FastPDF Formatter Reference

Version 3.1
This edition applies to the FastPDF Formatter.

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

support@makeafp.com

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

Chapter 1. FastPDF Formatter Functions ................................................................. 1
  Using the FastPDF Formatter Function Calls to Build a PDF Document ..................1
  Hierarchy of FastPDF Formatter Calls ....................................................................1
    Session Level calls .............................................................................................2
    Document Level Calls .........................................................................................2
    Page-level Calls ................................................................................................2
    Format of the Function Call Descriptions ..........................................................3
  2D Aztec Barcode ..................................................................................................5
  2D DataMatrix Barcode .........................................................................................6
  2D MaxiCode Barcode ..........................................................................................7
  2D MicroPDF417 Barcode .....................................................................................9
  2D PDF417 Barcode .............................................................................................10
  2D PDF417 Truncated Barcode ............................................................................11
  2D QR Code Barcode ...........................................................................................12
  Angle Text ...........................................................................................................14
  Arc Drawing .........................................................................................................16
  Barcode (Linear) ...................................................................................................18
  Barchart ...............................................................................................................20
  Barchart Element Label .......................................................................................22
  Barchart Legend Label ........................................................................................24
  Barchart - Stacked Barchart .................................................................................25
  Blank Page ..........................................................................................................28
  Bookmark ...........................................................................................................29
  Box Drawing ........................................................................................................31
  Cap Height of Font (Query) ................................................................................32
  Center Align Text ................................................................................................33
  Centimeter Value .................................................................................................34
  Character Space ..................................................................................................35
  Chart Colors .........................................................................................................36
  Chord Drawing ....................................................................................................38
  Circle Drawing ....................................................................................................39
  Close Document – Output PDF to File or Memory ............................................40
  Close Document – Output PDF as Attachment of Email ......................................42
  Close Document – Split PDF Pages ....................................................................44
  Close Page ..........................................................................................................45
  Color of Fill ..........................................................................................................46
  Color of Stroke ....................................................................................................48
Color Space Management ................................................................. 50
Compression Filter ........................................................................ 51
Compression Level ......................................................................... 52
Decrypt Input PDF Document ....................................................... 53
Default Input Data Encoding .......................................................... 54
Default Language Locale ............................................................... 55
Define an Inline Overlay Object ................................................... 56
Delete Page .................................................................................. 58
Document Description .................................................................... 59
Edit Page ...................................................................................... 60
End of Document File of Input PDF (Checking) ............................... 61
Ellipse Drawing ........................................................................... 62
Encrypt Output PDF Document ..................................................... 63
File Name of Default Input/Output File (Query) ............................... 64
Font Definition ............................................................................. 65
Font Changes ................................................................................ 68
Get Page – Getting PDF Page From Input Document .................... 70
Get Input PDF Page Width, Height and Number of Pages ............... 71
Get Text Field by a Location ........................................................ 72
Get Text Field by a Location Range ............................................. 75
Get Text Field by a Location Range and a Pettern ........................ 76
Get Image Information .................................................................. 78
Graphic Path Drawing .................................................................. 79
Graphic Position .......................................................................... 81
Horizontal Boxes ......................................................................... 82
Horizontal Line ............................................................................ 84
Horizontal Lines .......................................................................... 85
Image Resolution .......................................................................... 86
Lines (Multiple Connected Lines) .................................................. 87
Line Ending Style .......................................................................... 88
Line Join Style ............................................................................. 89
Line Type ...................................................................................... 90
Line Width ................................................................................... 92
Inch Value ................................................................................... 93
Import Image ................................................................................ 94
Import PDF File ........................................................................... 96
Include Image .............................................................................. 97
Include Overlay ............................................................................ 99
JPEG Image Quality ..................................................................... 101
Left Align Text ............................................................................ 102
Linechart ..................................................................................... 103
Graphic Linechart Legend ........................................................... 106
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version of PDF Format</td>
<td>168</td>
</tr>
<tr>
<td>Vertical Boxes</td>
<td>169</td>
</tr>
<tr>
<td>Vertical Line</td>
<td>171</td>
</tr>
<tr>
<td>Vertical Lines</td>
<td>172</td>
</tr>
<tr>
<td>View PDF File</td>
<td>173</td>
</tr>
<tr>
<td>Web Link</td>
<td>174</td>
</tr>
<tr>
<td>Word Space</td>
<td>175</td>
</tr>
<tr>
<td>X Absolute Position of Text</td>
<td>176</td>
</tr>
<tr>
<td>X Current Position of Text (Query)</td>
<td>177</td>
</tr>
<tr>
<td>X Relative Move Position of Text</td>
<td>178</td>
</tr>
<tr>
<td>Y Absolute Position of Text</td>
<td>179</td>
</tr>
<tr>
<td>Y Current Position of Text (Query)</td>
<td>180</td>
</tr>
<tr>
<td>Y Relative Move Position of Text</td>
<td>181</td>
</tr>
<tr>
<td><strong>Chapter 2. FastPDF Formatter Parameters</strong></td>
<td>182</td>
</tr>
<tr>
<td>FONTLIB – Specifies the Library Path of Fonts</td>
<td>182</td>
</tr>
<tr>
<td>OPENPW – Specifies the PDF Open Password</td>
<td>182</td>
</tr>
<tr>
<td>OWNERPW – Specifies the PDF Open Password</td>
<td>183</td>
</tr>
<tr>
<td>RESLIB – Specifies the Library Path of Images and PDF Templates</td>
<td>183</td>
</tr>
<tr>
<td><strong>Chapter 3. FastPDF Formatter Variables</strong></td>
<td>184</td>
</tr>
<tr>
<td>$Edt – End of PDF Document</td>
<td>184</td>
</tr>
<tr>
<td>$FileInName – Filename of the Default Input Data File</td>
<td>184</td>
</tr>
<tr>
<td>$PdOutFile – Filename of the Output PDF File</td>
<td>184</td>
</tr>
<tr>
<td>$PdfOutPath – Path Name of the PDF Output</td>
<td>184</td>
</tr>
<tr>
<td><strong>Chapter 4. String Manipulation Functions</strong></td>
<td>185</td>
</tr>
<tr>
<td>Comma Float</td>
<td>185</td>
</tr>
<tr>
<td>Comma Integer</td>
<td>186</td>
</tr>
<tr>
<td>Comma Digital String</td>
<td>186</td>
</tr>
<tr>
<td>Delete Characters</td>
<td>187</td>
</tr>
<tr>
<td>Extract Substrings</td>
<td>187</td>
</tr>
<tr>
<td>Find String</td>
<td>189</td>
</tr>
<tr>
<td>First Character</td>
<td>189</td>
</tr>
<tr>
<td>In Substitution Table</td>
<td>190</td>
</tr>
<tr>
<td>Insert String</td>
<td>190</td>
</tr>
<tr>
<td>Is Empty</td>
<td>191</td>
</tr>
<tr>
<td>Last Character</td>
<td>191</td>
</tr>
<tr>
<td>Left Copy</td>
<td>192</td>
</tr>
<tr>
<td>Left Copy and Pad</td>
<td>192</td>
</tr>
<tr>
<td>Left Copy and Right Trim</td>
<td>193</td>
</tr>
<tr>
<td>Left Trim</td>
<td>193</td>
</tr>
<tr>
<td>Left Trim for EBCDIC</td>
<td>194</td>
</tr>
</tbody>
</table>
Chapter 1. FastPDF Formatter Functions

