MakeAFP Weaver Reference

Version 3.7
This edition applies to the MakeAFP Weaver.

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

support@makeafp.com

When you send information to MakeAFP, you grant MakeAFP a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.
Chapter 1. MakeAFP Weaver Functions

Using the MakeAFP Weaver Function Calls to Build an AFP Document

Hierarchical of MakeAFP Weaver Calls

Session Level calls

Document Level Calls

Page-level Calls

Format of the Function Call Descriptions

2D Aztec Barcode by Drawing

2D DataMatrix Barcode by Drawing

2D MaxiCode Barcode by Drawing

2D MicroPDF417 Barcode by Drawing

2D PDF417 Barcode by Drawing

2D PDF417 Truncated Barcode by Drawing

2D QR Code Barcode by Drawing

Barcode (Linear) by Drawing

BCOCA Barcode (Linear)

BCOCA MaxiCode 2D Barcode

BCOCA PDF417 2D Barcode

BCOCA QR Code 2D Barcode

Begin of Index Name Group of Input AFP (Checking)

Begin Index Group

Begin Overstrike

Begin Underscore

Blank Page (Checking)

Box Drawing

Center Align 1-Byte Text

Center Align Japanese

Center Align Korean

Center Align Simplified Chinese

Center Align Traditional Chinese

Center Align SBCS-HOST/DBCS-HOST

Center Align UTF-16 Text

Center Align UTF-16 Text Converting from Legacy String

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

**Chapter 2. MakeAFP Weaver Parameters** .................................................................................................... 204

Conventions Used in This Chapter ....................................................................................................................... 204
CMR – Specifies a Colo Management Resource ................................................................................................. 205
CPGID – Specifies a Code Page Identifier .......................................................................................................... 205
FDEF – Specifies a Form Fefintion ......................................................................................................................... 206
FDEFLIB – Specifies the Library Path of Form Definitions .................................................................................. 206
FONT – Specifies an AFP FOCA Raster or Outline Font ...................................................................................... 207
FONT – Specifies an OpenType/TrueType Font .................................................................................................... 208
FONTLIB – Specifies the Library Path of Fonts .................................................................................................. 209
INDEXOBJ – Specifies Generating of the AFP Index Object File ....................................................................... 210
OBJT – Specifies an AFP Object or non-AFP Object ......................................................................................... 210
OBJTLIB – Specifies the Library Path of Image and CMR Objects ..................................................................... 211
OVLY – Specifies an Overlay ................................................................................................................................ 212
OVLYLIB – Specifies the Library Path of Overlays ............................................................................................ 212
PRMODE – Specifies the Type of Input Data and Processing Option .................................................................. 213
PSEG – Specifies a Page Segment ........................................................................................................................ 213
PSEGLIB – Specifies the Library Path of Page Segments .................................................................................. 214
RESLIB – Specifies the Library Path of AFP Resources and Objects .................................................................. 215
RESTYPE - Specifies the Types of Resources to be Retrieved .......................................................................... 215
Chapter 3. MakeAFP Weaver Variables ................................................................. 217

$Bng – Begin of AFP Page Group Index ................................................................. 217
$Eng – End of AFP Page Group Index ................................................................. 217
$Edt – End of AFP Document ............................................................................ 217
$MaxPaging – Maximum Number of AFP Page Buffers .................................... 217
$Page – Current AFP Page Buffer Number ......................................................... 218

Chapter 4. String Manipulation Functions .............................................................. 219

Comma Float ........................................................................................................ 219
Comma Integer .................................................................................................. 220
Comma Digital String ....................................................................................... 220
Delete Characters ............................................................................................. 221
Extract Substrings ............................................................................................ 221
Find String ......................................................................................................... 221
First Character .................................................................................................. 223
In Substitution Table .......................................................................................... 224
Insert String ..................................................................................................... 224
Is Empty ............................................................................................................ 225
Last Character .................................................................................................. 225
Left Copy ........................................................................................................... 226
Left Copy and Pad ............................................................................................. 226
Left Copy and Right Trim .................................................................................. 227
Left Trim ........................................................................................................... 227
Left Trim for EBCDIC ....................................................................................... 228
Left Trim and Concatenate Strings ................................................................... 228
Match String Comparing .................................................................................. 229
Pattern Searching ............................................................................................. 230
Remove String .................................................................................................. 230
Replace String .................................................................................................. 231
Reverse Find String .......................................................................................... 232
Right Copy ......................................................................................................... 233
Right Copy and Left Trim .................................................................................. 233
Right Copy and Pad .......................................................................................... 234
Right Copy and Right Trim ............................................................................... 234
Right Trim ......................................................................................................... 234
Right Trim for EBCDIC ..................................................................................... 235
Strings Concatenate ........................................................................................ 236
String Pad ......................................................................................................... 237
Substitute String .............................................................................................. 237
Substitute Change ............................................................................................ 238
Substring ........................................................................................................... 238
Stream I/O Routines ..........................................................281
String-Manipulation Routines ...............................................282
Time Routines ................................................................284
Memory and Other Routines ................................................285
Chapter 1. MakeAFP Weaver Functions

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

Using the MakeAFP Weaver Function Calls to Build an AFP Document

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

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

Hierarchy of MakeAFP Weaver Calls

MakeAFP Weaver has three levels of function calls:
• **Session Level Calls**
  These calls start the MakeAFP Weaver session, set overall session measurement units, and are issued only once for each program. Print and view the AFP file if needed.

• **Document Level Calls**
  These calls open and close an AFP document and place data (such as page group level indexes) at the document level.

• **Page-Level Calls**
  These calls open and close an AFP page, and format data within individual pages.

**Session Level calls**

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

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

- **Start Session**
  Starts the MakeAFP Weaver session.

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

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

**Document Level Calls**

- **Open Document**
  Opens an AFP document.

- **Begin Page Group Index**
  Begins a page group level index.

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

- **Close Page Group Index**
  Ends a page group level index.

- **Invoke Copy-Group**
  Invokes an AFP copy-group.

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

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

- **Close Document**
  Closes an AFP document.
Page-level Calls

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

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

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

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

**Put Fixed Paragraph**
Puts a boxed text paragraph.

**Draw Horizontal Line**
Draws a fixed-length horizontal line.

**Draw Vertical Line**
Draws a fixed-length vertical line.

**Set Horizontal X Position**
Specifies a horizontal X position.

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

**Set Vertical Y Position**
Specifies a vertical Y position.

**Query Position**
Queries current X or Y position.

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

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

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

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

**Set Text Orientation**
Sets text orientation for the subsequent text.

**Begin Underscore**
Begins an underscore.

**End Underscore**
Ends an underscore.

**Put Linear Barcode Data**
Puts a linear barcode data.

**Put 2D Barcode Data**
Puts a 2D barcode data.
Mask Area
Hides a rectangle area.

Get Index Value
Gets index tag value’s strings that can be used to create the string of barcode.

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

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

Close Page
Closes the page.

Format of the Function Call Descriptions
The function descriptions are listed in alphabetic order. Each function calls description includes the following sections:

Function
A description of the major purpose of the function.

Syntax
A diagram showing the function parameters.

Parameters
Explanation of each parameter.

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

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

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

MakeAFP provides default values to rarely used parameters to simplify the use of the MakeAFP Weaver function.
## 2D Aztec Barcode by Drawing

### Function

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

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

### Syntax

```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 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:**

![Barcode QR Code](image-url)
2D DataMatrix Barcode by Drawing

Function

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

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

Syntax

```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:

- 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 DataMatrix"
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
DataMatrix(data,1.2,1.5);
ClosePage();
CloseDoc();
```

Print / display:
2D MaxiCode Barcode by Drawing

Function

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

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

Syntax

```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 leftmost 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 sequences (includes Standard Error Correction)</td>
<td>93</td>
<td>135</td>
</tr>
<tr>
<td>5</td>
<td>Full ECC, like MODE4 but with Enhanced Error Correction</td>
<td>77</td>
<td>110</td>
</tr>
<tr>
<td>6</td>
<td>Reserved for the maintenance of scanner hardware</td>
<td>93</td>
<td>135</td>
</tr>
</tbody>
</table>
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.

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

Sample
```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:
```

![MaxiCode Sample Image](image-url)
2D MicroPDF417 Barcode by Drawing

Function

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

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

Syntax

```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:

<table>
<thead>
<tr>
<th>Value</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:

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

Print / display:
2D PDF417 Barcode by Drawing

Function

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

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

Syntax

```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:

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

Print / display:
2D PDF417 Truncated Barcode by Drawing

Function

Although MakeAFP Weaver supports the 2D barcodes defined in the latest IBM BCOC standard for ultra-fast speed formatting with a very small AFP BCOC data stream, most AFP viewers and transformers for AFP are not able to view or convert BCOC object into other formats, like PDF, HTML, and XML, the same BCOC 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 is an integer from 0 (only error recognition) to 8 (highest). The higher the security level, the more error correction codewords will be added to the symbol. The default value is 0, the security level is chosen automatically according to the number of input characters.

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

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

- DEG0: The barcode image is not rotated.
- DEG90: The barcode image is rotated 90 degrees clockwise
- DEG180: The barcode image is rotated 180 degrees clockwise
- DEG270: The barcode image is rotated 270 degrees clockwise
ocaColor
Valid AFP OCA color values are BLUE, RED, MAGENTA or PINK, GREEN, CYAN or TURQ, YELLOW, BLACK, BROWN, MUSTARD, DARKBLUE, DARKGREEN, DARKTURQ or DARKCYAN, ORANGE, PURPLE, or GRAY. Default is BLACK color.

Sample:

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

Print / display:

```
[Barcode]
```
2D QR Code Barcode by Drawing

Function

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

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

Syntax

```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,  
);
```

Micro QR Code for short messages:

```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:

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

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

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

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

Sample:

```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:
```

![QR Code Image]
Barcode (Linear) by Drawing

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

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

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

Syntax

```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:

- **CDB2OF7** - AIM USS-Codabar, Codabar 2-of-7
- **CODE11** - Code 11
- **CODE128** - CODE 128, A, B, and C auto-switching mode
- **CODE128B** - CODE 128, Set B, for suppress mode C in favor of mode B
- **CODE32** - Code 32, up to 8 digits
- **CODE39E** - Code 39 (3 of 9) Extended (full text)
- **CODE93** - Code 93
- **DL2OF5** - Data Logical Code 2 of 5
- **DPIDENT** - Deutsche Post Identcode, 11 digits
- **DPLEIT** - Deutsche Post Leitcode, 13 digits
- **EAN128** - EAN 128
- **EAN14** - EAN-14, 13 digits
- **IATA2OF5** - IATA Code 2 of 5
- **IND2OF5** - Industrial Code 2 of 5
- **ITF14** - ITF-14, 13 digits
- **ITL2OF5** - Interleaved Code 2-of-5
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGMARS</td>
<td>LOGMARS</td>
</tr>
<tr>
<td>MAT2OF5</td>
<td>Matrix Code 2-of-5</td>
</tr>
<tr>
<td>MSI</td>
<td>MSI Plessey</td>
</tr>
<tr>
<td>PHARMA</td>
<td>Pharmacode One-Track</td>
</tr>
<tr>
<td>PLESSEY</td>
<td>PLESSEY (an older code still popular in some industries)</td>
</tr>
<tr>
<td>TELEPENA</td>
<td>Telepen Alpha</td>
</tr>
<tr>
<td>TELEPENN</td>
<td>Telepen Numeric</td>
</tr>
<tr>
<td>APOST</td>
<td>Australia Post Standard customer, allows 8 Digits, 8 digits followed by 5 characters, 16 digits, 8 digits followed by 10 characters, 23 digits</td>
</tr>
<tr>
<td>APOSTRD</td>
<td>Australia Post Redirection, 8 digits</td>
</tr>
<tr>
<td>APOSTRP</td>
<td>Australia Post Reply Paid, 8 digits</td>
</tr>
<tr>
<td>APOSTRT</td>
<td>Australia Post Routing, 8 digits</td>
</tr>
<tr>
<td>DPOST</td>
<td>Dutch Post KIX, 11 characters</td>
</tr>
<tr>
<td>JPOST</td>
<td>Japan Postal barcode</td>
</tr>
<tr>
<td>KPOST</td>
<td>Korea Postal barcode</td>
</tr>
<tr>
<td>POSTNET</td>
<td>PostNet</td>
</tr>
<tr>
<td>PLANET</td>
<td>Planet</td>
</tr>
<tr>
<td>RM4SCC</td>
<td>Royal Mail 4 State</td>
</tr>
<tr>
<td>SPOST4</td>
<td>Singapore Postal 4-state barcode</td>
</tr>
<tr>
<td>USPS4S</td>
<td>USPS 4-state postal barcode, 20 digits, 5, 9, or 11 digits zip code can be appended using the dash (-) character</td>
</tr>
<tr>
<td>EAN</td>
<td>EAN, EAN-2/EAN-5/EAN-8/EAN-13, 2, 5, 7, and 12 digits, EAN-2 2 digits or EAN-5 5 digits can be appended using the plus (+) character</td>
</tr>
<tr>
<td>UPCA</td>
<td>UPC-A, 11 digits, EAN-2 2 digits or EAN-5 5 digits can be appended using the plus (+) character</td>
</tr>
<tr>
<td>UPCE</td>
<td>UPC-E, 6 digits, also 7 digits starting with 1, EAN-2 2 digits or EAN-5 5 digits can be appended using the plus (+) character</td>
</tr>
</tbody>
</table>

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

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

**width, height**
The width and height of barcode dimension.

**degree**
The rotation for the barcode. The valid values are:
- DEG0: The barcode is not rotated.
- DEG90: The barcode is rotated 90 degrees clockwise.
- EG180: The barcode is rotated 180 degrees clockwise.
- DEG270: The barcode is rotated 270 degrees clockwise.

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

```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:
```

```
!
```
BCOCA Barcode (Linear)

Function

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

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

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

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

Note: BCOCA linear barcode requires appropriate printer microcode support.

Syntax

```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    oca_color = BLACK,
    ushort    cmr_id = 0,
    ushort    process_mode = AUDIT
);
```

Parameters

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

- **APOST**: Postal barcode for Australia
- **CDB2OF7**: AIM USS-Codabar, Codabar 2-of-7
- **CODE128**: CODE 128, AIM USS-128
- **CODE93**: Code 93
- **EAN13**: EAN-13 (includes JAN-standard)
- **EAN2SUP**: EAN Two-digit supplemental
EAN5SUP  EAN Five-digit supplemental
EAN8     EAN-8 (includes JAN-short)
IND2OF5  Industrial 2-of-5
ITL2OF5  Interleaved 2-of-5, AIM USS-I 2/5
JPOST    Postal 4-State barcode for Japan
MAT2OF5  Matrix 2-of-5
MSI      Modified Plessey
POSTNET  POSTNET
RM4SCC   Royal Mail 4 State
UPCA     UPC/CGPC Version A
UPCE     UPC/CGPC Version E
UPC2SUPP UPC - two digit supplemental
UPC5SUPP UPC - five digit supplemental
USPS4S   USPS Intelligent 4-State barcode for USA postal, 20, 25, 29 or 31 digits required

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

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

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

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

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

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

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

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

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

fontid
The ID number of the font to be used when HRI (human-readable interpretation) is requested. Specify DEFAULT to use your device's default font.
Some bar code types have specific requirements for the type of HRI font used, like the UPC and EAN symbologies specify OCR-B for HRI, and some bar code types do not allow HRI at all, for example, Japan Postal barcode, POSTNET, and RM4SCC, where this field is ignored.

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

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

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

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

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

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

**process_mode**
Specifies the processing mode for the CMR:

<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...</td>
</tr>
</tbody>
</table>
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

cchar *data = "1234567890";
OpenPage(8.5,11);
BBBarCode(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</td>
<td>0123456789</td>
<td>Symbology: 8 digits</td>
</tr>
<tr>
<td>Modifier <code>X01</code></td>
<td></td>
<td>BCOCA range: 8 digits</td>
</tr>
<tr>
<td>Australia Post Bar</td>
<td>0123456789</td>
<td>Symbology: 8–16 digits</td>
</tr>
<tr>
<td>Modifier <code>X02</code></td>
<td></td>
<td>BCOCA range: 8–16 digits</td>
</tr>
<tr>
<td>Australia Post Bar</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz (space), # (number sign)</td>
<td>Symbology: 8–13 characters</td>
</tr>
<tr>
<td>Modifier <code>X03</code></td>
<td></td>
<td>BCOCA range: 8–13 characters</td>
</tr>
<tr>
<td>Australia Post Bar</td>
<td>0123456789 for sorting code 0–3 for customer information</td>
<td>Symbology: 8–24 digits</td>
</tr>
<tr>
<td>Modifier <code>X04</code></td>
<td></td>
<td>BCOCA range: 8–24 digits</td>
</tr>
<tr>
<td>Australia Post Bar</td>
<td>0123456789</td>
<td>Symbology: 8–23 digits</td>
</tr>
<tr>
<td>Modifier <code>X05</code></td>
<td></td>
<td>BCOCA range: 8–23 digits</td>
</tr>
<tr>
<td>Australia Post Bar</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz (space), # (number sign)</td>
<td>Symbology: 8–18 digits</td>
</tr>
<tr>
<td>Modifier <code>X06</code></td>
<td></td>
<td>BCOCA range: 8–18 digits</td>
</tr>
<tr>
<td>Australia Post Bar</td>
<td>0123456789 for sorting code 0–3 for customer information</td>
<td>Symbology: 8–39 digits</td>
</tr>
<tr>
<td>Modifier <code>X07</code></td>
<td></td>
<td>BCOCA range: 8–39 digits</td>
</tr>
<tr>
<td>Australia Post Bar</td>
<td>0123456789</td>
<td>Symbology: 8 digits</td>
</tr>
<tr>
<td>Modifier <code>X08</code></td>
<td></td>
<td>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 -$/+% (space) A total of 43 valid characters</td>
<td>Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>Code 93</td>
<td>0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ -$/+% space character a – representing Shift 1 b – representing Shift 2 c – representing Shift 3 d – representing Shift 4</td>
<td>Symbology: unlimited. BCOCA range: 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>EAN-8 (includes JAN-short)</td>
<td>0123456789</td>
<td>7 characters</td>
</tr>
<tr>
<td>EAN-13 (includes JAN-standard)</td>
<td>0123456789</td>
<td>12 characters</td>
</tr>
<tr>
<td>EAN Two-digit</td>
<td>0123456789</td>
<td>2 characters for Modifier <code>X00</code></td>
</tr>
<tr>
<td>Supplemental</td>
<td></td>
<td>14 characters for Modifier <code>X01</code></td>
</tr>
<tr>
<td>Symbology</td>
<td>Example Data</td>
<td>Details</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EAN Five-digit Supplemental</td>
<td>0123456789</td>
<td>5 characters for Modifier X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 characters for Modifier \X'01'</td>
</tr>
<tr>
<td>Industrial 2-of-5</td>
<td>0123456789</td>
<td>Symbology: unlimited. BCOCA 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>Symbology: unlimited. BCOCA 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</td>
<td>ABCDEFGHUKLM NOPQRSTUVWXYZ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- (hyphen)</td>
</tr>
<tr>
<td>Japan Postal 4-State Bar Code (Modifier \X'01')</td>
<td>0123456789</td>
<td>CC1,CC2,CC3,CC4,CC5,CC6,CC7,CC8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- (hyphen), start, stop</td>
</tr>
<tr>
<td>No length checking done.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix 2-of-5</td>
<td>0123456789</td>
<td>Symbology: unlimited. BCOCA 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>3 to 15 characters for Modifier \X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 to 14 characters for Modifier \X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 to 13 characters for all other modifiers</td>
</tr>
<tr>
<td>POSTNET</td>
<td>0123456789</td>
<td>5 characters for Modifier X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 characters for Modifier \X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 characters for Modifier \X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BCOCA range for Modifier \X'03': 0 to 50 characters, some printers may support a larger data length.</td>
</tr>
<tr>
<td>Royal Mail (RM4SCC, modifier X'00')</td>
<td>0123456789</td>
<td>ABCDEFGHUKLM NOPQRSTUVWXYZ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Royal Mail (Dutch KIX variation, modifier \X'01')</td>
<td>0123456789</td>
<td>ABCDEFGHUKLM NOPQRSTUWXYZ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>abcdefghijklmnopqrstuvwxyz</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>2 characters for Modifier X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 characters for Modifier \X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 characters for Modifier \X'02'</td>
</tr>
<tr>
<td>UPC Five-digit Supplemental Paperbacks)</td>
<td>0123456789</td>
<td>5 characters for Modifier X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 characters for Modifier \X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 characters for Modifier \X'02'</td>
</tr>
<tr>
<td>USPS 4-State</td>
<td>0123456789</td>
<td>20 digits for modifier X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25 digits for modifier \X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29 digits for modifier \X'02'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31 digits for modifier \X'03'</td>
</tr>
</tbody>
</table>
BCOCA MaxiCode 2D Barcode

Function

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

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

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

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

Syntax

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

Parameters

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

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

**symbol_mode**
Symbol mode, Valid values are:

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

**zipper**
Specifies whether or not a vertical zipper-like test pattern and contrast block is to be printed to the right of the symbol. The zipper provides a quick visual check for printing distortions. If the bar code is rotated, the zipper and block are rotated along with the symbol. Default is FALSE, a zipper pattern is not printed.
noESC
Specifies whether a backslash character `\' within the bar code data should be treated as an
escape sequence or not. Specify FALSE if the backslash should be treated as an escape, an
escape sequence is useful if you need to encode control characters like Carriage Return into
the barcode. The default value is TRUE, each backslash character within the bar code data is
treated as character data.

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

<table>
<thead>
<tr>
<th>Degree</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>

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

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

process_mode
Specifies the processing mode for the CMR:

<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 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);

BMAsiCode(data,       // Barcode data
         1,         // Barcode x position to 1"  
         1);        // Barcode y position to 1"

ClosePage();
CloseDoc();
```
BCOCA PDF417 2D Barcode

Function

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

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

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

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

Syntax

```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         oca_color = BLACK,
    ushort        cmr_id = 0,
    ushort        process_mode = AUDIT,
    char*         macro = NULL
);`
```

Parameters

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

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

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

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

rowsize
The number of data symbol characters per row. Each row consists of a start pattern, a left row indicator codeword, 1 to 30 data symbol characters, a right row indicator
codeword, and a stop pattern. The number of rows times the number of data symbol characters per row cannot exceed 928. The default value is 10.

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

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

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

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

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

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

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

**process_mode**
Specifies the processing mode for the CMR:

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

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

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

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

**macro**

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

**Sample**

```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();
```
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.

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

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

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

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

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

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

process_mode
Specifies the processing mode for the CMR:

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

INSTR

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

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

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

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

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

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

Sample

```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 of Index Name Group of Input AFP (Checking)

Function

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

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

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

Syntax

Bool Bng(void);

Parameters

None.

Sample

None.
Begin Index Group

Function

Begins an index page group.

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

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

Syntax

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

Parameters

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

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

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

Sample

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

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

    $MaxPaging = 50;  // Maximum paging is up to 50 pages
    SetUnit(IN_U600);  // Set default unit to inch
    Start();         // Start initiation, open default input, output and definition files, retrieves
```
Encoding("ibm-037","ibm-437");

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

while ($Edt == 0) // Until end of AFP document
{
    getField(660, 1080, custName); // Get customer name
    getField(4050, 900, mobileNo); // Get customer mobile number
    do {
        $Page++; // Point to next AFP page buffer
        getPage(); // Get next page
        // detecting if it is the first page of a group,
        // overlay O1OVL1E only used by at first page of
        // each page group
        bog = TriggerOvly("O1OVL1E");
    } while (!bog && !$Edt); // Until beginning of next page group or End of AFP file
    bog = 0; // Reset it for next group
    // Now got all pages of a page group and first page of next group, now it is ready to process new AFP output
    if (!$Edt) // If not end of AFP document
    
        grpPages = $Page -1; // Keep total number of pages per group, need to minus 1 page of the first page of next group
        sprintf(tmp, "%08d", ++groups); // Auto-converts ASCII to EBCDIC for indexes
        BgnIdx(tmp);
        PutIdx("Customer Name", custName);
        PutIdx("Mobile Number", mobileNo);
    
    for (i = 0; i < grpPages; i++)
    {
        $Page = i + 1; // Point to page buffer number to be opened
        sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
        BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90); // Add 1D barcode
        ClosePage(); // End of AFP page, write to AFP file
    }
    
    EndIdx(); // End of group level index

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

CloseDoc(); // End of AFP document, close AFP output
return 0;
}
Begin Overstrike

**Function**

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

**Syntax**

```c
void BgnOstrike( );
```

**Parameters**

No parameter to be specified.

**Sample**

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
::
::
BgnOstrike(); // begin overstriking of text
Ltxt("This is an overstrike text"); // text will be overstriked
::
::
EndOstrike() ; // end overstriking of text
::
ClosePage();
CloseDoc();
```
Begin Underscore

Function

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

Syntax

void BgnUscore( );

Parameters

No parameter to be specified.

Sample

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

BgnUscore(); // begin underscoring of text

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

EndUscore() // end underscoring of text

ClosePage();
CloseDoc();
Blank Page (Checking)

Function

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

Syntax

    bool BlankPage(void);

Parameters

No parameter to be specified.

Sample

    SetUnit(IN_U600);
    OpenDoc();
    GetPage();         // Read-in an AFP page
    :
    :
    if ( !BlankPage() )  // Output this AFP page if it is not a blank
        ClosePage();    // page
    :
    :
    CloseDoc();
Box Drawing

Function

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

Syntax

```plaintext
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

```plaintext
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Box(1,1,5,2,0.02);  // Draw box from (1",1"), size 5 x 2", red line thickness is 0.02"
Box(1,5,5,1,0.01);  // Draw box from (1",5"), size 5 x 1", blue line thickness is 0.01"
:
:
ClosePage();
CloseDoc();
```
Center Align 1-Byte Text

Function

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

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

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

Syntax

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

Parameters

data
The NULL-terminated ASCII data string.

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

Sample

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

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

Pos(2,2);  // current position at (2",2")
Ctxt("text is center aligned"); // Center text at (2",2")

ClosePage();
CloseDoc();
```
Center Align Japanese

Function

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

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

Syntax

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

Parameters

data
The NULL-terminated SJIS data string.

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

Sample

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

ClosePage();
CloseDoc();
```
Center Align Korean

Function

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

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

Syntax

```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 at 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 at (2",2")
Ckr("IBM 소프트웨어 솔루션"); // Center KSC text at
    :                 // (2",2")
    :
ClosePage();
CloseDoc();
```
Center Align Simplified Chinese

Function

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

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

Syntax

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

Parameters

data
The NULL-terminated GBK data string.

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

Sample

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

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

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

Syntax

```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 at 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 at (2",2")
Ctc("實現Win2000與Linux的雙領導");  // Center BIG5 text at (2",2")

:  

ClosePage();
CloseDoc();
```
**Center Align SBCS-HOST/DBCS-HOST**

**Function**

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

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

**Syntax**

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

**Parameters**

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

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

**Sample**

```cpp
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    
    Font2(3,4); // assume font 3 is EBCDIC font, // and font 4 is DBCS-HOST font
    
    Pos(2,2); // current position at (2",2")
Cdbcs("实现 Win2000 与 Linux 的双引导"); // Center DBCS text at (2",2")
    
ClosePage();
CloseDoc();
```
Center Align UTF-16 Text

Function

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

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

Syntax

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

Parameters

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

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

Sample

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

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
  
  Pos(2,2);                      // current position at (2",2")
  Font(2);                          // Assume font 2 is a TrueType font
                                  // with data type UTF16BE defined
  Cu16(data1);                    // center put UTF-16 at (2",2")
  
ClosePage();
CloseDoc();
```
Center Align UTF-16 Text Converting from Legacy String

Function

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

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

Syntax

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

Parameters

data
The NULL-terminated legacy codepage string.

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

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

Sample

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

DefaultCode("GB18030");   // set default codepage of input data
OpenPage(8.5,11);
    :  
Pos(2,2);                 // set current position at (2",2")
Font(2);                  // Assume font 2 is a TrueType font
    :  // with data type UTF16BE defined
Cu16c("test 测试");     // center put UTF-16 converting from
    :  // Chinese GB18030
       
ClosePage();
CloseDoc();
```
Center Align UTF-8 Text

Function

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

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

Syntax

```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 at 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\xaf\x95";

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);                         // current position at (2",2")
Font(2);                          // Assume font 2 is a TrueType font
// with data type UTF8 defined
Cu8(data1);                       // center put UTF-8 at (2",2")
:
:
ClosePage();
CloseDoc();
```
Center Align UTF-8 Text Converting from Legacy String

Function

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

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

Syntax

```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 MakeAFP document *Encoding Names* for more details about the available names.

