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1001300</td>
<td>GENERIC BAR CODE/OCR-B</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1005346</td>
<td>Latin 2 – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005347</td>
<td>Cyrillic – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005348</td>
<td>Latin 1 – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005349</td>
<td>Greece – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005350</td>
<td>Turkey – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005351</td>
<td>Israel – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005352</td>
<td>Arabic – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005353</td>
<td>Latin 4 – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>T1005354</td>
<td>Vietnamese – Windows</td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>TIV100037</td>
<td>USA/Canada - CECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>TIV10273</td>
<td>Germany F.R./Austria- CECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>TIV10274</td>
<td>Belgium - CECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>TIV10275</td>
<td>Brazil - CECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>TIV10277</td>
<td>Denmark/Norway - CECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>TIV10278</td>
<td>Finland/Sweden- CECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>TIV10280</td>
<td>ITALY- CECP</td>
<td>EBCDIC</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Code</td>
<td>Encryption</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>T1V10281</td>
<td>Japan (Latin) - CECP</td>
<td>281</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1V10282</td>
<td>Portugal - CECP</td>
<td>282</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1V10284</td>
<td>Spain/Latin America - CECP</td>
<td>284</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1V10285</td>
<td>UNITED KINGDOM - CECP</td>
<td>285</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1V10290</td>
<td>Japan (Katakana)</td>
<td>290</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1V10297</td>
<td>France - CECP</td>
<td>297</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1V10500</td>
<td>International #5</td>
<td>500</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1V10871</td>
<td>iceland - CECP</td>
<td>871</td>
<td>EBCDIC</td>
</tr>
</tbody>
</table>
## Appendix B. SBCS/DBCS AFP Code Pages and CPGID Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>CPGID</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1H00037</td>
<td>Traditional Chinese EBCDIC</td>
<td>037</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H00290</td>
<td>Japanese Katakana Extended</td>
<td>290</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H00833</td>
<td>Korean EBCDIC</td>
<td>833</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H00836</td>
<td>Simplified Chinese EBCDIC</td>
<td>836</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01002</td>
<td>Japanese DCF Rel 2 Compatibility</td>
<td>1002</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01027</td>
<td>Japanese Latin Extended</td>
<td>1027</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01030</td>
<td>Japanese Katakana Extended with Box Characters</td>
<td>1030</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01031</td>
<td>Japanese Latin Extended with Box Characters</td>
<td>1031</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01041</td>
<td>Japanese PC Extended</td>
<td>1041</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1H01043</td>
<td>Traditional Chinese PC</td>
<td>1043</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1H01114</td>
<td>Traditional Chinese PC BIG5 with Euro</td>
<td>1114</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1H01115</td>
<td>Simplified Chinese PC (GB)</td>
<td>1115</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1H01126</td>
<td>Korean PC</td>
<td>1126</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1H01150</td>
<td>Korean EBCDIC with Box Characters</td>
<td>1150</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01151</td>
<td>Simplified Chinese EBCDIC with Box Characters</td>
<td>1151</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01152</td>
<td>Traditional Chinese EBCDIC with Box Characters</td>
<td>1152</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H01159</td>
<td>Traditional Chinese EBCDIC with Euro</td>
<td>1159</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1H0252</td>
<td>Simplified Chinese PC (GB18030)</td>
<td>1252</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1HK0037</td>
<td>Japanese English</td>
<td>037</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1HK0290</td>
<td>Japanese Katakana</td>
<td>290</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T010300, T1I300, T1J300, T1K300</td>
<td>Japanese DBCS-HOST</td>
<td>300</td>
<td>DBCS-HOST</td>
</tr>
<tr>
<td>T10834</td>
<td>Korean DBCS-HOST (Small Set)</td>
<td>834</td>
<td>DBCS-HOST</td>
</tr>
<tr>
<td>T10835</td>
<td>Traditional Chinese DBCS-HOST</td>
<td>835</td>
<td>DBCS-HOST</td>
</tr>
<tr>
<td>T10837</td>
<td>Simplified Chinese DBCS-HOST (GB2312)</td>
<td>837</td>
<td>DBCS-HOST</td>
</tr>
<tr>
<td>T10941</td>
<td>Japanese SJIS-PC</td>
<td>941</td>
<td>DBCS-PC</td>
</tr>
<tr>
<td>T10947</td>
<td>Traditional Chinese BIG5-PC</td>
<td>947</td>
<td>DBCS-PC</td>
</tr>
<tr>
<td>T10951</td>
<td>Korean KSC-PC (Small Set)</td>
<td>951</td>
<td>DBCS-PC</td>
</tr>
<tr>
<td>T11362</td>
<td>Korean KSC-PC (Big Set)</td>
<td>1362</td>
<td>DBCS-PC</td>
</tr>
<tr>
<td>T11374</td>
<td>Traditional Chinese HKSCS-PC</td>
<td>1374</td>
<td>DBCS-PC</td>
</tr>
<tr>
<td>T11376</td>
<td>Traditional Chinese HKSCS-HOST</td>
<td>1376</td>
<td>DBCS-HOST</td>
</tr>
<tr>
<td>T11380</td>
<td>Simplified Chinese GB2312-PC (Small Set)</td>
<td>1380</td>
<td>DBCS-PC</td>
</tr>
<tr>
<td>T11385</td>
<td>Simplified Chinese GBK-PC (Big Set)</td>
<td>1385</td>
<td>DBCS-PC</td>
</tr>
<tr>
<td>T1K834</td>
<td>Korean DBCS-HOST (Big Set)</td>
<td>837</td>
<td>DBCS-HOST</td>
</tr>
<tr>
<td>T1K837</td>
<td>Simplified Chinese DBCS-HOST (GB18030)</td>
<td>837</td>
<td>DBCS-HOST</td>
</tr>
</tbody>
</table>
## Appendix C. Combined SBCS/DBCS CPGID Summary

<table>
<thead>
<tr>
<th>CPGID</th>
<th>Description</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>937</td>
<td>Combination of Traditional Chinese SBCS-HOST and DBCS-HOST</td>
<td>EBCDIC/DBCS-HOST (CP37/CP835)</td>
</tr>
<tr>
<td>939</td>
<td>Combination of Japanese SBCS-HOST and DBCS-HOST</td>
<td>EBCDIC/DBCS-HOST (CP1027/CP300)</td>
</tr>
<tr>
<td>943</td>
<td>Combination of Japanese ASCII and SJIS-PC for open systems</td>
<td>ASCII/SJIS-PC (CP897/CP941)</td>
</tr>
<tr>
<td>950</td>
<td>Combination of Traditional Chinese ASCII and BIG5-PC for open systems</td>
<td>ASCII/BIG5-PC (CP1114/CP947)</td>
</tr>
<tr>
<td>1363</td>
<td>Combination of Korean ASCII and KSC-PC for open systems</td>
<td>ASCII/KSC-PC (CP1126/CP1362)</td>
</tr>
<tr>
<td>1364</td>
<td>Combination of Korean SBCS-HOST and DBCS-HOST</td>
<td>EBCDIC/DBCS-HOST (CP833/CP834)</td>
</tr>
<tr>
<td>1375</td>
<td>Combination of Traditional Chinese ASCII and HKSCS-PC for open systems</td>
<td>ASCII/HKSCS-PC (CP1114/CP1374)</td>
</tr>
<tr>
<td>1377</td>
<td>Combination of Traditional Chinese SBCS-HOST and DBCS-HOST for HKCS</td>
<td>ASCII/GBK-PC (CP37/CP1376)</td>
</tr>
<tr>
<td>1386</td>
<td>Combination of Simplified Chinese ASCII and GBK-PC for open systems</td>
<td>ASCII/GBK-PC (CP1114/CP1385)</td>
</tr>
<tr>
<td>1388</td>
<td>Combination of Simplified Chinese SBCS-HOST and DBCS-HOST for GBK</td>
<td>EBCDIC/DBCS-HOST (CP836/CP837)</td>
</tr>
<tr>
<td>1392</td>
<td>Combination of Simplified Chinese ASCII and GB18030-PC for open systems</td>
<td>ASCII/GB18030</td>
</tr>
<tr>
<td>13767</td>
<td>Combination of Simplified Chinese SBCS-HOST and DBCS-HOST for GB18030</td>
<td>EBCDIC/DBCS-HOST (CP836/CP837)</td>
</tr>
</tbody>
</table>
Appendix D. Encoding Names and Alias