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

Using the FastPDF Formatter Function Calls to Build a PDF Document

A typical sequence of FastPDF Formatter function calls for a PDF application is as follows:

1. Initialize a FastPDF Formatter environment by calling the “Start” function. The “Start” session function must be called before using any other FastPDF Formatter functions.
2. Call the “Open Document” function.
3. Set default measurement units for the whole job by calling the “Set Unit” function.
4. Call the “Open Page” or “Get Page” function by either adding a new PDF page or reading an existing PDF page.
5. Specify a trigger to determine where to locate a text or page if the input file is a PDF document.
6. Get a text string by its location from an input PDF page, it can be for reengineering purposes, like adding dynamic new barcodes, bookmarks, etc.
7. 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.
8. If needed, specify any changes to the attributes of the data by calling the “Set Font” and “Set Color” functions before placing the data.
9. Compose the page using the appropriate function call:
   – Place a line of the text string in left, right, or center alignment.
   – Place the fixed or dynamic text paragraph in left, right, center, and fully justify alignment.
   – Draws line or box at a fixed or dynamic position.
   – Place the barcode at a fixed or dynamic position.
   – Include resource objects, such as images and overlays at a fixed or dynamic position.
   – Plots the graphic box, line, color area, barchart, or piechart.
10. Repeat steps from 5 to 9 until the page is finished.
11. Call the “Close Page” function.
12. Repeat steps 4 through 11 until all pages are done.
13. Call the “Close Document” function.

Hierarchy of FastPDF Formatter Calls

FastPDF Formatter has three levels of function calls:

- **Session Level Calls**
  These calls start the FastPDF Formatter session, such as setting the overall session measurement unit, view the generated PDF file if needed.

- **Document Level Calls**
  These calls open and close a PDF document.
• **Page-Level Calls**
  These calls open and close a PDF page, and format data or edit PDF within individual pages.

**Session Level calls**

- **Set Default Measurement Unit**
  Defines the default measurement unit for the whole job.

- **Start Session**
  Starts the FastPDF Formatter session.

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

**Document Level Calls**

- **Open Document**
  Opens a PDF document.

- **Close Document**
  Closes a PDF document.

**Page-level Calls**

- **Open Page**
  Opens a PDF page.

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

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

- **Include images and overlays**
  Includes the images and overlays as the object resources.

- **Put Text**
  Puts a line of text.

- **Put Paragraph**
  Puts a text paragraph with alignment control.

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

- **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 for text.

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

- **Set Vertical Y Position**
  Specifies a vertical Y position for text.
Query Position
Queries current X or Y text position.

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

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

Set Font
Specifies the font for subsequent text data.

Draw Linear Barcode
Draws a linear barcode.

Draw 2D Barcode Data
Draw a 2D barcode.

Plot Graphic
Plots graphic line, box, patterned color area, circle, arc, ellipse, etc.

Plot Barchart
Plots a graphic barchart.

Plot Piechart
Plots a graphic piechart.

Set Trigger
Specifies a trigger to determine where to identify a text data or page from the input PDF document.

Get Data Field
Gets a text string by its location from the input PDF document, to be used for the PDF reengineering purposes.

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 must also have default arguments. In other words, you cannot omit a middle parameter.

MakePDF provides default values to rarely used parameters to simplify the use of the FastPDF Formatter function.
2D Aztec Barcode

Function
Generates Aztec 2D barcode.