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

Sample

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

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

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

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

ClosePage();
CloseDoc();
```
Center Align UTF-8 Text Converting from UTF-16LE

Function

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

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

Syntax

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

Parameters

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

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

Sample

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

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    :
    :
Pos(2,2); // current position to (2",2")
Font(2);  // Assume font 2 is a TrueType font
          // with data type UTF8 defined
Cu8u(data); // center put UTF-8 converting from
            // UTF16-LE
    :
    :
ClosePage();
CloseDoc();
```
Centimeter Value

Function
   Specifies a value in centimeters.

Syntax
   float cm(
       float value
   );

Parameters
   value
       The value in centimeters.

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

Function

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

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

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

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

Syntax

```c
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

```c
SetUnit(IN_U600);

OpenDoc();

OpenPage(8.5,11);
   :
   :
ClosePage();
CloseDoc();```
Close Page

Function

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

With MakeAFP, you can open multiple pages by either the “Open Page” or the “Get Page” function requests, and then process different pages in an interleaved manner once each page is initialized, all the entire MO:DCA data stream will be kept in memory buffers in page-level, and only to be written to one of the AFP document files you opened until the page is closed with the “Close Page” function.

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

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

Syntax

```cpp
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

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

Function

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

Syntax

For OCA color:

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

For RGB color:

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

For CYMK color:

```c
ColorCMYK(
    UCHAR cyan_color_percentage,
    UCHAR yellow_color_percentage,
    UCHAR magenta_color_percentage,
    UCHAR black_color_percentage
);
```

Parameters

ocacolor

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

RGB values

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

CYMK color percentage values

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

Sample

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

Pos(5,5);                             // current position at (5,5) mm
ColorRGB(255,0,0);                   // RGB red color
Ltxt("RGB Red Color Text");
Pos(5,10.);                           // current position at (5,10) mm
ColorCMYK(0,0,0,100);                // CYMK black color
Ltxt("CMYK Black Color Text");
Pos(5,15.);                           // current position at (5,15) mm
Color(CYAN);                          // AFP OCA CYAN color
Ltxt("AFP OCA CYAN Color Text");

ClosePage();
CloseDoc();
```
**Color Management Resource Association**

**Function**

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

This function can be repeated to associate all CMRs required.

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

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

**Syntax**

Invokes CMR Association:

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

Revokes CMR Association:

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

**Parameters**

**cmr_id**

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

**process_mode**

Specifies the processing mode for the CMR:

<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 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();

RevokeCMR();     // revoke CMR association
CloseDoc();
```
Copy Group

Function

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

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

Syntax

void CopyGroup(
    char * copygroup_name,
    ushort docNo
);

Parameters

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

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

Sample

SetUnit(MM_U600);
OpenDoc();
    :
    :
OpenPage(210,297);
    :
ClosePage();
OpenPage(210,297);
CopyGroup("F2TRAY2");  // Call copy group F2TRAY2 you defined in form definition, for use the paper from input paper tray 2 for this page and subsequence pages
    :
ClosePage();
    :
CloseDoc();
Default Language Locale

Function

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

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

Syntax

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

Parameters

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

  Refer to MakeAFP document *How to specify a Locale* for more details about the locale names.
Encoding of AFP and PC Native

Function

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

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

Syntax

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

Parameters

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

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

Sample

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

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

    $MaxPaging = 50;            // Maximum paging is up to 50 pages
    SetUnit(IN_U600);              // Set default unit to inch
    Start();                    // Start initiation, open default input,
    // output and definition files, retrieves
    // AFP resources, allocate memory
    Encoding("ibm-037","ibm-437");   // AFP – CP037, PC - CP437
    OpenDoc();                  // Open an AFP document

    while ($Edt == 0)           // Until end of AFP document
    {
        $Page = 0;                // Reset AFP page buffer number
        do {
```
$Page++;    // Point to next AFP page buffer
GetPage();  // Get a page from existing AFP file
} while ($Eng == 0);  // Until end of each page group

// Now got all pages of a page group, now it is
// ready to compose the new AFP output
grpPages = $Page;  // keep total number of pages per group
for (i = 0; i < grpPages; i++)
{
    $Page = i + 1;    // Point to page buffer number to be opened
    again
    InclPseg("S1OWL", 0.3, 0.25);  // Add a new page segment image
    MaskArea(5, 0.4, 2, 0.75);  // Mask an area on every page
    sprintf(tmp, "Page %d of %d", $Page, grpPages);  // Generate pagination
    Font(1);  Pos(8, 0.45);  Rtxt(tmp);
    sprintf(tmp, "%06d", ++pageSN);  // generate page serial number
    Font(2);  Pos(0.2, 10.8);  Ltxt(tmp);  // With MakeAFP Weaver, you can
    // use a
    font encoded in ASCII
    GetIdx("Policy", policyNo);  // MakeAFP Weaver does auto-
    // conversion with the
    // encoding names defined with
    // Encoding() function
    sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);
    BarCode(CODE128, tmp, 0.3, 2, 0.2, DEG90);  // Add 1D barcode 128
    DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);  // Add 2D DataMatrix
    ClosePage();  // Close AFP page, write each page to AFP file
}
CloseDoc();  // Close AFP document and its file
#endif
#if _DEBUG
ViewAFP();  // Only view AFP output in debug mode
#endif
return 0;
End Index Group

**Function**

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

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

**Syntax**

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

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

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

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

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

$Page = 1; // Reset AFP page buffer number to 1 // for the next customer statement
End of Document File of Input AFP (Checking)

Function

Tests for end of document of input AFP file.

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

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

Syntax

Bool Edt(void);

Parameters

None.

Sample

None.
**End of Index Name Group of Input AFP (Checking)**

**Function**

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

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

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

**Syntax**

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

**Parameters**

None.

**Sample**

None.
End Overstrike

Function

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

Syntax

void EndOstrike();

Parameters

No parameter to be specified.

Sample

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

BgnOstrike();  // begin overstriking of text

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

EndOstrike();  // end overstriking of text

ClosePage();
CloseDoc();
End Underscore

Function

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

Syntax

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

Parameters

No parameter to be specified.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
BgnUscore();          // begin underscoring of text
Ltxt("This is an underscore text"); // text will be underscored
:
:
EndUscore();        // end underscoring of text
:
ClosePage();
CloseDoc();
```
Font Definition

Function

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

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

Syntax

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

```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 Weaver definition file with FONT parameter.

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

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

Sample

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

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

**AFP fonts’ calls in the MakeAFP Weaver program:**

```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
Pos(4.5, 2.5);
Font(times14);            // use font times14 for the subsequent texts
Lttx("Testing text");
Pos(6, 4);
Font2(helv11, song11);   // use fonts helv11 and song11 for the
                         // subsequent SBCS-HOST/DBCS-HOST output
```
Csc2("实现 Win2000 与 Linux 的双引导");
Font ID Query

**Function**
Queries the current font ID number.

**Syntax**

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

**Sample**

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.27, 11.67);
if ( FontID() == 2 )  // if current Font ID is 2
{
    ...
}
else
{
    ...
}
ClosePage();
CloseDoc();
```
Get Index Value

Function

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

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

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

Syntax

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

Parameters

**Index_name**

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

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

**Index_value**

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

Sample

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

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

    $MaxPaging = 50;            // Maximum paging is up to 50 pages
    SetUnit(IN_U600);              // Set default unit to inch
    Start();                    // Start initiation, open default input,
                          // output and definition files, retrieves
                          // AFP resources, allocate memory

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

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

do {
    $Page++;                // Point to next AFP page buffer
    GetPage();              // Get a page from existing AFP file
} while ($Eng == 0);      // Until end of each page group

// Now got all pages of a page group, now it is
// ready to compose the new AFP output
grpPages = $Page;         // keep total number of pages per group
for (i = 0; i < grpPages; i++) {
    $Page = i + 1;          // Point to page buffer number to be opened again
    InclPseg("S1OWL", 0.3, 0.25);  // Add a new page segment image
    MaskArea(5, 0.4, 2, 0.75); // Mask an area on every page
    sprintf(tmp, "Page %d of %d", $Page, grpPages); // Generate pagination
    Font(1);  Pos(8, 0.45);  Rtxt(tmp);     // With MakeAFP Weaver, you can
    // use a
    // font encoded in ASCII
    sprintf(tmp, "%06d", ++pageSN);  // generate page serial number
    Font(2);  Pos(0.2, 10.8);  Ltxt(tmp);
    GetIdx("Policy", policyNo);       // MakeAFP Weaver does auto-
    // conversion with the
    // encoding names defined with
    // Encoding() function
    sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);
    BarCode(CODE128, tmp, 0.3, 2, 2, 0.2, DEG90);  // Add 1D barcode 128
    DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);           // Add 2D DataMatrix
    ClosePage();            // Close AFP page, write each page to AFP file
}

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

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

return 0;
Get No-Operation Value

Function

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

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

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

Syntax

Char *GetNop(char *nop_value);

Parameters

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

Sample

Char nop_value1[512], nop_value2[512];
SetUnit(IN_U600);
$MaxPaging = 100;          // Maximum page buffers are 100, must be defined before Start() function call
Start();                    // Start a MakeAFP Weaver session
OpenDoc();

$Page = 3;                  // Get AFP page into page buffer 3
GetPage();
GetNop(nop_value1);        // Get first NOP value from page 3
GetNop(nop_value2);        // Get second NOP value from page 3

ClosePage();               // Close page 3 and write to AFP output file

$Page = 15;                 // Get AFP page into page buffer 15
GetPage();

ClosePage();               // Close page 15 and write to AFP output file
Get Page – Getting an Existing AFP Page

Function

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

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

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

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

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

Syntax

void GetPage(
    bool   remove_imm = false
);  

Parameters

remove_imm

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

Sample

SetUnit(IN_U600);
$MaxPaging = 100;          // Maximum page buffers are 100, must be
                          // defined before Start() function call
Start();                  // Start a MakeAFP Weaver session
OpenDoc();                :
$Page = 3;                 // Get AFP page into page buffer 3
GetPage();                :
ClosePage();             // Close page 3 and write to AFP output
file                      :
$Page = 15;              // Get AFP page into page buffer 15
GetPage();              :

ClosePage();  // Close page 15 and write to AFP output
file
Get Text Field by a Location

Function

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

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

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

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

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

Syntax

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

Parameters

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

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

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

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

Sample

```
/**************************************************************************/
/* This sample shows how to detect a trigger by an overlay name and     */
/* capture data fields from page 1, add barcode to existing AFP         */
/* AFP was encoded in CP-037, USA EBCDIC                               */
/**************************************************************************/
int main( )
```
unsigned int i, grpPages, pageSN = 0;
char tmp[80], mobileNo[20];
bool bog = 0;

$MaxPaging = 50;        // Maximum paging is up to 50 pages
SetUnit(IN_U600);       // Set default unit to inch
Start();                // Start initiation, open default input,
                        // output and definition files, retrieves
                        // AFP resources, allocate memory

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

OpenDoc();              // Open AFP document

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

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

while ($Edt == 0)       // Until end of AFP document
{
    GetField(4050, 900, mobileNo);   // Get customer mobile number
    do {
        $Page++;            // Point to next AFP page buffer
        GetPage();          // Get next page
        // Detecting if it is the first page of a group,
        // overlay O1OVL1E only used by at first page of
        // each page group
        bog = TriggerOvly("O1OVL1E");
        } while (!bog && !$Edt);  // Until beginning of next page group or end
                        // of
                        // AFP file
        bog = 0;                  // reset it for next group
        // Now got all pages of a page group and first page of next group, now
        // is ready to compose the new AFP output
        if (!$Edt)                // If not end of AFP document
            grpPages = $Page -1 ;   // keep total number of pages per group, need
                        // minus 1 page of the first page of next
                        // group
        for (i = 0; i < grpPages; i++)
        {
            $Page = i + 1;          // point to page buffer number to be opened
            if (*mobileNo)
            {
                sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
                BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90);
            }
        }
    ClosePage();              // Close AFP page, write to AFP file
}
MovePage(1, grpPages + 1);  // As we got first page of next group
                     // now need to move its contents to page
                     // buffer 1 for the next page group

$Page = 1;

CloseDoc();         // Close AFP document and its file
return 0;
}
**Get Text Field by a Location Area**

**Function**

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

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

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

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

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

**Syntax**

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

**Parameters**

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

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

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

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

**Sample**

```c
/****************************************************************************/
/* This sample shows how to capture a trigger from last page of each group, */
/* */
```
/* get a field from page 1 for add a barcode, mask an area and add a page */
/* segment. */
/* */
/* */
/* AFP was encoded in CP-037, USA EBCDIC */
*/

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

    $MaxPaging = 50;  // Maximum paging is up to 50 pages
    SetUnit(IN_U600);  // Set default unit to inch
    Start();  // Start initiation, open default input, output and definition files, retrieves AFP resources, allocate memory

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

    while ($Edt == 0)  // Until end of AFP document
    {
        $Page = 0;  // Reset AFP page buffer number
        do {
            $Page++;  // Point to next AFP page buffer
            GetPage();  // Get a page from existing AFP file
            if ($Page == 1)  // Get policy number from page 1
                GetField2(2448, 2448, 6080, 6110, policyNo);

            if ($Page > 2)
                eog = Trigger(3744, 2338, "Part 1");  // detecting if it is a last page of a page group, "Part 1" text
                // string only appears at last page
            
        } while (!eop);  // Until end of each page group
        eop = 0;  // reset it for next group

        grpPages = $Page;  // keep total number of pages per group
        for (i = 0; i < grpPages; i++)
        {
            $Page = i + 1;  // point to page buffer number to be opened again

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

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

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

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

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

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

return 0;
}
Get Text Field by a Location Area and a Pattern

Function

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

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

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

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

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

Syntax

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

Parameters

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

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

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

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

- '@' A single alphabetic character (A to Z or a to z)
- '#' A single numeric character (0 to 9)
- '&' A single alphabetic or numeric character
- '+' A single blank or numeric character
- '=' A single blank or alphabetic character
- '~' A single non-blank character
- '?' Any single character
To suppress the special syntactic significance of any "@#&+?~',", and match the character exactly, precede it with a "\" (backslash).

Sample

/**************************************************************************/
/* This sample shows how to capture a trigger and field from page 1 only, */
/* for adding barcode to existing AFP. AFP was encoded in CP-437, ASCII */
/**************************************************************************/
int main( )
{
    unsigned int i, grpPages, pageSN = 0;
    char tmp[80], savingsNo[20];
    bool bog = 0;
    $MaxPaging = 50;        // Maximum paging is up to 50 pages
    SetUnit(IN_U600);       // Set default unit to inch
    Start();                // Start initiation, open default input, output and definition files, retrieves AFP resources, allocate memory
    OpenDoc();              // Open AFP document
    $Page = 1;              // Set AFP page buffer number to 1 for the first page
    // of AFP file
    GetPage();              // Get first page of AFP file
    while ($Edt == 0)       // Until end of AFP document
    {
        // Get Savings A/C number from page 1 by a mask
        GetField3(180, 180, 1080, 1120, savingsNo, "###-####-####");
        do {
            $Page++;            // Point to next AFP page buffer
            GetPage();          // Get next page
            // detecting if it is the first page of a group,
            // "Page 1 of" text string only appears at
            // first page of each page group
            bog = Trigger2(2187, 2187, 1120, 1200, "Page 1 of");
        } while (!bog && !$Edt);  // Until beginning of next page group or end of AFP file
        bog = 0;                  // Reset it for next group
        // Now got all pages of a page group and first page of next group, now it is ready to compose new AFP output
        if (!$Edt)
            grpPages = $Page -1 ;   // keep total number of pages per group, need to minus 1 page of the first page of next group
        for (i = 0; i < grpPages; i++)
        {
            $Page = i + 1;          // Point to page buffer number to be opened again
            sprintf(tmp, "%d %d %s", ++pageSN, $Page, savingsNo);
            DataMatrix(tmp, 0.2, 2.2, 0.4, 0.4);
            ClosePage();            // Close AFP page, write to AFP file
MovePage(1, grpPages + 1); // As we got first page of next group previously,
// now need to move its contents to page buffer

$page = 1; // for next page group

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

return 0;
Get Text Field by a X-location Range and a Pattern

**Function**

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

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

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

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

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

**Syntax**

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

**Parameters**

**x1, x2**

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

**field_value**

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

**pattern**

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

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

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

**Sample**
None.
Get Text Field by a Y-location Range and a Pattern

Function

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

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

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

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

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

Syntax

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

Parameters

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

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

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

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

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

Sample
None.
Get Text Field Position

Function

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

Syntax

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

Parameters

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

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

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

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

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

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

Sample

```cpp
int main()
{
    UINT grpPages;
    USHORT x_found = 0, y_found = 0;
    bool eog = 0;
    char account[20];
    $MaxPaging = 100; // Maximum paging is up to 100
```
SetUnit(IN_U600);  // Set default unit to INCH
Start();  // Start initiation, open default input,
    // output and definition files, retrieves
    // AFP resources, allocate memory
OpenDoc();  // Open AFP document
while ($Edt == 0)  // Process until end of AFP document
{
    $Page = 0;
    do {
        $Page++;            // Point to next AFP page buffer
        getPage();  // Get a page from existing AFP file
        if ($Page == 1)
        {
            GetField(443, 447, account, 1);  // Get A/C number from page 1
            Box(0.7, 1.3, 2.5, 1.23, 0.007);  // draw a box around the address
            // Add 1d and 2D barcode
            BarCode(CODE128, account, 8.77, 2.5, 1.5, 0.37, DEG90);
            QRCode(account, 8.4, 1);
        }
        else
        {
            if (GetFieldPos(136, 136, 700, 2000, "Subtotal", x_found, y_found))
            {
                Vline(0.324, 2.1, (float )y_found/300 - 2, 0.007);  // AFP is 300 PELS
                Vline(8.04, 2.1, (float )y_found/300 - 2, 0.007);
                Hline(0.324, (float )y_found/300.0 + 0.1, 7.72, 0.007);
            }
        }
    } while ($Eng == 0);  // Until end of each indexed page group
    // Now got all pages of a page group, now it is
    // ready to compose the new AFP output
    grpPages = $Page;  // keep total number of pages per group
    for ($Page = 1; $Page <= grpPages; $Page++)
        closePage();  // Closeof each AFP page, write to AFP file
}
CloseDoc();  // Close AFP document and its file
#if defined DEBUG
    ViewAFP();  // Only view AFP output in debug mode
#endif
return 0;
Goto Page

Function

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

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

Syntax

```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()
Start();
OpenDoc();
:  GotoPage(3); // switch to page buffer 3
  OpenPage(8.5,11);
  ClosePage();
  :  GotoPage(15); // switch to page buffer 15
  OpenPage(8.5,11);
  ClosePage();
```
**Horizontal Line**

**Function**

Draws a horizontal line.

**Syntax**

```c
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**

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(220.297);
:
:
Color(RED); // defines color for the legacy line
Hline(10,10,100,1); // draws a horizontal blue line from (10,10)mm, its length is 100 mm, thickness is 1 mm
:
:
ClosePage();
CloseDoc();
```
Horizontal Lines

**Function**
Repeat drawing of horizontal lines.

**Syntax**
```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();
```
Inch Value

Function

Specifies a value in inches.

Syntax

    float inch(
        float    value
    );

Parameters

    value
    The value in inches.

Sample

    SetUnit(MM_U600);
    OpenDoc();
    OpenPage(220.297);
    :
    :
    Pos(10,10);                   // set X and Y position to (10,10)mm
    :
    :
    Pos( inch(2), 35);            // set X position to 2" and Y position to
    // 35mm
    :
    :
    ClosePage();
    CloseDoc();
Include Data-Object

Function

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

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

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

Syntax

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

Parameters

object_name

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

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

1. No filename extension
2. JPG
3. TIF
4. GIF
5. JP2
6. EPS
7. PDF
8. BMP
9. PCX
10. PCL
11. OBJ
Note: Some file extensions may not be supported by your AFP print server. Using legacy AFP object naming is recommended, which allows one to eight characters as the base filename. Your AFP print server may support the data-object resource file that has No filename extension or with extension .obj.

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

**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.

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

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

- **CENTER** Specifies that the center of the object is to be positioned at the center of the object placement area. Any portion of the object that falls outside the object placement area is trimmed.
- **LEFT** Specifies that the object is positioned at the upper, left-hand corner of the object placement area, an object that falls outside the object placement area as defined by the object_width & object_height parameters is not trimmed and could cause an exception condition by the IPDS printer.
- **FILL** Specifies that the center of the object is to be positioned coincident with the center of the object placement area. The object is then scaled so that it fills the object placement area in both the horizontal and vertical directions. This may require that the object be asymmetrically scaled by different scale factors in both horizontal and vertical directions.
- **FIT** Specifies scale to fit. The object is to be scaled to fit within the object placement area, as defined by the object_width & object_height parameters. The center of the object is placed in the center of the object placement area and the object is scaled up or down to fit the area. Scaling in the horizontal and vertical directions is symmetrical. This parameter ensures that the object is not being trimmed and presented in the object placement area with the largest possible size.
- **REPEAT** Specifies that the origin of the object is to be positioned with the origin of the object placement area. The object is then replicated in horizontal and vertical directions. If the last replicated data does not fit in the object area, it is trimmed to fit.
- **TRIM** Specifies position and trim. The object is positioned at the upper, left-hand corner of the object placement area. Any portion of the
object that falls outside the object placement area as defined by the object_width & object_height parameters is trimmed.

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

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

process_mode
Specifies the processing mode for the CMR:

<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. 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.</td>
</tr>
</tbody>
</table>

render_intent
Specify the rendering intent for the above object:

<table>
<thead>
<tr>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERCP</td>
<td>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.</td>
</tr>
<tr>
<td>SATUR</td>
<td>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.</td>
</tr>
<tr>
<td>PELCM</td>
<td>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...</td>
</tr>
</tbody>
</table>
different white points won't match colorimetrically but may match visually. This intent is typically used for vector graphics.

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

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

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

**object_color**
The color to be used as the default color or initial color for the object placement area. This parameter is used only for AFP objects of the PSEG, GOCA, BCOCA, and IOCA type. If the object type is non-AFP, this parameter is ignored. Colors specified must be one of the standard AFP OCA color, valid values are: 

- NONE
- DEFAULT
- BLACK
- BLUE
- BROWN
- GREEN
- RED
- PINK
- TURQ
- YELLOW
- DARKBLUE
- ORANGE
- PURPLE
- MUSTARD
- GRAY
- DARKGREEN
- DARKTURQ
- DARKCYAN

**Sample**
```plaintext
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 definstion file

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

ClosePage();
CloseDoc();
```
Include Overlay

**Function**

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

**Syntax**

```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("O1TEST02");          // Include an overlay at current position

ClosePage();
CloseDoc();
```
Include Page Segment

Function

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

Syntax

```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();
```
Left Align ASCII / EBCDIC Text

Function

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

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

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

Syntax

```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 at which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    
    Pos(2,2);                       // current position at (2",2")
    Font(3);                        // assume font 3 is ASCII font
    Ltxt("text is left aligned");   // left put text at (2",2")
    
    ClosePage();
    CloseDoc();
```
Left Align Japanese

Function

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

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

Syntax

```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 at which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    :
    :
Pos(2,2);                               // position at (2",2")
Font2(3,4);                               // assume font 3 is ASCII font,
    // and font 4 is SJIS font
Ljp("Alphabet が混在した文章のサンプルです"); // left put SJIS text at
    // (2",2")
    :
    :
ClosePage();
CloseDoc();
```
Left Align Korean

**Function**

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

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

**Syntax**

```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 at which the next character would be placed.

**Sample**

```cpp
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    :
    :
Pos(2,2);   // position at (2",2")
Font2(3,4); // assume font 3 is ASCII font,
            // and font 4 is KSC font
Lkr("IBM 소프트웨어 솔루션"); // left put KSC text
            // at (2",2")
    :
    :
ClosePage();
CloseDoc();
```
Left Align Simplified Chinese

**Function**

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

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

**Syntax**

```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 at 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 at (2",2")

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

```c
ClosePage();
CloseDoc();
```
Left Align Traditional Chinese

Function

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

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

Syntax

```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 at which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);                           // current position at (2",2")
Font2(3,4);                           // assume font 3 is ASCII font,
    // and font 4 is BIG5 font
Ltc("實現 Win2000 與 Linux 的双引導");      // left put BIG5 text at (2",2")
:
:
ClosePage();
CloseDoc();
```
Left Align SBCS-HOST/DBCS-HOST

Function

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

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

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

Syntax

```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 at which the next character would be placed.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    :
    :
Pos(2,2);   // current position at (2",2")
Font2(3,4); // assume font 3 is EBCDIC font,
            // and font 4 is DBCS-HOST font
Ldbcs("实现 Win2000 与 Linux 的双引导");  // left put DBCS text at (2",2")
    :
    :
ClosePage();
CloseDoc();
```
Left Align UTF-16 Text

Function

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

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

Syntax

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

Parameters

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

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

Sample

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

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);                      // current position at (2",2")
Font(2);                       // Assume font 2 is a TrueType font
    // with data type UTF16BE defined
Lu16(data1);                   // left put UTF-16 at (2",2")
:
:
ClosePage();
CloseDoc();
```
Left Align UTF-16 Text Converting from Legacy String

Function

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

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

Syntax

```c
void Lu16c(
    char*     data,
    char*     fromcode = NULL,
    bool     same_pos = TRUE
);
```

Parameters

data
The NULL-terminated legacy codepage string.

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

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

Sample

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

DefaultCode("GB18030");   // set default codepage of input data
OpenPage(8.5,11);
    :
Pos(2,2);       // set current position at (2",2")
Font(2);       // Assume font 2 is a TrueType font
    // with data type UTF16BE defined
Lu16c("test 测试");  // left put UTF-16 converting from
    // Chinese GB18030
    :
ClosePage();
CloseDoc();
```
Left Align UTF-8 Text
Function
Left aligns a single-line UTF-8 string at the current position.
Before calling this function, make sure the font ID you called with the “Font” function, was
defined with data type UTF8 by the FONT parameter in your MakeAFP definition file. Refer to
Chapter 3 for more details about how to define OpenType/TrueType fonts in AFP.

Syntax
void Lu8(
UChar8*
bool
);

data,
same_pos = TRUE

Parameters
data
The NULL-terminated UTF-8 string.
same_pos
Indicates whether the current position is updated at the end of this function. If this parameter
is set to TRUE, the current position remains at the origin position before this function is issued.
Otherwise, the current position is moved to the position at which the next character would
be placed.

Sample
/* UTF-8 string, "test" and CJK characters "测试" */
UChar8
data1[20] = "test\xe6\xb5\x8b\xe8\xaf\x95";
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);

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

Font(2);

// Assume font 2 is a TrueType font
// with data type UTF8 defined

Lu8(data1);

// left put UTF-8 at (2",2")
:
:

ClosePage();
CloseDoc();


Left Align UTF-16 Text Converting from Legacy String

Function

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

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

Syntax

```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 MakeAFP document Encoding Names for more details about the available names.

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

Sample

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

DefaultCode("GB18030");       // set default codepage of input data
OpenPage(8.5,11);
    :
    :
Pos(2,2);                        // set current position at (2",2")
Font(2);                           // Assume font 2 is a TrueType font
    // with data type UTF16BE defined
Lu8c("test 测试");              // left put UTF-8 converting from
    // Chinese GB18030
    :
    :
ClosePage();
CloseDoc();
```
Left Align UTF-8 Text Converting from UTF-16LE

Function

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

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

Syntax

```c
void Lu8u(
    UChar*     u16_data,
    bool       same_pos = TRUE
);
```

Parameters

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

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

Sample

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

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

Pos(2,2); // current position to (2",2")
Font(2);  // Assume font 2 is a TrueType font
          // with data type UTF8 defined
Lu8u(data); // Left put UTF-8 converting from
            // UTF16-LE

ClosePage();
CloseDoc();
```
Lines Per Inch

Function

Defines the default vertical baseline spacing in terms of lines per inch for the subsequent text.