<table>
<thead>
<tr>
<th>Encoding Name and Alias</th>
<th>Description</th>
<th>CPGID</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOCU-1, csBOCU-1, ibm-1214, ibm-1215</td>
<td>Binary Ordered Compression for Unicode, it combines the wide applicability of UTF-8 with the compactness of Standard Compression Scheme for Unicode</td>
<td></td>
</tr>
<tr>
<td>CESU-8, ibm-9400</td>
<td>CESU-8 is a Compatibility Encoding Scheme for wild-char (CESU) that serializes a Unicode code point as a sequence of one, two, three or six bytes</td>
<td></td>
</tr>
<tr>
<td>ebcDIC-xml-us</td>
<td>XML in EBCDIC-US</td>
<td></td>
</tr>
<tr>
<td>HZ, HZ-GB-2312</td>
<td>Simplified Chinese, International and national Standard</td>
<td></td>
</tr>
<tr>
<td>ibm-37, IBM037, ibm-037, ebcDIC-cp-us, ebcDIC-cp-ca, ebcDIC-cp-wt, ebcDIC-cp-nl, cslIBM037, cp037, 037, cpibm37, cp37, T1V10037</td>
<td>USA/Canada – CECP, EBCDIC</td>
<td>037</td>
</tr>
<tr>
<td>Ibm-259, IBM-Symbols, cslIBM-Symbols</td>
<td>Symbols, Set 7, EBCDIC</td>
<td>259</td>
</tr>
<tr>
<td>ibm-273, IBM273, CP273, cslIBM273, ebcDIC-de, cplb273, 273, T1V10273</td>
<td>Germany F.R./Austria- CECP, EBCDIC</td>
<td>273</td>
</tr>
<tr>
<td>ibm-277, IBM277, cp277, EBCDIC-CP-DK, EBCDIC-CP-NO, cslIBM277, ebcDIC-dk, cplb277, 277, T1V10277</td>
<td>Denmark/Norway - CECP, EBCDIC</td>
<td>277</td>
</tr>
<tr>
<td>ibm-278, IBM278, cp278, ebcDIC-cp-fi, ebcDIC-cp-se, cslIBM278, ebcDIC-sv, cplb278, 278, T1V10278</td>
<td>Finland/Sweden- CECP, EBCDIC</td>
<td>278</td>
</tr>
<tr>
<td>ibm-280, IBM280, CP280, ebcDIC-cp-it, cslIBM280, cpibm280, 280, T1V10280</td>
<td>ITALY- CECP, EBCDIC</td>
<td>280</td>
</tr>
<tr>
<td>ibm-284, IBM284, CP284, ebcDIC-cp-es, cslIBM284, cpibm284, 284, T1V10284</td>
<td>Spain/Latin America - CECP, EBCDIC</td>
<td>284</td>
</tr>
<tr>
<td>ibm-286, EBCDIC-AT-DE-A, cslEBCDICATDEA, T1000286</td>
<td>Austria/Germany F.R., Alt (3270), EBCDIC</td>
<td>286</td>
</tr>
<tr>
<td>ibm-290, IBM290, cp290, EBCDIC-JP-kana, cslIBM290, T1V10290</td>
<td>Japan (Katakana), EBCDIC</td>
<td>290</td>
</tr>
<tr>
<td>ibm-293, T1000293</td>
<td>APL EBCDIC</td>
<td>293</td>
</tr>
<tr>
<td>ibm-297, IBM297, cp297, ebcDIC-</td>
<td>France – CECP, EBCDIC</td>
<td>297</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>cp-fr</td>
<td>csIBM297, cpibm297, 297, T1V10297</td>
<td></td>
</tr>
<tr>
<td>ibm-420</td>
<td>IBM420, cp420, ebcdic-cp-ar1, csIBM420, 420, T1000420</td>
<td></td>
</tr>
<tr>
<td>ibm-424</td>
<td>IBM424, cp424, ebcdic-cp-he, csIBM424, 424, T1000424</td>
<td></td>
</tr>
<tr>
<td>ibm-437</td>
<td>IBM437, cp437, 437, csPC8CodePage437, windows-437, T1000437</td>
<td></td>
</tr>
<tr>
<td>ibm-500</td>
<td>IBM500, CP500, ebcdic-cp-be, csIBM500, ebcdic-cp-ch, cpibm500, 500, T1V10500</td>
<td></td>
</tr>
<tr>
<td>ibm-720</td>
<td>windows-720, DOS-720, PC Arabic, ASCII</td>
<td></td>
</tr>
<tr>
<td>ibm-737</td>
<td>IBM737, cp737, windows-737, 737</td>
<td></td>
</tr>
<tr>
<td>ibm-775</td>
<td>IBM775, cp775, csPC775Baltic</td>
<td></td>
</tr>
<tr>
<td>ibm-803</td>
<td>cp803, T1000803</td>
<td></td>
</tr>
<tr>
<td>ibm-808</td>
<td>IBM808, T1000808</td>
<td></td>
</tr>
<tr>
<td>ibm-813</td>
<td>iso-8859-7, greek, greek8, ELOT_928, ECMA-118, csISOLatinGreek, iso-ir-126, ISO_8859-7:1987, 8859_7, cp813, 813, T1000813</td>
<td></td>
</tr>
<tr>
<td>ibm-819</td>
<td>IBM819, cp819, latin1, 8859_1, csISOLatin1, iso-ir-100, ISO_8859-1:1987, 11, 819, T1000819</td>
<td></td>
</tr>
<tr>
<td>ibm-838</td>
<td>IBM-Thai, csIBMThai, cp838, 838, ibm-9030, ibm838, T1000838</td>
<td></td>
</tr>
<tr>
<td>ibm-848</td>
<td>IBM848, T1000848</td>
<td></td>
</tr>
<tr>
<td>ibm-849</td>
<td>IBM849, T1000849</td>
<td></td>
</tr>
<tr>
<td>ibm-850</td>
<td>IBM850, cp850, 850, csPC850Multilingual, windows-850, T1000850</td>
<td></td>
</tr>
<tr>
<td>ibm-851</td>
<td>IBM851, cp851, 851, csPC851, T1000851</td>
<td></td>
</tr>
<tr>
<td>ibm-852</td>
<td>IBM852, cp852, 852, csPCp852, windows-852, T1000852</td>
<td></td>
</tr>
<tr>
<td>ibm-855</td>
<td>IBM855, cp855, 855, csIBM855, csPCp855, windows-855, T1000855</td>
<td></td>
</tr>
<tr>
<td>ibm-856</td>
<td>cp856, 856, T1000856</td>
<td></td>
</tr>
<tr>
<td>ibm-857</td>
<td>IBM857, cp857, 857, csIBM857, windows-857, T1000857</td>
<td></td>
</tr>
<tr>
<td>ibm-858</td>
<td>IBM858, IBM00858, CCSID00858, CP00858, PC-Multilingual-850+euro, cp858, windows-858, T1000858</td>
<td></td>
</tr>
<tr>
<td>ibm-860</td>
<td>IBM860, cp860, 860, Portugal - Personal Computer, ASCII</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>367</td>
<td>ASCII</td>
</tr>
<tr>
<td>420</td>
<td>Arabic Bilingual, EBCDIC</td>
</tr>
<tr>
<td>424</td>
<td>Israel (Hebrew), EBCDIC</td>
</tr>
<tr>
<td>437</td>
<td>US Personal Computer, ASCII</td>
</tr>
<tr>
<td>500</td>
<td>International #5, EBCDIC</td>
</tr>
<tr>
<td>720</td>
<td>PC Arabic, ASCII</td>
</tr>
<tr>
<td>737</td>
<td>PC Greek, ASCII</td>
</tr>
<tr>
<td>775</td>
<td>PC Baltic, ASCII</td>
</tr>
<tr>
<td>803</td>
<td>Hebrew Character Set A, EBCDIC</td>
</tr>
<tr>
<td>808</td>
<td>Cyrillic, Russian with euro, ASCII</td>
</tr>
<tr>
<td>813</td>
<td>Greece - ISO/ASCII 8-Bit, ASCII</td>
</tr>
<tr>
<td>819</td>
<td>Latin-1, West European, ASCII</td>
</tr>
<tr>
<td>838</td>
<td>Thai EBCDIC</td>
</tr>
<tr>
<td>848</td>
<td>Cyrillic, Ukraine with Euro, ASCII</td>
</tr>
<tr>
<td>849</td>
<td>Cyrillic, Belarus Russian Euro, ASCII</td>
</tr>
<tr>
<td>850</td>
<td>PC Multilingual, ASCII</td>
</tr>
<tr>
<td>851</td>
<td>Greek - Personal Computer, ASCII</td>
</tr>
<tr>
<td>852</td>
<td>Latin2 Multilingual PC, ASCII</td>
</tr>
<tr>
<td>855</td>
<td>Cyrillic - Personal Computer, ASCII</td>
</tr>
<tr>
<td>856</td>
<td>Hebrew (old) - Personal Computer, ASCII</td>
</tr>
<tr>
<td>857</td>
<td>Latin5 PC, ASCII</td>
</tr>
<tr>
<td>858</td>
<td>PC - Multilingual with euro, ASCII</td>
</tr>
<tr>
<td>860</td>
<td>Portugal - Personal Computer, ASCII</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>csIBM860, T1000860</td>
<td>Iceland - Personal Computer, ASCII</td>
</tr>
<tr>
<td>ibm-861, IBM861, cp861, 861, cp-is, csIBM861, windows-861, T1000861</td>
<td>Hebrew - Personal Computer, ASCII</td>
</tr>
<tr>
<td>ibm-862, IBM862, cp862, 862, DOS-862, csPC862LatinHebrew, windows-862, T1000862</td>
<td>Canadian French - PC, ASCII</td>
</tr>
<tr>
<td>ibm-864, IBM864, cp864, csIBM864, T1000864</td>
<td>Arabic - Personal Computer, ASCII</td>
</tr>
<tr>
<td>ibm-865, IBM865, cp865, 865, csIBM865, T1000865</td>
<td>Nordic - Personal Computer, ASCII</td>
</tr>
<tr>
<td>ibm-866, IBM866, cp866, 866, csIBM866, windows-866, T1000866</td>
<td>Cyrillic #2 - Personal Computer, ASCII</td>
</tr>
<tr>
<td>ibm-867, cp867, T1000867</td>
<td>Israel - Personal Computer, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-868, IBM868, CP868, 868, csIBM868, cp-ar, T1000868</td>
<td>Urdu, ASCII</td>
</tr>
<tr>
<td>ibm-869, IBM869, cp869, 869, cp-gr, csIBM869, windows-869, T1000869</td>
<td>Greece - Personal Computer, ASCII</td>
</tr>
<tr>
<td>ibm-870, IBM870, CP870, ebcdic-cp-roece, ebcdic-cp-yu, csIBM870, T1000870</td>
<td>Latin2 Multilingual, EBCDIC</td>
</tr>
<tr>
<td>ibm-871, IBM871, ebcdic-cp-is, csIBM871, CP871, ebcdic-is, cpiibm871, 871, T1V10871</td>
<td>Iceland – CECP, EBCDIC</td>
</tr>
<tr>
<td>ibm-872, T1000872</td>
<td>Cyrillic PC with Euro, ASCII</td>
</tr>
<tr>
<td>ibm-874, ibm-9066, cp874, TIS-620, tis620.2533, eucTH, cp9066, ibm874, MS874, windows-874, T1000874</td>
<td>Thai - Personal Computer, ASCII</td>
</tr>
<tr>
<td>ibm-875, IBM875, cp875, 875, T1000875</td>
<td>Greece, EBCDIC</td>
</tr>
<tr>
<td>ibm-878, KOI8-R, ko8, csKOI8R, cp878, windows-20866, T1000878</td>
<td>Russian internet, ASCII</td>
</tr>
<tr>
<td>ibm-880, IBM880, cp880,EBCDIC-Cyrillic,csIBM880, windows-20880, T1000880</td>
<td>Cyrillic Multilingual, ASCII</td>
</tr>
<tr>
<td>ibm-896, T1000896</td>
<td>Japanese Katakana, ASCII</td>
</tr>
<tr>
<td>ibm-897, JIS_X0201, X0201, csHalfWidthKatakana, T1000897</td>
<td>Japanese, Half Width Katakana, ASCII</td>
</tr>
<tr>
<td>ibm-901, T1000901</td>
<td>Baltic – PC Multilingual, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-902, T1000902</td>
<td>Estonia, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-905, IBM905, CP905, ebcdic-cp-tr, csIBM905, windows-20905, T1000905</td>
<td>Latin3 Multilingual, EBCDIC</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>IBM-916, CP916, 916, T1000916</td>
<td>Hebrew ISO/ASCII 8-Bit, ASCII</td>
</tr>
<tr>
<td>IBM-918, IBM918, CP918, EBCDIC-CP-AR2, IBM918, EBCDIC-CP-AR2, T1000918</td>
<td>Urdu, EBCDIC</td>
</tr>
<tr>
<td>IBM-922, IBM922, CP922, 922, T1000922</td>
<td>Estonian, ASCII</td>
</tr>
<tr>
<td>IBM-924, IBM00924, CCSID00924, CP00924, EBCDIC-Latin9—Euro, T1000924</td>
<td>Latin 9 EBCDIC</td>
</tr>
<tr>
<td>IBM-930, IBM930, IBM-5026, CP930, CPIBM930, 930</td>
<td>Japanese, mixed EBCDIC/DBCS-HOST</td>
</tr>
<tr>
<td>IBM-931, IBM-5035, CP939, 939</td>
<td>Japanese, EBCDIC</td>
</tr>
<tr>
<td>IBM-932, IBM-942, CP932, SHIFT_JIS78, SJIS78, IBM-942_VSUB_VPUA, IBM-932_VSUB_VPUA</td>
<td>Japanese, SJIS without MS/IBM extensions</td>
</tr>
<tr>
<td>IBM-933, IBM933, CP933, CPIBM933, 933</td>
<td>Korean, mixed EBCDIC/DBCS-HOST</td>
</tr>
<tr>
<td>IBM-935, IBM935, CP935, CPIBM935, 935</td>
<td>Chinese (simplified), mixed EBCDIC/DBCS-HOST</td>
</tr>
<tr>
<td>IBM-936, Windows-936-2000, GBK, CP936, MS936, Windows-936</td>
<td>Windows Chinese(simplified), GBK</td>
</tr>
<tr>
<td>IBM-937, CP937, CPIBM937, 937</td>
<td>Chinese (traditional), mixed EBCDIC/DBCS-HOST</td>
</tr>
<tr>
<td>IBM-943, Shift_JIS, MS_Kanji, CSShiftJIS, Windows-31J, CSWindows31J, X-SJIS, X-MS-CP932, CP932, Windows-932, CP943C, IBM-943C, MS932, PKC, SJIS, IBM-943_VSUB_VPUA</td>
<td>Japanese, Shift-JIS standard with extension</td>
</tr>
<tr>
<td>IBM-949, CP949, 949, IBM-949_VASCII_VSUB_VPUA, CP949C, IBM-949_VSUB_VPUA</td>
<td>Windows Korean, (old) encoding</td>
</tr>
<tr>
<td>IBM-950, Windows-950-2000, Big5, CSBig5, Windows-950, X-Big5, CP950, 950</td>
<td>Windows Chinese(Traditional), ASCII</td>
</tr>
<tr>
<td>IBM-954, EUC-JP</td>
<td>Extended Unix Code Packed Format for</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ibm-964, EUC-TW, ibm-eucTW, cns11643, cp964, 964, ibm-964_VPUA</td>
<td>Extended Unix Code Packed Format for Chinese (Traditional)</td>
</tr>
<tr>
<td>ibm-971, ibm-971_VPUA</td>
<td>Extended Unix Code Packed Format for Korean</td>
</tr>
<tr>
<td>ibm-1004, T1001004, IBM PC Desktop Publishing, ASCII</td>
<td>1004</td>
</tr>
<tr>
<td>ibm-1006, cp1006, 1006</td>
<td>Urdu, 8-bit EBCDIC</td>
</tr>
<tr>
<td>ibm-1008, cp1008, T1001008</td>
<td>Arabic ISO/ASCII 8-Bit, ASCII</td>
</tr>
<tr>
<td>ibm-1025, cp1025, 1025, T1001025</td>
<td>Cyrillic Multilingual, EBCDIC</td>
</tr>
<tr>
<td>ibm-1026, IBM1026, CP1026, cslBM1026, 1026, T1001026</td>
<td>Latin5, Turkey, EBCDIC</td>
</tr>
<tr>
<td>ibm-1046, T1001046, Arabic Extended ISO/ASCII 8-Bit, ASCII</td>
<td>1046</td>
</tr>
<tr>
<td>ibm-1047, IBM1047, cpibm1047</td>
<td>Open systems Latin1, EBCDIC</td>
</tr>
<tr>
<td>ibm-1097, cp1097, 1097</td>
<td>Farsi, EBCDIC</td>
</tr>
<tr>
<td>ibm-1098, cp1098, 1098</td>
<td>Farsi, ASCII</td>
</tr>
<tr>
<td>ibm-1112, cp1112, 1112, T1001112</td>
<td>Baltic – Multilingual, EBCDIC</td>
</tr>
<tr>
<td>ibm-1122, cp1122, 1122, T1001122</td>
<td>Estonia, EBCDIC</td>
</tr>
<tr>
<td>ibm-1123, cp1123, 1123, cpibm1123, T1001123</td>
<td>Cyrillic, Ukraine EBCDIC</td>
</tr>
<tr>
<td>ibm-1124, cp1124, 1124, T1001124</td>
<td>ISO Cyrillic Ukraine, ASCII</td>
</tr>
<tr>
<td>ibm-1125, cp1125, T1001125</td>
<td>Cyrillic Ukraine ASCII</td>
</tr>
<tr>
<td>ibm-1129, T1001129</td>
<td>ISO Vietnamese, ASCII</td>
</tr>
<tr>
<td>ibm-1130, T1001130</td>
<td>Vietnamese EBCDIC</td>
</tr>
<tr>
<td>ibm-1131, cp1131, T1001131</td>
<td>Cyrillic Belarus EBCDIC</td>
</tr>
<tr>
<td>ibm-1132, T1001132</td>
<td>Lao EBCDIC</td>
</tr>
<tr>
<td>ibm-1133, T1001133</td>
<td>ISO Lao, ASCII</td>
</tr>
<tr>
<td>ibm-1137</td>
<td>Devanagari EBCDIC</td>
</tr>
<tr>
<td>ibm-1140, IBM01140, CCSID01140, CP01140, cp1140, cpibm1140, ebcdic-us-37+euro, T1001140</td>
<td>USA, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1141, IBM01141, CCSID01141, CP01141, cp1141, cpibm1141, ebcdic-de-273+euro, T1001141</td>
<td>Austria, Germany ECECP, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1142, IBM01142, CCSID01142, CP01142, cp1142, cpibm1142, ebcdic-dk-277+euro, ebcdic-no-277+euro, T1001142</td>