Syntax

```c
void Aztec(
    char*    data,
    float    x_pos,
    float    y_pos,
    UI16    symbol_size = 0,
    UI16    security_mode = 0,
    float    scale = 1.0,
    UI16    degree = 0
);
```

Parameters

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

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

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

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

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

degree
The rotation of 2D barcode image. Valid values are 0, 90, 180, and 270.

Sample:

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

Output:
2D DataMatrix Barcode

Function
Generates DataMatrix 2D barcode.

Syntax
void DataMatrix(
    char*    data,
    float    x_pos,
    float     y_pos,
    UI16    symbol_size = 0,
    UI16    security_mode = 0,
    float    scale = 1.0,
    UI16    degree = 0,
);

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 left most 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 though 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. Valid values are 0, 90, 180, and 270.

Sample:
char *data = "1234567890 this is testing of DataMatrix";
SetUnit(INCH);
OpenDoc();
    OpenPage(8.5,11);
        DataMatrix(data,1.2,1.5);
    ClosePage();
CloseDoc();

Output:
2D MaxiCode Barcode

Function
Generates MaxiCode 2D barcode.

Syntax

```c
void MaxiCode(
    char* data,
    float x_pos,
    float y_pos,
    mode symbol_mode = 4,
    char* postal_data = NULL,
    UI16 degree = 0
);
```

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>Structured Carrier Message for additional numeric postal code</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</td>
<td>84</td>
<td>126</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>Structured Carrier Message</td>
<td>84</td>
<td>126</td>
<td>50</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>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>Full ECC, like MODE4 but with Enhanced Error Correction</td>
<td>77</td>
<td>110</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>Reserved for the maintenance of scanner hardware</td>
<td>93</td>
<td>135</td>
<td>50</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)</td>
</tr>
</tbody>
</table>
mode 3). The remaining unused characters should be filled with the BLANK character (ASCII '20'x)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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. Valid values are 0, 90, 180, and 270.

**Sample**

```c
char *data = "1234567890 This is testing of MaxiCode";
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
MaxiCode(data,       // Barcode data
         1,         // Barcode x position at 1" 
         1,        // Barcode Y position at 1" 
         
ClosePage();
CloseDoc();
```

**Output:**

![MaxiCode Barcode Image]
2D MicroPDF417 Barcode

Function

Generates MicroPDF417 2D barcode.

Syntax

```c
void MPDF417(
    char*       data,
    float       x_pos,
    float       y_pos,
    float width = 0,
    float scale = 1.0,
    UI16        degree = 0
);
```

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. Valid values are 0, 90, 180, and 270.

Sample:

```c
char *data = "1234566789 MicroPDF417";
SetUnit(INCH);
OpenDoc();
    OpenPage(8.5,11);
    MDF417(data,1.2,1.5);
ClosePage();
CloseDoc();
```

Output:

```
|   |
|___|
```

- 9 -
2D PDF417 Barcode

Function

Generates PDF417 2D barcode.

Syntax

```c
void PDF417(
    char*     data,
    float     x_pos,
    float     y_pos,
    float     width = 0,
    UI16      security = 0,
    float     scale = 1.0,
    UI16      degree = 0
);
```

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
In 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. Valid values are 0, 90, 180, and 270.

Sample:

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

Output:
2D PDF417 Truncated Barcode

Function
Generates PDF417 Truncated 2D barcode.

Syntax
```c
void PDF417T(
    char*      data,
    float        x_pos,
    float      y_pos,
    float       width = 0,
    UI16        security = 0,
    float     scale = 1.0,
    UI16      degree = 0
);
```

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**
In 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. Valid values are 0, 90, 180, and 270.

Sample:
```c
char *data = "1234567890 this is testing of PDF417 Truncated";
SetUnit(INCH);
OpenDoc();
    OpenPage(8.5,11);
    PDF417T(data,1.2,1.5);    // position at (1.2",15")
ClosePage();
CloseDoc();
```

Output:
2D QR Code Barcode

Functions

Generate the QR Code 2D barcode and Micro QR Code 2D barcode.

Syntax

```c
void QRCode(
    char* data,
    float x_pos,
    float y_pos,
    float symbol_size = 0,
    UI16 security = 0,
    float scale = 1.0,
    bool to_utf8 = true,
    UI16 degree = 0
);```

Micro QR Code for short messages:

```c
void MQRCode(
    char* data,
    float x_pos,
    float y_pos,
    float symbol_size = 0,
    UI16 security = 0,
    float scale = 1.0,
    bool to_utf8 = true,
    UI16 degree = 0
);```

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.

to_utf8
Specifies whether convert input data string to UTF-8:
TRUE Converts legacy encoding data to UTF-8 by FastPDF Formatter, make sure that the default input data encoding is defined properly by the function of DefaultCode() first, otherwise the default input data encoding “Windows-1252” is being used for the internal data encoding conversion.

FALSE Input data is already in UTF-8 encoding.

degree
The rotation for the barcode. Valid values are 0, 90, 180, and 270.

Sample:

```c
char *data = "1234567890 this is testing of QR Code";
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);

QRCode(data,1.2,1.5); // position at (1.2",15")

ClosePage();
CloseDoc();

Output:
```

![QR Code Image]
Angle Text

Function

Rotates a line text string in a specifying angle.