Syntax

```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 measurement unit for the subsequent text.

Syntax

```c
void LineSp(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);

LineSp(4); // subsequent baseline spacing will be 4 mm

LineSp(inch(0.4)); // subsequent baseline spacing will be 0.4 inch

ClosePage();
CloseDoc();
```
Margin of Inline Text

**Function**

Sets the inline left margin for the subsequent text to be positioned with the “Next Line” and “Skip Lines” function calls.

**Syntax**

```cpp
void Margin(
    float margin
);
```

**Parameters**

**margin**

The left inline margin for the text in terms of the measurement unit.

**Sample**

```cpp
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
: :
lpi(8);
Margin(0.8); // left margin for the text is 0.8"
Skip(10);    // skip 10 lines
: :
: :
ClosePage();
CloseDoc();
```
Mask an Area

**Function**

Hides a hidden area from an AFP page.

You can hide areas that you do not want to display or print, for instance, you might hide an area that contains old OMR lines and then create a new barcode in the same area.

**Syntax**

```c
void MaskArea(
    float x_pos,
    float y_pos,
    float width,
    float height
);
```

**Parameters**

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

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

- **width**
  The width of the hidden area.

- **height**
  The height of the hidden area.

**Sample**

None.
Mask a Text Field

Function

Masks a text field string by the coordinate location of the data field on an AFP page. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

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

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```c
void MaskField(
    ushort    x,
    ushort    y,
    ushort        column,
    ushort        length,
    char          maskChar = '*',
    ushort       field_no = 1
);
```

Parameters

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

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

- **column**: Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

- **length**: Specifies the number of contiguous bytes (characters), starting at the column, that is to be masked.

- **maskChar**: Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

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

Sample

None.
**Mask Text Field by a Location Area**

**Function**

Masks a text field string by the coordinate location range of the data field on an AFP page. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

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

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

**Syntax**

```c
void MaskField2(
    ushort  x1,
    ushort  x2,
    ushort  y1,
    ushort  y2,
    ushort  column,
    ushort  length,
    char    maskChar = '*',
    ushort  field_no = 1
);
```

**Parameters**

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

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

- **column**
  Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

- **length**
  Specifies the number of contiguous bytes (characters), starting at the column, that is to be masked.

- **maskChar**
  Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

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

**Sample**

None.
Mask Text Field by a Location Area and a Pattern

Function

Masks a text field string by the coordinate location range of the data field on an AFP page and a matching pattern of symbols. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

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

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```c
void MaskField3(   
    ushort  x1,  
    ushort  x2,  
    ushort  y1,  
    ushort  y2,  
    ushort         column,  
    ushort         length,  
    char*       pattern,  
    char          maskChar = '*',  
);  
```

Parameters

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

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

**column**
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

**length**
Specifies the number of contiguous bytes (characters), starting at the column, that is to be masked.

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

- `'@'` A single alphabetic character (A to Z or a to z)
- `'#'` A single numeric character (0 to 9)
- `'&'` A single alphabetic or numeric character
- `'+'` A single blank or numeric character
- `'-='` A single blank or alphabetic character
- `'~'` A single non-blank character
- `'?'` Any single character
To suppress the special syntactic significance of any "@#&+?~¼", and match the character exactly, precede it with a "\" (backslash).

**maskChar**
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.

**Sample**
None.
Mask Text Field by an X-location Range and a Pattern

Function

Masks a text field string by the X-location range of the data field on an AFP page and a matching pattern of symbols. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

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

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```c
void MaskFieldX(
    ushort x1,  // X position range of the data field in PELS.
    ushort x2,  // With MakeAFP ShowPTX utility, you can dump the data fields and their coordinate locations in PELS.
    ushort column,  // Specifies the column number from the beginning of the text field, column 1 refers to the first byte.
    ushort length,  // Specifies the number of contiguous bytes (characters), starting at the column, that is to be masked.
    char* pattern,  // Specifies a character string or a pattern of symbols to be used for picking up a set of the character string that matches with the specified pattern. Valid pattern symbols are:
    char maskChar = '*',  // Specifies a character to be used to mask the text string, default value is an asterisk (*) character.
);```

Parameters

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

column
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

length
Specifies the number of contiguous bytes (characters), starting at the column, that is to be masked.

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

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

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

maskChar
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.
Sample

None.
Mask Text Field by a Y-location Range and a Pattern

Function

Masks a text field string by the Y-location range of the data field on an AFP page and a match pattern of symbols. You may need to mask some confidential number string, such as the credit card number string.

This function must be called after the AFP page is read-in by the “Get Page” function.

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

Make sure the “Encoding” function is called before this function is called if your AFP texts are encoded in EBCDIC so that MakeAFP Weaver can handle encoding conversion properly.

Syntax

```c
void MaskFieldY(
    ushort  y1,
    ushort  y2,
    ushort  column,
    ushort  length,
    char*   pattern,
    char    maskChar = '*',
);
```

Parameters

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

**column**
Specifies the column number from the beginning of the text field, column 1 refers to the first byte.

**length**
Specifies the number of contiguous bytes (characters), starting at the column, that is to be masked.

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

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

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

**maskChar**
Specifies a character to be used to mask the text string, default value is an asterisk (*) character.
Sample

None.
**Maximum Pagination**

**Function**

Defines the maximum number of AFP page buffers.

This function is mainly developed for calling from other programming languages, with Visual C++, you can set MakeAFP variable $MaxPaging directly.

You must call this function to set the value for MakeAFP variable $MaxPaging before calling the “Start” function which allocates the required memory for your paging buffers.

For the generation of pagination, such as "Page 347 of 1000", we need to keep composed AFP data in the AFP page buffers first. With MakeAFP Weaver, you can open multiple pages with the “Open Page” functions, and then process different pages in an interleaved manner once each page is initialized, all the composed AFP data stream will be kept in memory buffers in page-level, and finally, after you have completed all the formatting and counted all the pages of a page group, you have to put your pagination text in each page just before you close the page with the “Close Page” function.

**Syntax**

```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,
Start();  // it must be called before Start() function
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"
Pos(2.5,4);
ClosePage();
CloseDoc();
Move AFP Page between AFP Page Buffers

**Function**

Moves an AFP page from an AFP page buffer in which your AFP page was stored previously to another buffer.

**Syntax**

```c
void MovePage(
    ushort toPage,
    ushort fromPage
);
```

**Parameters**

- `toPage`:
  Specifies an AFP page buffer number to move an AFP page to.

- `fromPage`:
  Specifies an AFP page buffer number to move an AFP page from.

**Sample**

```c
int main( )
{
    unsigned int i, grpPages, pageSN, groups;
    char tmp[80], mobileNo[20], custName[60];
    bool bog = 0;
    $MaxPaging = 50;  // Maximum paging is up to 50 pages
    SetUnit(IN_U600);           // Set default unit to inch
    Start();             // Start initiation, open default input,
                           // output and definition files, retrieves
                           // AFP resources, allocate memory
    Encoding("ibm-037","ibm-437");
    OpenDoc();                // OPne AFP document
    $Page = 1;                // Set AFP page buffer number to 1 for the first
                           // page of AFP file
    GetPage();               // Get first page of AFP file
    while ($Edt == 0)        // Until end of AFP document
    {
        getField(660, 1080, custName);      // Get customer name
        getField(4050, 900, mobileNo);      // Get customer mobile number
        $Page++;            // Point to next AFP page buffer
```
GetPage(); // Get next page

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

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

bog = 0; // Reset it for next group

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

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

sprintf(tmp, "%08d", ++groups);

BgnIdx(tmp); // Auto-converts ASCII to EBCDIC for indexes

PutIdx("Customer Name", custName);
PutIdx("Mobile Number", mobileNo);

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

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

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

EndIdx(); // End of group level index

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

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

} CloseDoc(); // Close AFP document and its file
return 0;
**Next Line**

**Function**

Starts a new text line from the left inline margin defined by the “Margin” function call, it increments the current baseline coordinate position by the amount of baseline increment defined by either the “Lines Per Inch” or “Line Spacing” function call.

**Syntax**

```java
void NextLine (  
    void  
);  
```

**Parameters**

No parameter to be specified.

**Sample**

```java
SetUnit(IN_U600);  
OpenDoc();  
OpenPage(8.5,11);  
:   
:  
1pi(8);  
Margin(0.8);  // left margin for the text is 0.8"  
NextLine();  // Jump to next new line position  
:   
:   
ClosePage();  
CloseDoc();  
```
Open Document

Function

Opens an AFP document, you must call this function to initialize an AFP document before you open an AFP page, and you must close this AFP document by the “Close Document” function before you end your program.

MakeAFP Weaver transfers AFP resources into each AFP output document file if the AFP resource inline is specified by the MakeAFP definition file.

Syntax

```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 Sample:

```c
void main()
{
    Start();  // Start initiation, open default input, output and definition files, getting AFP resources

    OpenDoc();  // Open an document, open its AFP file

    : : :

    CloseDoc()  // Close AFP document and iys AFP file
}
```
Opening Page – Adding New Page

Function

Opens a new AFP page. Once the page formatting is completed, you can close the page with the “Close Page” function. The initial current position is at the page origin (the top left corner of the logical page specified by the Form Definition).

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

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

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

Syntax

```c
void OpenPage(
    float   page_width,
    float   page_height
);
```

Parameters

- **page_width**
  - Width of the page.

- **page_height**
  - Length of the page.

Sample

```c
SetUnit(IN_U600);
$MaxPaging = 100;  // Maximum page buffers are 100, must be defined before Start() function call
Start();            // Start a MakeAFP Weaver session
OpenDoc();
:
$Page = 3;          // switch to page 3 for open page 3
OpenPage(8.5,11);
:
ClosePage();       // Close page 3 and write to AFP output file
:
$Page = 15;         // switch to page 15 for open page 15
OpenPage(8.5,11);
:
```
ClosePage(); // Close page 15 and write to AFP output
file

:
Paragraph of 1-Byte Text

Function

Formats a line of the 1-byte texts into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

You need to define an ASCII or EBCDIC encoded font with the “Font” function. MakeAFP Weaver converts data encoding internally, based on the encoding of AFP font defined.

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

Make sure your default input data encoding and language locale are defined properly by the functions of DefaultCode() and DefaultLocale() before calling this function, otherwise the default encoding “Windows-1252” and locale “en_US” is being used for the paragraph internal processing.

Syntax

```cpp
void ParTxt(
    char*  text,
    float  paragraph_width,
    alignmode  alignment = LEFT,
    bool  same_pos = FALSE
);
```

Parameters

text
The ASCII or EBCDIC text to be aligned into a fixed-width paragraph. Newline character (\n or \x0a) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

- \[F=xx\] \(xx\) are the SBCS font ID in two characters of hex code value, for instance, "01" for 1st font, "03" for 3rd font, etc
- \[U=01\] Turns on underscore
- \[U=00\] Turns off underscore
- \[C=xx\] \(xx\) are the color ID in two characters of hex code value:
  - BLUE "01" RED "02"
  - PINK "03" MAGENTA "03"
  - GREEN "04" CYAN "05"
  - TURQ "05" YELLOW "06"
  - BLACK "08" DARKBLUE "09"
  - BROWN "10" ORANGE "0A"
  - PURPLE "0B" DARKGREEN "0C"
  - DARKCYAN "0D" DARKTURQ "0D"
  - MUSTARD "0E" GRAY "0F"

- \[C=rrrggbb\] \(rr, gg, bb\) are the RGB values in two characters of hex code value respectively, the valid value is from "00" through "FF".

- \[C=ccmmyykk\] \(cc, mm, yy, kk\) are the CMYK values in two characters of hex code value respectively, the valid value is from "00" through "64".
Make sure your default input data encoding is defined properly by the function of DefaultCode() before calling this function with toCode parameter, otherwise default input data encoding is "Windows-1252".

**paragraph_width**
The width of the paragraph.

**alignment**
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

- LEFT: Text is left aligned
- RIGHT: Text is right aligned
- CENTER: Text is centered
- JUSTIFY: Text is justified

**same_pos**
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

**Sample**

```c
char *msg = "The paragraph of text will be right-aligned nicely";

Encoding("ibm-037", "ibm-437"); // PC codepage 437, USAN ASCII
   // AFP codepage 037, USA EBCIDC

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

OpenPage(8.5,11);
 :
LineSp(0.25); // Line spacing is 0.25", 4 LPI
ParTxt(msg,3,RIGHT); // text is right aligned into 3" width
// paragraph
 :
ClosePage();
```
Function

Formats a line of the Japanese text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Syntax

```c
void ParJp ( 
char*      text, 
float      paragraph_width, 
alignmode  alignment = LEFT, 
bool       same_pos = FALSE 
); 
```

Parameters

text
The SJIS Japanese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\x0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

- `[F=xx}` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
- `[U=01}` Turns on underscore
- `[U=00}` Turns off underscore
- `[C=xx}` xx are the color ID in two characters of hex code value:
  - BLUE "01" RED "02"
  - PINK "03" MAGENTA "03"
  - GREEN "04" CYAN "05"
  - TURQ "05" YELLOW "06"
  - BLACK "08" DARKBLUE "09"
  - BROWN "10" ORANGE "0A"
  - PURPLE "0B" DARKGREEN "0C"
  - DARKCYAN "0D" DARKTURQ "0D"
  - MUSTARD "0E" GRAY "0F"
  - `[C=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=ccmmyykk}` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the Japanese paragraph.

alignment
Specifies how the Japanese text in the fixed paragraph should be formatted. The valid values are:

- LEFT Japanese text is left-aligned
Japanese text is right-aligned
Japanese text is centered
Japanese text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

char *msg = "The Japanese ひらがな、漢字、数字 will be center-aligned";
SetUnit(IN_U600);
OpenDoc( );
OpenPage(8.5,11);
LineSp(0.25);  // Line spacing is 0.25", 4 LPI
Font2(1,2);
ParJp(msg,3,CENTER);  // Japanese is center aligned into 3" width
                      // paragraph
ClosePage();
CloseDoc();
Paragraph of Korean

Function

Formats a line of the Korean text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Syntax

```c
void ParKr(
    char*  text,
    float  paragraph_width,
    alignmode  alignment = LEFT,
    bool   same_pos = FALSE
);
```

Parameters

text
The KSC Korean to be aligned into a fixed-width paragraph. Newline character (\n or \x0a) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

- `[F=xx]` xx are the SBCS font ID in two characters of hex code value, for instance, "01" for 1st font, "03" for 3rd font, etc
- `[U=01]` Turns on underscore
- `[U=00]` Turns off underscore
- `[C=xx]` xx are the color ID in two characters of hex code value:
  - BLUE "01"
  - RED "02"
  - PINK "03"
  - MAGENTA "03"
  - GREEN "04"
  - CYAN "05"
  - TURQ "05"
  - YELLOW "06"
  - BLACK "08"
  - DARKBLUE "09"
  - BROWN "10"
  - ORANGE "0A"
  - PURPLE "0B"
  - DARKGREEN "0C"
  - DARKCYAN "0D"
  - ORANGE "0E"
  - GRAY "0F"
- `[C=rrrggb]` rr, gg, bb are the RGB values in two characters of hex code value respectively, valid value is from “00” through “FF”.
- `[C=ccmmyykk]` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, valid value is from “00” through “64”.

paragraph_width
The width of the Korean paragraph.

alignment
Specifies how the Korean text in the fixed paragraph should be formatted. The valid values are:

- LEFT Korean text is left-aligned
RIGHT  Korean text is right-aligned  
CENTER  Korean text is centered  
JUSTIFY  Korean text is justified  

**same_pos**
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

**Sample**

```c
char *msg = "The Korean 온 가족의 티셔츠가 내 품에 will be center-aligned";
OpenPage(8.5,11);
LineSp(0.25);  // Line spacing is 0.25", 4 LPI
Font2(1,2);
ParKr(msg,3,CENTER);  // Korean is center aligned into 3" width  
                      // paragraph
ClosePage();
```
Paragraph of Simplified Chinese

Function

Formats a line of the Simplified Chinese text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Syntax

```c
void ParSc(
    char*  text,
    float  paragraph_width,
    alignmode alignment = LEFT,
    bool   same_pos = FALSE
);
```

Parameters

text
The GBK Simplified Chinese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\x0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

- `[F=xx]` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
- `[U=01]` Turns on underscore
- `[U=00]` Turns off underscore
- `[C=xx]` xx are the color ID in two characters of hex code value:
  - `BLUE` "01" RED "02"
  - `PINK` "03" MAGENTA "03"
  - `GREEN` "04" CYAN "05"
  - `TURQ` "05" YELLOW "06"
  - `BLACK` "08" DARKBLUE "09"
  - `BROWN` "10" ORANGE "0A"
  - `PURPLE` "0B" DARKGREEN "0C"
  - `DARKCYAN` "0D" DARKTURQ "0D"
  - `DARKCYAN` "0D" DARKTURQ "0D"
  - `MUSTARD` "0E" GRAY "0F"
- `[C=rrrggbb]` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
- `[C=ccmmyykk]` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the Chinese paragraph.

alignment
Specifies how the Chinese text in the fixed paragraph should be formatted. The valid values are:
LEFT  Chinese text is left-aligned
RIGHT Chinese text is right-aligned
CENTER Chinese text is centered
JUSTIFY Chinese text is justified

**same_pos**
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

**Sample**

```c
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";
OpenPage(8.5,11);
  
LineSp(0.25); // Line spacing is 0.25", 4 LPI
Font2(1,2);
ParSc(msg,3,CENTER); // Chinese is center aligned into 3" width  
  // paragraph
ClosePage();
```
Paragraph of Traditional Chinese

Function

Formats a line of the Traditional Chinese text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Syntax

```c
void ParTc(
    char*  text,
    float  paragraph_width,
    alignmode alignment = LEFT,
    bool   same_pos = FALSE
);
```

Parameters

text

The BIG5 Traditional Chinese to be aligned into a fixed-width paragraph. Newline character (‘\n’ or ‘\0a’) is allowed to split your text line, and the following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

- `[F=xx]` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
- `[U=01]` Turns on underscore
- `[U=00]` Turns off underscore
- `[C=xx]` xx are the color ID in two characters of hex code value:
  - BLUE "01" RED "02"
  - PINK "03" MAGENTA "03"
  - GREEN "04" CYAN "05"
  - TURQ "05" YELLOW "06"
  - BLACK "08" DARKBLUE "09"
  - BROWN "10" ORANGE "0A"
  - PURPLE "0B" DARKGREEN "0C"
  - DARKCYAN "0D" DARKTURQ "0D"
  - MUSTARD "0E" GRAY "0F"
- `[C=rrrgggbb]` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
- `[C=ccmmyykk]` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width

The width of the Chinese paragraph.

alignment

Specifies how the Chinese text in the fixed paragraph should be formatted. The valid values are:
LEFT  Chinese text is left-aligned
RIGHT Chinese text is right-aligned
CENTER Chinese text is centered
JUSTIFY Chinese text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```c
char *msg = "The Chinese 越来越多的电脑用户 will be center-aligned";
OpenPage(8.5,11);
    LineSp(0.25); // Line spacing is 0.25", 4 LPI
    Font2(1,2);
    ParTc(msg,3,CENTER); // Chinese is center aligned into 3" width paragraph
ClosePage();
```
Paragraph of UTF-16 Text

Function

Formats a line of the Unicode UTF-16LE text into a fixed-width paragraph. Native UTF-16 string on Windows is in little-endian (UTF-16LE) encoding, this function internally translates it to UTF-16BE which is used by AFP.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Make sure your default language locale is defined properly by the function DefaultLocale() before calling this function, otherwise the default locale is “en_US”.

Syntax

```c
void ParU16(
    UChar*     text,
    float      paragraph_width,
    alignmode    alignment = LEFT,
    bool         same_pos = FALSE
);
```

Parameters

text
The UTF-16 text to be aligned into a fixed-width paragraph. The following escape control codes in UTF-16LE can be inserted into your text data for dynamic control of font, color, and underscore switching:

- `[F=xx]` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
- `[U=01]` Turns on underscore
- `[U=00]` Turns off underscore
- `[C=xx]` xx are the color ID in two characters of hex code value:
  - BLUE: "01"
  - RED: "02"
  - PINK: "03"
  - MAGENTA: "03"
  - GREEN: "04"
  - CYAN: "05"
  - TURQ: "05"
  - YELLOW: "06"
  - BLACK: "08"
  - DARKBLUE: "09"
  - BROWN: "10"
  - ORANGE: "0A"
  - PURPLE: "0B"
  - DARKGREEN: "0C"
  - DARKCYAN: "0D"
  - DARKTURQ: "0D"
  - MUSTARD: "0E"
  - GRAY: "0F"

- `[C=rrrggbb]` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.

- `[C=ccmmyykk]` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the paragraph.
alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

- LEFT  Text is left aligned
- RIGHT Text is right aligned
- CENTER Text is centered
- JUSTIFY Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```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"); // Simplified Chinese of China
ChartoU16(msg16, 128, msg); // Converts GB18030 to UTF-16LE
OpenPage(8.5, 11);
:
LPI(4); // Line spacing is 4 lines per inch
         // for paragraph
ParU16(msg16, 3, CENTER); // Chinese text is right aligned
         // into 3" width paragraph
:
ClosePage();
```
Paragraph of UTF-16 Text Converting from Legacy String

Function

Formats a line of the Unicode UTF-16LE text converting from the legacy codepage/charset string into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Make sure your default PC encoding and language locale are defined properly by the functions of DefaultCode() and DefaultLocale() before calling this function, otherwise default PC encoding is “Windows-1252” and locale is “en_US”.

Syntax

```c
void ParU16c(
    char*     text,
    float      paragraph_width,
    alignmode    alignment = LEFT,
    bool         same_pos = FALSE
);
```

Parameters

text
The legacy codepage string to be converted to UTF-16 and aligned into a fixed-width paragraph. The following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

- `[F=xx]` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
- `[U=01]` Turns on underscore
- `[U=00]` Turns off underscore
- `[C=xx]` xx are the color ID in two characters of hex code value:
  - BLUE       "01" RED        "02"
  - PINK       "03" MAGENTA    "03"
  - GREEN      "04" CYAN       "05"
  - TURQ       "05" YELLOW     "06"
  - BLACK      "08" DARKBLUE   "09"
  - BROWN      "10" ORANGE     "0A"
  - PURPLE     "0B" DARKGREEN  "0C"
  - DARKCYAN   "0D" DARKTURQ   "0D"
  - MUSTARD    "0E" GRAY       "0F"
- `[C=rrrggbb]` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
- `[C=ccmmyykk]` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the paragraph.
alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

- LEFT Text is left aligned
- RIGHT Text is right aligned
- CENTER Text is centered
- JUSTIFY Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```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"); // Simplified Chinese of China

OpenPage(8.5,11);

LPI(4); // Line spacing is 4 lines per inch
        // for paragraph

ParU16c(msg, 3, CENTER); // Chinese text is right aligned
                          // into 3" width paragraph

ClosePage();
```
Paragraph of UTF-8 Text

Function

Formats a line of the Unicode UTF-8 text into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Make sure your default language locale is defined properly by the function DefaultLocale() before calling this function, otherwise the default locale is “en_US”.

Syntax

```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 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=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=ccmmyykk]``` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from "00" through "64".

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

- **alignment**: Specifies how the text data in the fixed paragraph should be formatted. The valid values are:
  - LEFT Text is left aligned
RIGHT  Text is right aligned
CENTER Text is centered
JUSTIFY Text is justified

**same_pos**
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

**Sample**

```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"); // Simplified Chinese of China
ChartoU8(msg8, 256, msg); // Converts GB18030 to UTF-8

OpenPage(8.5,11);
:
LPI(4); // Line spacing is 4 lines per inch
// for paragraph
ParU8(msg8, 3, CENTER); // Chinese text is right aligned
// into 3" width paragraph
:
ClosePage();
```
Paragraph of UTF-8 Text Converting from Legacy String

Function

Formats a line of the Unicode UTF-8 text converting from the legacy codepage/charset string into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Make sure your default PC encoding and language locale are defined properly by the functions of DefaultCode() and DefaultLocale() before calling this function, otherwise default PC encoding is “Windows-1252” and locale is “en_US”.

Syntax

```c
void ParU8c(
    char*     text,
    float      paragraph_width,
    alignmode    alignment = LEFT,
    bool         same_pos = FALSE
);
```

Parameters

text
The legacy codepage string to be converted to UTF-8 and aligned into a fixed-width paragraph. The following escape control codes can be inserted into your text data for dynamic control of font, color, and underscore switching:

- `[F=xx]` xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
- `[U=01]` Turns on underscore
- `[U=00]` Turns off underscore
- `[C=xx]` xx are the color ID in two characters of hex code value:
  - BLUE  "01"
  - PINK  "03"
  - GREEN "04"
  - TURQ  "05"
  - BLACK "08"
  - BROWN "10"
  - PURPLE "0B"
  - DARKCYAN "0D"
  - MUSTARD "0E"
  - BLUE  "0F"

- `[C=rrggbbs]` rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
- `[C=cmmmyykk]` cc, mm, yy, kk are the CMYK values in two characters of hex code value respectively, the valid value is from “00” through “64”.

paragraph_width
The width of the paragraph.
alignment
Specifies how the text data in the fixed paragraph should be formatted. The valid values are:

- LEFT: Text is left aligned
- RIGHT: Text is right aligned
- CENTER: Text is centered
- JUSTIFY: Text is justified

same_pos
Indicates whether the current position remains at the origin position before this function is issued. The default value is FALSE, the current position is moved to the position at which the next text would be placed.

Sample

```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"); // Simplified Chinese of China

OpenPage(8.5,11);

: LPI(4); // Line spacing is 4 lines per inch
         // for paragraph
ParU8c(msg, 3, CENTER); // Chinese text is right aligned
             // into 3" width paragraph
:
ClosePage();
```
Paragraph of UTF-8 Text Converting from UTF-16LE

Function

Formats a line of the Unicode UTF-8 text converting from the Unicode UTF-16LE string into a fixed-width paragraph.

You can call the “Lines Per Inch” or “Line Spacing” function first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Make sure your default language locale is defined properly by the function DefaultLocale() before calling this function, otherwise the default locale is “en_US”.

Syntax

```c
void ParU8u(
    UChar* u16_text,
    float paragraph_width,
    alignmode alignment = LEFT,
    bool same_pos = FALSE
);
```

Parameters

u16_text
The UTF-16LE text to be aligned into a fixed-width paragraph. The following escape formatting control codes in UTF-16LE can be inserted into your text data for dynamic control of font, color, and underscore switching:

- \[F=xx\] xx are the SBCS font ID in two characters of hex code value, for instance, “01” for 1st font, “03” for 3rd font, etc
- \[U=01\] Turns on underscore
- \[U=00\] Turns off underscore
- \[C=xx\] xx are the color ID in two characters of hex code value:
  - BLUE “01” RED “02”
  - PINK “03” MAGENTA “03”
  - GREEN “04” CYAN “05”
  - TURQ “05” YELLOW “06”
  - BLACK “08” DARKBLUE “09”
  - BROWN “10” ORANGE “0A”
  - PURPLE “0B” DARKGREEN “0C”
  - DARKCYAN “0D” DARKTURQ “0D”
  - MUSTARD “0E” GRAY “0F”
- \[C=rrrgggb\] rr, gg, bb are the RGB values in two characters of hex code value respectively, the valid value is from “00” through “FF”.
- \[C=ccmmyyyy\] 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";

UChar msg16[128]; // Defines a buffer UTF-16

DefaultCode("gb18030");  // Text is in Chinese GB18030

DefaultLocale("zh_CN"); // language locale is Simplified Chinese

ChartoU16(msg16, 128, msg); // Converts GB18030 to UTF-16LE

OpenPage(8.5,11);

: LPI(4); // Line spacing is 4 lines per inch // for paragraph

ParU8u(msg16, 3, CENTER); // Chinese texts are right aligned // into 3" width paragraph

: ClosePage();
```
Point Value

**Function**

Specifies a value in point.

**Syntax**

```c
float pt(
    float value
);
```

**Parameters**

- `value`  
  The value in point.

**Sample**

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8,11);
:
:
Pos(2.5,4);        // set X and Y position to (2.5",4")
:
:
Pos(pt(20),3.5);   // set X position to 20 points and Y position to // 3.5"
:
:
ClosePage();
CloseDoc();
```
Position of Text

Function

Sets the absolute horizontal position (X) and absolute vertical position (Y) for the output text on the page. The origin position on the page is at (0, 0).

Syntax

```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 not valid.