<td>Denmark, Norway ECECP, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1143, IBM01143, CCSID01143 CP01143, cp1143, cpibm1143, ebcdic-fi-278+euro, ebcdic-se-278+euro, T1001143</td>
<td>Finland, Sweden ECECP, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1144, IBM01144, CCSID01144, CP01144, cp1144, cpibm1144, ebcdic-it-280+euro, T1001144</td>
<td>Italy ECECP, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ibm-1145</td>
<td>Spain, Latin America ECECP, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1146</td>
<td>UK ECECP, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1147</td>
<td>France ECECP, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1148</td>
<td>International Latin 1, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1149</td>
<td>Iceland, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1153</td>
<td>Latin2 Multilingual with Euro, EBCDIC</td>
</tr>
<tr>
<td>ibm-1154</td>
<td>Cyrillic Multilingual with Euro, EBCDIC</td>
</tr>
<tr>
<td>ibm-1155</td>
<td>EBCDIC Turkey with Euro, EBCDIC</td>
</tr>
<tr>
<td>ibm-1156</td>
<td>EBCDIC Baltic - Multi with Euro, EBCDIC</td>
</tr>
<tr>
<td>ibm-1157</td>
<td>EBCDIC Estonia with Euro, EBCDIC</td>
</tr>
<tr>
<td>ibm-1158</td>
<td>EBCDIC Cyrillic Ukraine with Euro, EBCDIC</td>
</tr>
<tr>
<td>ibm-1160</td>
<td>Thailand EBCDIC with Euro, EBCDIC</td>
</tr>
<tr>
<td>ibm-1161</td>
<td>Thai, with the Euro sign</td>
</tr>
<tr>
<td>ibm-1162</td>
<td>Windows Thai, ASCII with the Euro sign</td>
</tr>
<tr>
<td>ibm-1164</td>
<td>Vietnamese, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ibm-1200</td>
<td>Unicode wild-char Big Endian, a Unicode character set with two or 4 bytes encoding units</td>
</tr>
<tr>
<td>ibm-1202</td>
<td>Unicode wild-char Little Endian, a Unicode character set with two or 4 bytes encoding units</td>
</tr>
<tr>
<td>ibm-1204</td>
<td>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.</td>
</tr>
<tr>
<td>ibm-1208</td>
<td>Unicode UTF-8, a Unicode character set with multibyte characters</td>
</tr>
<tr>
<td>ibm-1232</td>
<td>Unicode UTF-32 Big Endian, a Unicode character set with four-byte encoding units</td>
</tr>
<tr>
<td>ibm-1234</td>
<td>Unicode UTF-32 Little Endian, a Unicode</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UTF32_LittleEndian, ibm-1235</td>
<td>character set with four-byte encoding units</td>
</tr>
<tr>
<td>ibm-1236, UTF-32, ISO-10646-UCS-4, ibm-1237, csUCS4, ucs-4</td>
<td>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.</td>
</tr>
<tr>
<td>ibm-1250, ibm-5346, windows-1250, cp1250, T1001250</td>
<td>Windows, Latin 2, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-1251, ibm-5347, windows-1251, cp1251, T1001251</td>
<td>Windows Cyrillic, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm1252, ibm-5348, windows-1252, cp1252, T1001252</td>
<td>Windows, Latin 1, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-1253, ibm-5349, windows-1253, cp1253, T1001253</td>
<td>Windows Greek, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-1254, ibm-5350, windows-1254, cp1254, T1001254</td>
<td>Windows Turkish, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-1255, ibm-9447, windows-1255, cp1255</td>
<td>Windows Hebrew, with the Euro sign</td>
</tr>
<tr>
<td>ibm-1256, windows-1256-2000, windows-1256, cp1256</td>
<td>Windows Arabic, with the Euro sign</td>
</tr>
<tr>
<td>ibm-1257, windows-1257, cp1257, ibm-9449, ibm-9449, ibm-5353, T1001257</td>
<td>Windows Baltic Rim, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-1258, ibm-5354, windows-1258, cp1258, T1001258</td>
<td>Windows Vietnamese, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-1276, Adobe-Standard-Encoding, csAdobeStandardEncoding, T1001276</td>
<td>Adobe Standard Encoding, ASCII</td>
</tr>
<tr>
<td>ibm-1277, Adobe-Latin1-Encoding, T1001277</td>
<td>Adobe Latin1 Encoding, ASCII</td>
</tr>
<tr>
<td>ibm-1364, cp1364</td>
<td>Korean, mixed EBCDIC/DBCS-HOST with Euro sign</td>
</tr>
<tr>
<td>ibm-1371, cpIBM1371</td>
<td>Chinese (Traditional), EBCDIC/DBCS-HOST with the Euro sign</td>
</tr>
<tr>
<td>ibm-1373, windows-950</td>
<td>Chinese (traditional), Windows BIG5</td>
</tr>
<tr>
<td>ibm-1375, Big5-HKSCS, Big5hk, HKSCS-BIG5, Big5HKSCS</td>
<td>Chinese (traditional) BIG5 with Hong Kong Supplementary Character Set (HKSCS)</td>
</tr>
<tr>
<td>ibm-1377, T11376, HKSCS-HOST</td>
<td>Chinese (traditional) BIG5 with Hong Kong Supplementary Character Set (HKSCS) for IBM hots</td>
</tr>
<tr>
<td>ibm-1381, cp1381, 1381</td>
<td>Chinese (Simplified) GB standard</td>
</tr>
<tr>
<td>ibm-1383, GB2312, csGB2312, EUC-CN, ibm-eucCN, hp15CN, cp1383, 1383, ibm-1383_VPUA</td>
<td>Extended Unix Code Packed Format for Chinese (Simplified)</td>
</tr>
<tr>
<td>ibm-1386, cp1386, windows-936, ibm-1386_VSUB_VPUA</td>
<td>Chinese (Simplified) GBK PC standard</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ibm-1388, ibm-9580</td>
<td>Chinese EBCDIC/DBCS-HOST for GBK with the Euro sign</td>
</tr>
<tr>
<td>ibm-1390, cpibm1390</td>
<td>Japanese, mixed EBCDIC/DBCS-HOST with the Euro sign</td>
</tr>
<tr>
<td>ibm-1392, windows-54936, gb18030, GB18030</td>
<td>Chinese GB18030 PC standard</td>
</tr>
<tr>
<td>ibm-1399</td>
<td>Japan, EBCDIC, host MBCS (Latin-Kanji), with the Euro sign</td>
</tr>
<tr>
<td>ibm-4899, cpibm4899</td>
<td>Old EBCDIC Hebrew, with the Euro sign</td>
</tr>
<tr>
<td>ibm-4909</td>
<td>Greek ISO, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-4971, cpibm4971</td>
<td>EBCDIC Greek, with the Euro sign</td>
</tr>
<tr>
<td>ibm-5471, MS950_HKSCS, hkbig5, big5-hkscsunicode3.0</td>
<td>Chinese (traditional) Big5-HKSCS-2001 Hong Kong with Unicode 3.0 mappings</td>
</tr>
<tr>
<td>ibm-5478, GB_2312-80, chinese, iso-ir-58, csISO8GB231280, gb2312-1980, GB2312.1980-0</td>
<td>Extended Unix Code Packed Format for Chinese (Simplified)</td>
</tr>
<tr>
<td>ibm-9005, ISO-8859-7, greek, greek8, ELOT_928, ECMA-118, csISOLatinGreek, iso-ir-126, ISO_8859-7:1987, windows-28597, sun_eu_greek</td>
<td>ISO Greek (with euro update), ASCII</td>
</tr>
<tr>
<td>ibm-12712, cpibm12712, ebcdic-he</td>
<td>EBCDIC Hebrew (new sheqel, control characters update), with the Euro sign</td>
</tr>
<tr>
<td>ibm-16684</td>
<td>Japanese, Jis + Roman Jis Host</td>
</tr>
<tr>
<td>ibm-16804, cpibm16804, ebcdic-ar</td>
<td>Arabic, EBCDIC with the Euro sign</td>
</tr>
<tr>
<td>ISO-8859-10, iso-ir-157, l6, ISO_8859-10:1992, csISOLatin6, latin6</td>
<td>Nordic</td>
</tr>
<tr>
<td>x-iscii-de, windows-57002, iscii-dev</td>
<td>Windows Devanagari</td>
</tr>
<tr>
<td>x-iscii-be, windows-57003, iscii-bng, windows-57006, x-iscii-as</td>
<td>Windows Bengali</td>
</tr>
<tr>
<td>x-iscii-pa, windows-57011, iscii-gur</td>
<td>Windows Punjabi</td>
</tr>
<tr>
<td>x-iscii-gu, windows-57010, iscii-guj</td>
<td>Windows Gujarati</td>
</tr>
<tr>
<td>x-iscii-or, windows-57007, iscii-ori</td>
<td>Windows Oriya</td>
</tr>
<tr>
<td>x-iscii-ta, windows-57004, iscii-tml</td>
<td>Windows Tamil</td>
</tr>
<tr>
<td>x-iscii-te, windows-57005, iscii-tlg</td>
<td>Windows Telugu</td>
</tr>
<tr>
<td>x-iscii-ka, windows-57008, iscii-kr</td>
<td>Windows Kannada</td>
</tr>
<tr>
<td>x-iscii-ma, windows-57009, iscii-mlm</td>
<td>Windows Malayalam</td>
</tr>
<tr>
<td>macOS Version</td>
<td>Character Set Name</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>0.2-10.2</td>
<td>Macintosh, ASCII</td>
</tr>
<tr>
<td>2566-10.2</td>
<td>Chinese (traditional) BIG5 with Hong Kong Supplementary Character Set (HKSCS)</td>
</tr>
<tr>
<td>29-10.2</td>
<td>Macintosh, Central Euro, ASCII</td>
</tr>
<tr>
<td>35-10.2</td>
<td>Macintosh, Turkish, ASCII</td>
</tr>
<tr>
<td>6-10.2</td>
<td>Macintosh, Greek, ASCII</td>
</tr>
<tr>
<td>7_3-10.2</td>
<td>Macintosh, Cyrillic, ASCII</td>
</tr>
<tr>
<td>UTF-7, windows-65000</td>
<td>Unicode UTF-7, a Unicode character set with 7-bit characters; used primarily for email headers</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Code</th>
<th>Language</th>
<th>Code</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>Afar</td>
<td>cy</td>
<td>Welsh</td>
</tr>
<tr>
<td>ab</td>
<td>Abkhazian</td>
<td>da</td>
<td>Danish</td>
</tr>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td>de</td>
<td>German</td>
</tr>
<tr>
<td>am</td>
<td>Amharic</td>
<td>dz</td>
<td>Bhutani</td>
</tr>
<tr>
<td>ar</td>
<td>Arabic</td>
<td>el</td>
<td>Greek</td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
<td>en</td>
<td>English</td>
</tr>
<tr>
<td>ay</td>
<td>Aymara</td>
<td>eo</td>
<td>Esperanto</td>
</tr>
<tr>
<td>az</td>
<td>Azerbaijani</td>
<td>es</td>
<td>Spanish</td>
</tr>
<tr>
<td>ba</td>
<td>Bashkir</td>
<td>et</td>
<td>Estonian</td>
</tr>
<tr>
<td>be</td>
<td>Byelorussian</td>
<td>eu</td>
<td>Basque</td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarian</td>
<td>fa</td>
<td>Persian</td>
</tr>
<tr>
<td>bh</td>
<td>Bihari</td>
<td>fo</td>
<td>Faroese</td>
</tr>
<tr>
<td>bi</td>
<td>Bislama</td>
<td>fr</td>
<td>French</td>
</tr>
<tr>
<td>bn</td>
<td>Bengali; Bangla</td>
<td>fy</td>
<td>Frisian</td>
</tr>
<tr>
<td>bo</td>
<td>Tibetan</td>
<td>ga</td>
<td>Irish</td>
</tr>
<tr>
<td>br</td>
<td>Breton</td>
<td>gd</td>
<td>Scots Gaelic</td>
</tr>
<tr>
<td>ca</td>
<td>Catalan</td>
<td>gl</td>
<td>Galician</td>
</tr>
<tr>
<td>co</td>
<td>Corsican</td>
<td>gn</td>
<td>Guarani</td>
</tr>
<tr>
<td>cs</td>
<td>Czech</td>
<td>gu</td>
<td>Gujarati</td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
<td>Code</td>
<td>Language</td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
<td>------</td>
<td>-------------------</td>
</tr>
<tr>
<td>ha</td>
<td>Hausa</td>
<td>rn</td>
<td>Kirundi</td>
</tr>
<tr>
<td>he</td>
<td>Hebrew</td>
<td>ro</td>
<td>Romanian</td>
</tr>
<tr>
<td>hi</td>
<td>Hindi</td>
<td>ru</td>
<td>Russian</td>
</tr>
<tr>
<td>hr</td>
<td>Croatian</td>
<td>rw</td>
<td>Kinyarwanda</td>
</tr>
<tr>
<td>hu</td>
<td>Hungarian</td>
<td>sa</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>hy</td>
<td>Armenian</td>
<td>sd</td>
<td>Sindhi</td>
</tr>
<tr>
<td>ia</td>
<td>Interlingua</td>
<td>sg</td>
<td>Sangho</td>
</tr>
<tr>
<td>id</td>
<td>Indonesian</td>
<td>sh</td>
<td>Serbo-Croatian</td>
</tr>
<tr>
<td>ie</td>
<td>Interlingue</td>
<td>si</td>
<td>Sinhalese</td>
</tr>
<tr>
<td>ik</td>
<td>Inupiak</td>
<td>sk</td>
<td>Slovak</td>
</tr>
<tr>
<td>is</td>
<td>Icelandic</td>
<td>sl</td>
<td>Slovenian</td>
</tr>
<tr>
<td>it</td>
<td>Italian</td>
<td>sm</td>
<td>Samoan</td>
</tr>
<tr>
<td>iu</td>
<td>Inuktitut</td>
<td>sn</td>
<td>Shona</td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
<td>so</td>
<td>Somali</td>
</tr>
<tr>
<td>jw</td>
<td>Javanese</td>
<td>sq</td>
<td>Albanian</td>
</tr>
<tr>
<td>ka</td>
<td>Georgian</td>
<td>sr</td>
<td>Serbian</td>
</tr>
<tr>
<td>kk</td>
<td>Kazakh</td>
<td>ss</td>
<td>Siswati</td>
</tr>
<tr>
<td>kl</td>
<td>Greenlandic</td>
<td>st</td>
<td>Sesotho</td>
</tr>
<tr>
<td>km</td>
<td>Cambodian</td>
<td>su</td>
<td>Sundanese</td>
</tr>
<tr>
<td>kn</td>
<td>Kannada</td>
<td>sv</td>
<td>Swedish</td>
</tr>
<tr>
<td>ko</td>
<td>Korean</td>
<td>sw</td>
<td>Swahili</td>
</tr>
<tr>
<td>ks</td>
<td>Kashmiri</td>
<td>ta</td>
<td>Tamil</td>
</tr>
<tr>
<td>ku</td>
<td>Kurdish</td>
<td>te</td>
<td>Telugu</td>
</tr>
<tr>
<td>ky</td>
<td>Kirghiz</td>
<td>tg</td>
<td>Tajik</td>
</tr>
<tr>
<td>la</td>
<td>Latin</td>
<td>th</td>
<td>Thai</td>
</tr>
<tr>
<td>ln</td>
<td>Lingala</td>
<td>ti</td>
<td>Tigrinya</td>
</tr>
<tr>
<td>lo</td>
<td>Laotian</td>
<td>tk</td>
<td>Turkmen</td>
</tr>
<tr>
<td>lt</td>
<td>Lithuanian</td>
<td>tl</td>
<td>Tagalog</td>
</tr>
<tr>
<td>lv</td>
<td>Latvian, Lettish</td>
<td>tn</td>
<td>Setswana</td>
</tr>
<tr>
<td>mg</td>
<td>Malagasy</td>
<td>to</td>
<td>Tonga</td>
</tr>
<tr>
<td>mi</td>
<td>Maori</td>
<td>tr</td>
<td>Turkish</td>
</tr>
<tr>
<td>mk</td>
<td>Macedonian</td>
<td>ts</td>
<td>Tsonga</td>
</tr>
<tr>
<td>ml</td>
<td>Malayalam</td>
<td>tt</td>
<td>Tatar</td>
</tr>
<tr>
<td>mn</td>
<td>Mongolian</td>
<td>tw</td>
<td>Taiwan</td>
</tr>
<tr>
<td>mo</td>
<td>Moldavian</td>
<td>ug</td>
<td>Uighur</td>
</tr>
<tr>
<td>mr</td>
<td>Marathi</td>
<td>uk</td>
<td>Ukrainian</td>
</tr>
<tr>
<td>ms</td>
<td>Malay</td>
<td>ur</td>
<td>Urdu</td>
</tr>
<tr>
<td>mt</td>
<td>Maltese</td>
<td>uz</td>
<td>Uzbek</td>
</tr>
<tr>
<td>my</td>
<td>Burmese</td>