Syntax

```c
void AngleTxt(
    char * text,
    float  xpos,
    float  ypos,
    float  angle,
    float  radius = 0,
    float  axis_y = 0
);
```

```c
void AngleU8(
    UChar8 * utf8_text,
    float  xpos,
    float  ypos,
    float  angle,
    float  radius = 0,
    float  axis_y = 0
);
```

```c
void AngleU16(
    UChar * utf16le_text,
    float  xpos,
    float  ypos,
    float  angle,
    float  radius = 0,
    float  axis_y = 0
);
```

Parameters

text
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

utf8_text
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_text
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

x_pos, y_pos
The position of the top left corner of the text.

angle
The angle in degrees.

radius
Specifies a radius value for move the text string in the rotated coordinate system on the x-axis, so
that the text string will be presented on a radius.

**axis_y**
Specifies a value for move the text string also in the rotated coordinate system on the y-axis. The origin is measured in bottom-up coordinates.

**Sample**

```cpp
SetUnit(MM);
OpenDoc();
OpenPage(DIN_A4);

Font("Arial", 16, BOLD);
int angle = 0;
float x = PageWidth()*0.5;
float y = PageHeight()*0.35;
for (int i = 0; i < 12; i++)
{
    SetColor(0x80C080 | (SI32)angle << 6);
    AngleTxt("Circlat PDF Text", x, y, angle, 12, CapHeight()*0.5);
    angle += 30.0;
}
ClosePage();
CloseDoc();
```

**Output:**

![Output Image]
Arc Drawing

Function

Plots a graphic arc.

The thickness and type of border can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GArc(
    float     x_pos,
    float     y_pos,
    float      radius,
    float      start_angle,
    float      end_angle,
    PathFill_mode     fill_mode = STROKE_PATH
);  
void GArc2(
    float     x_pos,
    float     y_pos,
    float      width,
    float      height,
    float      start_angle,
    float      end_angle,
    PathFill_mode     fill_mode = STROKE_PATH
);  
```

Parameters

- **x_pos, y_pos**
The (X, Y) coordinate of the center of the arc or midpoint of the bounding rectangle.

- **radius**
The radius of the arc.

- **Width, height**
The width and height of the bounding rectangle.

- **start_angle**
The start angle is measured anticlockwise.

- **end_angle**
The end angle is measured anticlockwise.

- **fill_mode**
In the graphic filling mode, you may use the following values:
  
  - FILL_PATH: Fill color into the arc does not stroke arc boundary
  - FILL_STROKE: Fill color into the arc and stroke arc boundary
  - STROKE_PATH: Stroke arc boundary does not fill color into the arc

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage();

ColorRGB(165,255,165);  // Set a RGB fill color
StrokeColor(DARKBLUE);  // Set a stroke color by color alias
```
\texttt{GArc(100, 100, 40, 0, 140, FILL\_STROKE);}

\texttt{ColorRGB(255,165,165);} \\
\texttt{GArc2(100, 180, 80, 120, 0, 140, FILL\_STROKE);} \\
\texttt{ClosePage();}
\texttt{CloseDoc();}

\textbf{Output:}

\begin{center}
\includegraphics[width=0.5\textwidth]{example.png}
\end{center}
Barcode (Linear)

Function

Generates 1D linear barcode, it 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(
    BarType barcode_type,
    char*  data,
    float x_pos,
    float y_pos,
    float width,
    float height,
    UI16  degree = 0,
);
```

Parameters

**Barcode_type**

The barcode encoding, followings are supported:

- **CDB2OF7**  AIM USS-Codabar, Codabar 2-of-7
- **CODE11**  Code 11
- **CODE128**  CODE 128, A, B, and C auto-switching mode
- **CODE128B**  CODE 128, Set B, for suppress mode C in favor of mode B
- **CODE32**  Code 32, up to 8 digits
- **CODE39**  AIM USS-39, Code 39 (3 of 9)
- **CODE39E**  Code 39 (3 of 9) Extended (full text)
- **CODE93**  Code 93
- **DL2OF5**  Data Logical Code 2 of 5
- **DPIENT**  Deutsche Post Identcode, 11 digits
- **DPLEIT**  Deutsche Post Leitcode, 13 digits
- **EAN128**  EAN 128
- **EAN14**  EAN-14, 13 digits
- **IATA2OF5**  IATA Code 2 of 5
- **IND2OF5**  Industrial Code 2 of 5
- **ITF14**  ITF-14, 13 digits
- **ITL2OF5**  Interleaved Code 2-of-5
- **LOGMARS**  LOGMARS
- **MAT2OF5**  Matrix Code 2-of-5
- **MSI**  MSI Plessey
- **PHARMA**  Pharmacode One-Track
- **PLESSEY**  PLESSEY (an older code still popular in some industries)
- **TELEPENA**  Telepen Alpha
- **TELEPENN**  Telepen Numeric
- **APOST**  Australia Post Standard customer allows 8 Digits,
  8 digits followed by 5 characters, 16 digits,
  8 digits followed by 10 characters, 23 digits
APOSTRD  Australia Post Redirection, 8 digits
APOSTRP  Australia Post Reply Paid, 8 digits
APOSTRT  Australia Post Routing, 8 digits
DPOST    Dutch Post KIX, 11 characters
JPOST    Japan Postal barcode
KPOST    Korea Postal barcode
POSTNET  PostNet
PLANET   Planet
RM4SCC   Royal Mail 4 State
SPOST4   Singapore Postal 4-state barcode
USPS4S   USPS 4-state postal barcode, 20 digits, 5, 9, or 11 digits zip code can be appended using the + character
EAN      EAN, EAN-2/EAN-5/EAN-8/EAN-13, 2, 5, 7, and 12 digits, EAN-2 2 digits or EAN-5 5 digits can be appended using the + character
UPCA     UPC-A, 11 digits, EAN-2 2 digits or EAN-5 5 digits can be appended using the + character
UPCE     UPC-E, 6 digits, also 7 digits starting with 1, EAN-2 2 digits or EAN-5 5 digits can be appended using the + character

data
Either ASCII or EBCDIC input data. Make sure the PRMODE parameter is specified in your MakePDF 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 left most element of the barcode symbol.

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

degree
The rotation for the barcode. The valid values are 0, 90, 180, and 270.

Sample:

char *data = "123456789012345678901234567890";
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
:
:
BarCode(CODE128,   // Barcode type is Code 128
  data,                  // Input data field
  1,                     // barcode X position at 1"
  1,                     // barcode Y position at 1"
  2,                     // barcode dimension width 2"
  0.35)  // barcode dimension height 0.35"
  :
  :
ClosePage();
CloseDoc();

Output:
**Barchart**

**Function**

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

With the `GBarchartElement()` and `GBarchartLegend()` functions, you can control how to place the legend labels and indicate whether to present the element labels or not.

With the `GChartColors()` function, you can override the default RGB color values predefined for the graphic barchart and piechart.

**Syntax**