**y_position**
The value of the absolute vertical position from the page origin. Negative values are not valid.

Sample

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

Pos(5,10);  // set x position at 5 mm and
            // y position at 10 mm

ClosePage();
CloseDoc();
```
Print AFP File

Function

Submits the generated AFP file directly to your AFP/IPDS print server. It calls a printing to submit command provided by your AFP/IPDS Print server software.

It must be specified after the “Close Document” function request.

Notes: With your debug property setting “Program Arguments”, you may have to define a fully-qualified AFP output filename with the flag parameter "-o", for example:

   -d afp2pcl.def -i afp2pcl.txt -o c:\makeafp\samples\test\afp2pcl\afp2pcl.afp

To let PrintAFP() function to submit the AFP output file from a specific path during your development debug running.

Syntax

void PrintAFP(
    char*  print_command,
    ushort      docNo = 1
    char*  winPrinter = NULL
);

Parameters

print_command
The printing submit command is provided by your AFP/IPDS print server or its client software. You must install the client software provided by your vendor if you want to submit the AFP file remotely.

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

winPrinter
Optional, only to be used for print AFP to a Windows PCL printer. Specifies the name of your Windows PCL printer, default is print to your Windows default printer if it is not specified.

Sample

Submit the generated AFP file to IBM Infoprint Manager:

    Start();
    SetUnit(IN_U600);
    OpenDoc();
    :
    :
    CloseDoc();
    
    // Call IBM Infoprint Manager print command:  pdpr
    // job attribute file is:  d:\ipmdata\att\test01.att
    // IPDS printer name is:  prt1
    PrintAFP("pdpr -X d:\ipmdata\att\test01.att -p prt1");
    
    // Call IBM AFP Workbench Viewer "Print It" program

// to print AFP to a Windows PCL printer named
// "Infoprint 1145 PCL by IP"

PrintAFP("\"d:\AFP Viewer\ftdwp\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

```c
void PutIdx(
    char*   index_name,
    char*   index_value,
    ushort  docNo = 1,
    bool    autoConvert = true
);
```

Parameters

Make AFP Weaver puts the characters strings of index_name and index_value “as is” without any conversion, you may need to call one of the Make AFP 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 Make AFP Weaver definition file properly.

**Index_name**
The name of the index, up to 250 characters, including blanks, for example, “Account Number”.

**Index_value**
The value of the index, up to 250 characters, including blanks, for example, “1234-567-4567”.

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

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

Sample

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

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

    $MaxPaging = 50;       // Maximum paging is up to 50 pages
    SetUnit(IN_U600);     // Set default unit to inch
    Start();              // Start initiation, open default input,

    ...
// output and definition files, retrieves
// AFP resources, allocate memory

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

OpenDoc(); // Open AFP document

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

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

while ($Edt == 0) // Until end of AFP document
{
    GetField(660, 1080, custName); // Get customer name
    GetField(4050, 900, mobileNo);  // Get customer mobile number
    do {
        $Page++; // Point to next AFP page buffer
        getPage(); // Get next page
        // detecting if it is the first page of a group,
        // overlay O1OVL1E only used by at first page of
        // each page group
        bog = TriggerOvly("O1OVL1E");
    } while (!bog && !$Edt); // Until beginning of next page group or
                          // End of AFP file

    bog = 0; // Reset it for next group

    // Now got all pages of a page group and first page of next group, now
    // is ready to process new AFP output
    if (!$Edt) // If not end of AFP document
    {
        grpPages = $Page -1; // Keep total number of pages per group,
                              // need to minus 1 page of the first
                              // page of next group
        sprintf(tmp, "%08d", ++groups);
        BgnIdx(tmp); // Auto-converts ASCII to EBCDIC for indexes
        PutIdx("Customer Name", custName);
        PutIdx("Mobile Number", mobileNo);
        for (i = 0; i < grpPages; i++)
        {
            $Page = i + 1; // Point to page buffer number to be
                           // opened
            sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
            BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90); // Add 1D barcode
            ClosePage(); // End of AFP page, write to AFP file
        }
    }

    EndIdx(); // End of group level index

    MovePage(1, grpPages + 1); // As we got first page of next group
                              // previously, now need move its
                              // contents
                              // to page buffer 1 for the next page
    $Page = 1; // Reset page buffer to 1 for next group
}
CloseDoc(); // End of AFP document, close AFP output
return 0;
}
Rendering Color Intent

Function

Specifies the color rendering intent for the subsequent AFP pages or an overlay created by MakeAFP Weaver, to modify the final appearance of the color object.

This function can be repeated to define the rendering intents for all object types.

Syntax

Invokes Color Rendering:

```
void Render(
    objt_type     object_type,
    render_type     render_intent
);
```

Revokes Color Rendering:

```
void RevokeRender( );
```

Parameters

object_type
Specify the object type to which the rendering intent applies:

- **IOCA**: The AFP IOCA image object.
- **OBJT**: The non-AFP data-object, such as JPEG/TIFF/GIF, etc.
- **PTOCA**: The AFP PTOCA text object.
- **GOCA**: The AFP GOCA vector graphic object.

render_intent
Specify the rendering intent for the above object:

- **PERCP**: The Perceptual rendering intent. With this rendering intent, gamut mapping is vendor-specific, and colors are adjusted to give a pleasing appearance. This intent is typically used to render continuous-tone images.
- **SATUR**: The Saturation rendering intent. With this rendering intent, gamut mapping is vendor-specific, and colors are adjusted to emphasize saturation. This intent results in vivid colors and is typically used for business graphics.
- **PELCM**: The Media-relative colorimetric rendering intent. In-gamut colors are rendered accurately, and out-of-gamut colors are mapped to the nearest value within the gamut. Colors are rendered concerning the source white point and are adjusted for the media white point. Therefore colors printed on two different media with different white points won't match colorimetrically but may match visually. This intent is typically used for vector graphics.
- **ABSCM**: The ICC-absolute colorimetric rendering intent. In-gamut colors are rendered accurately, and out-of-gamut colors are mapped to the nearest value within the gamut. Colors are rendered only concerning the source white point and are not adjusted for the media white point. Therefore colors printed on two different media
with different white points should match colorimetrically, but may not match visually. This intent is typically used for logos.

Sample

```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 1-Byte Text

**Function**

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

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

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

**Syntax**

```c
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**

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
Font(3);                // assume font 3 is an ASCII font
:
Pos(2,2);               // current position at (2",2")
Rtxt("text is right aligned");  // right align text at (2",2")
:
:
ClosePage();
CloseDoc();
```
Right Align Japanese

Function

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

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

Syntax

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

Parameters

data
The NULL-terminated SJIS data string.

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

Sample

```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 at (2",2")
Rjp("Alphabet が混在した文章のサンプルです"); // right align SJIS text at
// (2",2")
:
:
ClosePage();
CloseDoc();
```
Right Align Korean

Function

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

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

Syntax

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

Parameters

data
The NULL-terminated KSC data string.

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

Sample

```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 at (2",2")
Rkr("IBM 소프트웨어 솔루션");  // right align KSC text
// at (2",2")

ClosePage();
CloseDoc();```

Right Align Simplified Chinese

Function

Right aligns a single-line of the Simplified Chiense text string at the current position.

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

Syntax

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

Parameters

data

The NULL-terminated GBK data string.

same_pos

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

Sample

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

Font2(3,4);                             // assume font 3 is ASCII font,
                                          // and font 4 is Gb18030 font

Pos(2,2);  // current position at (2",2")
Rsc("实现 Win2000 与 Linux 的双引导");  // right align GBK text at (2",2")

ClosePage();
CloseDoc();
```
Right Align Traditional Chinese

Function

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

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

Syntax

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

Parameters

data
The NULL-terminated BIG5 data string.

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

Sample

```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 at (2",2")
Rtc("實現 Win2000 與 Linux 的雙引导"); // right align BIG5 text at (2",2")
:
ClosePage();
CloseDoc();
```
**Right Align SBCS-HOST/DBCS-HOST**

**Function**

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

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

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

**Syntax**

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

**Parameters**

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

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

**Sample**

```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 at (2",2")
Rdbcs("实现 Win2000 与 Linux 的双引导"); // right align DBCS text at (2",2")
    ::
ClosePage();
CloseDoc();
```
Right Align UTF-16 Text

Function

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

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

Syntax

```c
void Ru16(
    UChar*     data,
    bool     same_pos = TRUE
);```

Parameters

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

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

Sample

```c
/* UTF-16 string, "test" and CJK characters "测试" */
UChar     data1[20] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
Pos(2,2);                      // current position at (2",2")
Font(2);                        // Assume font 2 is a TrueType font
                                // with data type UTF16BE defined
Ru16(data1);                   // right put UTF-16 at (2",2")
ClosePage();
CloseDoc();
```
Right Align UTF-16 Text Converting from Legacy String

Function

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

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

Syntax

```c
void Ru16c(
    char*     data,
    char*     fromcode = NULL,
    bool     same_pos = TRUE
);
```

Parameters

data
The NULL-terminated legacy codepage string.

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

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

Sample

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

DefaultCode("GB18030");   // set default converter for input data
OpenPage(8.5,11);

Pos(2,2);                  // set current position at (2",2")
Font(2);                   // Assume font 2 is a TrueType font
                          // with data type UTF16BE defined

Ru16c("test 测试");      // right put UTF-16 converting from
                          // Chinese GB18030

ClosePage();
CloseDoc();
```
Right Align UTF-8 Text

Function

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

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

Syntax

```c
void Ru8(
    UChar8*     data,
    bool     same_pos = TRUE
);
```

Parameters

data
The NULL-terminated UTF-8 string.

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

Sample

```c
/* UTF-8 string, "test" and CJK characters "测试" */
UChar8 data1[20] = "test\xe6\xb5\xb8\xe8\xaf\xa6\xe8\xaf\xb9";

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
:
:
Pos(2,2);                           // current position at (2",2")
Font(2);                           // Assume font 2 is a TrueType font
    // with data type UTF8 defined
Ru8(data1);                        // right put UTF-8 at (2",2")
:
:
ClosePage();
CloseDoc();
```
Right Align UTF-8 Text Converting from Legacy String

**Function**

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

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

**Syntax**

```c
void Ru8c(
    char*     data, // The NULL-terminated legacy codepage string.
    char*     fromcode = NULL, // The encoding name of the source string to be converted into UTF-8. Default is NULL, default encoding name predefined by the DefaultCode() function is used. Refer to MakeAFP document Encoding Names for more details about the available names.
    bool     same_pos = TRUE // Indicates whether the current position is updated at the end of this function. If this parameter is set to TRUE, the current position remains at the origin position before this function is issued. Otherwise, the current position is moved to the position at which the next character would be placed.
);
```

**Parameters**

- **data**
The NULL-terminated legacy codepage string.

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

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

**Sample**

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

DefaultCode("GB18030"); // set default converter for input data
Pos(2,2); // current position at (2",2")
Font(2); // Assume font 2 is a TrueType font
    // with data type UTF8 defined
Ru8c("test 测试"); // right put UTF-8 converting from 
                // Chinese GB18030

ClosePage();
CloseDoc();
```
Right Align UTF-8 Text Converting from UTF-16LE

Function

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

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

Syntax

```c
void Ru8u(
    UChar* u16_data,
);```

Parameters

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

Sample

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

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

: : Pos(2,2); // current position to (2",2")
Font(2); // Assume font 2 is a TrueType font
        // with data type UTF8 defined
Ru8u(data); // Right put UTF-8 converting from
            // UTF16-LE

: :

ClosePage();
CloseDoc();
```
Set Default Unit

Function

Sets default measurement unit and IPDS printer default units per inch, it must be called before calling the “Open Document” or “Open Page” function.

MakeAFP Weaver default is IN_U600 if you do not call this function.

Syntax

```c
void SetUnit(
    unit    makeafp_unit
);
```

Parameters

**makeafp_unit**

You can specify one of the following value:

- CM_U240   CM, 240 units per inch
- CM_U300   CM, 300 units per inch
- CM_U360   CM, 360 units per inch
- CM_U480   CM, 480 units per inch
- CM_U600   CM, 600 units per inch
- CM_U720   CM, 720 units per inch
- CM_U1440  CM, 1440 units per inch
- MM_U240   MM, 240 units per inch
- MM_U300   MM, 300 units per inch
- MM_U360   MM, 360 units per inch
- MM_U480   MM, 480 units per inch
- MM_U600   MM, 600 units per inch
- MM_U720   MM, 720 units per inch
- MM_U1440  MM, 1440 units per inch
- IN_U240   Inch, 240 units per inch
- IN_U300   Inch, 300 units per inch
- IN_U360   Inch, 360 units per inch
- IN_U480   Inch, 480 units per inch
- IN_U600   Inch, 600 units per inch
- IN_U720   Inch, 720 units per inch
- IN_U1440  Inch, 1440 units per inch
- PT_U240   Point, 240 units per inch
- PT_U300   Point, 300 units per inch
- PT_U360   Point, 360 units per inch
- PT_U480   Point, 480 units per inch
- PT_U600   Point, 600 units per inch
- PT_U720   Point, 720 units per inch
- PT_U1440  Point, 1440 units per inch

Sample

None.
Skip Lines

Function

Skips baseline position by a specific number of lines, and begins a new text line from left inline margin defined by the “Margin” function call, it increments the current baseline coordinate position by the number of lines times the baseline increment defined by either the “Lines Per Inch” or “Line Spacing” function call.

Syntax

```c
void Skip(
    float  lines
);
```

Parameters

- **lines**
  The number of lines to skip.

Sample

```c
SetUnit(IN_U600);
OpenDoc();
OpenPage(8.5,11);
    :
    :
LineSp(0.25);       // Line spacing is 0.25", 4 LPI
Margin(0.8);        // left margin for the text is 0.8"
Skip(10.5);         // Skip 10.5 lines, baseline increment is 
                    // 0.25" x 11.5 = 2.625"
    :
    :
ClosePage();
CloseDoc();
```
Start Session

Function

Starts a MakeAFP Weaver session before calling any other MakeAFP Weaver functions.

This function starts and establishes initiation of a MakeAFP Weaver session, opens a default input data file either in text or binary mode and output AFP document file in binary mode, parses the parameters defined in the MakeAFP Weaver definition file, and merges all the AFP resources and OpenType/TrueType fonts required by your program either by generating external AFP resource file or putting them inline within the output AFP document file; and it also retrieves AFP and OpenType/TrueType fonts information required by MakeAFP for text formatting and alignments.

Syntax

```c
char* Start(
    char* command_line_arguments = NULL,
);
```

Parameters

`command_line_arguments`

It is mainly provided for calling from other programming languages, with which you may want to specify the command-line arguments directly, instead of specifying arguments while issuing commands.

Refer to Chapter 2. Running MakeAFP Weaver in Batch Mode, MakeAFP Weaver Users’ Guide, for more details about command-line flag-arguments supported by MakeAFP Weaver.

Sample

```c
void main() {
    Start();  // Start initiation, open default AFP input, 
             // AFP output and definition files, getting 
             // AFP resources and AFP font Information
    ...
    ...
}
```
Text Orientation

Function

Sets the combination of inline and baseline orientations in which the subsequent text will be presented.

Syntax

```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

This figure illustrates changes in orientation with no change in character rotation.
Trigger by a Location and a Pattern

Function

Defines a location and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

Syntax

```c
bool Trigger(
    ushort x_pos,
    ushort y_pos,
    char* mask
);
```

Parameters

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

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

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

  - `'@'` A single alphabetic character (A to Z or a to z)
  - `'#'` A single numeric character (0 to 9)
  - `'#'` A single alphabetic or numeric character
  - `'+'` A single blank or numeric character
  - `'+'` A single blank or alphabetic character
  - `'|'` A single non-blank character
  - `'|'` Any single character

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

You can specify an empty string as the mask if you only need to detect a trigger by its unique position without comparing of text string.

Sample

```c
/*****************************************************************************/
```
/** This sample shows how to capture a trigger from last page of each group,**
/* get a field from page 1 for add a barcode, mask an area and add a page
/* segment.
/*
/*
/* AFP was encoded in CP-037, USA EBCDIC
*/

/* int main()
 { */
 unsigned int i, grpPages, pageSN = 0;
 char tmp[80], policyNo[20];
 bool eog = 0;
 $MaxPaging = 50; // Maximum paging is up to 50 pages
 SetUnit(IN_U600); // Set default unit to inch
 Start(); // Start initiation, open default input,
 // output and definition files, retrieves
 // AFP resources, allocate memory
 Encoding("ibm-037","ibm-437"); // AFP - CP037, PC - CP437
 OpenDoc(); // Open AFP document
 while ($Edt == 0) // Until end of AFP document
 { $Page = 0; // Reset AFP page buffer number
   do {
     $Page++; // Point to next AFP page buffer
    getPage(); // Get a page from existing AFP file
     if ($Page == 1) // Get policy number from page 1
       getField2(2448, 2448, 6080, 6110, policyNo);
     if ($Page > 2) // detecting if it is a last page
       eog = trigger(3744, 2338, "Part 1"); // a page group, "Part 1" text
       // string only appears at
       // of last page
     } while (!eop); // Until end of each page group
     eop = 0; // reset it for next group
     // Now got all pages of a page group, now it is
     // ready to compose the new AFP output
     grpPages = $Page; // keep total number of pages per group
     for (i = 0; i < grpPages; i++)
     { $Page = i + 1; // point to page buffer number to be opened
     Increment("S1OWL", 0.3, 0.25); // Add a page segment image
     MaskArea(5, 0.4, 2, 0.75); // Mask an area on every page
     sprintf(tmp, "Page %d of %d", $Page, grpPages); // generate pagination
Font(1); Pos(8, 0.45); Rtxt(tmp); // You can use an ASCII encoded font
     // directly with MakeAFP Weaver
sprintf(tmp, "%06d", ++pageSN); // generate page serial number
Font(2); Pos(0.2, 10.8); Ltxt(tmp);
sprintf(tmp, "%d %d %s", pageSN, $Page, policyNo);
BarCode(CODE128, tmp, 0.3, 2, 2, 0.2, DEG90); // Add 1D and 2D barcodes
DataMatrix(tmp, 5.4, 0.8, 0.4, 0.4);
ClosePage(); // Close AFP page, write to AFP file
}
}

CloseDoc(); // Close AFP document and its file
return 0;
}
Trigger by a location area and a Pattern

Function

Defines a location area and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

Syntax

```c
bool Trigger2(  
    ushort x1,  
    ushort x2,  
    ushort y1,  
    ushort y2,  
    char* mask)
```

Parameters

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

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

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

- `'@'`  
  A single alphabetic character (A to Z or a to z)

- `'#'`  
  A single numeric character (0 to 9)

- `'#'`  
  A single alphabetic or numeric character

- `'+`  
  A single blank or numeric character

- `'=.'`  
  A single blank or alphabetic character

- `'+~'`  
  A single non-blank character

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

Sample

```c
/*****************************************************************************/
```
int main()
{
    unsigned int i, grpPages, pageSN = 0;
    char tmp[80], policyNo[20];
    bool eog = 0;
    $MaxPaging = 50;  // Maximum paging is up to 50 pages
    SetUnit(IN_U600); // Set default unit to inch
    Start();  // Start initiation, open default input,
    // output and definition files, retrieves
    // AFP resources, allocate memory
    Encoding("ibm-037","ibm-437"); // AFP - CP037, PC - CP437
    OpenDoc(); // Open AFP document
    while ($Edt == 0) // Until end of AFP document
    {
        $Page = 0; // Reset AFP page buffer number
        do {
            $Page++; // Point to next AFP page buffer
            GetPage(); // Get a page from existing AFP file
            if ($Page == 1) // Get policy number from page 1
                GetField2(2448, 2448, 6080, 6110, policyNo);
            if ($Page > 2)
                eog = Trigger(3744, 2338, "Part ##-#"); // detecting if it is a last
            } while (!eop); // Until end of each page group
            eop = 0; // reset it for next group
            // Now got all pages of a page group, now it is
            // ready to compose the new AFP output
            grpPages = $Page; // keep total number of pages per group
            for (i = 0; i < grpPages; i++)
            {
                $Page = i + 1; // point to page buffer number to be opened
                Inc1Pseg("S1OWL", 0.3, 0.25); // Add a page segment image
                MaskArea(5, 0.4, 2, 0.75); // Mask an area on every page
                sprintf(tmp, "Page %d of %d", $Page, grpPages); // generate pagination
                Font(1); Pos(8, 0.45); Rttx(tmp); // You can use an ASCII encoded font
// directly with MakeAFP Weaver

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

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

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

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

return 0;
Trigger by an X-location Ranger and a Pattern

Function

Defines an X-location range and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

Syntax

```cpp
bool TriggerX(
    ushort   x1,
    ushort   x2,
    char*    mask
);
```

Parameters

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

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

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

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

Sample

None.
Trigger by a Y-location Ranger and a Pattern

Function

Defines a Y-location range and a string or a pattern of symbols to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

Syntax

```c
bool TriggerY(
    ushort  y1,
    ushort  y2,
    char*   mask
);
```

Parameters

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

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

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

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

Sample

None.
Trigger by Name of Copy Group

Function

Defines an AFP copy-group (also called medium map) name to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

This function must be called after an AFP page has been read-in from the existing AFP input file with the “Open Page” function so that you can check if the copy-group name is invoked before this page.

Syntax

```cpp
bool TriggerCopygroup(char* copygroup);
```

Parameters

- **copygroup**
  Specifies an AFP copy-group (also called medium map) name to uniquely identify the end of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

  With the MakeAFP ShowPTX utility, you can find out which copy-group name and where it is invoked in your existing AFP.

Sample

None.
Trigger by Name of Data-Object Image

Function

Defines a data object image (like JPEG/TIFF/GIF) name to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

This function must be called after an AFP page has been read-in from the existing AFP input file with the “Open Page” function so that you can check up if the data-object name is included in this page.

Syntax

```c
bool TriggerObjt(
    char* data_object
);
```

Parameters

data_object

Specifies a data-object name to uniquely identify the first page of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

With the MakeAFP ShowPTX utility, you can find out which data-object name and where it is included in your existing AFP.

Sample

None.
Trigger by Name of Overlay

Function

Defines an overlay name to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

This function must be called after an AFP page has been read-in from the existing AFP input file with the “Open Page” function so that you can check up if the overlay name is included in this page.

Syntax

```c
bool TriggerOvly(
   char*    overlay
);
```

Parameters

`overlay`

Specifies an overlay name to uniquely identify the first page of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

With the MakeAFP ShowPTX utility, you can find out which overlay name and where it is included in your existing AFP.

Sample

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

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

    $MaxPaging = 50;       // Maximum paging is up to 50 pages
    SetUnit(IN_U600);      // Set default unit to inch
    Start();               // Start initiation, open default input,
                            // output and definition files, retrieves
                            // AFP resources, allocate memory
```
Encoding("ibm-037","ibm-437");

OpenDoc();          // Open AFP document

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

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

while ($Edt == 0)    // Until end of AFP document
{
    GetField(660, 1080, custName);     // Get customer name
    GetField(4050, 900, mobileNo);     // Get customer mobile number
    do {
        $Page++;                        // Point to next AFP page buffer
        GetPage();                      // Get next page
        // detecting if it is the first page of a group,
        // overlay O1OVL1E only used by at first page of
        // each page group
        bog = TriggerOvly("O1OVL1E");
    } while (!bog && !$Edt);          // Until beginning of next page group or
                                     // End of AFP file
    bog = 0;                         // Reset it for next group
    // Now got all pages of a page group and first page of next group, now
    // it // is ready to process new AFP output
    if (!$Edt)                        // If not end of AFP document
        grpPages = $Page -1;          // Keep total number of pages per group,
                                      // need to minus 1 page of the first
                                      // page of next group
        sprintf(tmp, "%08d", ++groups);
    BgnIdx(tmp);                     // Auto-converts ASCII to EBCDIC for indexes
    PutIdx("Customer Name", custName);
    PutIdx("Mobile Number", mobileNo);
    for (i = 0; i < grpPages; i++) {
        $Page = i + 1;               // Point to page buffer number to be
                                      // opened
        sprintf(tmp, "%d %d %s", ++pageSN, $Page, mobileNo);
        BarCode(CODE128, tmp, 0.25, 2.2, 2, 0.2, DEG90);  // Add 1D barcode
        ClosePage();                 // Close AFP page, write to AFP file
    }
    EndIdx();                      // End of group level index
    MovePage(1, grpPages + 1);     // As we got first page of next group
                                      // previously, now need move its
                                      // contents
                                      // to page buffer 1 for the next page group
$Page = 1;      // Reset page buffer to 1 for next group
}
CloseDoc();      // Close AFP document and its file
#ifdef _DEBUG
ViewAFP();      // Only view AFP output in debug mode
#endif
return 0;
}
Trigger by Name of Page Segment

Function

Defines a page segment name to uniquely identify the first page of a page group or a specific page, or to be used with the “Get Field” function to identify a text string to be captured from an AFP page. It returns a TRUE bool if the trigger is found.

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

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

This function must be called after an AFP page has been read-in from the existing AFP input file with the “Open Page” function so that you can check up if the page segment name is included in this page.

Syntax

```c
bool TriggerPseg(
    char* page_segment
);
```

Parameters

page_segment

Specifies a page segment name to uniquely identify the first page of a page group or a page. It must be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

With the MakeAFP ShowPTX utility, you can find out which page segment name and where it is included in your existing AFP.

Sample

None.
**Vertical Line**

**Function**

Draws a vertical line.

**Syntax**

```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(RED);  // defines color for the legacy line
Vline(10,10,100,1);    // draw a vertical blue line from
            // (10,10)mm, its length is 100 mm,
            // thickness is 1 mm
 :
 :
ClosePage();
CloseDoc();
```
**Vertical Lines**

**Function**

Repeat drawing vertical lines.

**Syntax**

```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();
```
View AFP File

Function

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

AFP viewer for Windows can be easily integrated with MakeAFP Weaver by the “View AFP” function, so that you can view the AFP file just generated immediately during your development or before printing.

With Windows Explorer, you can select “Tools → Folder Options → File Types → New” to link the AFP type file to an AFP viewer.

During your development, you can run the program in debug or execute mode with your MS Visual Studio C++ compiler. In your project settings, you can define the “Working directory” in which you can keep your input file and MakeAFP definition file, and then define “Program arguments” as -d definition_file -i input_file -o output_afp_file.

* “-i input_file” is an optional parameter, for your development testing or for developing the overlay with MakeAFP, you may just key in the data within your program.

MakeAFP Weaver calls the AFP viewer automatically if an error message has taken place during your development or production, or once the 100 pages limitation is reached if it is running in demo mode without any software license key or hardware key.

Syntax

```c
void ViewAFP(
    ushort      docNo = 1,
    char*       AFPviewer = NULL
);
```

Parameters

docNo
Speifies which AFP document to be opened by AFP Viewer, valid values are 1 through 10, the default value is 1.

AFPviewer
The program name of the AFP viewer, fully qualified with path name in your hard disk, default is using your default AFP Viewer on the Windows system, if this parameter is not specified.

Sample

```c
Start();

SetUnit(IN_U600);
OpenDoc();
:
:
CloseDoc(); // ViewAFP() must be called after AFP file is closed by CloseDoc() function

#if defined DEBUG
```
ViewAFP(1, "d:\AFP Viewer\ftdwinvw.exe"); // only view AFP in debug
#endif // mode
**X Absolute Position**

**Function**

Sets the new horizontal absolute position (X) for the output text on the page. The origin position on the page is at (0, 0).

**Syntax**

```c
void Xpos(
    float  x_position
);
```

**Parameters**

- **x_position**
  The value of the absolute horizontal position from the page origin. Negative values are not valid.

**Sample**

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Xpos(5); // Set x position at 5 mm
:
:
ClosePage();
CloseDoc();
```
X Current Position (Query)

Function

Queries the current horizontal position on the page.

Syntax

float GetXpos( );

Sample

SetUnit(IN_U600);
OpenDoc();
OpenPage(8.27, 11.67);

if ( GetXpos() > 5.5 ) // if current X position is more than
    // 5.5"
{
    : 
    :
} else
{
    : 
    :
}

ClosePage();
CloseDoc();
X Move Relative Position

Function

Moves horizontal position (X) relative to the current horizontal coordinate position.

Syntax

```c
void Xmove(
    float x_move
);
```

Parameters

**x_move**  
The value of horizontal movement relative to the current presentation horizontal position (X). Positive value moves the position to the right; negative value moves the position to the left.