<td>vi</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>na</td>
<td>Nauru</td>
<td>vo</td>
<td>Volapuk</td>
</tr>
<tr>
<td>ne</td>
<td>Nepali</td>
<td>wo</td>
<td>Wolof</td>
</tr>
<tr>
<td>nl</td>
<td>Dutch</td>
<td>xh</td>
<td>Xhosa</td>
</tr>
<tr>
<td>no</td>
<td>Norwegian</td>
<td>yi</td>
<td>Yiddish</td>
</tr>
<tr>
<td>oc</td>
<td>Occitan</td>
<td>yo</td>
<td>Yoruba</td>
</tr>
<tr>
<td>om</td>
<td>(Afan) Oromo</td>
<td>za</td>
<td>Zhuang</td>
</tr>
<tr>
<td>or</td>
<td>Oriya</td>
<td>zh</td>
<td>Chinese</td>
</tr>
<tr>
<td>pa</td>
<td>Punjabi</td>
<td>zu</td>
<td>Zulu</td>
</tr>
<tr>
<td>pl</td>
<td>Polish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ps</td>
<td>Pashto, Pushto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qu</td>
<td>Quechua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rm</td>
<td>Rhaeto-Romance</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ISO 3166 – Country/Area Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Country/Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF</td>
<td>AFGHANISTAN</td>
</tr>
<tr>
<td>AX</td>
<td>ALAND ISLANDS</td>
</tr>
<tr>
<td>AL</td>
<td>ALBANIA</td>
</tr>
<tr>
<td>DZ</td>
<td>ALGERIA</td>
</tr>
<tr>
<td>AS</td>
<td>AMERICAN SAMOA</td>
</tr>
<tr>
<td>AD</td>
<td>ANDORRA</td>
</tr>
<tr>
<td>AO</td>
<td>ANGOLA</td>
</tr>
<tr>
<td>AI</td>
<td>ANGUILLA</td>
</tr>
<tr>
<td>AQ</td>
<td>ANTARCTICA</td>
</tr>
<tr>
<td>AG</td>
<td>ANTIGUA AND BARBUDA</td>
</tr>
<tr>
<td>AR</td>
<td>ARGENTINA</td>
</tr>
<tr>
<td>AM</td>
<td>ARMENIA</td>
</tr>
<tr>
<td>AW</td>
<td>ARUBA</td>
</tr>
<tr>
<td>AU</td>
<td>AUSTRALIA</td>
</tr>
<tr>
<td>AT</td>
<td>AUSTRIA</td>
</tr>
<tr>
<td>AZ</td>
<td>AZERBAIJAN</td>
</tr>
<tr>
<td>BS</td>
<td>BAHAMAS</td>
</tr>
<tr>
<td>BH</td>
<td>BAHRAIN</td>
</tr>
<tr>
<td>BD</td>
<td>BANGLADESH</td>
</tr>
<tr>
<td>BB</td>
<td>BARBADOS</td>
</tr>
<tr>
<td>BY</td>
<td>BELARUS</td>
</tr>
<tr>
<td>BE</td>
<td>BELGIUM</td>
</tr>
<tr>
<td>BZ</td>
<td>BELIZE</td>
</tr>
<tr>
<td>BJ</td>
<td>BENIN</td>
</tr>
<tr>
<td>BM</td>
<td>BERMUDA</td>
</tr>
<tr>
<td>BT</td>
<td>BHUTAN</td>
</tr>
<tr>
<td>BO</td>
<td>BOLIVIA</td>
</tr>
<tr>
<td>BA</td>
<td>BOSNIA AND HERZEGOVINA</td>
</tr>
<tr>
<td>BW</td>
<td>BOTSWANA</td>
</tr>
<tr>
<td>BV</td>
<td>BOUVET ISLAND</td>
</tr>
<tr>
<td>BR</td>
<td>BRAZIL</td>
</tr>
<tr>
<td>IO</td>
<td>BRITISH INDIAN OCEAN TERRITORY</td>
</tr>
<tr>
<td>BN</td>
<td>BRUNEI DARUSSALAM</td>
</tr>
<tr>
<td>BG</td>
<td>BULGARIA</td>
</tr>
<tr>
<td>BF</td>
<td>BURKINA FASO</td>
</tr>
<tr>
<td>BI</td>
<td>BURUNDI</td>
</tr>
<tr>
<td>KH</td>
<td>CAMBODIA</td>
</tr>
<tr>
<td>CM</td>
<td>CAMEROON</td>
</tr>
<tr>
<td>CA</td>
<td>CANADA</td>
</tr>
<tr>
<td>CV</td>
<td>CAPE VERDE</td>
</tr>
<tr>
<td>KY</td>
<td>CAYMAN ISLANDS</td>
</tr>
<tr>
<td>CF</td>
<td>CENTRAL AFRICAN REPUBLIC</td>
</tr>
<tr>
<td>TD</td>
<td>CHAD</td>
</tr>
<tr>
<td>CL</td>
<td>CHILE</td>
</tr>
<tr>
<td>CN</td>
<td>CHINA</td>
</tr>
<tr>
<td>CX</td>
<td>CHRISTMAS ISLAND</td>
</tr>
<tr>
<td>CC</td>
<td>COCOS (KEELING) ISLANDS</td>
</tr>
<tr>
<td>CO</td>
<td>COLOMBIA</td>
</tr>
<tr>
<td>KM</td>
<td>COMOROS</td>
</tr>
<tr>
<td>Code</td>
<td>Country</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>CG</td>
<td>CONGO</td>
</tr>
<tr>
<td>CD</td>
<td>CONGO, THE DEMOCRATIC REPUBLIC OF THE</td>
</tr>
<tr>
<td>CK</td>
<td>COOK ISLANDS</td>
</tr>
<tr>
<td>CR</td>
<td>COSTA RICA</td>
</tr>
<tr>
<td>CI</td>
<td>COTE D’IVOIRE</td>
</tr>
<tr>
<td>HR</td>
<td>CROATIA</td>
</tr>
<tr>
<td>CU</td>
<td>CUBA</td>
</tr>
<tr>
<td>CY</td>
<td>CYPRUS</td>
</tr>
<tr>
<td>CZ</td>
<td>CZECH REPUBLIC</td>
</tr>
<tr>
<td>DK</td>
<td>DENMARK</td>
</tr>
<tr>
<td>DJ</td>
<td>DJIBOUTI</td>
</tr>
<tr>
<td>DM</td>
<td>DOMINICA</td>
</tr>
<tr>
<td>DO</td>
<td>DOMINICAN REPUBLIC</td>
</tr>
<tr>
<td>EC</td>
<td>ECUADOR</td>
</tr>
<tr>
<td>EG</td>
<td>EGYPT</td>
</tr>
<tr>
<td>SV</td>
<td>EL SALVADOR</td>
</tr>
<tr>
<td>GQ</td>
<td>EQUATORIAL GUINEA</td>
</tr>
<tr>
<td>ER</td>
<td>ERITREA</td>
</tr>
<tr>
<td>EE</td>
<td>ESTONIA</td>
</tr>
<tr>
<td>ET</td>
<td>ETHIOPIA</td>
</tr>
<tr>
<td>FK</td>
<td>FALKLAND ISLANDS (MALVINAS)</td>
</tr>
<tr>
<td>FO</td>
<td>FAROE ISLANDS</td>
</tr>
<tr>
<td>FJ</td>
<td>FIJI</td>
</tr>
<tr>
<td>FI</td>
<td>FINLAND</td>
</tr>
<tr>
<td>FR</td>
<td>FRANCE</td>
</tr>
<tr>
<td>GF</td>
<td>FRENCH GUIANA</td>
</tr>
<tr>
<td>PF</td>
<td>FRENCH POLYNESIA</td>
</tr>
<tr>
<td>TF</td>
<td>FRENCH SOUTHERN TERRITORIES</td>
</tr>
<tr>
<td>GA</td>
<td>GABON</td>
</tr>
<tr>
<td>GM</td>
<td>GAMBIA</td>
</tr>
<tr>
<td>GE</td>
<td>GEORGIA</td>
</tr>
<tr>
<td>DE</td>
<td>GERMANY</td>
</tr>
<tr>
<td>GH</td>
<td>GHANA</td>
</tr>
<tr>
<td>GI</td>
<td>GIBRALTAR</td>
</tr>
<tr>
<td>GR</td>
<td>GREECE</td>
</tr>
<tr>
<td>GL</td>
<td>GREENLAND</td>
</tr>
<tr>
<td>GD</td>
<td>GRENADE</td>
</tr>
<tr>
<td>GP</td>
<td>GUADELOUPE</td>
</tr>
<tr>
<td>GU</td>
<td>GUAM</td>
</tr>
<tr>
<td>GT</td>
<td>GUATEMALA</td>
</tr>
<tr>
<td>GG</td>
<td>GUERNSEY</td>
</tr>
<tr>
<td>GN</td>
<td>GUINEA</td>
</tr>
<tr>
<td>GW</td>
<td>GUINEA-BISSAU</td>
</tr>
<tr>
<td>GY</td>
<td>GUYANA</td>
</tr>
<tr>
<td>HT</td>
<td>HAITI</td>
</tr>
<tr>
<td>HM</td>
<td>HEARD ISLAND AND MCDONALD ISLANDS</td>
</tr>
<tr>
<td>VA</td>
<td>HOLY SEE (VATICAN CITY STATE)</td>
</tr>
<tr>
<td>HN</td>
<td>HONDURAS</td>
</tr>
<tr>
<td>HK</td>
<td>HONG KONG</td>
</tr>
<tr>
<td>HU</td>
<td>HUNGARY</td>
</tr>
<tr>
<td>IS</td>
<td>ICELAND</td>
</tr>
<tr>
<td>IN</td>
<td>INDIA</td>
</tr>
<tr>
<td>Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ID</td>
<td>INDONESIA</td>
</tr>
<tr>
<td>IR</td>
<td>IRAN (ISLAMIC REPUBLIC OF)</td>
</tr>
<tr>
<td>IQ</td>
<td>IRAQ</td>
</tr>
<tr>
<td>IE</td>
<td>IRELAND</td>
</tr>
<tr>
<td>IM</td>
<td>ISLE OF MAN</td>
</tr>
<tr>
<td>IL</td>
<td>ISRAEL</td>
</tr>
<tr>
<td>IT</td>
<td>ITALY</td>
</tr>
<tr>
<td>JM</td>
<td>JAMAICA</td>
</tr>
<tr>
<td>JP</td>
<td>JAPAN</td>
</tr>
<tr>
<td>JE</td>
<td>JERSEY</td>
</tr>
<tr>
<td>JO</td>
<td>JORDAN</td>
</tr>
<tr>
<td>KZ</td>
<td>KAZAKHSTAN</td>
</tr>
<tr>
<td>KE</td>
<td>KENYA</td>
</tr>
<tr>
<td>KI</td>
<td>KIRIBATI</td>
</tr>
<tr>
<td>KP</td>
<td>KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF</td>
</tr>
<tr>
<td>KR</td>
<td>KOREA, REPUBLIC OF</td>
</tr>
<tr>
<td>KW</td>
<td>KUWAIT</td>
</tr>
<tr>
<td>KG</td>
<td>KYRGYZSTAN</td>
</tr>
<tr>
<td>LA</td>
<td>LAO PEOPLE'S DEMOCRATIC REPUBLIC</td>
</tr>
<tr>
<td>LV</td>
<td>LATVIA</td>
</tr>
<tr>
<td>LB</td>
<td>LEBANON</td>
</tr>
<tr>
<td>LS</td>
<td>LESOTHO</td>
</tr>
<tr>
<td>LR</td>
<td>LIBERIA</td>
</tr>
<tr>
<td>LY</td>
<td>LIBYAN ARAB JAMAHIRIYA</td>
</tr>
<tr>
<td>LI</td>
<td>LIECHTENSTEIN</td>
</tr>
<tr>
<td>LT</td>
<td>LITHUANIA</td>
</tr>
<tr>
<td>LU</td>
<td>LUXEMBOURG</td>
</tr>
<tr>
<td>MO</td>
<td>MACAO</td>
</tr>
<tr>
<td>MK</td>
<td>MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF</td>
</tr>
<tr>
<td>MG</td>
<td>MADAGASCAR</td>
</tr>
<tr>
<td>MW</td>
<td>MALAWI</td>
</tr>
<tr>
<td>MY</td>
<td>MALAYSIA</td>
</tr>
<tr>
<td>MV</td>
<td>MALDIVES</td>
</tr>
<tr>
<td>ML</td>
<td>MALI</td>
</tr>
<tr>
<td>MT</td>
<td>MALTA</td>
</tr>
<tr>
<td>MH</td>
<td>MARSHALL ISLANDS</td>
</tr>
<tr>
<td>MQ</td>
<td>MARTINIQUE</td>
</tr>
<tr>
<td>MR</td>
<td>MAURITANIA</td>
</tr>
<tr>
<td>MU</td>
<td>MAURITIUS</td>
</tr>
<tr>
<td>YT</td>
<td>MAYOTTE</td>
</tr>
<tr>
<td>MX</td>
<td>MEXICO</td>
</tr>
<tr>
<td>FM</td>
<td>MICRONESIA, FEDERATED STATES OF</td>
</tr>
<tr>
<td>MD</td>
<td>MOLDOVA, REPUBLIC OF</td>
</tr>
<tr>
<td>MC</td>
<td>MONACO</td>
</tr>
<tr>
<td>MN</td>
<td>MONGOLIA</td>
</tr>
<tr>
<td>ME</td>
<td>MONTENEGRO</td>
</tr>
<tr>
<td>MS</td>
<td>MONTSERRAT</td>
</tr>
<tr>
<td>MA</td>
<td>MOROCCO</td>
</tr>
<tr>
<td>MZ</td>
<td>MOZAMBIQUE</td>
</tr>
<tr>
<td>MM</td>
<td>MYANMAR</td>
</tr>
<tr>
<td>NA</td>
<td>NAMIBIA</td>
</tr>
<tr>
<td>NR</td>
<td>NAURU</td>
</tr>
<tr>
<td>2-digit code</td>
<td>Country</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>NP</td>
<td>NEPAL</td>
</tr>
<tr>
<td>NL</td>
<td>NETHERLANDS</td>
</tr>
<tr>
<td>AN</td>
<td>NETHERLANDS ANTILLES</td>
</tr>
<tr>
<td>NC</td>
<td>NEW CALEDONIA</td>
</tr>
<tr>
<td>NZ</td>
<td>NEW ZEALAND</td>
</tr>
<tr>
<td>NI</td>
<td>NICARAGUA</td>
</tr>
<tr>
<td>NE</td>
<td>NIGER</td>
</tr>
<tr>
<td>NG</td>
<td>NIGERIA</td>
</tr>
<tr>
<td>NU</td>
<td>NIUE</td>
</tr>
<tr>
<td>NF</td>
<td>NORFOLK ISLAND</td>
</tr>
<tr>
<td>MP</td>
<td>NORTHERN MARIANA ISLANDS</td>
</tr>
<tr>
<td>NO</td>
<td>NORWAY</td>
</tr>
<tr>
<td>OM</td>
<td>OMAN</td>
</tr>
<tr>
<td>PK</td>
<td>PAKISTAN</td>
</tr>
<tr>
<td>PW</td>
<td>PALAU</td>
</tr>
<tr>
<td>PS</td>
<td>PALESTINIAN TERRITORY, OCCUPIED</td>
</tr>
<tr>
<td>PA</td>
<td>PANAMA</td>
</tr>
<tr>
<td>PG</td>
<td>PAPUA NEW GUINEA</td>
</tr>
<tr>
<td>PY</td>
<td>PARAGUAY</td>
</tr>
<tr>
<td>PE</td>
<td>PERU</td>
</tr>
<tr>
<td>PH</td>
<td>PHILIPPINES</td>
</tr>
<tr>
<td>PN</td>
<td>PITCAIRN</td>
</tr>
<tr>
<td>PL</td>
<td>POLAND</td>
</tr>
<tr>
<td>PT</td>
<td>PORTUGAL</td>
</tr>
<tr>
<td>PR</td>
<td>PUERTO RICO</td>
</tr>
<tr>
<td>QA</td>
<td>QATAR</td>
</tr>
<tr>
<td>RE</td>
<td>REUNION</td>
</tr>
<tr>
<td>RO</td>
<td>ROMANIA</td>
</tr>
<tr>
<td>RU</td>
<td>RUSSIAN FEDERATION</td>
</tr>
<tr>
<td>RW</td>
<td>RWANDA</td>
</tr>
<tr>
<td>SH</td>
<td>SAINT HELENA</td>
</tr>
<tr>
<td>KN</td>
<td>SAINT KITTS AND NEVIS</td>
</tr>
<tr>
<td>LC</td>
<td>SAINT LUCIA</td>
</tr>
<tr>
<td>PM</td>
<td>SAINT PIERRE AND MIQUELON</td>
</tr>
<tr>
<td>VC</td>
<td>SAINT VINCENT AND THE GRENADINES</td>
</tr>
<tr>
<td>WS</td>
<td>SAMOA</td>
</tr>
<tr>
<td>SM</td>
<td>SAN MARINO</td>
</tr>
<tr>
<td>ST</td>
<td>SAO TOME AND PRINCIPE</td>
</tr>
<tr>
<td>SA</td>
<td>SAUDI ARABIA</td>
</tr>
<tr>
<td>SN</td>
<td>SENEGAL</td>
</tr>
<tr>
<td>RS</td>
<td>SERBIA</td>
</tr>
<tr>
<td>SC</td>
<td>SEYCHELLES</td>
</tr>
<tr>
<td>SL</td>
<td>SIERRA LEONE</td>
</tr>
<tr>
<td>SG</td>
<td>SINGAPORE</td>
</tr>
<tr>
<td>SK</td>
<td>SLOVAKIA</td>
</tr>
<tr>
<td>SI</td>
<td>SLOVENIA</td>
</tr>
<tr>
<td>SB</td>
<td>SOLOMON ISLANDS</td>
</tr>
<tr>
<td>SO</td>
<td>SOMALIA</td>
</tr>
<tr>
<td>ZA</td>
<td>SOUTH AFRICA</td>
</tr>
<tr>
<td>GS</td>
<td>SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS</td>
</tr>
<tr>
<td>ES</td>
<td>SPAIN</td>
</tr>
<tr>
<td>LK</td>
<td>SRI LANKA</td>
</tr>
<tr>
<td>Code</td>
<td>Country Name</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>SD</td>
<td>SUDAN</td>
</tr>
<tr>
<td>SR</td>
<td>SURINAME</td>
</tr>
<tr>
<td>SJ</td>
<td>SVALBARD AND JAN MAYEN</td>
</tr>
<tr>
<td>SH</td>
<td>ST. HELENA</td>
</tr>
<tr>
<td>PM</td>
<td>ST. PIERRE AND MIQUELON</td>
</tr>
<tr>
<td>SZ</td>
<td>SWAZILAND</td>
</tr>
<tr>
<td>SE</td>
<td>SWEDEN</td>
</tr>
<tr>
<td>CH</td>
<td>SWITZERLAND</td>
</tr>
<tr>
<td>SY</td>
<td>SYRIAN ARAB REPUBLIC</td>
</tr>
<tr>
<td>TW</td>
<td>TAIWAN</td>
</tr>
<tr>
<td>TJ</td>
<td>TAJIKISTAN</td>
</tr>
<tr>
<td>TZ</td>
<td>TANZANIA, UNITED REPUBLIC OF</td>
</tr>
<tr>
<td>TH</td>
<td>THAILAND</td>
</tr>
<tr>
<td>TL</td>
<td>TIMOR-LESTE</td>
</tr>
<tr>
<td>TG</td>
<td>TOGO</td>
</tr>
<tr>
<td>TK</td>
<td>TOKELAU</td>
</tr>
<tr>
<td>TO</td>
<td>TONGA</td>
</tr>
<tr>
<td>TT</td>
<td>TRINIDAD AND TOBAGO</td>
</tr>
<tr>
<td>TN</td>
<td>TUNISIA</td>
</tr>
<tr>
<td>TR</td>
<td>TURKEY</td>
</tr>
<tr>
<td>TM</td>
<td>TURKMENISTAN</td>
</tr>
<tr>
<td>TC</td>
<td>TURKS AND CAICOS ISLANDS</td>
</tr>
<tr>
<td>TV</td>
<td>TUVALU</td>
</tr>
<tr>
<td>UG</td>
<td>UGANDA</td>
</tr>
<tr>
<td>UA</td>
<td>UKRAINE</td>
</tr>
<tr>
<td>AE</td>
<td>UNITED ARAB EMIRATES</td>
</tr>
<tr>
<td>GB</td>
<td>UNITED KINGDOM</td>
</tr>
<tr>
<td>US</td>
<td>UNITED STATES</td>
</tr>
<tr>
<td>UM</td>
<td>UNITED STATES MINOR OUTLYING ISLANDS</td>
</tr>
<tr>
<td>UY</td>
<td>URUGUAY</td>
</tr>
<tr>
<td>UZ</td>
<td>UZBEKISTAN</td>
</tr>
<tr>
<td>VU</td>
<td>VANUATU</td>
</tr>
<tr>
<td>VA</td>
<td>VATICAN CITY STATE (HOLY SEE)</td>
</tr>
<tr>
<td>VE</td>
<td>VENEZUELA</td>
</tr>
<tr>
<td>VN</td>
<td>VIET NAM</td>
</tr>
<tr>
<td>VG</td>
<td>VIRGIN ISLANDS (BRITISH)</td>
</tr>
<tr>
<td>VI</td>
<td>VIRGIN ISLANDS (U.S.)</td>
</tr>
<tr>
<td>WF</td>
<td>WALLIS AND FUTUNA</td>
</tr>
<tr>
<td>EH</td>
<td>WESTERN SAHARA</td>
</tr>
<tr>
<td>YE</td>
<td>YEMEN</td>
</tr>
<tr>
<td>YU</td>
<td>YUGOSLAVIA</td>
</tr>
<tr>
<td>ZM</td>
<td>ZAMBIA</td>
</tr>
<tr>
<td>ZW</td>
<td>ZIMBABWE</td>
</tr>
</tbody>
</table>
Appendix F. RGB Color Values