```c
void GBarchart(
    float height,  // The height of the barchart graphic.
    float bar_width,  // The width of each bar.
    float gap_barset,  // The gap of each bar set.
    uint vertical_scale_unit,  // The unit size of each scale on the vertical axis.
    ushort vertical_scales,  // The number of scales on the vertical axis.
    ushort number_bars,  // The number of bar elements in each bar set.
    ushort bar_sets,  // The number of bar-sets.
    float* values,  // The 1 dimension or 2 dimensions array of the data values.
    char* horizontal_labels[],  // The array of label texts for the horizontal axis. You can insert a new line control code \n or \x0a to split the text into multiple lines on output. This value is ignored if your number of bars is 1.
    char* legend_texts[],  // The array of legend texts.
    bool 3D_barchart = TRUE,  // Specify whether it is a 3D barchart or not, the default is TRUE, plots the 3D barchart.
    bool gridline = TRUE
);```

**Parameters**

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

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

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

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

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

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

- **Bar_sets**
  The number of bar-sets.

- **values**
  The 1 dimension or 2 dimensions array of the data values.

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

- **legend_texts**
  The array of legend texts.

- **3D_barchart**
  Specify whether it is a 3D barchart or not, the default is TRUE, plots the 3D barchart.
**legend_texts**
The array of texts for the legend.

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

**Sample**

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

char *lbs[] = {"Mon\nHoliday 1", "Tue\nHoliday 2", "Wed", "Thu",
               "Fri", "Sat", "Sun"};

Start();
SetUnit(INCH);                // Set default unit to inch
OpenDoc();
OpenPage(8.5,11);              // Page size is 8.5" by 11"
LPI(8);                       // Line space is 8 LPI
Font("Arial", 8);
GPos(1, 2.3);                 // Set position of barchart
GBarchart(2,                  // Max height of barchart
          0.15,               // Width of each bar
          0.2,                // Gap between each bar sets
          10,                 // Unit size of v-axis scale
          10,                 // Number of v-axis scales
          3,                  // Number of bars per bar set
          7,                  // Number of bar set
          &data1[0][0],       // Data array of input data
          lbs,               // Label texts below H-axis
          legend);           // Legend texts
ClosePage();
CloseDoc();
```

**Output:**

![Graph output](image)
Barchart Element Label

Function

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

This function must be called before the Barchart function.

Syntax

void GBarchartElement(
    ElementLabelPos pos = ELEMENT_LABEL_NONE,
    char*     format = "%.1f",
    UI16      fontID = DEFAULT
);

Parameters

element_label_position
Specifies where to place the bar element labels, valid values are:

- ELEMENT_LABEL_ABOVE Place labels on top of each bar
- ELEMENT_LABEL_MIDDLE Place labels at the middle of each bar
- ELEMENT_LABEL_NONE No element labels

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

fontid
Specifies a font handle number to be used to present the bar element labels. Default is using the currently active font.

You can call the function of Font() to specify a font that returns its font handle number for you.

Sample

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

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

char *legend[] = { "East", "West", "North" };
SetUnit(INCH);  // Set default unit to inch
OpenDoc();

OpenPage(8.5,11);  // page size is 8.5" by 11"
LPI(8);  // set line space to 8 LPI

GBarchartElement(ELEMENT_LABEL_ABOVE, "%.0f");  // Defines for element

GBarchartLegend(1.3, 0.4, 0.12, false, Font("Arial", 9, BOLD));  // by a bold font

Font("Arial", 8);

GPos(1, 7.5);  // Set position of barchart
GBarchart(2, 0.15, 0.2, 10, 10, 3, 7, data, lbls, legend, false, false);  // don't plot grid lines

ClosePage();
CloseDoc();

Output:
**Barchart Legend Label**

**Function**

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

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

**Syntax**