Sample

```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**

**Function**

Sets the new vertical absolute position for the output text on the page. The origin position on the page is at (0, 0).

**Syntax**

```c
void Ypos(
    float    y_position
);
```

**Parameters**

**y_position**
The value of the vertical position absolute from the page origin. Negative values are not valid.

**Sample**

```c
SetUnit(MM_U600);
OpenDoc();
OpenPage(210,297);
:
:
Ypos(15);                // Set Y position at 15 mm
:
:
ClosePage();
CloseDoc();
```
**Y Current Position (Query)**

**Function**

Queries the current vertical position on the page.

**Syntax**

```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 Move Relative Position

Function

Moves vertical position (Y) relative to the current vertical coordinate position.

Syntax

void Ymove(
    float y_move
);

Parameters

y_move
The value of vertical movement relative to the current presentation vertical position (Y).
Positive values move the position down; negative values move the position up.

Sample

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

: :

Ymove(25);  // Move 25 mm down

: :

Ymove(-10);  // Move 10 mm up

: :

ClosePage();
CloseDoc();


Chapter 2. MakeAFP Weaver Parameters

This chapter describes the MakeAFP Weaver parameters to be defined in the MakeAFP definition file, including the syntax rules and values.

Conventions Used in This Chapter

Highlighting

This chapter uses the following highlighting conventions:

- **Bold** Identifies commands, keywords, and other items, whose names are predefined by the MakeAFP or must be entered as-is.
- **Italic** Identifies parameters whose actual names or values you supply.

Syntax Notation

This chapter uses the following syntax notation:

- Italics within a command represent variables for which you must supply a value for. For instance:
  
  \texttt{FONTLIB=pathname}
  
  means that you enter \texttt{FONTLIB=} as shown and then replace the variable \texttt{pathname} with a value that represents any valid path name.

- Do not enter the following symbols as part of the command:

  Vertical bar \texttt{|}
  
  Braces \texttt{{ } }
  
  Brackets \texttt{[ ]}
  
  Underscore \texttt{_}

  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:
    
    \texttt{PRMODE= \{ EBCDIC | SOSI1 | SOSI2 \}}
    
    means that when you enter \texttt{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:
    
    \texttt{FONT1 = \{ CDF | CHS, CDP \} [,height_point] [,scale_ratio]}
means that height_point and scale_ration are the optional parameters.

- An underscore, _, indicates the default value, which MakeAFP uses if you do not specify the parameter with a non-default value. For instance:

  RESTYPE = { NONE | ALL | ......

  means that if the RESTYPE parameter is not entered, MakeAFP Weaver uses the default value of NONE for the RESTYPE parameter.

---

**CMR – Specifies a Colo Management Resource**

**Function**

Specifies a CMR for the AFP color management.

A separate CMR parameter is required for each CMR file, up to a maximum of 16 CMRs can be specified in a MakeAFP definition file.

**Syntax**

CMR\_n = cmr\_file

**Parameter**

\_n

The CMR identifier number, when adding a CMR parameter, it is recommended that you use the next available number, beginning with 1 (one), a maximum of 16 CMRs can be specified in a MakeAFP definition file.

\_cmr\_file

Specifies the file name of an AFP CMR with or without file name extension of *.cmr.

**Sample**

```plaintext
objtlib = d:\afp_cmr

cmr1 = EUROISOCC001000.cmr  Specify CMR of Europe ISO Coated

cmr2 = JPSTD2CC001000.cmr  Specify CMR of Japan Standard V2
```

---

**CPGID – Specifies a Code Page Identifier**

**Function**

Specifies the two through four digits Code Page Global Identifier that defines an IBM-registered code page ID, which is required whenever the index values and attribute names are specified with the MakeAFP Weaver indexing functions.

The Code Page Global Identifier is used by an AFP viewer or AFP archiving system client software, which must display indexing information. This software use this identifier with code page translation tables to represent the index attribute and value data.

For more information about IBM code pages, refer to *IBM AFP Fonts: Technical Reference for Code Pages*, S544–3802 and MakeAFP’s documents *IBM CodePage Name and CPGID Summary, and Encoding Alias Names*.

**Syntax**

CPGID = codepageID
Parameter

*codepageID*

Any valid code page ID, which is a three through the four-character decimal value that defines an IBM-registered code page ID.

If this parameter is not specified, MakeAFP Weaver uses code page ID 850 (Personal Computer - Multilingual Page ASCII) as the default.

**Sample**

cpgid = 437  
Code page ID for US English ASCII

---

**FDEF – Specifies a Form Definition**

**Function**

Specifies the file name of form definition to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver. Once it is specified, then the form definition resources from input AFP file is being removed from the output AFP.

The form definition defines the placement of the page on the form, the input and output bins to use, duplex printing, and so on. You must call a form definition when you print your job. If the AFP file doesn't contain an inline form definition, then you can either specify a form definition by name while you submit your print job or use the default form definition set up by your AFP print server installation.

**Syntax**

\[
FDEF = fdefname
\]

**Parameter**

*fdefname*

Any valid form definition name. The form definition name can be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix F1, if there is one.

**Sample**

fdef = F1TEST01

---

**FDEFLIB – Specifies the Library Path of Form Definitions**

**Function**

Specifies the directories in which form definitions are stored.

**Syntax**

\[
FDEFLIB = pathlist
\]
Parameter

**pathlist**

Any valid search path. You must use a semicolon (;) to separate multiple paths. MakeAFP Weaver searches the paths in the order you specified.

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

1. No filename extension
2. FDE
3. FIL
4. FDEF38PP

Some FROMDEF file extensions may not be supported by your AFP print server.

Sample

```
fdeflib = c:\makeafp\reslib;d:\ipmwin\reslib
```

**FONT – Specifies an AFP FOCA Raster or Outline Font**

**Function**

Specifies a single-byte or double-byte font to be used by the MakeAFP Weaver for the AFP text data stream to be added.

A separate FONT parameter is required for each font, up to a maximum of 32 fonts can be specified in a MakeAFP definition file.

**Syntax**

```
FONTn = { CDF | CHS, CDP } [,height_point] [,scale_ratio]
```

**Parameter**

- **n**
  
The Font identifier number, when adding a font parameter, it is recommended that you use the next available number, beginning with 1 (one), a maximum of 127 fonts can be specified in a MakeAFP definition file.

- **CDF**
  
  Any valid AFP coded font name, up to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix X0 for AFP raster font or XZ for AFP outline font.

- **CHS**
  
  Any valid AFP character set name, up to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix C0 for AFP raster font or CZ for AFP outline font.

- **CDP**
  
  Any valid AFP coded page name, up to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix T1. Make sure you select the correct coded page for your input data, refer to Appendix A to Appendix B for more information.

- **height_point**
  
  Specifies the height of the AFP outline font in points (Each point is equal to 1/72 of one inch).

- **scale_ratio**
Optional, specifies the ratio of font width scaling in percent with an outline font. For instance, specifying scale ratio 200 yields a font with characters string width twice as wide (200% as wide) as normal.

*Font width scaling may not be supported by some AFP viewers.*

**Sample**

```
font1 = x0gt10
font2 = c0d0gt12,t1d0base
font3 = xzhe00,12.5
font4 = czh210,t1v10037,10
```

**FONT – Specifies an OpenType/TrueType Font**

**Function**

Specifies an OpenType/TrueType font with font type extension (.ttf, .otf or .ttc) to be used in MakeAFP formatting, and also indicates the user’s input data type to be used with the font.

A separate FONT parameter is required for each OpenType/TrueType font, up to a maximum of 127 fonts can be specified in a MakeAFP definition file.

**Syntax**

```
FONTn = { ttf_filename | (ttf_filename, font_index) }, encoding, height_point [, scale_tatio]
```

**Parameter**

- **n**
  The Font identifier number, when adding a font parameter, it is recommended that you use the next available number, beginning with 1 (one), a maximum of 127 fonts can be specified in a MakeAFP definition file.

- **ttf_filename**
  Any valid filename of the OpenType/TrueType font or TrueType font collection with file extensions of ttf, otf and ttc.

- **ttf_index**
  The index number of a TrueType font within a TrueType Collection (*.ttc), which includes multiple TrueType fonts in a single file, default is 0 referring to the first TrueType font in TTC. TTC is mostly used for the East Asian CJK fonts.

- **encoding**
  Specifies the encoding to be used to use OpenType/TrueType fonts.

Most of the legacy AFP data stream is encoded by the EBCDIC-based and ASCII-based encoding schemes, although now OpenType/TrueType fonts are encoded in Unicode UTF-16, you can continue using the legacy encoding with AFP, but you must indicate its legacy encoding scheme by an AFP code page name. Code point conversions from the legacy encodings, such as from ASCII, EBCDIC, and DBCS-HOST to UTF-16BE, are performed in the presentation device for AFP, for example, by the IPDS printer or AFP viewer.

The following encoding types are allowed:

```
T1xxxxxx         ASCII, EBCDIC, DBCS-HOST codepage name defined by IBM, refer to Appendix A to B for more details about IBM codepage name.
```
UTF8
Unicode data is encoded in UTF-8. Simple code point conversions from UTF8 to UTF-16BE, is performed quickly in the presentation device for AFP.

UTF16BE
Unicode data is encoded in UTF-16 big-endian.

height_point
Specifies the height of the outline font in points (Each point is equal to 1/72 of one inch).

scale_ratio
Optional, specifies the ratio of font width scaling in percent with an outline font. For instance, specifying scale ratio 200 yields a font with characters string width twice as wide (200% as wide) as normal.

*Font width scaling may not be supported by some AFP viewers.

Sample

font1=tahoma.ttf,T1000437,12
Specifies a TrueType font, height 12 and encoding is by ASCII codepage T1000437 (US English for PC)

font2=(simsun18030.ttc,1),UTF16BE,11,120
Specifies the second font in the TrueType font collection, height 11, width scale 120% and encoding is by Unicode UTF-16BE

font3 = xzhe00,12.5
Specifies an AFP FOCA outline font by coded font name and font height

---

**FONTLIB – Specifies the Library Path of Fonts**

**Function**

Specifies the directories in which AFP fonts and OpenType/TrueType fonts are stored.

**Syntax**

```
FONTLIB = pathlist
```

**Parameter**

**pathlist**

Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.

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

1. No filename extension
2. OLN
3. 600
4. 480
5. 360
6. 300
7. 240
8. ECP
9. CDP
10. CHS
11. CDF
12. CFT
13. FONTOlN
14. FONT240
15. FONT300
**INDEXOBJ – Specifies Generating of the AFP Index Object File**

**Function**

Specifies whether the AFP index object file is to be generated or not. MakeAFP Weaver puts group-level index entries into the index object file.

To achieve the best AFP data loading performance with an AFP archiving system, you need the AFP index file to be loaded together with the AFP document file and AFP resource file.

Refer to IBM *Content Manager OnDemand for Multiplatforms Administration Guide* for more details about loading a previously indexed AFP file directly.

**Syntax**

\[ \text{INDEXOBJ} = \{ \text{YES} \mid \text{NO} \} \]

**Parameter**

**YES**

Specifies that the AFP index object file is generated to be used by an AFP archiving system or AFP Viewer. MakeAFP Weaver generates the AFP index object file with the file name extension .ind.

Make sure the CPGID parameter is defined in your Make AFP Weaver definition file properly.

**NO**

This is the default value, there is no AFP Index object file to be generated. The Index Object file is not required for printing.

**OBJT – Specifies an AFP Object or non-AFP Object**

**Function**

Specifies the file name of the AFP or non-AFP object to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver.

A separate OBJT parameter is required for each object, and a maximum of 127 objects can be specified in a MakeAFP definition file.

If you want objects to be loaded to the printer before the page begins printing, or if objects are used repeatedly and need to be available in the printer memory during printing, then you must define them with OBJT parameters to let MakeAFP Weaver build a catalog of objects being used in the AFP file to hard load them into the printer memory before printing starts.

**Syntax**

\[ \text{OBJT} = \text{objtname}, \text{type} \]
**Parameter**

**objtname**  
Any valid object name exclusive of the filename extension. The float-quoted object name can be 1 to 125 alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @).

**type**  
Indicates type of the object:

- **BCOCA**  
  AFP BCOCA barcode object
- **GOCA**  
  AFP GOCA graphic object
- **IOCA**  
  AFP IOCA image object
- **PSEG**  
  AFP Page Segment image object
- **BMP**  
  Windows Device Dependent Bit Map
- **EPS**  
  Encapsulated Postscript
- **EPSTR**  
  EPS with Transparency
- **GIF**  
  Graphics Interchange Format
- **PCX**  
  Paintbrush Picture File Format
- **JPEG**  
  JPEG file Interchange Format
- **JPEG2**  
  JPEG2000 file Interchange Format
- **PCL**  
  PCL Page Object
- **PDF**  
  PDF Single Page Object
- **PDFSPOTR**  
  PDF Single Page Object with Transparency
- **TIFF**  
  Tag Image File Format

The above objects require appropriate support of the IPDS printer and AFP print server to print.

**Sample**

```
objt = FLOWER1,JPEG
objt = "Orchid Flower",TIFF
```

**OBJTLIB – Specifies the Library Path of Image and CMR Objects**

**Function**

Specifies the directories in which AFP objects, non-AFP objects, and CMRs (Color Management Resources) are stored.

**Syntax**

```
OBJTLIB = pathlist
```

**Parameter**

**pathlist**

Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.

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

1. No filename extension
2. JPG
3. TIF
4. GIF
Note: Some file extensions may not be supported by your AFP print server.

Sample

objtlib = c:\makeafp\imglib;d:\ipmwin\reslib

---

**OVLY – Specifies an Overlay**

**Function**

Specifies the file name of the overlay to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver.

A separate OVLY parameter is required for each overlay, and a maximum of 127 overlays can be specified in a MakeAFP definition file.

If you want overlays to be loaded to the printer before the page begins printing, or if overlays are used repeatedly and need to be available in the printer memory during printing, then you must define them with OVLY parameters to let MakeAFP Weaver build a catalog of overlays being used in the AFP file to hard load them into printer memory before printing starts.

**Syntax**

```
OVLY = ovlyname
```

**Parameter**

`ovlyname`

Any valid overlay name. The overlay name can be one to eight alphanumeric characters (a-z, A-Z, 0–9) and special characters (# $ @), including the two-character prefix O1, if there is one.

**Sample**

```
ovly = O1CDP01
ovly = O1CDP02
```

---

**OVLYLIB – Specifies the Library Path of Overlays**

**Function**

Specifies the directories in which AFP overlays are stored.

**Syntax**

```
OVLYLIB = pathlist
```

**Parameter**

`pathlist`

Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.
When MakeAFP Weaver finds more than one overlay with the same base filename in the same directory, it selects the matching overlay by the following file extension search order:

1. No filename extension
2. 600
3. 480
4. 360
5. 300
6. 240
7. OVL
8. OLY
9. OVR
10. OVLY38PP
11. AFP

Some file extensions may not be supported by your AFP print server.

Sample

```text
ovlylib = c:\makeafp\reslib;d:\ipmwin\reslib
```

---

**PRMODE – Specifies the Type of Input Data and Processing Option**

**Function**

Specifies the type of input data and whether MakeAFP Weaver must perform optional processing on that data or not.

MakeAFP Weaver default is ASCII input data if you do not specify this parameter.

**Syntax**

```
PRMODE= { EBCDIC | SOSI1 | SOSI2 }
```

**Parameter**

- **EBCDIC**
  Specifies that input data is EBCDIC encoding from IBM mainframes.

- **SOSI1**
  Specifies that input data is SBCS-HOST/DBCS-HOST encoding and each SO(shift-out), SI(shift-in) code is to be converted to a white-space character.

- **SOSI2**
  Specifies that input data is SBCS-HOST/DBCS-HOST encoding and each SO(shift-out), SI(shift-in) code is to be escaped.

Sample

```text
prmode = sos1
```

---

**PSEG – Specifies a Page Segment**

**Function**

Specifies the file name of the page segment to be embedded in the AFP resource file or inline within the AFP document file generated by MakeAFP Weaver.

A separate PSEG parameter is required for each page segment, and a maximum of 127 page segments can be specified in a MakeAFP definition file.
If you want page segments to be loaded to the printer before the page begins printing, or if page segments are used repeatedly and need to be available in the printer memory during printing, then you must define them with PSEG parameters to let MakeAFP Weaver build a catalog of page segments being used in the AFP file to hard load them into printer memory before printing starts.

Syntax

\[\text{PSEG} = \text{psegname}\]

Parameter

\textit{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

\begin{itemize}
  \item pseg = S1CDP01
  \item pseg = S1CDP02
\end{itemize}

\section*{PSEGLIB – Specifies the Library Path of Page Segments}

Function

Specifies the directories in which page segments are stored.

Syntax

\[\text{PSEGLIB} = \text{pathlist}\]

Parameter

\textit{pathlist}

Any valid search path. You must use a semicolon (;) to separate the multiple paths. MakeAFP Weaver searches the paths in the order in which they are specified.

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

1. No filename extension
2. 600
3. 480
4. 360
5. 300
6. 240
7. PSG
8. PSE
9. PSEG38PP
10. AFP

Some file extensions may not be supported by your AFP print server.

Sample

\begin{itemize}
  \item pseglib = c:\makeafp\reslib;d:\ipmwin\reslib
\end{itemize}
**RESLIB – Specifies the Library Path of AFP Resources and Objects**

**Function**

Specifies the directories in which form definitions, overlays, page segments, AFP & non-AFP objects, AFP fonts, and OpenType/TrueType fonts are stored.

**Syntax**

```
RESLIB = pathlist
```

**Parameter**

`pathlist`

Any valid search path. You must use a semicolon (`;`) to separate multiple paths. MakeAFP Weaver searches the paths in the order you specified.

**Sample**

```
reslib = c:\makeafp\reslib;d:\ipmwin\reslib;c:\winnt\fonts
```

**RESTYPE - Specifies the Types of Resources to be Retrieved**

**Function**

Specifies the types of resources that should be transferred from the input AFP file and retrieved from the resource directories if any new resourced is used, and whether the resources are being embedded inline within the AFP output document or as a separated AFP resource file.

**Syntax**

```
```

**Parameter**

MakeAFP Weaver supports the specification of the parameters in any combination.

- **NONE**
  Specifies that no AFP resources file has been created or AFP resources are written inline within the AFP output document. This is the default, make sure that all AFP resources are available on the AFP print server.

- **ALL**
  Specifies that all AFP resources, OpenType/TrueType fonts, and non-AFP data-objects are embedded in the resource file or inline within the AFP output document.

- **CMR**
  Specifies that all CMRs (Color Management Resources) are embedded in the resource file or inline within the AFP document file.

- **FDEF**
  Specifies that the form definition is embedded in the resource file or inline within the AFP output document.

- **FONT**
  Specifies that all AFP fonts and OpenType/TrueType fonts are embedded in the resource file or inline within the AFP output document.

- **OBJT**
Specifies that all AFP objects or non-AFP objects are embedded in the resource file or inline within the AFP output document.

**OVLY**
Specifies that all overlays are embedded in the resource file or inline within the AFP output document.

**PSEG**
Specifies that all page segments are embedded in the resource file or inline within the AFP output document.

**INLINE**
Specifies that resources are embedded inline within the AFP document file, otherwise a separate AFP resource file with the filename extension .res is being generated, which can be used by an AFP archiving system directly, like IBM Content Manager OnDemand.

AFP print server treats inline resources as the private AFP resources, and they will be purged from IPDS printer memory automatically after the job is printed successfully.

**Sample**

Include all AFP resources inline:

```
restype = all,inline
```

Include form definition, overlays, non-AFP object and page segments inline for viewing by IBM AFP viewer and IBM DB2 Content Manager OnDemand:

```
restype = fdef,ovly,pseg,objt,inline
```
Chapter 3. MakeAFP Weaver Variables

In addition to the MakeAFP Weaver functions described in Chapter 1, there are several variables maintained by MakeAFP Weaver for internal use or exchanging of information during the data formatting. Some of the MakeAFP Weaver variables described below are accessible from your program.

$Bng – Begin of AFP Page Group Index

Indicates whether the “Begin of Name Group” of AFP index boundary has been detected after the “Get Page” function is called for the reading of an AFP page from the input AFP file. The “Get Page” function reads an AFP page from the AFP input file and also sets $Bng variable to TRUE if a “Begin of Name Group” boundary is detected.

$Eng – End of AFP Page Group Index

Indicates whether the “End of Name Group” of AFP index boundary has been detected after the “Get Page” function is called for the reading of an AFP page from the input AFP file. The “Get Page” function reads an AFP page from the AFP input file and also sets $Eng variable to TRUE if an “End of Name Group” boundary is detected.

$Edt – End of AFP Document

Indicates whether the “End of AFP Document” has been detected after the “Get Page” function is called for the reading of an AFP page from the input AFP file. The “Get Page” function reads an AFP page from the AFP input file and also sets $Edt variable to TRUE if the “End of Document” boundary is detected.

$MaxPaging – Maximum Number of AFP Page Buffers

Defines the maximum number of AFP page buffers. For generating page pagination, such as “Page 347 of 1000”, we need to keep composed AFP data in the AFP page buffers first. With MakeAFP Weaver, you can open multiple pages by “Open Page” function calls, and then process different pages in an interleaved manner once each page is initialized, all the composed AFP data stream will be kept in memory buffers in page-level, and finally, after you have completed all the formatting and counted all the pages of a page group, you have to put your pagination text and OMR in each page just before you end the page with “Close Page” function.

Big value takes big memory, only define this value as big as your maximum requirement for pagination. MakeAFP Weaver default value is 1, you can override its value before you start a MakeAFP Weaver session by calling the function of “Start();”. MakeAFP Weaver reports an error message if its value is not enough for your job.
$Page – Current AFP Page Buffer Number

Defines the current AFP page buffer number. With $Page variable, you can directly switch to any AFP page buffer is to be opened with the “Open Page” function request, or access it again.
Chapter 4. String Manipulation Functions

Although MS Visual Studio C++ provides comprehensive powerful functions for file input & output handling, searching and sorting, memory buffer manipulation, data conversion, string manipulation, directory control, etc, you still may need some of MakeAFP’s complementary functions specially developed for data formatting requirements to assist your MakeAFP application developments.

Refer to Microsoft MSDN library for more detailed information about the functions provided by MS Visual Studio C++ in its run-time library routines, iostream library and standard C++ library,

The descriptions of the MakeAFP Weaver functions for string manipulation are listed in alphabetic order. The description of Each function includes the following sections:

Function
A description of the major purpose of the function.

Syntax
A diagram showing the function parameters.

Parameters
Explanation of each parameter.

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

Default Values
When calling these functions, every parameter must be specified in the order shown in this chapter. MakeAFP provides default values to some parameters for simplifying the use of the function, so you can omit them by default values when you invoke the function, but when your program omits parameters for a function that provides default values, your program must omit all the parameters that follow. In other words, you cannot omit a parameter in the middle.

Comma Float

Function
Formats a float using commas as the thousandth separators and a specified number of significant fractional digits.

Syntax

```c
char *CommaFloat(
    double float _value,
    ushort fraction_digits
);
```
Parameters

float _value
Source float value.

fraction_digits
The number of significant fractional digits.

Sample

float total = 129894.5698;
printf("Total Amount: %s", CommaFloat(total,2));

Output: Total Amount: 129,894.57

Comma Integer

Function
Formats a 64-bit integer using commas as the thousandth separators.

Syntax

char *CommaInt(
    _int64 integer_value
);

Parameters

integer_value
Source 64-bit integer value.

fraction_digits
The number of significant fractional digits.

Sample

_int64 total = 1298945698123;
printf("Total Amount: %s", CommaInt(total));

Output: Total Amount: 1,298,945,698,123

Comma Digital String

Function
Formats a digital string using commas as the thousandth separators and a specified number of significant fractional digits, removing the leading zeros.

Syntax

char *CommaDigit(
    str* digital_string,
    ushort fraction_digits
);

char *CommaDigit2(
    str* digital_string,
);

---

Comma Float

Function
Formats a float value using commas as the thousandth separators and a specified number of significant fractional digits.

Syntax

char *CommaFloat(
    float value,
    fraction_digits
);

Parameters

value
Source float value.

fraction_digits
The number of significant fractional digits.

Sample

float total = 129894.5698;
printf("Total Amount: %s", CommaFloat(total,2));

Output: Total Amount: 129,894.57
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 *Delete(
    char*  string,
    ushort  start_col,
    ushort  length
);
```

**Parameters**

string
Source data string.

start_col
Starting character position to delete.

length
The length of characters to be deleted.

Sample

```c
char data[30] = "This is a string testing";
printf("After Deleted: %s", Delete(data,11,7));

Output:

After Deleted: This is a testing
```

Extract Substrings

**Function**

Extracts a substring or multiple substrings delimited by the given separator(s).

**Syntax**

Extract once by a delimiter:
char *Extract1(
    char* srcStr,
    ushort order_pos,
    char* delimiter
);

Extract multiple time by a delimiter:

void Extract(
    char* dstStr_array[],
    char* srcStr,
    char delimiter,
    char qualifier
);

void Extract2(
    char* dstStr_array[],
    char* srcStr,
    char* delimiters
);

Parameters

dstStr_array
Destination array of the strings extracted.

srcStr
Source string comprising of delimited character(s) substrings.

order_pos
Order position number of the substring to be extracted.

delimiters
Set of delimiter characters.

delimiter
A delimiter character.

qualifier
A character as the qualifier.

Sample

char src1[256] = "substring1;substring2;substring3;substring4";
char src2[256] = "field1:,123,456,000.00:,field3:,12,341.00:,field5";
char src3[256] = "test1,'168,456,000.00',test3,'88,666.00'";
char *dst[20];

printf("Extracted 3rd substrings is: %s", Extract1(src1, 3,""));

Extract(dst,src2,:,");

printf("Extracted substrings 1 are: %s %s %s %s", 
    dst[0], dst[1], dst[2], dst[3]);

Extract(dst, src3, ',', '\');

printf("Extracted substrings 2 are: %s %s %s %s", 
    dst[0], dst[1], dst[2], dst[3]);
Output:

Extracted 3rd substrings is: substring3
Extracted substrings 1 are: field1 123,456,000.00 field3 12,341.00
Extracted substrings 2 are: test1 168,456,000.00 test3 88,666.00

Find String

Function

Checks whether a string is in the data string returns its position if found, otherwise returns 0.

Syntax

```
int Find1(
    char* str,
    char* search,
    int start_pos
)
```
```
int Find2(
    char* str,
    char* search,
    int start_pos,
    int stop_pos
)
```

Parameters

- **str**
  Source data string.

- **search**
  Search string.

- **start_pos**
  The position to start the search.

- **stop_pos**
  The position to stop the search.

Sample

```
char str[80] = "This is data string search testing";
int pos = Find2(str, "search", 9, 30);
```

Return:

21

First Character

Function

Returns the position of the first non-white-space character.

Syntax

```
int FristChar(
    char* string
);
```
Parameters

string
Source data string.

Sample

char str[80] = "This is data string";
int pos = FirstChar(str);

Return: 8

In Substitution Table

Function
Checks whether a string is in the substitution table. Returns 1 if the string is found in the substitution table, otherwise returns 0.

Syntax

int InSubst(
    char* subst_tbl[][2],
    char* search
);

Parameters

sbst_tbl
Substitution table.

search
search string.

Sample

char *payment [] [2] = { {"001", "Cash Payment"},
                        {"005", "Master Payment"},
                        {"003", "Visa Payment"},
                        {"007", "Check Payment"},
                        {"011", "GOTO payment"},
                        {"\0", "\0"} }; // End of table

int intab = InSubst(payment, "002");
int intab = InSubst(payment, "007");

Return:
0
1

Insert String

Function
Inserts a character string after the specified position.

Syntax

char *Insert(
    char* srcStr,
    ushort pos,
    ushort insertStr
);
Parameters

string
Source data string.

pos
Character position where you insert a string after.

insertStr
The string to be inserted in the destination string.

Sample

cchar src[256] = "This is data string";
printf("%s", Insert(src,8,"inserted "));

Output:
This is inserted data string

Is Empty

Function
Checks if a string is either of 0-byte length or contains white-space characters only. Returns 1 if it is empty otherwise returns 0.

Syntax

int IsEmpty(
    char* string
);

Parameters

string
data string.

Sample

cchar str1[80] = "This is data string";
cchar str2[80] = "    ";
cchar str3[80] = NULL;
int empty = IsEmpty(str1);
    empty = IsEmpty(str2);
    empty = IsEmpty(str3);

Return:
  0
  1
  1

Last Character

Function
Returns the position of the last non-white-space character.

Syntax

int LastChar(
    char* string
);
Parameters

**string**
Source data string.

Sample

char str[80] = "This is data string ";
int pos = LastChar(str);

Return: 19

Left Copy

**Function**
Copies characters from the left of the source string to destination string with null-terminated.

**Syntax**

```c
char *Lcp(
    char* dstStr,
    char* srcStr,
    ushort length
);
```

**Parameters**

**dstStr**
Destination string.

**srcStr**
Source string.

**length**
The 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 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, new line control codes on the right side of the destination will be trimmed before being terminated with NULL.

Syntax

```c
char *LcpRtrim(
    char* dstStr,
    char* srcStr,
    ushort length
);
```

Parameters

dstStr
Destination string.

srcStr
Source string.

length
The number of characters to be copied.

Sample

```c
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

```c
char *Ltrim(
    char* string
);
```

Parameters

string
Data string to be left trim.