For reference convenience, some RGB color values you may use are listed here by the color names with their RGB values.

<table>
<thead>
<tr>
<th>Color Name</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aliceblue</td>
<td>rgb(240, 248, 255)</td>
</tr>
<tr>
<td>antiquewhite</td>
<td>rgb(250, 235, 215)</td>
</tr>
<tr>
<td>aqua</td>
<td>rgb(0, 255, 255)</td>
</tr>
<tr>
<td>aquamarine</td>
<td>rgb(127, 255, 212)</td>
</tr>
<tr>
<td>azure</td>
<td>rgb(240, 255, 255)</td>
</tr>
<tr>
<td>beige</td>
<td>rgb(245, 245, 220)</td>
</tr>
<tr>
<td>bisque</td>
<td>rgb(255, 228, 196)</td>
</tr>
<tr>
<td>black</td>
<td>rgb(0, 0, 0)</td>
</tr>
<tr>
<td>blanchedalmond</td>
<td>rgb(255, 235, 205)</td>
</tr>
<tr>
<td>blue</td>
<td>rgb(0, 0, 255)</td>
</tr>
<tr>
<td>blueviolet</td>
<td>rgb(138, 43, 226)</td>
</tr>
<tr>
<td>brown</td>
<td>rgb(165, 42, 42)</td>
</tr>
<tr>
<td>burlywood</td>
<td>rgb(222, 184, 135)</td>
</tr>
<tr>
<td>cadetblue</td>
<td>rgb(95, 158, 160)</td>
</tr>
<tr>
<td>chartreuse</td>
<td>rgb(127, 255, 0)</td>
</tr>
<tr>
<td>chocolate</td>
<td>rgb(210, 105, 30)</td>
</tr>
<tr>
<td>coral</td>
<td>rgb(255, 127, 80)</td>
</tr>
<tr>
<td>cornflowerblue</td>
<td>rgb(100, 149, 237)</td>
</tr>
<tr>
<td>cornsilk</td>
<td>rgb(255, 248, 220)</td>
</tr>
<tr>
<td>crimson</td>
<td>rgb(220, 20, 60)</td>
</tr>
<tr>
<td>cyan</td>
<td>rgb(0, 255, 255)</td>
</tr>
<tr>
<td>darkblue</td>
<td>rgb(0, 0, 139)</td>
</tr>
<tr>
<td>darkcyan</td>
<td>rgb(0, 139, 139)</td>
</tr>
<tr>
<td>darkgoldenrod</td>
<td>rgb(184, 134, 11)</td>
</tr>
<tr>
<td>darkgray</td>
<td>rgb(169, 169, 169)</td>
</tr>
<tr>
<td>darkgreen</td>
<td>rgb(0, 100, 0)</td>
</tr>
<tr>
<td>darkgrey</td>
<td>rgb(169, 169, 169)</td>
</tr>
<tr>
<td>darkkhaki</td>
<td>rgb(189, 183, 107)</td>
</tr>
<tr>
<td>darkmagenta</td>
<td>rgb(139, 0, 139)</td>
</tr>
<tr>
<td>darkolivegreen</td>
<td>rgb(85, 107, 47)</td>
</tr>
<tr>
<td>darkorange</td>
<td>rgb(255, 140, 0)</td>
</tr>
<tr>
<td>darkorchid</td>
<td>rgb(153, 50, 204)</td>
</tr>
<tr>
<td>darkred</td>
<td>rgb(139, 0, 0)</td>
</tr>
<tr>
<td>darksalmon</td>
<td>rgb(233, 150, 122)</td>
</tr>
<tr>
<td>darkseagreen</td>
<td>rgb(143, 188, 143)</td>
</tr>
<tr>
<td>darkslateblue</td>
<td>rgb(72, 61, 139)</td>
</tr>
<tr>
<td>darkslategray</td>
<td>rgb(47, 79, 79)</td>
</tr>
<tr>
<td>darkturquoise</td>
<td>rgb(0, 206, 209)</td>
</tr>
<tr>
<td>darkviolet</td>
<td>rgb(148, 0, 211)</td>
</tr>
<tr>
<td>Color Name</td>
<td>RGB Color</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>lightsteelblue</td>
<td>rgb(176, 196, 222)</td>
</tr>
<tr>
<td>lightyellow</td>
<td>rgb(255, 255, 224)</td>
</tr>
<tr>
<td>lime</td>
<td>rgb(0, 255, 0)</td>
</tr>
<tr>
<td>limegreen</td>
<td>rgb(50, 205, 50)</td>
</tr>
<tr>
<td>linen</td>
<td>rgb(250, 240, 230)</td>
</tr>
<tr>
<td>magenta</td>
<td>rgb(255, 0, 255)</td>
</tr>
<tr>
<td>maroon</td>
<td>rgb(128, 0, 0)</td>
</tr>
<tr>
<td>mediumaquamarine</td>
<td>rgb(102, 205, 170)</td>
</tr>
<tr>
<td>mediumblue</td>
<td>rgb(0, 0, 205)</td>
</tr>
<tr>
<td>mediummorchid</td>
<td>rgb(186, 85, 211)</td>
</tr>
<tr>
<td>mediumpurple</td>
<td>rgb(147, 112, 219)</td>
</tr>
<tr>
<td>mediumseagreen</td>
<td>rgb(60, 179, 113)</td>
</tr>
<tr>
<td>mediumslateblue</td>
<td>rgb(123, 104, 238)</td>
</tr>
<tr>
<td>mediumspringgreen</td>
<td>rgb(0, 250, 154)</td>
</tr>
<tr>
<td>mediumturquoise</td>
<td>rgb(72, 209, 204)</td>
</tr>
<tr>
<td>mediumvioletred</td>
<td>rgb(199, 21, 133)</td>
</tr>
<tr>
<td>midnightblue</td>
<td>rgb(25, 25, 112)</td>
</tr>
<tr>
<td>mintcream</td>
<td>rgb(245, 255, 250)</td>
</tr>
<tr>
<td>mistyrose</td>
<td>rgb(255, 228, 225)</td>
</tr>
<tr>
<td>moccasin</td>
<td>rgb(255, 228, 181)</td>
</tr>
<tr>
<td>navajowhite</td>
<td>rgb(255, 222, 173)</td>
</tr>
<tr>
<td>navy</td>
<td>rgb(0, 0, 128)</td>
</tr>
<tr>
<td>oldlace</td>
<td>rgb(253, 245, 230)</td>
</tr>
<tr>
<td>olive</td>
<td>rgb(128, 128, 0)</td>
</tr>
<tr>
<td>olivedrab</td>
<td>rgb(107, 142, 35)</td>
</tr>
<tr>
<td>orange</td>
<td>rgb(255, 165, 0)</td>
</tr>
<tr>
<td>orangered</td>
<td>rgb(255, 69, 0)</td>
</tr>
<tr>
<td>orchid</td>
<td>rgb(218, 112, 214)</td>
</tr>
<tr>
<td>palegoldenrod</td>
<td>rgb(238, 232, 170)</td>
</tr>
<tr>
<td>palegreen</td>
<td>rgb(152, 251, 152)</td>
</tr>
<tr>
<td>paleturquoise</td>
<td>rgb(175, 238, 238)</td>
</tr>
<tr>
<td>palevioletred</td>
<td>rgb(219, 112, 147)</td>
</tr>
<tr>
<td>papayawhip</td>
<td>rgb(255, 239, 213)</td>
</tr>
<tr>
<td>peachpuff</td>
<td>rgb(255, 218, 185)</td>
</tr>
<tr>
<td>peru</td>
<td>rgb(205, 133, 63)</td>
</tr>
<tr>
<td>pink</td>
<td>rgb(255, 192, 203)</td>
</tr>
<tr>
<td>plum</td>
<td>rgb(221, 160, 221)</td>
</tr>
<tr>
<td>powderblue</td>
<td>rgb(176, 224, 230)</td>
</tr>
<tr>
<td>purple</td>
<td>rgb(139, 69, 19)</td>
</tr>
<tr>
<td>red</td>
<td>rgb(128, 0, 128)</td>
</tr>
<tr>
<td>rosybrown</td>
<td>rgb(188, 143, 143)</td>
</tr>
<tr>
<td>royalblue</td>
<td>rgb(65, 105, 225)</td>
</tr>
<tr>
<td>saddlebrown</td>
<td>rgb(139, 69, 19)</td>
</tr>
<tr>
<td>salmon</td>
<td>rgb(250, 128, 114)</td>
</tr>
</tbody>
</table>
Appendix G. 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