```c
void GBarchartLegend(
    float  x_rel_pos = DEFAULT,
    float  y_rel_pos  = DEFAULT,
    float      legend_box_size = DEFAULT,
    bool  vertical_layout = TRUE,
    UI16     fontid = DEFAULT,
);
```

**Parameters**

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

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

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

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

- **fontid**
  Specifies a font handle number to be used to present the legend label text of the barchart. Default is using the currently active font.

You can call the function of Font() to specify a font that returns its font handle number for you.

**Sample**

See sample for the function of Barchart Element Label.
Barchart - Stacked Barchart

Function

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

With the GBarchartElement () and GBarchartLegend () functions, you can control how to place the legend labels and indicate whether to present the element labels or not.

With the GChartColors () function, you can override the default RGB color values predefined for the graphic barchart and piechart.

Syntax

```c
void GStackchart(
    float   height,
    float   bar_width,
    float   gap_barset,
    uint  vertical_scale_unit,
    ushort      vertical_scales_positive,
    ushort      vertical_scales_negative,
    ushort      number_bars,
    ushort      bar_sets,
    float*  values,
    char*  horizontal_labels[],
    char*  legend_texts[],
    bool        3D_barchart = TRUE,
    bool       gridline = TRUE
);
```

Parameters

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

**bar_width**
The width of each bar.

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

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

**vertical_scales_positive**
The number of scales on the vertical axis for positive values.

**vertical_scales_negative**
The number of scales on the vertical axis for negative values.

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

**Bar_sets**
The number of bar-sets.

**values**
The 1 dimension or 2 dimensions array of the data values.

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

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

legend_texts
The array of texts for the legend.

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

Sample

```c
int main()
{
    float data[] = {45.00, 55.45, -27.73,
                    45.00, 67.34, -33.67,
                    45.00, 50.68, -25.34};
    char *lbls[] = {"Sept", "Oct", "Nov"};
    char *legend[] = {"Fix Charges", "Call Charges", "Discount"};
    if(__argc == 1 || *__argv[1] == '?')
    {
        printf("\n\nUsage:  BARCHART -o output_file\n\n");
        exit(0);
    }
    Start();
    SetUnit(INCH);                // Set default units to inch
    OpenDoc();
    OpenPage(8.5,11);            // page size is 8.5" by 11"
    GBarchartElement(ELEMENT_LABEL_ABOVE, "%.2f");  // Defines for element
    GBarchartLegend(0.3,          // plus relative 1.3" for X
                    0.4,          // plus relative 0.4" for Y
                    0.10,         // legend box size is 0.1"
                    FALSE);        // place legend horizontally
    UI16 f2 = Font("Arial", 8, BOLD);     // get font ID
    Font("Arial", 8);
    GPos(1, 3);              // set position of barchart
    GStackchart(1.8, // height of barchart
                0.77, // width of each bar
                0.5, // gap between each bar
                20, // unit size of v-axis scale
                7, // number of v-axis scales for positive values
                2, // number of v-axis scales for negative values
                3, // number of stacks per bar
                3, // number of bars
                data, // data array of input data
                lbls, // label texts below H-axis
                legend, // legend texts
                false, // not 3D barchart
                true, // plot grid lines
                f2);        // font ID for the total value of bar
    ClosePage();
```
CloseDoc();

#ifdef _DEBUG
    ViewPDF();  // View PDF file only in debug mode
#endif

    return 0;
}

Output:

![Chart showing financial data]

- Fix Charges
- Call Charges
- Discount
Blank Page

Function

Checks if the current page is blank.

Syntax

bool BlankPage(void);

Parameters

No parameter to be specified.

Sample

SetUnit(INCH);
OpenDoc();
GetPage(); // Read-in a PDF page

if ( BlankPage() ) // If it is a blank page, then delete it
    DeletePage();

CloseDoc();
**Bookmark**

**Function**

Adds a page-level bookmark to the PDF document.

**Syntax**

```c
SI32 Bookmark(
    char *    title,
    UI32      dest_page_no,
    SI32      parent_bookmark_handle = -1,
);
```

```c
bool BookmarkStyle(
    SI32    bookmark_handle,
    FontStyleName  font_style = REGULAR,
    UI32    color = BLACK
);
```

**Parameters**

- **title**
  Defines the title of the bookmark.

- **dest_page_no**
  Defines the destination page which should be opened when clicking on the bookmark.

- **Parent_bookmark_handle**
  Defines the parent bookmark handle number previously created if any, or -1 if the bookmark should appear as the root node.

- **bookmark_handle**
  The bookmark handle number returned by the related AddBookmark() function call.

- **font_style**
  The font style to be used to display the bookmark title, supported values are: REGULAR, ITALIC, and BOLD

- **color**
  The color to be used to display the bookmark title.

**Sample**

```c
Start();
SetUnit(PT);
OpenDoc();

// create some pages to test the bookmark
char str[30];
for (int i = 1; i < 9; i++)
{
    OpenPage();
    Font("Arial", 36);
    sprintf(str, "Page %d", i);
```
Pos(50, 50);
Ltxt(str);
ClosePage();
}

SI32 bm;

bm = Bookmark("Chapter 1", 1, -1);
  Bookmark("Section 1", 1, bm);
  Bookmark("Section 2", 1, bm);
  Bookmark("Section 3", 2, bm);
  Bookmark("Section 4", 3, bm);

bm = Bookmark("Chapter 2", 5, -1);
  Bookmark("Section 1", 5, bm);
  Bookmark("Section 2", 6, bm);
  Bookmark("Section 3", 7, bm);
  Bookmark("Section 4", 8, bm);
  Bookmark("Section 5", 8, bm);

CloseDoc();

Output:
Box Drawing

Function

Plots a graphic rectangle box.

The thickness and type of border are defined by GLineWidth() and GLineType() functions.

Syntax