Sample

```c
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.

---

**Left Trim and Concatenate Strings**

**Function**

Trims white-space characters from the left sides of multiple data strings before concatenating them.

**Syntax**

```c
char *LtrimCat(
    char* separator,
    char* string,...
);
```

Parameters
characters to be inserted between concatenated strings.

string,...
variable-argument lists of multiple strings.

Sample
char str1[] = "This is a string testing.";
char str2[] = "The second string.";
printf("After Concatenation: %s", LtrimCat(" ", str1, str2));
Output: After Concatenation: This is a string testing. The second string.

Match String Comparing

Function
Recursively compares a string to a pattern, returning 1 if a match is found or 0 if not.

Syntax
int Match(
    char* string,
    char* pattern,
    bool ignore_case
);  

Parameters
string
The NULL-terminated strings to compare.

pattern
The NULL-terminated pattern string to be used for the comparison. The general syntax of the
pattern is:
```
`*'
Matches any sequence of characters (zero or more)
`?
Matches any single character
[SET]
Matches any character in the specified set
[!SET] or [^SET]
Matches any character, not in the specified set
```

A set is composed of characters or ranges; a range looks like `character hyphen character" (as
in 0-9 or A-Z). [0-9a-zA-Z_] is the minimal set of characters allowed in the [..] pattern construct.

To suppress the special syntactic significance of any of "[]??!^-\", inside or outside a []
construct, and match the character exactly, precede it with a "\" (backslash).

ignore_case
Specifies whether the upper and lower case is ignored.

Sample
char str1[] = "This is data string";
char str2[] = "TX 20890";
char str3[] = "Answer?";

int rc = Match(str1, ".*", 0);
rc = Match(str2, "[A-Z][A-Z] [0-9][0-9][0-9][0-9][0-9]", 0);
rc = Match(str3, "\?", 0);

Return: 0
  1
  1
Pattern Searching

**Function**

Searches a string for a set of characters that match a specified pattern, returns the characters if it is finding a match.

**Syntax**

```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);
Return:
12345
45-57
88 USA
```

Remove String

**Function**

Removes a single instant of string or multiple instances of the string.

**Syntax**

```c
char *Remove1(char* string, char* pattern, int start_pos);
```

- Remove once:
  ```c
  char *Remove1(
  ```
char*      srcStr,
char*      rmStr
);

Remove all:
char *Remove(
char*      srcStr,
char*      rmStr
);

Parameters

srcStr
Source string.

rmStr
The string to be removed from the source string.

Sample

c char str[] = "This is a string testing string.";
printf("After Removed once: %s\n", Remove1(str," string"));
printf("After Removed all: %s\n", Remove(str," string"));
Output: After Removed once: This is a testing string.
After Removed all: This is a testing.

Replace String

Function
Replaces a single instant of string or multiple instances of the string.

Syntax
Replace once:
char *Replace1(
char* srcStr,
char* tgtStr,
char* newStr
);

Replace all:
char *Replace(
char* srcStr,
char* tgtStr,
char* newStr
);

Parameters

srcStr
Source string.

tgtStr
Target string to be replaced.

newStr
New string to be used to replace target string.
Sample

```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
int Rfind1(
    char* str,
    char* search,
    int start_pos
)
Or
int Rfind2(
    char* str,
    char* search,
    int start_pos,
    int stop_pos
)
```

**Parameters**

- `str`: Source data string.
- `search`: Search string.
- `start_pos`: The position to start the search.
- `stop_pos`: The position to stop the search.

**Sample**

```c
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**

```c
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**

```c
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.
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 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 destination string where the white-space, carriage return, new line 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, new line control codes, from the right side of the source string.

**Syntax**

```c
char *Rtrim(
    char*    string
);```

**Parameters**

- **string**  
  Data string to be right trimmed.

**Sample**

```c
char str[] = "This is a string testing       ";

printf("Right Trimmed: %s", Rtrim(str));
```

**Output:**

```
Right Trimmed: This is a string testing
```

---

**Right Trim and Concatenate Strings**

**Function**
Trims white-space, carriage return, new line control codes from the right sides of multiple data strings before concatenating them.

**Syntax**

```c
char *RtrimCat(
    char* separator,
    char* string,...
);
```

**Parameters**

- `separator`: characters to be inserted between concatenated strings.
- `string,...`: variable-argument lists of multiple stringss.

**Sample**

```c
char str1[] = "This is a string testing.      ";
char str2[] = "The second string.   ";
printf("After Concatenation: %s", RtrimCat(" ", str1, str2));
```

**Output:**

After Concatenation: This is a string testing. The second string.

---

### Right Trim for EBCDIC

**Function**

Trims EBCDIC white-space, carriage return, new line control codes, from the right side of the source EBCDIC string.

**Syntax**

```c
char *E_Rtrim(
    char* string
);
```

**Parameters**

- `string`: Data string to be right trimmed.

**Sample**

Refer to the sample for the Rtrim function.

---

### Strings Concatenate

**Function**

Concatenates multiple strings into a string.

**Syntax**

```c
char *Strcat(
    char* string,...
);
```
Parameters

String...
Variable-argument lists of multiple strings.

Sample

char str1[] = "This is a string testing. ";
char str2[] = "The second string.";
printf("After Concatenation: %s", Strcat(str1, str2));
Output:

After Concatenation: This is a string testing. The second string.

String Pad

Function
Pads a character to the right side of the string.

Syntax

char *StrPad(
    char    *srcStr,
    ushort    length,
    char    pad_char
);

Parameters

srcStr
Source string.

length
Length of the new destination string to be returned.

pad_char
The character to be padded to the right of the destination string if the length of the source string is less than the specified length.

Sample

char str[] = "The string testing";
printf("Result: %s", StrPad(str, 25, '\.'));
Output:

Result: The string testing.......

Substitute String

Function
Returns a string with a substitution found in the substitute table you defined, otherwise returns NULL.

Syntax

char *Subst(
    char*    subst_tbl[][2],
    char*    srtcStr
);
**Parameters**

- **Subst_tbl**
  Substitution table.
- **srcStr**
  Source string.

**Sample**

```c
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* src, int start, int end);
```
char *SubStr(
    char* dstStr,
    char* srcStr,
    ushort from_pos,
    ushort length
);

char *SubStr2(
    char* srcStr,
    ushort from_pos,
    ushort length
);

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of substring.

Sample

char str[] = "This is a substring testing";
char dst[20];
printf("Result: %s, SubStr(dst,str,11,9)));

Output:
Result: substring

**Substring Left Trim**

**Function**

Gets the specified length of the substring from the specified position of the source string, and trims white-space characters from the left of the destination string.

**Syntax**

```c
char *SubStrLtrim(
    char* dstStr,
    char* srcStr,
    ushort from_pos,
    ushort length
);
```
char *SubStrLtrim2(
    char* srcStr,
    ushort from_pos,
    ushort length
);

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of the substring.

Sample

char str[] = "This is a      substring testing";
char dst[20];
printf("Result: %s", SubStrLtrim(dst,str,11,14));
Output:
           Result: substring

Substr Pad

Function

Gets the specified length of the substring from the specified position of the source string, and pads with pad character to the right of the destination string if appropriate.

Syntax

char *SubStrPad(
    char* dstStr,
    char* srcStr,
    ushort from_pos,
    ushort length,
    char    pad_char
);

char *SubStrPad2(
    char* srcStr,
    ushort from_pos,
    ushort length,
    char    pad_char
);

Parameters

dstStr
Destination string.

srcStr
Source string.
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
);

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"
char dstl[20];
SubStrRtrim(dstl, str1, 11, 16);
```
printf("Result: %s", dst1);
Output:

Result: substring

---

**Substring Trim Both Sides**

**Function**

Gets the specified length of the substring from the specified position of the source string, and trims white-space, carriage return, new line control codes from both sides of the destination string.

**Syntax**

```c
char *SubStrTrim(
    char* dstStr,
    char* srcStr,
    ushort from_pos,
    ushort length
);
char *SubStrTrim2(
    char* srcStr,
    ushort from_pos,
    ushort length
);
```

**Parameters**

- **dstStr**
  Destination string.

- **srcStr**
  Source string.

- **from_pos**
  The starting position to start from.

- **length**
  Length of the substring.

**Sample**

```c
char str1[] = "This is a substring testing";
char dst1[20];
printf("Result: %s", SubStrTrim(dst1, str1, 11, 20));
Output:

Result: substring
```

---

**System Date and Time**

**Function**

Returns a formatted time and date string.
**Syntax**

```c
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)
- `%u` - 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**

```c
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**

```c
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
);
```

**Parameters**

**thai_string**
The Thai ASCII data string to be used for composition.

**Sample**

No sample was provided.

Translate Digits to Simplified Chinese Figures

**Function**
Translate ASCII digits to Simplified Chinese figures in GB18030 encoding.

**Syntax**

```c
char *DigitGBK(
    char* dst,
    char* src
);
```

**Parameters**

**dst**
Destination of Simfilied Chinese figures string encoding in GD18030.

**src**
Source of digits string encoded in ASCII.

**Sample**

```c
printf("Chinese figures: %s", DigitGBK("123.45"));
```
**Translate Digits to Traditional Chinese Figures**

**Function**
Translate ASCII digits to Traditional Chinese figures in BIG5 encoding.

**Syntax**
```c
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**
```c
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**
```c
char *Trim(
    char* string
);
```

**Parameters**
- **string**
  Data string to be trimmed.

**Sample**
```c
char str[] = "     This is a string testing     ";
printf("Both Sides Trimmed: %s", Trim(str));
```

**Output:**

```
Both Sides Trimmed: This is a string testing
```
Trim Both Sides for EBCDIC

Function

Trims EBCDIC white-space characters from both sides of the source string, as well as carriage return and new line control codes from the right side of the source EBCDIC string.

Syntax

```
char *E_Trim(
    char*    string
);
```

Parameters

```
string
    EBCDIC data string to be trimmed.
```

Sample

Refer to the sample for the Trim function.

Trim and Concatenate Strings

Function

Trims white-space characters from both sides and carriage-return and line-feed control codes from the right side of the source strings before concatenating them into the destination string.

Syntax

```
char *TrimCat(
    char     *separators,
    char     *srcStr,...
);
```

Parameters

```
separators
    Characters to be inserted between concatenated strings.
```

```
srcStr,...
    Variable-argument lists of multiple strings.
```

Sample

```
char str1[] = "      This is a string testing.    ";
char str2[] = "   This is 2nd string.    ";
printf("Result: %s", TrimCat(" ***** ", str1, str2));
```

Output:

```
Result: This is a string testing. ***** This is 2nd string.
```
Chapter 5. Conversion Functions

When developing applications around legacy and Unicode characters, it is required to convert between legacy ASCII/DBCS-PC and EBCDIC/DBCS-HOST, between Unicode and legacy text data, or between Unicode encodings.

**Codepage/Charset TO UTF-16 Conversion**

**Function**

Converts from codepage/charset stream to Unicode UTF-16, and returns the length of the UTF-16 output.

**Syntax**

```c
int32_t  ChartoU16(
    UChar   *target,    // Pointer to the targeted UTF-16 output buffer.
    int32_t  *targetCapacity,  // The maximum size of the targeted UTF-16 buffer.
    char    *source,  // Pointer to the input source buffer, in bytes.
    int32_t  sourceLen = -1,  // Length of the input source, or default -1 for NULL-terminated input.
    char  fromCode = NULL  // The name of the source encoding. Default is NULL, uses the encoding name pre-defined and loaded by the “DefaultCode” function. Refer to MakeAFP document Encoding Names for more details about the available names.
);```

**Parameters**

- **target**
  Point to the targeted UTF-16 output buffer.

- **targetCapacity**
  The maximum size of the targeted UTF-16 buffer.

- **source**
  Pointer to the input source buffer, in bytes.

- **sourceLen**
  Length of the input source, or default -1 for NULL-terminated input.

- **fromCode**
  The name of the source encoding. Default is NULL, uses the encoding name pre-defined and loaded by the “DefaultCode” function. Refer to MakeAFP document Encoding Names for more details about the available names.

**Codepage/Charset TO UTF-8 Conversion**

**Function**

Converts from codepage/charset stream to Unicode UTF-8 and returns the length of the UTF-8 output.
Syntax

```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**
  Maximum size the targeted UTF-8 buffer.

- **source**
  Pointer to the input source buffer, in bytes.

- **sourceLen**
  Length of the input source, or default -1 for NULL-terminated input.

- **fromCode**
  The name of the source encoding. Default is NULL, uses the encoding name pre-defined and loaded by the “DefaultCode” function. Refer to MakeAFP document *Encoding Names for more details about the available names.*

### Default Encoding Names

**Function**

Defines the current default input data encoding names.

Make sure you have defined a correct encoding name before calling data encoding conversion functions and paragraph functions.

**Syntax**

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

**Parameters**

- **codeName**
  The name of the default encoding, default is “windows-1252”. Refer to MakeAFP document *Encoding Names for more details about the available names.*

### Universal Conversion

**Function**

Converts from one external charset to another, like conversion between legacy ASCII/DBCS-PC and EBCDIC/DBCS-HOST, between Unicode and legacy text data or
between Unicode encodings. External string used as source or target for the conversion is always treated as a byte stream. It returns the length of the complete target output.

**Syntax**

```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 MakeAFP document *Encoding Names* for more details about the available names.

- **fromCode**
  The name of the source encoding. Refer to MakeAFP document *Encoding Names* for more details about the available names.

- **target**
  Point to the target output buffer.

- **targetCapacity**
  The maximum size of the target buffer, in bytes.

- **source**
  Pointer to the input source buffer.

- **sourceLen**
  Length of the input source, in bytes, or default -1 for NULL-terminated input.

---

**UTF-16 to Codepage/Charset Conversion**

**Function**

Converts from Unicode UTF-16 to a codepage/charset stream and returns the length of the complete target output.

**Syntax**

```c
int32_t  U16toChar(
    char   *target,
    int32_t  *targetCapacity,
    UChar    *source,
    int32_t   sourceLen = -1,
    char  toCode = NULL
);
```

**Parameters**

- **target**
  Point to the target output buffer.

- **targetCapacity**
  The maximum size of the target buffer, in bytes.
UTF-16 to UTF-8 Conversion

Function

Converts from Unicode UTF-16 to UTF-8 and returns the length of the complete UTF-8 target output.

Syntax

```c
int32_t U16toU8(
    UChar8 *target,
    int32_t *targetCapacity,
    UChar *source,
    int32_t sourceLen = -1
);
```

Parameters

- `target`
  Point to the target UTF-8 output buffer.

- `targetCapacity`
  The maximum size of the UTF-8 target buffer.

- `source`
  Pointer to the UTF-16 input source buffer.

- `sourceLen`
  Length of the UTF-16 input source, or default -1 for NULL-terminated input.

UTF-32 to UTF-16 Conversion

Function

Converts from Unicode UTF-32 to UTF-16 and returns the length of the complete UTF-16 target output.

Syntax

```c
int32_t U32toU16(
    UChar *target,
    int32_t *targetCapacity,
    UChar32 *source,
    int32_t sourceLen = -1
);
```
Parameters

target
Point to the target UTF-16 output buffer.

targetCapacity
The maximum size of the UTF-16 target buffer.

source
Pointer to the UTF-32 input source buffer.

sourceLen
Length of the UTF-32 input source, or default -1 for UTF-32 NULL-terminated input.

---

UTF-32 to UTF-8 Conversion

Function
Converts from Unicode UTF-32 to UTF-8 and returns the length of the complete UTF-8 target output.

Syntax

```c
int32_t U32toU8(
    UChar8   *target,
    int32_t  *targetCapacity,
    UChar32   *source,
    int32_t   sourceLen = -1
);
```

Parameters

target
Point to the target UTF-8 output buffer.

targetCapacity
The maximum size of the UTF-8 target buffer.

source
Pointer to the UTF-32 input source buffer.

sourceLen
Length of the UTF-32 input source, or default -1 for UTF-32 NULL-terminated input.

---

UTF-8 to UTF-16 Conversion

Function
Converts from Unicode UTF-8 to UTF-16 and returns the length of the complete UTF-16 target output.

Syntax

```c
int32_t U8toU16(
    UChar   *target,
    int32_t  *targetCapacity,
    UChar8    *source,
    int32_t   sourceLen = -1
);
```

Parameters

target
Point to the target UTF-8 output buffer.

targetCapacity
The maximum size of the UTF-8 target buffer.

source
Pointer to the UTF-32 input source buffer.

sourceLen
Length of the UTF-32 input source, or default -1 for UTF-32 NULL-terminated input.
Parameters

**target**
Point to the target UTF-16 output buffer.

**targetCapacity**
The maximum size of the UTF-16 target buffer.

**source**
Pointer to the UTF-8 input source buffer.

**sourceLen**
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

---

**UTF-8 to Codepage/Charset Conversion**

**Function**
Converts from Unicode UTF-8 to a codepage/charset stream returns the length of the complete target output.

**Syntax**

```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 MakeAFP document *Encoding Names for more details about the available names.*

**Syntax**

```c
int32_t  U8toChar(
    UChar   *target,
    int32_t  *targetCapacity,
    UCha8    *source,
    int32_t   sourceLen = -1,
    char  toCode = NULL
);
```
Parameters

**target**
Point to the target UTF-16 output buffer.

**targetCapacity**
The maximum size of the UTF-16 target buffer.

**source**
Pointer to the UTF-8 input source buffer.

**sourceLen**
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

---

### Vietnamese Codepage/Charset Codepage/Charset Conversion

**Function**

Converts Vietnamese from one external charset to another, like conversion between legacy PC formats, between Unicode and legacy text data or between Unicode encodings. External string used as source or target for the conversion is always treated as a byte stream. It returns the length of the complete target output.