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_memcpy</td>
<td>Copy characters from one buffer to another until a given character or given number of characters has been copied</td>
</tr>
<tr>
<td>memchr</td>
<td>Return a pointer to the first occurrence, within a specified number of characters, of a given character in the buffer</td>
</tr>
<tr>
<td>memcmp</td>
<td>Compare the specified number of characters from two buffers</td>
</tr>
<tr>
<td>memcpy</td>
<td>Copy specified number of characters from one buffer to another</td>
</tr>
<tr>
<td>_memicmp</td>
<td>Compare the specified number of characters from two buffers without regard to case</td>
</tr>
<tr>
<td>memmove</td>
<td>Copy specified number of characters from one buffer to another</td>
</tr>
<tr>
<td>memset</td>
<td>Use given character to initialize a specified number of bytes in the buffer</td>
</tr>
<tr>
<td>_swab</td>
<td>Swap bytes of data and store them at the specified location</td>
</tr>
</tbody>
</table>

Character-Classification Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Character test condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum, iswalnum, _ismbcalnum</td>
<td>True if alphanumeric</td>
</tr>
<tr>
<td>isalpha, iswalpha, _ismbcalpha</td>
<td>True if alphabetic</td>
</tr>
<tr>
<td>_isascii, iswascii</td>
<td>True if ASCII</td>
</tr>
<tr>
<td>iscntrl, iswcntrl</td>
<td>True if control character</td>
</tr>
<tr>
<td>_iscsym, _iscsymf</td>
<td>True if the letter, underline, or digit</td>
</tr>
<tr>
<td>isdigit, iswdigit, _ismbcdigit</td>
<td>True if the decimal digit</td>
</tr>
<tr>
<td>isgraph, iswgraph, _ismbcgraph</td>
<td>True if printable other than space</td>
</tr>
<tr>
<td>islower, iswlower, _ismbclower</td>
<td>True if lowercase</td>
</tr>
<tr>
<td><strong>Routine</strong></td>
<td><strong>Use</strong></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>_cgets, _cgetws</td>
<td>Read string from the console</td>
</tr>
<tr>
<td>_cprintf, _cwprintf</td>
<td>Write formatted data to the console</td>
</tr>
<tr>
<td>_cputs</td>
<td>Write a string to console</td>
</tr>
<tr>
<td>_cscanf, _cwscanf</td>
<td>Read formatted data from the console</td>
</tr>
<tr>
<td>_getch, _getwch</td>
<td>Read a character from the console</td>
</tr>
<tr>
<td>_getche, _getwche</td>
<td>Read a character from the console and echo it</td>
</tr>
<tr>
<td>_inp</td>
<td>Read one byte from the specified I/O port</td>
</tr>
<tr>
<td>_inpd</td>
<td>Read float word from specified I/O port</td>
</tr>
<tr>
<td>_inpw</td>
<td>Read 2-byte word from specified I/O port</td>
</tr>
<tr>
<td>_kbhit</td>
<td>Check for keystroke at the console; use before attempting to read from the console</td>
</tr>
<tr>
<td>_outp</td>
<td>Write one byte to specified I/O port</td>
</tr>
<tr>
<td>_outpd</td>
<td>Write float word to specified I/O port</td>
</tr>
<tr>
<td>_outpw</td>
<td>Write the word to the specified I/O port</td>
</tr>
<tr>
<td>_putch, _putwch</td>
<td>Write character to console</td>
</tr>
<tr>
<td>_ungetch, _ungetwch</td>
<td>&quot;Unget&quot; last character read from the console so it becomes the next character read</td>
</tr>
</tbody>
</table>