```c
void GBox(
    float   x_pos,
    float   y_pos,
    float   width,
    float   height,
    PathFill_mode fill_mode = STROKE_PATH,
    float   corner_radius = 0
);
```

Parameters

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

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

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

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

- **fill_mode**
  In the graphic filling mode, you may use the following values:
  - `FILL_PATH` Fill color into the box does not stroke box boundary
  - `FILL_STROKE` Fill color into the box and stroke box boundary
  - `STROKE_PATH` Stroke box boundary does not fill color into box

- **corner_radius**
The radius of the rounded box corners, default is 0.

Sample

```c
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
GLineWidth(2); // set line width to zero.02" for box border
GBox(0.5, 4, 7.5, 1.5); // box size 7.5" x 1" at (0.5",4"),
  : 
  ClosePage();
CloseDoc();
```
Cap Height of Font (Query)

Function

Gets the cap height of the currently active font. The cap height is a typographic value that specifies the maximum height of a character in the currently active font without serifs. The return value is in the current default measurement unit.

Syntax

float CapHeight();

Sample

SetUnit(MM);
OpenDoc();
OpenPage(DIN_A4);
Font("Arial", 15, BOLD);
int angle = 0;
float x = PageWidth()*0.5;
float y = PageHeight()*0.35;
for (int i = 0; i < 12; i++)
{
  Color(0x80C080 | (SI32)angle << 6);
  AngleTxt("Circlat PDF Text", x, y, angle, 12, CapHeight()*0.5);
  angle += 30.0;
}
ClosePage();
CloseDoc();

Output:
Center Align Text

Function
Center aligns a line text string.

Syntax

```c
void Ctxt( 
    char * data, 
); 

void Cu8( 
    UChar8 * utf8_data, 
); 

void Cu16( 
    UChar * utf16le_data, 
); 
```

Parameters

data
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

utf8_data
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_data
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

Sample

```c
DefaultCode("gb18030");       // input data is Chinese GB18030
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
Font("Arial", 10);              // Define a font for English
 :
Pos(2,2);                      // Current text position at (2",2")
Ctxt("text is center aligned"); // Center align text at (2",2")
 :
Font("SimSun", 10, REGULAR, cpUnicode); // Define a font for Chinese, uses 
                                        // Unicode encoding with this font
 :
Pos(2,2.5);                    // Current text position at (2",2.5")
Ctxt("实现 Windows 与 Linux 的双引导"); // Center align Chinese text, 
                                          // Converts Chinese to UTF-16LE
ClosePage();
CloseDoc();
```
Centimeter Value

Function
Specifications a value in centimeters.

Syntax
float cm(
    float value
);

Parameters
value
The value in centimeters.

Sample
SetUnit(INCH);
OpenDoc();
OpenPage(8,11);

Pos(2.5,4);                   // set X and Y position to (2.5",4")

Pos(cm(2),3.5);               // Set X position to 2 cm and Y position to 3.5"

ClosePage();
CloseDoc();
Character Space

Function
Sets the inter-character spacing.

Syntax

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

Parameters

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

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage(210,297);
    :
    :
CharSpace(1);        // 1 MM extra space between chars.
Ctxt("Extra Intercharacter spacing");
CharSpace();        // character space reset to default
    :
    :
ClosePage();
CloseDoc();
```
Chart Colors

Function

Optional function for overriding the default RGB color values predefined for the graphic barchart and piechart.

Syntax

```c
void GChartColors(
    UI32*     RGB_colors,
    UI16     num_of_colors
);
```

Parameters

**RGB_colors**

An array of RGB color in hex values predefined for graphic barchart and piechart, up to 30 values can be defined. The default RGB color values predefined by MakePDF are:

```c
{     // RGB color values are defined in: 0x00[blue][green][red]
0x0000ff00, /* GREEN        */   0x00ffff8e, /* SKY_BLUE    */
0x00c0c000, /* TURQUOISE    */   0x006bc6ff, /* TAN         */
0x00ff9f9f, /* SKY_GRAY     */   0x00c1ffc1, /* SKY_GREEN   */
0x00ff6060, /* LIGHT_BLUE   */   0x00FF00FF, /* HOT_PINK    */
0x005b99ff, /* MEDIUM_BROWN */   0x00ff9f9f, /* LIGHT_BLUE  */
0x0000ffff, /* YELLOW       */   0x00b1e2ff, /* PEACH       */
0x00cc0099, /* PURPLE       */   0x003399ff, /* ORANGE      */
0x0080ff80, /* LIGHT_GREEN  */   0x003030ff, /* RED         */
0x000000ff, /* FOREST_GREEN */   0x008080ff, /* LIGHT_PINK  */
0x00800000, /* DARK_BLUE    */   0x000080c0, /* BROWN       */
0x000000c0, /* FOREST_GREEN */   0x00000080, /* DARK_RED    */
0x00c0c0c0  /* GRAY        */
}
```

**num_of_colors**

The number of RGB color values to be overridden.