**Syntax**

```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>Description</td>
<td>Encoding</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>T1000855</td>
<td>Cyrillic - Personal Computer</td>
<td>855 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000856</td>
<td>Hebrew - Personal Computer</td>
<td>856 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000857</td>
<td>Latin5 PC</td>
<td>857 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000858</td>
<td>PC - Multilingual with euro</td>
<td>858 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000860</td>
<td>Portugal - Personal Computer</td>
<td>860 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000861</td>
<td>Iceland - Personal Computer</td>
<td>861 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000862</td>
<td>Hebrew - Personal Computer</td>
<td>862 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000863</td>
<td>Canadian French - PC</td>
<td>863 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000864</td>
<td>Arabic - Personal Computer</td>
<td>864 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000865</td>
<td>Nordic - Personal Computer</td>
<td>865 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000866</td>
<td>Cyrillic #2 - Personal Computer</td>
<td>866 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000867</td>
<td>Israel - Personal Computer</td>
<td>867 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000868</td>
<td>Greece - Personal Computer</td>
<td>869 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000869</td>
<td>Iceland - Personal Computer</td>
<td>870 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000870</td>
<td>Cyrillic PC with Euro</td>
<td>872 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000871</td>
<td>Thai - Personal Computer</td>
<td>874 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000872</td>
<td>Greek .</td>
<td>875 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000873</td>
<td>OCR-AN ASCII</td>
<td>876 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000874</td>
<td>OCR-B ASCII</td>
<td>877 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000875</td>
<td>Cyrillic Multilingual</td>
<td>880 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000876</td>
<td>Thailand</td>
<td>889 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000877</td>
<td>OCR - A</td>
<td>892 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000878</td>
<td>OCR - B</td>
<td>893 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000879</td>
<td>Japan PC #1</td>
<td>897 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000880</td>
<td>Symbols, Set 7 ASCII</td>
<td>899 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000881</td>
<td>PC, Baltic - Multilingual w Euro</td>
<td>901 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000882</td>
<td>8-bit Estonia with Euro</td>
<td>902 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000883</td>
<td>People's Republic of China - PC</td>
<td>903 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000884</td>
<td>Republic of China (ROC) - PC</td>
<td>904 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000885</td>
<td>Latin3 Multilingual</td>
<td>905 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000886</td>
<td>APL ASCII</td>
<td>910 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000887</td>
<td>Latin2 ISO/ANSI 8-BIT</td>
<td>912 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000888</td>
<td>Latin 3, ISO/ASCII</td>
<td>913 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000889</td>
<td>Latin4 ISO/ANSI 8-BIT</td>
<td>914 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000890</td>
<td>Cyrillic ISO/ASCII 8-Bit</td>
<td>915 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000891</td>
<td>Hebrew ISO/ASCII 8-Bit</td>
<td>916 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000892</td>
<td>Latin5 ISO/ANSI 8-BIT</td>
<td>920 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000893</td>
<td>PC, Baltic - Multilingual</td>
<td>921 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000894</td>
<td>Estonia PC</td>
<td>922 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000895</td>
<td>Latin 9</td>
<td>923 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000896</td>
<td>Latin 9 EBCDIC</td>
<td>924 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000897</td>
<td>DCF REL 2 Compatibility</td>
<td>1002 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000898</td>
<td>U.S. Text Subset</td>
<td>1003 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000899</td>
<td>IBM PC Desktop Publishing</td>
<td>1004 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000900</td>
<td>Arabic ISO/ASCII 8-Bit</td>
<td>1008 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000901</td>
<td>Cyrillic Multilingual</td>
<td>1025 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000902</td>
<td>Latin5</td>
<td>1026 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000903</td>
<td>Japanese (Latin) Extended</td>
<td>1027 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000904</td>
<td>Hebrew Publishing</td>
<td>1028 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000905</td>
<td>Arabic Extended ISO/ASCII 8-Bit</td>
<td>1029 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000906</td>
<td>MICR, E13-B Combined</td>
<td>1032 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000907</td>
<td>MICR, CMC-7 Combined</td>
<td>1033 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000908</td>
<td>Symbols, Adobe ASCII</td>
<td>1038 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000909</td>
<td>GML List Symbols</td>
<td>1039 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>T1000910</td>
<td>Japanese Extended - PC</td>
<td>1041 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000911</td>
<td>Simplified Chinese Extended - PC</td>
<td>1042 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000912</td>
<td>Traditional Chinese Extended PC</td>
<td>1043 ASCII</td>
<td></td>
</tr>
<tr>
<td>T1000913</td>
<td>Arabic Extended ISO/ASCII 8-Bit</td>
<td>1046 ASCII</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td>Code</td>
<td>Code</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>T1001068</td>
<td>Text With Numeric Spacing</td>
<td>1068</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001069</td>
<td>Latin4 EBCDIC</td>
<td>1069</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001087</td>
<td>Symbols, Adobe</td>
<td>1087</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001091</td>
<td>Symbol Set 7, Modified</td>
<td>1091</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001092</td>
<td>Symbol Set 7, Modified - PC</td>
<td>1092</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001093</td>
<td>IBM LOGO</td>
<td>1093</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001110</td>
<td>Latin2 Multilingual</td>
<td>1110</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001111</td>
<td>Latin2 ISO/ANSI 8-BIT</td>
<td>1111</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001112</td>
<td>Baltic - Multilingual, EBCDIC</td>
<td>1112</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001113</td>
<td>Estonia, EBCDIC</td>
<td>1122</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001114</td>
<td>Cyrillic, Ukraine EBCDIC</td>
<td>1123</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001115</td>
<td>Cyrillic, Ukraine ISO-8</td>
<td>1124</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001116</td>
<td>PC, Cyrillic Ukrainian</td>
<td>1125</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001117</td>
<td>Vietnamese ISO-8</td>
<td>1129</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001118</td>
<td>Vietnamese EBCDIC</td>
<td>1130</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001119</td>
<td>PC, Cyrillic, Belo Russian</td>
<td>1131</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001120</td>
<td>Lao EBCDIC</td>
<td>1132</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001121</td>
<td>Lao ISO-8</td>
<td>1133</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001122</td>
<td>Japan Alphanumeric Katakana</td>
<td>1139</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001123</td>
<td>USA, Canada ECECP</td>
<td>1140</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001124</td>
<td>Austria, Germany ECECP</td>
<td>1141</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001125</td>
<td>Denmark, Norway ECECP</td>
<td>1142</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001126</td>
<td>Finland, Sweden ECECP</td>
<td>1143</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001127</td>
<td>Italy ECECP</td>
<td>1144</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001128</td>
<td>Spain, Latin America ECECP</td>
<td>1145</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001129</td>
<td>UK ECECP</td>
<td>1146</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001130</td>
<td>France ECECP</td>
<td>1147</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001131</td>
<td>International ECECP</td>
<td>1148</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001132</td>
<td>Iceland ECECP</td>
<td>1149</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001133</td>
<td>Latin2 Multilingual with Euro</td>
<td>1153</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001134</td>
<td>Windows Turkish</td>
<td>1254</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001135</td>
<td>Windows Baltic Rim</td>
<td>1257</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001136</td>
<td>Windows Vietnamese</td>
<td>1258</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001137</td>
<td>Apple Latin 1</td>
<td>1275</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001138</td>
<td>Adobe PS Standard</td>
<td>1276</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001139</td>
<td>Adobe PS ISO Latin 1</td>
<td>1277</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001140</td>
<td>Apple Greece</td>
<td>1280</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001141</td>
<td>Apple Turkey</td>
<td>1281</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001142</td>
<td>Apple Central Europe</td>
<td>1282</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001143</td>
<td>Apple Cyrillic</td>
<td>1283</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001144</td>
<td>GENERIC BAR CODE/OCR-B</td>
<td>1300</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001145</td>
<td>Latin 2 – Windows</td>
<td>1250</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001146</td>
<td>Cyrillic – Windows</td>
<td>1251</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001147</td>
<td>Latin 1 – Windows</td>
<td>1252</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001148</td>
<td>Greece – Windows</td>
<td>1253</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001149</td>
<td>Turkey – Windows</td>
<td>1254</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001150</td>
<td>Israel – Windows</td>
<td>1255</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001151</td>
<td>Arabic – Windows</td>
<td>1256</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001152</td>
<td>Latin 4 – Windows</td>
<td>1257</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001153</td>
<td>Vietnamese – Windows</td>
<td>1258</td>
<td>ASCII</td>
</tr>
<tr>
<td>T1001154</td>
<td>USA/Canada - CECP</td>
<td>37</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001155</td>
<td>Germany F.R./Austria - CECP</td>
<td>273</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001156</td>
<td>Belgium - CECP</td>
<td>274</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001157</td>
<td>Brazil - CECP</td>
<td>275</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001158</td>
<td>Denmark/Norway - CECP</td>
<td>277</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001159</td>
<td>Finland/Sweden - CECP</td>
<td>278</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>T1001160</td>
<td>ITALY- CECP</td>
<td>280</td>
<td>EBCDIC</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>T1H01252</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>T10300, 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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CP37/CP835)</td>
</tr>
<tr>
<td>939</td>
<td>Combination of Japanese SBCS-HOST and DBCS-HOST</td>
<td>EBCDIC/DBCS-HOST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CP1027/CP300)</td>
</tr>
<tr>
<td>943</td>
<td>Combination of Japanese ASCII and SJIS-PC for open systems</td>
<td>ASCII/SJIS-PC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CP1114/CP947)</td>
</tr>
<tr>
<td>1363</td>
<td>Combination of Korean ASCII and KSC-PC for open systems</td>
<td>ASCII/KSC-PC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CP1126/CP1362)</td>
</tr>
<tr>
<td>1364</td>
<td>Combination of Korean SBCS-HOST and DBCS-HOST</td>
<td>EBCDIC/DBCS-HOST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(CP1114/CP1374)</td>
</tr>
<tr>
<td>1377</td>
<td>Combination of Traditional Chinese SBCS-HOST and DBCS-HOST for HKSCS</td>
<td>ASCII/GBK-PC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(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 UTF-16 (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, csIBM037, cp037, 037, cpIBM37, cp37, T1V10037</td>
<td>USA/Canada – CECP, EBCDIC</td>
<td>037</td>
</tr>
<tr>
<td>ibm-259, IBM-Symbols, csIBM-Symbols</td>
<td>Symbols, Set 7, EBCDIC</td>
<td>259</td>
</tr>
<tr>
<td>ibm-273, IBM273, CP273, csIBM273, ebcDIC-de, cpIBM273, 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, csIBM277, ebcDIC-dk, cpIBM277, 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, csIBM278, ebcDIC-sv, cpIBM278, 278, T1V10278</td>
<td>Finland/Sweden- CECP, EBCDIC</td>
<td>278</td>
</tr>
<tr>
<td>ibm-280, IBM280, CP280, ebcDIC-cp-it, csIBM280, cpIBM280, 280, T1V10280</td>
<td>ITALY- CECP, EBCDIC</td>
<td>280</td>
</tr>
<tr>
<td>ibm-284, IBM284, CP284, ebcDIC-cp-es, csIBM284, cpIBM284, 284, T1V10284</td>
<td>Spain/Latin America- CECP, EBCDIC</td>
<td>284</td>
</tr>
<tr>
<td>ibm-286, EBCDIC-AT-DE-A, csEBCDIC-ATDEA, T1000286</td>
<td>Austria/Germany F.R., Alt (3270), EBCDIC</td>
<td>286</td>
</tr>
<tr>
<td>ibm-290, IBM290, cp290, EBCDIC-JP-kana, csIBM290, 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-cp-fr, csIBM297, cpIBM297, 297,</td>
<td>France – CECP, EBCDIC</td>
<td>297</td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td>Encoding</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>IBM-420, IBM420, cp420, ebcdic-cp-ar1, csIBM420, 420, T1000420</td>
<td>Arabic Bilingual, EBCDIC</td>
<td>420</td>
</tr>
<tr>
<td>IBM-424, IBM424, cp424, ebcdic-cp-he, csIBM424, 424, T1000424</td>
<td>Israel (Hebrew), EBCDIC</td>
<td>424</td>
</tr>
<tr>
<td>IBM-437, IBM437, cp437, 437, csPC8CodePage437, windows-437, T1000437</td>
<td>US Personal Computer, ASCII</td>
<td>437</td>
</tr>
<tr>
<td>IBM-500, IBM500, CP500, ebcdic-cp-be, csIBM500, ebcdic-cp-ch, cpiIBM500, 500, T1V10500</td>
<td>International #5, EBCDIC</td>
<td>500</td>
</tr>
<tr>
<td>IBM-720, windows-720, DOS-720</td>
<td>PC Arabic, ASCII</td>
<td>720</td>
</tr>
<tr>
<td>IBM-737, IBM737, cp737, windows-737, 737</td>
<td>PC Greek, ASCII</td>
<td>737</td>
</tr>
<tr>
<td>IBM-775, IBM775, cp775, csPC775Baltic</td>
<td>PC Baltic, ASCII</td>
<td>775</td>
</tr>
<tr>
<td>IBM-803, cp803, T1000803</td>
<td>Hebrew Character Set A, EBCDIC</td>
<td>803</td>
</tr>
<tr>
<td>IBM-808, T1000808</td>
<td>Cyrillic, Russian with euro, ASCII</td>
<td>808</td>
</tr>
<tr>
<td>IBM-819, IBM819, cp819, latin1, 8859_1, csISOLatin1, iso-ir-100, ISO_8859, ISO_8859-1:1987, I1, 819, T1000819</td>
<td>Latin-1, West European, ASCII</td>
<td>819</td>
</tr>
<tr>
<td>IBM-838, IBM-Thai, csIBMThai, cp838, 838, ibm-9030, ibm838, T1000838</td>
<td>Thai EBCDIC</td>
<td>838</td>
</tr>
<tr>
<td>IBM-848, T1000848</td>
<td>Cyrillic, Ukraine with Euro, ASCII</td>
<td>848</td>
</tr>
<tr>
<td>IBM-849, T1000849</td>
<td>Cyrillic, Belarus Russian Euro, ASCII</td>
<td>849</td>
</tr>
<tr>
<td>IBM-850, IBM850, cp850, 850, csPC850Multilingual, windows-850, T1000850</td>
<td>PC Multilingual, ASCII</td>
<td>850</td>
</tr>
<tr>
<td>IBM-851, IBM851, cp851, 851, csPC851, T1000851</td>
<td>Greek - Personal Computer, ASCII</td>
<td>851</td>
</tr>
<tr>
<td>IBM-852, IBM852, cp852, 852, csPcp852, windows-852, T1000852</td>
<td>Latin2 Multilingual PC, ASCII</td>
<td>852</td>
</tr>
<tr>
<td>IBM-855, IBM855, cp855, 855, csiIBM855, csPCp855, windows-855, T1000855</td>
<td>Cyrillic - Personal Computer, ASCII</td>
<td>855</td>
</tr>
<tr>
<td>IBM-856, cp856, 856, T1000856</td>
<td>Hebrew (old) - Personal Computer, ASCII</td>
<td>856</td>
</tr>
<tr>
<td>IBM-857, IBM857, cp857, 857, csiIBM857, windows-857, T1000857</td>
<td>Latin5 PC, ASCII</td>
<td>857</td>
</tr>
<tr>
<td>IBM-858, IBM00858, CCSID00858, CP00858, PC-Multilingual-850+euro, cp858, windows-858, T1000858</td>
<td>PC - Multilingual with euro, ASCII</td>
<td>858</td>
</tr>
<tr>
<td>IBM-860, IBM860, cp860, 860, csIBM860, T1000860</td>
<td>Portugal - Personal Computer, ASCII</td>
<td>860</td>
</tr>
<tr>
<td>Code</td>
<td>Names</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ibm-861, IBM861, cp861, 861, cp-is, csIBM861, windows-861, T1000861</td>
<td>Iceland - Personal Computer, ASCII</td>
<td>861</td>
</tr>
<tr>
<td>ibm-862, IBM862, cp862, 862, DOS-862, csPC862LatinHebrew, windows-862, T1000862</td>
<td>Hebrew - Personal Computer, ASCII</td>
<td>862</td>
</tr>
<tr>
<td>ibm-863, IBM863, cp863, 863, csIBM863, T1000863</td>
<td>Canadian French - PC, ASCII</td>
<td>863</td>
</tr>
<tr>
<td>ibm-864, IBM864, cp864, csIBM864, T1000864</td>
<td>Arabic - Personal Computer, ASCII</td>
<td>864</td>
</tr>
<tr>
<td>ibm-865, IBM865, cp865, 865, csIBM865, T1000865</td>
<td>Nordic - Personal Computer, ASCII</td>
<td>865</td>
</tr>
<tr>
<td>ibm-866, IBM866, cp866, 866, csIBM866, windows-866, T1000866</td>
<td>Cyrillic #2 - Personal Computer, ASCII</td>
<td>866</td>
</tr>
<tr>
<td>ibm-867, cp867, T1000867</td>
<td>Israel - Personal Computer, ASCII with Euro sign</td>
<td>867</td>
</tr>
<tr>
<td>ibm-868, IBM868, CP868, 868, csIBM868, cp-ar, T1000868</td>
<td>Urdu, ASCII</td>
<td>868</td>
</tr>
<tr>
<td>ibm-869, IBM869, cp869, 869, cp-gr, csIBM869, windows-869, T1000869</td>
<td>Greek - Personal Computer, ASCII</td>
<td>869</td>
</tr>
<tr>
<td>ibm-870, IBM870, CP870, ebcdic-cp-roeco, ebcdic-cp-yu, csIBM870, T1000870</td>
<td>Latin2 Multilingual, EBCDIC</td>
<td>870</td>
</tr>
<tr>
<td>ibm-871, IBM871, ebcdic-cp-is, csIBM871, CP871, ebcdic-is, cpibm871, 871, T1V10871</td>
<td>Iceland – CECP, EBCDIC</td>
<td>871</td>
</tr>
<tr>
<td>ibm-872, T1000872</td>
<td>Cyrillic PC with Euro, ASCII</td>
<td>872</td>
</tr>
<tr>
<td>ibm-874, ibm-9066, cp874, TIS-620, tis80.2533, eucTH, cp9066, ibm874, MS874, windows-874, T1000874</td>
<td>Thai - Personal Computer, ASCII</td>
<td>874</td>
</tr>
<tr>
<td>ibm-875, IBM875, cp875, 875, T1000875</td>
<td>Greece, EBCDIC</td>
<td>875</td>
</tr>
<tr>
<td>ibm-878, KOI8-R, koi8, csKOI8R, cp878, windows-20866, T1000878</td>
<td>Russian internet, ASCII</td>
<td>878</td>
</tr>
<tr>
<td>ibm-880, IBM880, cp880,EBCDIC-Cyrillic,csIBM880, windows-20880, T1000880</td>
<td>Cyrillic Multilingual, ASCII</td>
<td>880</td>
</tr>
<tr>
<td>ibm-896, T1000896</td>
<td>Japanese Katakana, ASCII</td>
<td>896</td>
</tr>
<tr>
<td>ibm-897, JIS_X0201, X0201, csHalfWidthKatakana, T1000897</td>
<td>Japanese, Half Width Katakana, ASCII</td>
<td>897</td>
</tr>
<tr>
<td>ibm-901, T1000901</td>
<td>Baltic – PC Multilingual, ASCII with Euro sign</td>
<td>901</td>
</tr>
<tr>
<td>ibm-902, T1000902</td>
<td>Estonia, ASCII with Euro sign</td>
<td>902</td>
</tr>
<tr>
<td>ibm-905, IBM905, CP905, ebcdic-cp-tr, csIBM905, windows-20905, T1000905</td>
<td>Latin3 Multilingual, EBCDIC</td>
<td>905</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4:1988, l4, 8859_4, cp914, 914, windows-28594, T1000914</td>
<td>Latin5, Cyrillic ISO/ASCII 8-Bit, ASCII</td>
<td>915</td>
</tr>
<tr>
<td>ibm-916, cp916, 916, T1000916</td>
<td>Urdu, EBCDIC</td>
<td>918</td>
</tr>
<tr>
<td>ibm-918, IBM918, CP918, ebcddic-cp-ar2, csIBM918, ebcddic-cp-ar2, T1000918</td>
<td>Latin5 ISO/ANSI 8-BIT, ASCII</td>
<td>920</td>
</tr>
<tr>
<td>ibm-921, iso-8859-13, 8859_13, cp921, 921, windows-28603, T1000921</td>
<td>Estonian, ASCII</td>
<td>922</td>
</tr>
<tr>
<td>ibm-922, IBM922, cp922, 922, T1000922</td>
<td>Latin 9, ASCII</td>
<td>923</td>
</tr>
<tr>
<td>ibm-923, iso-8859-15, Latin-9, I9, 8859_15, latin0, csisolatin0, csisolatin9, iso8859_15_fdlis, cp923, 923, windows-28605, T1000923</td>
<td>Japanese, mixed EBCDIC/DBCS-HOST</td>
<td>930</td>
</tr>
<tr>
<td>ibm-924, IBM00924, CCSID00924, CP00924, ebcddic-Latin9—euro, T1000924</td>
<td>Japanese, SJIS without MS/IBM extensions</td>
<td>932</td>
</tr>
<tr>
<td>ibm-930, ibm930, ibm-5026, cp930, cpibm930, 930</td>
<td>Chinese (simplified), mixed EBCDIC/DBCS-HOST</td>
<td>933</td>
</tr>
<tr>
<td>ibm-931, ibm-5035, cp939, 939</td>
<td>Chinese (traditional), mixed EBCDIC/DBCS-HOST</td>
<td>934</td>
</tr>
<tr>
<td>ibm-932, ibm-942, cp932, shift_jis78, sjis78, ibm-942_VSUB_VPUA, ibm-932_VSUB_VPUA</td>
<td>Japanese, Shift-JIS standard with extension</td>
<td>936</td>
</tr>
<tr>
<td>ibm-933, ibm933, cp933, cpibm933, 933</td>
<td>Chinese (simplified), mixed EBCDIC/DBCS-HOST</td>
<td>937</td>
</tr>
<tr>
<td>ibm-935, ibm935, cp935, cpibm935, 935</td>
<td>Chinese (traditional), mixed EBCDIC/DBCS-HOST</td>
<td>938</td>
</tr>
<tr>
<td>ibm-936, windows-936-2000, GBK, CP936, MS936, windows-936</td>
<td>Japanese, Shift-JIS standard with extension</td>
<td>940</td>
</tr>
<tr>
<td>ibm-937, cp937, cpibm937, 937</td>
<td>Japanese, Shift-JIS standard with extension</td>
<td>941</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, pck, sjis, ibm-943_VSUB_VPUA</td>
<td>Windows Korean, (old) encoding</td>
<td>942</td>
</tr>
<tr>
<td>ibm-949, cp949, 949, ibm-949_VASCII_VSUB_VPUA, cp949c, ibm-949_VSUB_VPUA</td>
<td>Windows Chinese(Traditional), ASCII BIG5</td>
<td>950</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<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>
<td>964</td>
</tr>
<tr>
<td>ibm-971, ibm-971_VPUA</td>
<td>Extended Unix Code Packed Format for Korean</td>
<td>971</td>
</tr>
<tr>
<td>Ibm-1004, T1001004</td>
<td>IBM PC Desktop Publishing, ASCII</td>
<td>1004</td>
</tr>
<tr>
<td>Ibm-1006, cp1006, 1006</td>
<td>Urdu, 8-bit EBCDIC</td>
<td>1006</td>
</tr>
<tr>
<td>Ibm-1008, cp1008, T1001008</td>
<td>Arabic ISO/ASCII 8-Bit, ASCII</td>
<td>1008</td>
</tr>
<tr>
<td>ibm-1025, cp1025, 1025, T1001025</td>
<td>Cyrillic Multilingual, EBCDIC</td>
<td>1025</td>
</tr>
<tr>
<td>ibm-1026, IBM1026, CP1026, csIBM1026, 1026, T1001026</td>
<td>Latin5, Turkey, EBCDIC</td>
<td>1026</td>
</tr>
<tr>
<td>ibm-1046, T1001046</td>
<td>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>
<td>1047</td>
</tr>
<tr>
<td>ibm-1097, cp1097, 1097</td>
<td>Farsi, EBCDIC</td>
<td>1097</td>
</tr>
<tr>
<td>ibm-1098, cp1098, 1098</td>
<td>Farsi, ASCII</td>
<td>1098</td>
</tr>
<tr>
<td>ibm-1112, cp1112, 1112, T1001112</td>
<td>Baltic – Multilingual, EBCDIC</td>
<td>1112</td>
</tr>
<tr>
<td>ibm-1122, cp1122, 1122, T1001122</td>
<td>Estonia, EBCDIC</td>
<td>1122</td>
</tr>
<tr>
<td>ibm-1123, cp1123, 1123, cpibm1123, T1001123</td>
<td>Cyrillic, Ukraine EBCDIC</td>
<td>1123</td>
</tr>
<tr>
<td>ibm-1124, cp1124, 1124, T1001124</td>
<td>ISO Cyrillic Ukraine, ASCII</td>
<td>1124</td>
</tr>
<tr>
<td>ibm-1125, cp1125, 1125, T1001125</td>
<td>Cyrillic Ukraine ASCII</td>
<td>1125</td>
</tr>
<tr>
<td>ibm-1129, T1001129</td>
<td>ISO Vietnamese, ASCII</td>
<td>1129</td>
</tr>
<tr>
<td>ibm-1130, T1001130</td>
<td>Vietnamese EBCDIC</td>
<td>1130</td>
</tr>
<tr>
<td>ibm-1131, cp1131, 1131, T1001131</td>
<td>Cyrillic Belarus EBCDIC</td>
<td>1131</td>
</tr>
<tr>
<td>ibm-1132, T1001132</td>
<td>Lao EBCDIC</td>
<td>1132</td>
</tr>
<tr>
<td>ibm-1133, T1001133</td>
<td>ISO Lao, ASCII</td>
<td>1133</td>
</tr>
<tr>
<td>ibm-1137</td>
<td>Devanagari EBCDIC</td>
<td>1137</td>
</tr>
<tr>
<td>ibm-1140, IBM01140, CCSID01140, CP01140, cp1140, cpiibm1140, ebcdic-us-37+euro, T1001140</td>
<td>USA, EBCDIC with the Euro sign</td>
<td>1140</td>
</tr>
<tr>
<td>ibm-1141, IBM01141, CCSID01141, CP01141, cp1141, cpiibm1141, ebcdic-de-273+euro, T1001141</td>
<td>Austria, Germany ECECP, EBCDIC with the Euro sign</td>
<td>1141</td>
</tr>
<tr>
<td>ibm-1142, IBM01142, CCSID01142, CP01142, cp1142, cpiibm1142, ebcdic-dk-277+euro, ebcdic-no-277+euro, T1001142</td>
<td>Denmark, Norway ECECP, EBCDIC with the Euro sign</td>
<td>1142</td>
</tr>
<tr>
<td>ibm-1143, IBM01143, CCSID01143 CP01143, cp1143, cpiibm1143, ebcdic-fi-278+euro, ebcdic-se-278+euro, T1001143</td>
<td>Finland, Sweden ECECP, EBCDIC with the Euro sign</td>
<td>1143</td>
</tr>
<tr>
<td>ibm-1144, IBM01144, CCSID01144, CP01144, cp1144, cpiibm1144, ebcdic-it-280+euro, T1001144</td>
<td>Italy ECECP, EBCDIC with the Euro sign</td>
<td>1144</td>
</tr>
<tr>
<td>ibm-1145, IBM01145, CCSID01145, CP01145, cp1145, cpiibm1145,</td>
<td>Spain, Latin America ECECP, EBCDIC with the Euro sign</td>
<td>1145</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>CCSID</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>ebc dic-es-284+euro, T1001145</td>
<td>UK ECECP, EBCDIC with the Euro sign</td>
<td>1146</td>
</tr>
<tr>
<td>ibm-1146, IBM01146, CCSID01146, CP01146, cp1146, cpibm1146, ebc dic-gb-285+euro, T1001146</td>
<td>France ECECP, EBCDIC with the Euro sign</td>
<td>1147</td>
</tr>
<tr>
<td>ibm-1147, IBM01147, CCSID01147, CP01147, cp1147, cpibm1147, ebc dic-fr-297+euro, T1001147</td>
<td>International Latin 1, EBCDIC with the Euro sign</td>
<td>1148</td>
</tr>
<tr>
<td>ibm-1148, IBM01148, CCSID01148, CP01148, cp1148, cpibm1148, ebc dic-international-500+euro, T1001148</td>
<td>Iceland, EBCDIC with the Euro sign</td>
<td>1149</td>
</tr>
<tr>
<td>ibm-1149, IBM01149, CCSID01149, CP01149, cp1149, cpibm1149, ebc dic-is-871+euro, T1001149</td>
<td>Latin2 Multilingual with Euro, EBCDIC</td>
<td>1153</td>
</tr>
<tr>
<td>ibm-1153, cpibm1153, T1001153</td>
<td>Cyrillic Multilingual with Euro, EBCDIC</td>
<td>1154</td>
</tr>
<tr>
<td>ibm-1154, cpibm1154, T1001154</td>
<td>EBCDIC Turkey with Euro, EBCDIC</td>
<td>1155</td>
</tr>
<tr>
<td>ibm-1155, cpibm1155, T1001155</td>
<td>EBCDIC Baltic - Multi with Euro, EBCDIC</td>
<td>1156</td>
</tr>
<tr>
<td>ibm-1157, cpibm1157, T1001157</td>
<td>EBCDIC Estonia with Euro, EBCDIC</td>
<td>1157</td>
</tr>
<tr>
<td>ibm-1158, cpibm1158, T1001158</td>
<td>EBCDIC Cyrillic Ukraine with Euro, EBCDIC</td>
<td>1158</td>
</tr>
<tr>
<td>ibm-1160, cpibm1160, T1001160</td>
<td>Thailand EBCDIC with Euro, EBCDIC</td>
<td>1160</td>
</tr>
<tr>
<td>Ibm-1161, windows-874-2000, TIS-620, windows-874, MS874, T1001161</td>
<td>Thai, with the Euro sign</td>
<td>1161</td>
</tr>
<tr>
<td>ibm-1162, TIS-620, Windows-874, T1001162</td>
<td>Windows Thai, ASCII, with the Euro sign</td>
<td>1162</td>
</tr>
<tr>
<td>ibm-1164, cpibm1164, T1001164</td>
<td>Vietnamese, EBCDIC with the Euro sign</td>
<td>1164</td>
</tr>
<tr>
<td>Ibm-1200, UTF-16BE, x-utf-16be, UTF16BE, ibm-1201, ibm-13488, ibm-13489, ibm-17584, ibm-17585, ibm-21680, ibm-21681, ibm-25776, ibm-25777, ibm-29872, ibm-29873, ibm-61955, ibm-61956, windows-1201, cp1200, cp1201, UTF16_BigEndian, T11200</td>
<td>Unicode UTF-16 Big Endian, a Unicode character set with two or 4 bytes encoding units</td>
<td>1200</td>
</tr>
<tr>
<td>Ibm-1202, UTF-16LE, x-utf-16le, ibm-1203, ibm-13490, ibm-13491, ibm-17586, ibm-17587, ibm-21682, ibm-21683, ibm-25778, ibm-25779, ibm-29874, ibm-29875, UTF16_LittleEndian, windows-1200, UTF16LE</td>
<td>Unicode UTF-16 Little Endian, a Unicode character set with two or 4 bytes encoding units</td>
<td>1202</td>
</tr>
<tr>
<td>Ibm-1204, UTF-16, ISO-10646-UCS-2, unicode, ibm-1205, csUnicode, ucs-2, UTF16</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>
<td>1204</td>
</tr>
<tr>
<td>Ibm-1208, UTF-8, UTF8, ibm-1209, ibm-5304, ibm-5305, ibm-13496, ibm-13497, ibm-17592, ibm-17593, windows-65001, cp1208</td>
<td>Unicode UTF-8, a Unicode character set with multibyte characters</td>
<td>1208</td>
</tr>
<tr>
<td>ibm-1232, UTF-32BE, UTF32_BigEndian, ibm-1233, ibm-9424</td>
<td>Unicode UTF-32 Big Endian, a Unicode character set with four-byte encoding units</td>
<td>1232</td>
</tr>
<tr>
<td>ibm-1234, UTF-32LE, UTF32_LittleEndian, ibm-1235</td>
<td>Unicode UTF-32 Little Endian, a Unicode character set with four-byte encoding units</td>
<td>1234</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</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>
<td>1236</td>
</tr>
<tr>
<td>Ibm-1250, ibm-5346, windows-1250, cp1250, T1001250</td>
<td>Windows, Latin 2, ASCII with Euro sign</td>
<td>1250</td>
</tr>
<tr>
<td>Ibm-1251, ibm-5347, windows-1251, cp1251, T1001251</td>
<td>Windows Cyrillic, ASCII with Euro sign</td>
<td>1251</td>
</tr>
<tr>
<td>Ibm1252, ibm-5348, windows-1252, cp1252, T1001252</td>
<td>Windows, Latin 1, ASCII with Euro sign</td>
<td>1252</td>
</tr>
<tr>
<td>Ibm-1253, ibm-5349, windows-1253, cp1253, T1001253</td>
<td>Windows Greek, ASCII with Euro sign</td>
<td>1253</td>
</tr>
<tr>
<td>Ibm-1254, ibm-5350, windows-1254, cp1254, T1001254</td>
<td>Windows Turkish, ASCII with Euro sign</td>
<td>1254</td>
</tr>
<tr>
<td>Ibm-1255, ibm-9447, windows-1255, cp1255</td>
<td>Windows Hebrew, with the Euro sign</td>
<td>1255</td>
</tr>
<tr>
<td>Ibm-1256, windows-1256-2000, windows-1256, cp1256</td>
<td>Windows, Latin 2, ASCII with Euro sign</td>
<td>1256</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>
<td>1257</td>
</tr>
<tr>
<td>Ibm-1258, ibm-5354, windows-1258, cp1258, T1001258</td>
<td>Windows Vietnamese, ASCII with Euro sign</td>
<td>1258</td>
</tr>
<tr>
<td>Ibm-1276, Adobe-Standard-Encoding, csAdobeStandardEncoding, T1001276</td>
<td>Adobe Standard Encoding, ASCII</td>
<td>1276</td>
</tr>
<tr>
<td>Ibm-1277, Adobe-Latin1-Encoding, T1001277</td>
<td>Adobe Latin1 Encoding, ASCII</td>
<td>1277</td>
</tr>
<tr>
<td>Ibm-1364, cp1364</td>
<td>Korean, mixed EBCDIC/DBCS-HOST with Euro sign</td>
<td>1364</td>
</tr>
<tr>
<td>Ibm-1371, cpibm1371</td>
<td>Chinese (Traditional), EBCDIC/DBCS-HOST with the Euro sign</td>
<td>1371</td>
</tr>
<tr>
<td>Ibm-1373, windows-950</td>
<td>Chinese (traditional), Windows BIG5</td>
<td>1373</td>
</tr>
<tr>
<td>Ibm-1375, Big5-HKSCS, big5hk, HKSCS-BIG5, BigSHKSCS</td>
<td>Chinese (traditional)BIG5 with Hong Kong Supplementary Character Set (HKSCS)</td>
<td>1375</td>
</tr>
<tr>
<td>Ibm-1377, T11376, HKSCS-HOST</td>
<td>Chinese (traditional)BIG5 with Hong Kong Supplementary Character Set (HKSCS) for IBM hosts</td>
<td>1377</td>
</tr>
<tr>
<td>Ibm-1381, cp1381, 1381</td>
<td>Chinese (Simplified) GB standard</td>
<td>1381</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>
<td>1383</td>
</tr>
<tr>
<td>Ibm-1386, cp1386, windows-936, ibm-1386_VSUB_VPUA</td>
<td>Chinese (Simplified) GBK PC standard</td>
<td>1386</td>
</tr>
<tr>
<td>Ibm-1388, ibm-9580</td>
<td>Chinese EBCDIC/DBCS-HOST for GBK with the Euro sign</td>
<td>1388</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ibm-1390, cpibm1390</td>
<td>Japanese, mixed EBCDIC/DBCS-HOST with the Euro sign</td>
<td>1390</td>
</tr>
<tr>
<td>ibm-1392, windows-54936, gb18030, GB18030</td>
<td>Chinese GB18030 PC standard</td>
<td>1392</td>
</tr>
<tr>
<td>ibm-1399</td>
<td>Japan, EBCDIC, host MBCS (Latin-Kanji), with the Euro sign</td>
<td>1399</td>
</tr>
<tr>
<td>ibm-4899, cpibm4899</td>
<td>Old EBCDIC Hebrew, with the Euro sign</td>
<td>4899</td>
</tr>
<tr>
<td>ibm-4909</td>
<td>Greek ISO, ASCII with Euro sign</td>
<td>4909</td>
</tr>
<tr>
<td>ibm-4971, cpibm4971</td>
<td>EBCDIC Greek, with the Euro sign</td>
<td>4971</td>
</tr>
<tr>
<td>ibm-5471, MS950_HKSCS, hkbig5, big5-hkscsunidec3.0</td>
<td>Chinese (traditional) Big5-HKSCS-2001 Hong Kong with Unicode 3.0 mappings</td>
<td>5471</td>
</tr>
<tr>
<td>ibm-5478, GB_2312-80, chinese, iso-ir-58, csISO58GBK231280, gb2312-1980, GB2312.1980-0</td>
<td>Extended Unix Code Packed Format for Chinese (Simplified)</td>
<td>5478</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>
<td>9005</td>
</tr>
<tr>
<td>ibm-12712, cpibm12712, ebcdic-he</td>
<td>EBCDIC Hebrew (new sheqel, control characters update), with the Euro sign</td>
<td>12712</td>
</tr>
<tr>
<td>ibm-16684</td>
<td>Japanese, Jis + Roman Jis Host</td>
<td>939</td>
</tr>
<tr>
<td>ibm-16804, cpibm16804, ebcdic-ar</td>
<td>Arabic, EBCDIC with the Euro sign</td>
<td>16804</td>
</tr>
<tr>
<td>ISO-8859-10, iso-ir-157, l6, ISO_8859-10:1992, csISOLatin6, latin6</td>
<td>Nordic</td>
<td></td>
</tr>
<tr>
<td>x-iscii-de, windows-57002, iscii-dev</td>
<td>Windows Devanagari</td>
<td></td>
</tr>
<tr>
<td>x-iscii-be, windows-57003, isci-bng, windows-57006, x-iscii-as</td>
<td>Windows Bengali</td>
<td></td>
</tr>
<tr>
<td>x-iscii-pa, windows-57011, iscii-gur</td>
<td>Windows Punjabi</td>
<td></td>
</tr>
<tr>
<td>x-iscii-gu, windows-57010, iscii-guj</td>
<td>Windows Gujarati</td>
<td></td>
</tr>
<tr>
<td>x-iscii-or, windows-57007, iscii-ori</td>
<td>Windows Oria</td>
<td></td>
</tr>
<tr>
<td>x-iscii-ta, windows-57004, isci-tml</td>
<td>Windows Tamil</td>
<td></td>
</tr>
<tr>
<td>x-iscii-te, windows-57005, isci-tlg</td>
<td>Windows Telugu</td>
<td></td>
</tr>
<tr>
<td>x-iscii-ka, windows-57008, iscii-knd</td>
<td>Windows Kannada</td>
<td></td>
</tr>
<tr>
<td>x-iscii-ma, windows-57009, isci-mlm</td>
<td>Windows Malayalam</td>
<td></td>
</tr>
<tr>
<td>macos-0_2-10.2, macintosh, mac, csMacintosh, windows-10000</td>
<td>Macintosh, ASCII</td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>macos-2566-10.2, Big5-HKSCS, big5hk, HKSCS-BIG5</td>
<td>Chinese (traditional) BIG5 with Hong Kong Supplementary Character Set (HKSCS)</td>
<td></td>
</tr>
<tr>
<td>macos-29-10.2, x-mac-centraleurroman, windows-10029, x-mac-ce, macce</td>
<td>Macintosh, Central Euro, ASCII</td>
<td></td>
</tr>
<tr>
<td>macos-35-10.2, x-mac-turkish, windows-10081, mactr</td>
<td>Macintosh, Turkish, ASCII</td>
<td></td>
</tr>
<tr>