---

**Console and Port I/O Routines**

<table>
<thead>
<tr>
<th><strong>ismbchira</strong></th>
<th>True if Hiragana</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ismbckata</strong></td>
<td>True if Katakana</td>
</tr>
<tr>
<td><strong>ismbclegal</strong></td>
<td>True if Legal multibyte character</td>
</tr>
<tr>
<td><strong>ismbcl0</strong></td>
<td>True if Japan-level 0 multibyte character</td>
</tr>
<tr>
<td><strong>ismbcl1</strong></td>
<td>True if Japan-level 1 multibyte character</td>
</tr>
<tr>
<td><strong>ismbcl2</strong></td>
<td>True if Japan-level 2 multibyte character</td>
</tr>
<tr>
<td><strong>ismbcsymbol</strong></td>
<td>True if Nonalphanumeric multibyte character</td>
</tr>
<tr>
<td><strong>isprint</strong></td>
<td>True if the printable character</td>
</tr>
<tr>
<td><strong>ispunct</strong></td>
<td>True if punctuation</td>
</tr>
<tr>
<td><strong>isspace</strong></td>
<td>True if white-space</td>
</tr>
<tr>
<td><strong>isupper</strong></td>
<td>True if uppercase</td>
</tr>
<tr>
<td><strong>iswctype</strong></td>
<td>The property specified by desc argument</td>
</tr>
<tr>
<td><strong>isxdigit</strong></td>
<td>True if the hexadecimal digit</td>
</tr>
<tr>
<td><strong>mblen</strong></td>
<td>Return the length of valid multibyte character; result depends on LC_CTYPE category setting of the current</td>
</tr>
</tbody>
</table>
## Data-Conversion Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Find the absolute value of the integer</td>
</tr>
<tr>
<td>atof</td>
<td>Convert string to float</td>
</tr>
<tr>
<td>atoi, _atoi64</td>
<td>Convert string to int</td>
</tr>
<tr>
<td>atol</td>
<td>Convert string to long</td>
</tr>
<tr>
<td>_ecvt</td>
<td>Convert float to the string of specified length</td>
</tr>
<tr>
<td>_fcvt</td>
<td>Convert float to string with a specified number of digits following the decimal point</td>
</tr>
<tr>
<td>_gcvt</td>
<td>Convert float number to string; store string in the buffer</td>
</tr>
<tr>
<td>_itoa, _i64toa, _itow, _i64tow</td>
<td>Convert int to string</td>
</tr>
<tr>
<td>labs</td>
<td>Find the absolute value of the long integer</td>
</tr>
<tr>
<td>_ltoa, _ltow</td>
<td>Convert long to string</td>
</tr>
<tr>
<td>_mbbtombc</td>
<td>Convert 1-byte multibyte character to corresponding 2-byte multibyte character</td>
</tr>
<tr>
<td>_mbcjistojms</td>
<td>Convert Japan Industry Standard (JIS) character to Japan Microsoft (JMS) character</td>
</tr>
<tr>
<td>_mbcjstmstojis</td>
<td>Convert JMS character to JIS character</td>
</tr>
<tr>
<td>_mbctohira</td>
<td>Convert multibyte character to 1-byte hiragana code</td>
</tr>
<tr>
<td>_mbctokata</td>
<td>Convert multibyte character to 1-byte katakana code</td>
</tr>
<tr>
<td>_mbctombb</td>
<td>Convert 2-byte multibyte character to corresponding 1-byte multibyte character</td>
</tr>
<tr>
<td>mbstowcs</td>
<td>Convert sequence of multibyte characters to the corresponding sequence of wide characters</td>
</tr>
<tr>
<td>mbtowc</td>
<td>Convert multibyte character to corresponding wide character</td>
</tr>
<tr>
<td>strtod, wcstod</td>
<td>Convert string to a float</td>
</tr>
<tr>
<td>strtol, wcstol</td>
<td>Convert string to a long integer</td>
</tr>
<tr>
<td>strtoul, wcstoul</td>
<td>Convert string to an unsigned long integer</td>
</tr>
<tr>
<td>strxfrm, wcxsxfrm</td>
<td>Transform string into collated form based on locale-specific information</td>
</tr>
<tr>
<td>_toascii</td>
<td>Convert character to ASCII code</td>
</tr>
<tr>
<td>tolower, towlower, _mbctolower</td>
<td>Test character and convert to lowercase if currently uppercase</td>
</tr>
<tr>
<td>_tolower</td>
<td>Convert character to lowercase unconditionally</td>
</tr>
<tr>
<td>toupper, towupper, _mbctoupper</td>
<td>Test character and convert to uppercase if currently lowercase</td>
</tr>
<tr>
<td>_toupper</td>
<td>Convert character to uppercase unconditionally</td>
</tr>
<tr>
<td>_ultoa, _ultow</td>
<td>Convert unsigned long to string</td>
</tr>
<tr>
<td>wcstombs</td>
<td>Convert sequence of wide characters to the corresponding sequence of multibyte characters</td>
</tr>
<tr>
<td>Routine</td>
<td>Use</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>wctomb</td>
<td>Convert wide character to corresponding multibyte character</td>
</tr>
<tr>
<td>_wtof</td>
<td>Convert wide-character string to a float</td>
</tr>
<tr>
<td>_wtoi, _wtoi64</td>
<td>Convert wide-character string to int or _int64</td>
</tr>
<tr>
<td>_wtol</td>
<td>Convert wide-character string to long</td>
</tr>
</tbody>
</table>

**Directory-Control Routines**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_chdir, _wchdir</td>
<td>Change current working directory</td>
</tr>
<tr>
<td>_chdrive</td>
<td>Change current drive</td>
</tr>
<tr>
<td>_getcwd, _wgetcwd</td>
<td>Get current working directory for default drive</td>
</tr>
<tr>
<td>_getdcwd, _wgetdcwd</td>
<td>Get current working directory for the specified drive</td>
</tr>
<tr>
<td>_getdiskfree</td>
<td>Populates a _diskfree_t structure with information about a disk drive.</td>
</tr>
<tr>
<td>_getdrive</td>
<td>Get current (default) drive</td>
</tr>
<tr>
<td>_getdrives</td>
<td>Returns a bitmask representing the currently available disk drives.</td>
</tr>
<tr>
<td>_mkdir, _wmkdir</td>
<td>Make new directory</td>
</tr>
<tr>
<td>_rmdir, _wrmdir</td>
<td>Remove directory</td>
</tr>
<tr>
<td>_searchenv,</td>
<td>Search for given file on specified paths</td>
</tr>
<tr>
<td>_wsearchenv</td>
<td></td>
</tr>
</tbody>
</table>

**File-Handling Routines (File Descriptor)**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_chsize</td>
<td>Change file size</td>
</tr>
<tr>
<td>_filelength</td>
<td>Get file length</td>
</tr>
<tr>
<td>_fstat, _fstat64, _fstat64</td>
<td>Get file-status information on the descriptor</td>
</tr>
<tr>
<td>_isatty</td>
<td>Check for character device</td>
</tr>
<tr>
<td>_locking</td>
<td>Lock areas of the file</td>
</tr>
<tr>
<td>_setmode</td>
<td>Set file-translation mode</td>
</tr>
</tbody>
</table>

**File-Handling Routines (Path or Filename)**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_access, _waccess</td>
<td>Check file-permission setting</td>
</tr>
<tr>
<td>_chmod, _wchmod</td>
<td>Change file-permission setting</td>
</tr>
<tr>
<td>_fullpath, _wfullpath</td>
<td>Expand a relative path to its absolute pathname</td>
</tr>
<tr>
<td>_get_osfhandle</td>
<td>Return operating-system file handle associated with existing stream FILE pointer</td>
</tr>
<tr>
<td>_makepath, _wmakepath</td>
<td>Merge path components into the single, full path</td>
</tr>
</tbody>
</table>
### File-Handling Routines (Open File)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>fopen</td>
<td>Opens a file and returns a pointer to the open file.</td>
</tr>
<tr>
<td>_fsopen</td>
<td>Open a stream with file sharing and returns a pointer to the open file.</td>
</tr>
<tr>
<td>_open</td>
<td>Opens a file and returns a file descriptor to the opened file.</td>
</tr>
<tr>
<td>_sopen</td>
<td>Open a file with file sharing and returns a file descriptor to the open file.</td>
</tr>
<tr>
<td>_fdopen</td>
<td>Associates a stream with a file that was previously opened for low-level I/O and returns a pointer to the open stream.</td>
</tr>
<tr>
<td>_fileno</td>
<td>Gets the file descriptor associated with a stream.</td>
</tr>
<tr>
<td>_open_osfhandle</td>
<td>Associates C run-time file descriptor with an existing operating-system file handle.</td>
</tr>
<tr>
<td>_pipe</td>
<td>Creates a pipe for reading and writing.</td>
</tr>
<tr>
<td>freopen</td>
<td>Reassign a file pointer.</td>
</tr>
</tbody>
</table>

### Low-Level I/O Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_close</td>
<td>Close file</td>
</tr>
<tr>
<td>_commit</td>
<td>Flush file to disk</td>
</tr>
<tr>
<td>_creat, _wcreat</td>
<td>Create file</td>
</tr>
<tr>
<td>_dup</td>
<td>Return next available file descriptor for a given file</td>
</tr>
<tr>
<td>_dup2</td>
<td>Create the second descriptor for a given file</td>
</tr>
<tr>
<td>_eof</td>
<td>Test for end of file</td>
</tr>
<tr>
<td>_lseek, _lseeki64</td>
<td>Reposition file pointer to a given location</td>
</tr>
<tr>
<td>_open, _wopen</td>
<td>Open file</td>
</tr>
<tr>
<td>_read</td>
<td>Read data from the file</td>
</tr>
<tr>
<td>_sopen, _wopen</td>
<td>Open file for file sharing</td>
</tr>
<tr>
<td>Routine</td>
<td>Use</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>_tell, _telli64</td>
<td>Get current file-pointer position</td>
</tr>
<tr>
<td>_umask</td>
<td>Set file-permission mask</td>
</tr>
<tr>
<td>_write</td>
<td>Write data to file</td>
</tr>
</tbody>
</table>

## Stream I/O Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearerr</td>
<td>Clear error indicator for the stream</td>
</tr>
<tr>
<td>fclose</td>
<td>Close stream</td>
</tr>
<tr>
<td>_fcloseall</td>
<td>Close all open streams except stdin, stdout, and stderr</td>
</tr>
<tr>
<td>_fdopen, wfdopen</td>
<td>Associate stream with file descriptor of open file</td>
</tr>
<tr>
<td>feof</td>
<td>Test for end of file on stream</td>
</tr>
<tr>
<td>ferror</td>
<td>Test for error on stream</td>
</tr>
<tr>
<td>fflush</td>
<td>Flush stream to buffer or storage device</td>
</tr>
<tr>
<td>fgetc, fgetwc</td>
<td>Read a character from the stream (function versions of getc and</td>
</tr>
<tr>
<td></td>
<td>getwc)</td>
</tr>
<tr>
<td>_fgetchar, _fgetwchar</td>
<td>Read a character from stdin (function versions of getchar and</td>
</tr>
<tr>
<td></td>
<td>getwchar)</td>
</tr>
<tr>
<td>fgetpos</td>
<td>Get position indicator of stream</td>
</tr>
<tr>
<td>fgets, fgetws</td>
<td>Read string from stream</td>
</tr>
<tr>
<td>_fileno</td>
<td>Get file descriptor associated with the stream</td>
</tr>
<tr>
<td>_flushall</td>
<td>Flush all streams to buffer or storage device</td>
</tr>
<tr>
<td>fopen, _wfopen</td>
<td>Open stream</td>
</tr>
<tr>
<td>fprintf, fwprintf</td>
<td>Write formatted data to stream</td>
</tr>
<tr>
<td>fputc, fputwc</td>
<td>Write a character to a stream (function versions of putc and</td>
</tr>
<tr>
<td></td>
<td>putwc)</td>
</tr>
<tr>
<td>_fputchar, _fputwchar</td>
<td>Write character to stdout (function versions of putchar and</td>
</tr>
<tr>
<td></td>
<td>putwchar)</td>
</tr>
<tr>
<td>fputs, fputws</td>
<td>Write a string to the stream</td>
</tr>
<tr>
<td>fread</td>
<td>Read unformatted data from the stream</td>
</tr>
<tr>
<td>freopen, _wfreopen</td>
<td>Reassign FILE stream pointer to new file or device</td>
</tr>
<tr>
<td>fscanf, fwscanf</td>
<td>Read formatted data from the stream</td>
</tr>
<tr>
<td>fseek</td>
<td>Move file position to the given location</td>
</tr>
<tr>
<td>fsetpos</td>
<td>Set position indicator of stream</td>
</tr>
<tr>
<td>_fopen, _wfopen</td>
<td>Open stream with file sharing</td>
</tr>
<tr>
<td>ftell</td>
<td>Get current file position</td>
</tr>
<tr>
<td>fwrite</td>
<td>Write unformatted data items to stream</td>
</tr>
<tr>
<td>getc, getwc</td>
<td>Read a character from the stream (macro versions of fgetc and</td>
</tr>
<tr>
<td></td>
<td>fgetwc)</td>
</tr>
<tr>
<td>getchar, getwchar</td>
<td>Read a character from stdin (macro versions of fgetchar and</td>
</tr>
<tr>
<td></td>
<td>fgetwchar)</td>
</tr>
<tr>
<td>_getmaxstdio</td>
<td>Returns the number of simultaneously open files permitted at the</td>
</tr>
<tr>
<td></td>
<td>stream I/O level.</td>
</tr>
<tr>
<td>gets, getstdio</td>
<td>Read line from stdin</td>
</tr>
</tbody>
</table>
### String-Manipulation Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_mbscoll</code>, <code>_mbsicoll</code>, <code>_mbsncoll</code>, <code>_mbsnicoll</code></td>
<td>Compare two multibyte-character strings using multibyte code page information (<code>_mbsicoll</code> and <code>_mbsnicoll</code> are case-insensitive)</td>
</tr>
<tr>
<td><code>_mbsdec</code>, <code>_strdec</code>, <code>_wcsdec</code></td>
<td>Move string pointer back one character</td>
</tr>
<tr>
<td><code>_mbsinc</code>, <code>_strinc</code>, <code>_wcsinc</code></td>
<td>Advance string pointer by one character</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>_mbslen</code></td>
<td>Get number of multibyte characters in the multibyte-character string; dependent upon OEM code page</td>
</tr>
<tr>
<td><code>_mbsnbcat</code></td>
<td>Append, at most, the first <code>n</code> bytes of one multibyte-character string to another</td>
</tr>
<tr>
<td><code>_mbsnbcmp</code></td>
<td>Compare first <code>n</code> bytes of two multibyte-character strings</td>
</tr>
<tr>
<td><code>_mbsnbcnt</code></td>
<td>Return number of multibyte-character bytes within supplied character count</td>
</tr>
<tr>
<td><code>_mbsnbcpy</code></td>
<td>Copy <code>n</code> bytes of the string</td>
</tr>
<tr>
<td><code>_mbsnbicmp</code></td>
<td>Compare <code>n</code> bytes of two multibyte-character strings, ignoring case</td>
</tr>
<tr>
<td><code>_mbsnbset</code></td>
<td>Set first <code>n</code> bytes of multibyte-character string to the specified character</td>
</tr>
<tr>
<td><code>_mbsncnt</code></td>
<td>Return number of multibyte characters within supplied byte count</td>
</tr>
<tr>
<td><code>_mbssncmp</code>, <code>_strnncmp</code>, <code>_wcsncmp</code></td>
<td>Find the next character in the string</td>
</tr>
<tr>
<td><code>_mbssninc</code>, <code>_strninc</code>, <code>_wcsninc</code></td>
<td>Advance string pointer by <code>n</code> characters</td>
</tr>
<tr>
<td><code>_mbsspnmp</code>, <code>_strspnmp</code>, <code>_wcsbspnmp</code></td>
<td>Return pointer to the first character in the given string that is not in another given string</td>
</tr>
<tr>
<td><code>_mbstrlen</code></td>
<td>Get number of multibyte characters in the multibyte-character string; locale-dependent</td>
</tr>
<tr>
<td><code>_scprintf</code>, <code>_scwprintf</code></td>
<td>Return the number of characters in a formatted string</td>
</tr>
<tr>
<td><code>_snscanf</code>, <code>_snwscanf</code></td>
<td>Read formatted data of a specified length from the standard input stream.</td>
</tr>
<tr>
<td><code>strftime</code>, <code>wcsftime</code></td>
<td>Format date-and-time string</td>
</tr>
<tr>
<td><code>strcat</code>, <code>wcscat</code>, <code>_mbscat</code></td>
<td>Append one string to another</td>
</tr>
<tr>
<td><code>strchr</code>, <code>wcschr</code>, <code>_mbschr</code></td>
<td>Find the first occurrence of a specified character in the string</td>
</tr>
<tr>
<td><code>strcmp</code>, <code>wcscmp</code>, <code>_mbscmp</code></td>
<td>Compare two strings</td>
</tr>
<tr>
<td><code>strcoll</code>, <code>wcscoll</code>, <code>_stricoll</code>, <code>_wcscoll</code>, <code>_strnicoll</code>, <code>_wcscoll</code>, <code>_strmicoll</code>, <code>_wcscoll</code></td>
<td>Compare two strings using current locale code page information ( <code>_stricoll</code>, <code>_wcscoll</code>, <code>_stricoll</code>, and <code>_wcscoll</code> are case-insensitive)</td>
</tr>
<tr>
<td><code>strcpy</code>, <code>wcscopy</code>, <code>_mbscopy</code></td>
<td>Copy one string to another</td>
</tr>
<tr>
<td><code>strcspn</code>, <code>wcscspn</code>, <code>_mbcspsn</code></td>
<td>Find the first occurrence of character from the specified character set in string</td>
</tr>
<tr>
<td><code>_strdup</code>, <code>_wcsdup</code>, <code>_mbsdup</code></td>
<td>Duplicate string</td>
</tr>
<tr>
<td><code>strerror</code>, <code>wcsererror</code></td>
<td>Map error number to the message string</td>
</tr>
<tr>
<td><code>strerror</code>, <code>wcsererror</code></td>
<td>Map user-defined error message to a string</td>
</tr>
<tr>
<td><code>strftime</code>, <code>wcsftime</code></td>
<td>Format date-and-time string</td>
</tr>
<tr>
<td><code>_stricmp</code>, <code>_wcscmp</code>, <code>_mbscmp</code></td>
<td>Compare two strings without regard to case</td>
</tr>
<tr>
<td><code>strlen</code>, <code>wcslen</code>, <code>_mbslen</code>, <code>_mbstrlen</code></td>
<td>Find the length of the string</td>
</tr>
<tr>
<td><code>_strlwr</code>, <code>wcslwr</code>, <code>_mblwr</code></td>
<td>Convert string to lowercase</td>
</tr>
<tr>
<td>Function</td>
<td>Use</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>strncat, wcsncat, _mbncat</td>
<td>Append characters of the string</td>
</tr>
<tr>
<td>strncmp, wcsncmp, _mbsncmp</td>
<td>Compare characters of two strings</td>
</tr>
<tr>
<td>strncpy, wcsncpy, _mbsncpy</td>
<td>Copy characters of one string to another</td>
</tr>
<tr>
<td>_strnicmp, _wcsnicmp, _mbsnicmp</td>
<td>Compare characters of two strings without regard to case</td>
</tr>
<tr>
<td>_strncpy, _wcsncpy, _mbsncpy</td>
<td>Set first n characters of the string to the specified character</td>
</tr>
<tr>
<td>strpbrk, wcsppbrk, _mbspbrk</td>
<td>Find the first occurrence of character from one string in another string</td>
</tr>
<tr>
<td>strrchr, wcsrchr, _mbstrchr</td>
<td>Find the last occurrence of a given character in the string</td>
</tr>
<tr>
<td>_strrev, _wcsrev, _mbstrrev</td>
<td>Reverse string</td>
</tr>
<tr>
<td>_strset, _wcsset, _mbset</td>
<td>Set all characters of the string to the specified character</td>
</tr>
<tr>
<td>strspn, wcspspn, _mbsspnpn</td>
<td>Find the first substring from one string in another string</td>
</tr>
<tr>
<td>strstr, wcssstr, _mbssstr</td>
<td>Find the first occurrence of the specified string in another string</td>
</tr>
<tr>
<td>strtok, wcstok, _mbstok</td>
<td>Find the next token in the string</td>
</tr>
<tr>
<td>_strupr, _wcsupr, _mbsupr</td>
<td>Convert string to uppercase</td>
</tr>
<tr>
<td>strxfrm, wcsxfrm</td>
<td>Transform string into collated form based on locale-specific information</td>
</tr>
<tr>
<td>vsprintf, _vstprintf</td>
<td>Write formatted output using a pointer to a list of arguments</td>
</tr>
</tbody>
</table>

**Time Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>asctime, _wasctime</td>
<td>Convert time from type struct tm to character string</td>
</tr>
<tr>
<td>clock</td>
<td>Return elapsed CPU time for the process</td>
</tr>
<tr>
<td>ctime, _ctime64, _wctime, _wctime64</td>
<td>Convert time from type time_t or _time64_t to a character string</td>
</tr>
<tr>
<td>difftime</td>
<td>Compute the difference between two times</td>
</tr>
<tr>
<td>_ftime, _ftime64</td>
<td>Store current system time in the variable of type struct _timeb or type struct _timeb64</td>
</tr>
<tr>
<td>_futime, _futime64</td>
<td>Set modification time on open file</td>
</tr>
<tr>
<td>gmtime, _gmtime64</td>
<td>Convert time from type time_t to struct tm or from type _time64_t to struct tm</td>
</tr>
<tr>
<td>localtime, _localtime64</td>
<td>Convert time from type time_t to struct tm or from type _time64_t to struct tm with local correction</td>
</tr>
<tr>
<td>mktime, _mktime64</td>
<td>Convert time to calendar value</td>
</tr>
<tr>
<td>_strdate, _wstrdate</td>
<td>Return current system date as a string</td>
</tr>
<tr>
<td>strftime, wcsftime</td>
<td>Format date-and-time string for international use</td>
</tr>
<tr>
<td>_strftime, _wstrtime</td>
<td>Return current system time as a string</td>
</tr>
</tbody>
</table>
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 the specified file using either current time or time value stored

---

Memory and Other Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>Stops the program</td>
</tr>
<tr>
<td>assert</td>
<td>Stops the program if an expression isn't true</td>
</tr>
<tr>
<td>atexit</td>
<td>Sets a function to be called when the program exits</td>
</tr>
<tr>
<td>bsearch</td>
<td>Perform a binary search</td>
</tr>
<tr>
<td>calloc</td>
<td>Allocates a two-dimensional chunk of memory</td>
</tr>
<tr>
<td>exit</td>
<td>Stop the program</td>
</tr>
<tr>
<td>free</td>
<td>Frees memory available for future allocation</td>
</tr>
<tr>
<td>getenv</td>
<td>Get environment information about a variable</td>
</tr>
<tr>
<td>longjmp</td>
<td>Start execution at a certain point in the program</td>
</tr>
<tr>
<td>qsort</td>
<td>Perform a quicksort</td>
</tr>
<tr>
<td>malloc</td>
<td>Allocates memory</td>
</tr>
<tr>
<td>raise</td>
<td>Send a signal to the program</td>
</tr>
<tr>
<td>rand</td>
<td>Returns a pseudorandom number</td>
</tr>
<tr>
<td>realloc</td>
<td>Changes the size of previously allocated memory</td>
</tr>
<tr>
<td>setjmp</td>
<td>Set execution to start at a certain point</td>
</tr>
<tr>
<td>signal</td>
<td>Register a function as a signal handler</td>
</tr>
<tr>
<td>srand</td>
<td>Initialize the random number generator</td>
</tr>
<tr>
<td>system</td>
<td>Perform a system call</td>
</tr>
<tr>
<td>va_arg</td>
<td>Use variable-length parameter lists</td>
</tr>
</tbody>
</table>