<td>macos-6-10.2, x-mac-greek, windows-10006, macgr</td>
<td>Macintosh, Greek, ASCII</td>
<td></td>
</tr>
<tr>
<td>macos-7_3-10.2, x-mac-cyrillic, windows-10007, maccy</td>
<td>Macintosh, Cyrillic, ASCII</td>
<td></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>
<td></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.

<table>
<thead>
<tr>
<th>Language</th>
<th>Language_Country</th>
</tr>
</thead>
</table>

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>ha</td>
<td>Hausa</td>
<td>rn</td>
<td>Kirundi</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>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>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>ku</td>
<td>Koryu</td>
<td>ta</td>
<td>Tamil</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>Laothian</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>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>
<tr>
<td>Code</td>
<td>Country / Area</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>AFGHANISTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AX</td>
<td>ALAND ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AL</td>
<td>ALBANIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DZ</td>
<td>ALGERIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AS</td>
<td>AMERICAN SAMOA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>ANDORRA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AO</td>
<td>ANGOLA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AI</td>
<td>ANGUILLA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AQ</td>
<td>ANTARCTICA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG</td>
<td>ANTIGUA AND BARBUDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR</td>
<td>ARGENTINA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td>ARMENIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AW</td>
<td>ARUBA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>AUSTRALIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT</td>
<td>AUSTRIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AZ</td>
<td>AZERBAIJAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>BAHAMAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BH</td>
<td>BAHRAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BD</td>
<td>BANGLADESH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>BARBADOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BY</td>
<td>BELARUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>BELGIUM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BZ</td>
<td>BELIZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>BERMUDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>BHUTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BO</td>
<td>BOLIVIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td>BOSNIA AND HERZEGOVINA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW</td>
<td>BOTSWANA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BV</td>
<td>BOUVET ISLAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>BRAZIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>BRITISH INDIAN OCEAN TERRITORY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BN</td>
<td>BRUNEI DARUSSALAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BG</td>
<td>BULGARIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>BURKINA FASO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BI</td>
<td>BURUNDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KH</td>
<td>CAMBODIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM</td>
<td>CAMEROON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>CANADA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV</td>
<td>CAPE VERDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KY</td>
<td>CAYMAN ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF</td>
<td>CENTRAL AFRICAN REPUBLIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TD</td>
<td>CHAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CL</td>
<td>CHILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>CHINA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CX</td>
<td>CHRISTMAS ISLAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>Cocos (Keeling) Islands</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO</td>
<td>COLOMBIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KM</td>
<td>COMOROS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Country</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td>CONGO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>CONGO, THE DEMOCRATIC REPUBLIC OF THE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK</td>
<td>COOK ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>COSTA RICA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CI</td>
<td>COTE D’IVOIRE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HR</td>
<td>CROATIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CU</td>
<td>CUBA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CY</td>
<td>CYPRUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CZ</td>
<td>CZECH REPUBLIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DK</td>
<td>DENMARK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DJ</td>
<td>DJIBOUTI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DM</td>
<td>DOMINICA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td>DOMINICAN REPUBLIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td>ECUADOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EG</td>
<td>EGYPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV</td>
<td>EL SALVADOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GQ</td>
<td>EQUATORIAL GUINEA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ER</td>
<td>ERITREA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>ESTONIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ET</td>
<td>ETHIOPIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FK</td>
<td>FALKLAND ISLANDS (MALVINAS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FO</td>
<td>FAROE ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FJ</td>
<td>FIJI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td>FINLAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td>FRANCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GF</td>
<td>FRENCH GUIANA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PF</td>
<td>FRENCH POLYNESIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TF</td>
<td>FRENCH SOUTHERN TERRITORIES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>GABON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM</td>
<td>GAMBIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>GEORGIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>GERMANY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GH</td>
<td>GHANA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GI</td>
<td>GIBRALTAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GR</td>
<td>GREECE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GL</td>
<td>GREENLAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD</td>
<td>GRENADE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GP</td>
<td>GUADELOUPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GU</td>
<td>GUAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GT</td>
<td>GUATEMALA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GG</td>
<td>GUERNSEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GN</td>
<td>GUINEA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GW</td>
<td>GUINEA-BISSAU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GY</td>
<td>GUYANA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HT</td>
<td>HAITI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HM</td>
<td>HEARD ISLAND AND MCDONALD ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VA</td>
<td>HOLY SEE (VATICAN CITY STATE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HN</td>
<td>HONDURAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK</td>
<td>HONG KONG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HU</td>
<td>HUNGARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>ICELAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>INDIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Country Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>INDONESIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR</td>
<td>IRAN (ISLAMIC REPUBLIC OF)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IQ</td>
<td>IRAQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td>IRELAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>ISLE OF MAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>ISRAEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IT</td>
<td>ITALY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JM</td>
<td>JAMAICA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JP</td>
<td>JAPAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JE</td>
<td>JERSEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JO</td>
<td>JORDAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KZ</td>
<td>KAZAKHSTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KE</td>
<td>KENYA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KY</td>
<td>KIRIBATI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KP</td>
<td>KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KR</td>
<td>KOREA, REPUBLIC OF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KW</td>
<td>KUWAIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KG</td>
<td>KYRGYZSTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>LAO PEOPLE'S DEMOCRATIC REPUBLIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LV</td>
<td>LATVIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB</td>
<td>LEBANON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS</td>
<td>LESOTHO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR</td>
<td>LEBERIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LY</td>
<td>LIBYAN ARAB JAMAHIRIYA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LI</td>
<td>LIECHTENSTEIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT</td>
<td>LITHUANIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td>LUXEMBOURG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MO</td>
<td>MACAO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MK</td>
<td>MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td>MADAGASCAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW</td>
<td>MALAWI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MY</td>
<td>MALAYSIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MV</td>
<td>MALDIVES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>MALI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MT</td>
<td>MALTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MH</td>
<td>MARSHALL ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQ</td>
<td>MARTINIQUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MR</td>
<td>MAURITANIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MU</td>
<td>MAURITIUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YT</td>
<td>MAYOTTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MX</td>
<td>MEXICO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FM</td>
<td>MICRONESIA, FEDERATED STATES OF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD</td>
<td>MOLDOVA, REPUBLIC OF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC</td>
<td>MONACO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MN</td>
<td>MONGOLIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ME</td>
<td>MONTENEGRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS</td>
<td>MONTSERRAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MA</td>
<td>MOROCCO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MZ</td>
<td>MOZAMBIQUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>MYANMAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NA</td>
<td>NAMIBIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR</td>
<td>NAURU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-letter code</td>
<td>Country Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>NEPAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NL</td>
<td>NETHERLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AN</td>
<td>NETHERLANDS ANTILLES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td>NEW CALEDONIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZ</td>
<td>NEW ZEALAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NI</td>
<td>NICARAGUA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>NIGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NG</td>
<td>NIGERIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NU</td>
<td>NIUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>NORFOLK ISLAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>NORTHERN MARIANA ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td>NORWAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OM</td>
<td>OMAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK</td>
<td>PAKISTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW</td>
<td>PALAU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>PALESTINIAN TERRITORY, OCCUPIED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PA</td>
<td>PANAMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG</td>
<td>PAPUA NEW GUINEA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PY</td>
<td>PARAGUAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE</td>
<td>PERU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PH</td>
<td>PHILIPPINES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PN</td>
<td>PITCAIRN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL</td>
<td>POLAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT</td>
<td>PORTUGAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>PUERTO RICO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QA</td>
<td>QATAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RE</td>
<td>REUNION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RO</td>
<td>ROMANIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td>RUSSIAN FEDERATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RW</td>
<td>RWANDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td>SAINT HELENA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KN</td>
<td>SAINT KITTS AND NEVIS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC</td>
<td>SAINT LUCIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM</td>
<td>SAINT PIERRE AND MIQUELON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td>SAINT VINCENT AND THE GRENADINES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td>SAMOA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM</td>
<td>SAN MARINO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>SAO TOME AND PRINCIPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA</td>
<td>SAUDI ARABIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SN</td>
<td>SENEGAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RS</td>
<td>SERBIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>SEYCHELLES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SL</td>
<td>SIERRA LEONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SG</td>
<td>SINGAPORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SK</td>
<td>SLOVAKIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>SLOVENIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB</td>
<td>SOLOMON ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SO</td>
<td>SOMALIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZA</td>
<td>SOUTH AFRICA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GS</td>
<td>SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ES</td>
<td>SPAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LK</td>
<td>SRI LANKA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Country Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>SUDAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR</td>
<td>SURINAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SJ</td>
<td>SVALBARD AND JAN MAYEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td>ST. HELENA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM</td>
<td>ST. PIERRE AND MIQUELON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SZ</td>
<td>SWAZILAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>SWEDEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>SWITZERLAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY</td>
<td>SYRIAN ARAB REPUBLIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TW</td>
<td>TAIWAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TJ</td>
<td>TAJIKISTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TZ</td>
<td>TANZANIA, UNITED REPUBLIC OF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TH</td>
<td>THAILAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TL</td>
<td>TIMOR-LESTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TG</td>
<td>TOGO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TK</td>
<td>TOKELAU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO</td>
<td>TONGA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TT</td>
<td>TRINIDAD AND TOBAGO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TN</td>
<td>TUNISIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR</td>
<td>TURKEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td>TURKMENISTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TC</td>
<td>TURKS AND CAICOS ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TV</td>
<td>TUVALU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UG</td>
<td>UGANDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UA</td>
<td>UKRAINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>UNITED ARAB EMIRATES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GB</td>
<td>UNITED KINGDOM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>UNITED STATES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UM</td>
<td>UNITED STATES MINOR OUTLYING ISLANDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UY</td>
<td>URUGUAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UZ</td>
<td>UZBEKISTAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VU</td>
<td>VANUATU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VA</td>
<td>VATICAN CITY STATE (HOLY SEE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VE</td>
<td>VENEZUELA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VN</td>
<td>VIET NAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VG</td>
<td>VIRGIN ISLANDS (BRITISH)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI</td>
<td>VIRGIN ISLANDS (U.S.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WF</td>
<td>WALLIS AND FUTUNA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EH</td>
<td>WESTERN SAHARA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YE</td>
<td>YEMEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YU</td>
<td>YUGOSLAVIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZM</td>
<td>ZAMBIA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZW</td>
<td>ZIMBABWE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix F. Visual Studio C++ Run-Time Routines

For reference convenience, some Visual Studio C++ routines you may use are listed here by the categories. Refer to Microsoft MSDN Library for more details.

Buffer-Manipulation Routines

<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 given character or given number of characters has been copied</td>
</tr>
<tr>
<td>memchr</td>
<td>Return pointer to first occurrence, within specified number of characters, of given character in buffer</td>
</tr>
<tr>
<td>memcmp</td>
<td>Compare 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 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 specified number of bytes in the buffer</td>
</tr>
<tr>
<td>_swab</td>
<td>Swap bytes of data and store them at 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</td>
<td>True if letter, underscore, or digit</td>
</tr>
<tr>
<td>_iscsymf</td>
<td>True if letter or underscore</td>
</tr>
<tr>
<td>isdigit, iswdigit, _ismbcdigit</td>
<td>True if 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>_ismbchira</td>
<td>True if Hiragana</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>_ismbckata</td>
<td>True if Katakana</td>
</tr>
<tr>
<td>_ismbclegal</td>
<td>True if Legal multibyte character</td>
</tr>
<tr>
<td>_ismbcl0</td>
<td>True if Japan-level 0 multibyte character</td>
</tr>
<tr>
<td>_ismbcl1</td>
<td>True if Japan-level 1 multibyte character</td>
</tr>
<tr>
<td>_ismbcl2</td>
<td>True if Japan-level 2 multibyte character</td>
</tr>
<tr>
<td>_ismbcsymbol</td>
<td>True if Nonalphanumeric multibyte character</td>
</tr>
<tr>
<td>isprint, iswprint, _ismbcprint</td>
<td>True if printable character</td>
</tr>
<tr>
<td>ispunct, iswpunct, _ismbcpunct</td>
<td>True if punctuation</td>
</tr>
<tr>
<td>isspace, iswspace, _ismbcspace</td>
<td>True if white-space</td>
</tr>
<tr>
<td>isupper, iswupper, _ismbcupper</td>
<td>True if uppercase</td>
</tr>
<tr>
<td>iswctype</td>
<td>Property specified by desc argument</td>
</tr>
<tr>
<td>isxdigit, iswxdigit</td>
<td>True if hexadecimal digit</td>
</tr>
<tr>
<td>mblen</td>
<td>Return length of valid multibyte character; result depends on LC_CTYPE category setting of current</td>
</tr>
</tbody>
</table>

### Console and Port I/O Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_cgets, _cgetws</td>
<td>Read string from console</td>
</tr>
<tr>
<td>_cprintf, _cwprintf</td>
<td>Write formatted data to console</td>
</tr>
<tr>
<td>_cputs</td>
<td>Write string to console</td>
</tr>
<tr>
<td>_cscanf, _cwscanf</td>
<td>Read formatted data from console</td>
</tr>
<tr>
<td>_getch, _getwch</td>
<td>Read character from console</td>
</tr>
<tr>
<td>_getche, _getwche</td>
<td>Read character from console and echo it</td>
</tr>
<tr>
<td>_inp</td>
<td>Read one byte from 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 console; use before attempting to read from 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 word to 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 console so it becomes next character read</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 absolute value of 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 string of specified length</td>
</tr>
<tr>
<td>_fcvt</td>
<td>Convert float to string with specified number of digits following decimal point</td>
</tr>
<tr>
<td>_gcvt</td>
<td>Convert float number to string; store string in buffer</td>
</tr>
<tr>
<td>_itoa, _i64toa, _itow, _i64tow</td>
<td>Convert int to string</td>
</tr>
<tr>
<td>labs</td>
<td>Find absolute value of 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>_mbcji стоимjs</td>
<td>Convert Japan Industry Standard (JIS) character to Japan Microsoft (JMS) character</td>
</tr>
<tr>
<td>_mbcmстоjис</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 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 float</td>
</tr>
<tr>
<td>strtol, wcstol</td>
<td>Convert string to long</td>
</tr>
<tr>
<td>strtoul, wcstoul</td>
<td>Convert string to unsigned long integer</td>
</tr>
<tr>
<td>strxfrm, wcxfrm</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 corresponding sequence of multibyte characters</td>
</tr>
</tbody>
</table>
wctomb  Convert wide character to corresponding multibyte character
_wtof   Convert wide-character string to a float
_wtoi, _wtoi64 Convert wide-character string to int or _int64
_wtol   Convert wide-character string to long

Directory-Control Routines

<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 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, _wsearchenv</td>
<td>Search for given file on specified paths</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, _fstati64</td>
<td>Get file-status information on descriptor</td>
</tr>
<tr>
<td>_isatty</td>
<td>Check for character device</td>
</tr>
<tr>
<td>_locking</td>
<td>Lock areas of 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 path name</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 single, full path</td>
</tr>
<tr>
<td>Routine</td>
<td>Use</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>_mktemp,</td>
<td>Create unique filename</td>
</tr>
<tr>
<td>_wmktemp</td>
<td></td>
</tr>
<tr>
<td>_open_osfhandle</td>
<td>Associate C run-time file descriptor with existing operating-system file handle</td>
</tr>
<tr>
<td>remove,</td>
<td>Delete file</td>
</tr>
<tr>
<td>_wremove</td>
<td></td>
</tr>
<tr>
<td>rename,</td>
<td>Rename file</td>
</tr>
<tr>
<td>_wrename</td>
<td></td>
</tr>
<tr>
<td>_splitpath,</td>
<td>Parse path into components</td>
</tr>
<tr>
<td>_wsplitpath</td>
<td></td>
</tr>
<tr>
<td>_stat, _stat64, _stati64, _wstat, _wstat64, _wstati64</td>
<td>Get file-status information on named file</td>
</tr>
<tr>
<td>_umask</td>
<td>Set default permission mask for new files created by program</td>
</tr>
<tr>
<td>_unlink,</td>
<td>Delete file</td>
</tr>
<tr>
<td>_wunlink</td>
<td></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 given file</td>
</tr>
<tr>
<td>_dup2</td>
<td>Create second descriptor for 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 given location</td>
</tr>
<tr>
<td>_open, _wopen</td>
<td>Open file</td>
</tr>
<tr>
<td>_read</td>
<td>Read data from file</td>
</tr>
<tr>
<td>_sopen, _wopen</td>
<td>Open file for file sharing</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 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 character from stream (function versions of getc and getwc)</td>
</tr>
<tr>
<td>_fgetchar, _fgetwchar</td>
<td>Read 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 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 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 string to stream</td>
</tr>
<tr>
<td>fread</td>
<td>Read unformatted data from 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 stream</td>
</tr>
<tr>
<td>fseek</td>
<td>Move file position to given location</td>
</tr>
<tr>
<td>fsetpos</td>
<td>Set position indicator of stream</td>
</tr>
<tr>
<td>_fsopen, _wfsopen</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 character from stream (macro versions of fgetc and fgetwc)</td>
</tr>
<tr>
<td>getchar, getwchar</td>
<td>Read character from stdin (macro versions of fgetchar and 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, getws</td>
<td>Read line from stdin</td>
</tr>
<tr>
<td>Routine</td>
<td>Use</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>__getw</td>
<td>Read binary int from stream</td>
</tr>
<tr>
<td>perror</td>
<td>Displays a string version of the current error to STDERR</td>
</tr>
<tr>
<td>printf, wprintf</td>
<td>Write formatted data to stdout</td>
</tr>
<tr>
<td>putc, putwc</td>
<td>Write character to a stream (macro versions of fputc and fputwc)</td>
</tr>
<tr>
<td>putchar, putwchar</td>
<td>Write character to stdout (macro versions of fputchar and fputwchar)</td>
</tr>
<tr>
<td>puts, __putws</td>
<td>Write line to stream</td>
</tr>
<tr>
<td>__putw</td>
<td>Write binary int to stream</td>
</tr>
<tr>
<td>remove</td>
<td>Erase a file</td>
</tr>
<tr>
<td>rename</td>
<td>Rename a file</td>
</tr>
<tr>
<td>rewind</td>
<td>Move file position to beginning of stream</td>
</tr>
<tr>
<td>__rmtmp</td>
<td>Remove temporary files created by tmpfile</td>
</tr>
<tr>
<td>scanf, wscanf</td>
<td>Read formatted data from stdin</td>
</tr>
<tr>
<td>setbuf</td>
<td>Control stream buffering</td>
</tr>
<tr>
<td>__setmaxstdio</td>
<td>Set a maximum for the number of simultaneously open files at the stream I/O level.</td>
</tr>
<tr>
<td>setvbuf</td>
<td>Control stream buffering and buffer size</td>
</tr>
<tr>
<td>__snprintf, __snwprintf</td>
<td>Write formatted data of specified length to string</td>
</tr>
<tr>
<td>__snscanf, __snwscanf</td>
<td>Read formatted data of a specified length from the standard input stream.</td>
</tr>
<tr>
<td>sprintf, swprintf</td>
<td>Write formatted data to string</td>
</tr>
<tr>
<td>sscanf, swscanf</td>
<td>Read formatted data from string</td>
</tr>
<tr>
<td>__tempnam, __wtempnam</td>
<td>Generate temporary filename in given directory</td>
</tr>
<tr>
<td>tmpfile</td>
<td>Create temporary file</td>
</tr>
<tr>
<td>tmpnam, __wtmpnam</td>
<td>Generate temporary filename</td>
</tr>
<tr>
<td>ungetc, ungetwc</td>
<td>Push character back onto stream</td>
</tr>
<tr>
<td>vfprintf, vfwprintf</td>
<td>Write formatted data to stream</td>
</tr>
<tr>
<td>vprintf, vwprintf</td>
<td>Write formatted data to stdout</td>
</tr>
<tr>
<td>__vsnprintf, __vsnwprintf</td>
<td>Write formatted data of specified length to buffer</td>
</tr>
<tr>
<td>vsprintf, vswprintf</td>
<td>Write formatted data to buffer</td>
</tr>
</tbody>
</table>

**String-Manipulation Routines**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>__mbscoll, __mbsicoll, __mbsncoll, __mbsnicoll</td>
<td>Compare two multibyte-character strings using multibyte code page information (__mbsicoll and __mbsnicoll are case-insensitive)</td>
</tr>
<tr>
<td>__mbsdec, __strdec, __wcsdec</td>
<td>Move string pointer back one character</td>
</tr>
<tr>
<td>__mbsinc, __strinc, __wcsinc</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>_mbslen</td>
<td>Get number of multibyte characters in multibyte-character string; dependent upon OEM code page</td>
</tr>
<tr>
<td>_mbsnbcat</td>
<td>Append, at most, first n bytes of one multibyte-character string to another</td>
</tr>
<tr>
<td>_mbsnbcmp</td>
<td>Compare first n bytes of two multibyte-character strings</td>
</tr>
<tr>
<td>_mbsnbcnt</td>
<td>Return number of multibyte-character bytes within supplied character count</td>
</tr>
<tr>
<td>_mbsnbcpy</td>
<td>Copy n bytes of string</td>
</tr>
<tr>
<td>_mbsnbicmp</td>
<td>Compare n bytes of two multibyte-character strings, ignoring case</td>
</tr>
<tr>
<td>_mbsnbset</td>
<td>Set first n bytes of multibyte-character string to specified character</td>
</tr>
<tr>
<td>_mbsncnt</td>
<td>Return number of multibyte characters within supplied byte count</td>
</tr>
<tr>
<td>_mbsnextc, _strnextc, wcsnextc</td>
<td>Find next character in string</td>
</tr>
<tr>
<td>_mbssninc, _strninc, wcsninc</td>
<td>Advance string pointer by n characters</td>
</tr>
<tr>
<td>_mbssspnp, _strspnp, wcsspnp</td>
<td>Return pointer to first character in given string that is not in another given string</td>
</tr>
<tr>
<td>_mbstrlen</td>
<td>Get number of multibyte characters in multibyte-character string; locale-dependent</td>
</tr>
<tr>
<td>_scprintf, _scwprintf</td>
<td>Return the number of characters in a formatted string</td>
</tr>
<tr>
<td>_snscanf, _snwscanf</td>
<td>Read formatted data of a specified length from the standard input stream.</td>
</tr>
<tr>
<td>sprintf, _stprintf</td>
<td>Write formatted data to a string</td>
</tr>
<tr>
<td>strcat, wcscat, _mbscat</td>
<td>Append one string to another</td>
</tr>
<tr>
<td>strchr, wcschr, _mbschr</td>
<td>Find first occurrence of specified character in string</td>
</tr>
<tr>
<td>strcmp, wcscmp, _mbscmp</td>
<td>Compare two strings</td>
</tr>
<tr>
<td>strcoll, wcscoll, _stricoll, _wcsicoll, _strnicoll, _wcsnicoll</td>
<td>Compare two strings using current locale code page information (_stricoll, _wcsicoll, _strnicoll, and _wcsnicoll are case-insensitive)</td>
</tr>
<tr>
<td>strcpy, wcscpy, _mbscopy</td>
<td>Copy one string to another</td>
</tr>
<tr>
<td>strcsppn, wcscsppn, _mbcsppn</td>
<td>Find first occurrence of character from specified character set in string</td>
</tr>
<tr>
<td>_strdup, _wcsdup, _mbsdup</td>
<td>Duplicate string</td>
</tr>
<tr>
<td>strerror, _wcserro</td>
<td>Map error number to message string</td>
</tr>
<tr>
<td>_strerror, _wcserro</td>
<td>Map user-defined error message to string</td>
</tr>
<tr>
<td>strftime, wcsftime</td>
<td>Format date-and-time string</td>
</tr>
<tr>
<td>_stricmp, _wcsicmp, _mbscmp</td>
<td>Compare two strings without regard to case</td>
</tr>
<tr>
<td>strlen, wcslen, _mbslen, _mbstrlen</td>
<td>Find length of string</td>
</tr>
<tr>
<td>_strlwr, _wcsllwr, _mbslwr</td>
<td>Convert string to lowercase</td>
</tr>
</tbody>
</table>
### strncat, wcsncat, _mbsncat
Append characters of string

### strncmp, wcsncmp, _mbsncmp
Compare characters of two strings

### strncpy, wcsncpy, _mbsncpy
Copy characters of one string to another

### _strnicmp, _wcsnicmp, _mbsnicmp
Compare characters of two strings without regard to case

### _strncpy, _wcsnset, _mbsnset
Set first \( n \) characters of string to specified character

### strstr, wcsstr, _mbsstr
Find first occurrence of specified string in another string

### strtok, wcstok, _mbstok
Find next token in string

### _strupr, _wcsupr, _mbsupr
Convert string to uppercase

### strxfrm, wcsxfrm
Transform string into collated form based on locale-specific information

### vsprintf, _vstprintf
Write formatted output using a pointer to a list of arguments

---

### Time Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>asctime, _wasctime</td>
<td>Convert time from type \texttt{struct tm} to character string</td>
</tr>
<tr>
<td>clock</td>
<td>Return elapsed CPU time for process</td>
</tr>
<tr>
<td>ctime, _ctime64, _wctime, _wctime64</td>
<td>Convert time from type \texttt{time_t} or \texttt{_time64_t} to character string</td>
</tr>
<tr>
<td>difftime</td>
<td>Compute difference between two times</td>
</tr>
<tr>
<td>_ftime, _ftime64</td>
<td>Store current system time in variable of type \texttt{struct _timeb} or \texttt{type struct _timeb64}</td>
</tr>
<tr>
<td>_ftime64</td>
<td></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 \texttt{time_t} to \texttt{struct tm} or from type \texttt{_time64_t} to \texttt{struct tm}</td>
</tr>
<tr>
<td>localtime, _localtime64</td>
<td>Convert time from type \texttt{time_t} to \texttt{struct tm} or from type \texttt{_time64_t} to \texttt{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 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 string</td>
</tr>
</tbody>
</table>
time, _time64
_getctime64
_tzset
_utime, _utime64,
_wutime, _wutime64
Set current system time as type time_t or as type _time64_t
Set external time variables from environment time variable TZ
Set modification time for 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>