Sample 1

```c
float data[]  = { 45, 65, 35, 70, 40, 50, 55};
char *label[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", 
                 "Saturday"};
UI32 colors[] = {0x0000ff, 0x00ff00, 0xff0000, 0xffbf2f, 0xff0080, 
                 0xc0c0c0};
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
Font("Arial" 8);
LPI(8);    // Set line space to 8 LPI for label texts
PiechartExplode(1, 0.14);    // Explode sectors 1 and 3, with distant 0.14"
PiechartExplode(3, 0.14);
GChartColors(colors,6);       // override default predefined RGB colors

PiechartLabel(0.1, "{percent}%\n{label}", TRUE);    // change defaults of label

GPos(2, 2);                   // Set piechart position – center of piechart

GPiechart(1.6,         // Piechart width
           1,         // Piechart height
           0.13,     // Piechart shadow depth
           6,        // Sectors of the piechart
           data,     // Input data array
           label,    // Label text array
           30);      // Starting angle);

Output:

![Pie Chart Image]
Chord Drawing

Function

Plots a graphic elliptical chord.

The thickness and type of border can be defined by the GLineWidth() and GLineType() functions.

Syntax

    void GChord(
        float    x_pos,
        float    y_pos,
        float    width,
        float    height,
        float    start_angle,
        float    end_angle,
        PathFill_mode   fill_mode = STROKE_PATH
    );

Parameters

    x_pos, y_pos
    The (X, Y) coordinate of the midpoint of the bounding rectangle.

    Width, height
    The width and height of the bounding rectangle.

    start_angle, end_angle
    The start and end angles are measured anticlockwise.

    fill_mode
    In the graphic filling mode, you may use the following values:

    FILL_PATH    Fill color into the box does not stroke box boundary
    FILL_STROKE  Fill color into the box and stroke box boundary
    STROKE_PATH  Stroke box boundary does not fill color into box

Sample

    SetUnit(PT);
    OpenDoc();
    OpenPage();
    Color(PINK); // Set fill color
    StrokeColor(DARKBLUE); // Set stroke color
    GChord(190, 170, 100, 50, 0, 100, FILL_STROKE);
    ClosePage();
    CloseDoc();

    Output:
Circle Drawing

Function

Plots a graphic circle.

The thickness and type of border can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GCircle(
    float    x_pos,
    float    y_pos,
    float    radius,
    PathFill_mode    fill_mode = STROKE_PATH
);
```

Parameters

- **x_pos, y_pos**
  The (X, Y) coordinate of the center of the circle.

- **radius**
  The radius of the circle.

- **fill_mode**
  In the graphic filling mode, you may use the following values:
  - **FILL_PATH** Fill color into the circle does not stroke circle boundary
  - **FILL_STROKE** Fill color into circle and stroke circle boundary
  - **STROKE_PATH** Stroke circle boundary does not fill color into the circle

Sample

```c
OpenDoc();
SetUnit(PT);
OpenPage(DIN_A4);
GLineWidth(1); // Set line width to 1 point
ColorRGB(220,220,220); // Set RGB gray color as the fill color
StrokeColor(BLACK); // Set black for stroking circle line
Circle(90, 90, 40, FILL_STROKE); // Draw circle at (90, 90), radius 40
ClosePage();
CloseDoc();
```

Output:

![Circle Drawing](image-url)
Close Document – Output PDF to File or Memory

Function

Closes the PDF document previously opened with the OpenDoc() function call, and outputs PDF
document either to a PDF file or a memory block.

Syntax

```c
void CloseDoc(
    char *  pdf_file = NULL
);
void CloseDocW(
    UChar *  pdf_file_utf16 = NULL
);
char* CloseDoc2Memory(
    UI32  &pdf_data_len
);
```

Parameters

**pdf_file**

Specifies the PDF output filename, the default value is empty without specifying any filename,
PDF filename is previously specified by the OpenDoc() function. FastPDF Formatter adds PDF
filename extension *.pdf if it is not specified.

The PDF file name can a fully qualified with the pathname, alternatively you can specify the
output path by the command-line flag -o parameter.

You can specify an empty PDF filename with the OpenDoc() function, then specify a PDF
filename with the CloseDoc() function, so that you could dynamically control the PDF output
filename. For instance, you may need such PDF filename control for the e-Statement multiple
outputs, that need PDF files to be dynamically generated by the account numbers or customers’
names.

**pdf_data_len**

Gets the length of the PDF memory file generated, CloseDoc() returns its memory-buffer address,
so that you can read a PDF file from a memory-buffer block directly for other purposes. The
memory PDF file is being generated if an empty filename is specified with the OpenDoc()
function.

Sample

```c
OpenDoc();  // First PDF filename is token from command-line

CloseDoc();  // Close first PDF document and close file

OpenDoc("test.pdf");  // Second PDF filename is specified here

CloseDoc();  // Close Second PDF document and close file
```
Close Document – Output PDF as Attachment of Email

Function

Closes the PDF document previously started with the OpenDoc() function call, and split PDF as the attachments of email files. It writes RFC822-compliant email files with filename extension *.eml in high performance.

You must specify an empty PDF filename with the OpenDoc(NULL) function, so that the PDF document can be generated in a memory block, and then be split as the attachments of email files.

Page group level bookmarks must be generated at the first page of each page-group, by the Bookmark("receiver_name <receiver_email_id>,file_base_name", page_number) function, to be used by CloseDoc2Email() function to split PDF. You may use your client’s account number to form the base name of the email file and PDF attachment file.

This function is only allowed with FastPDF Formatter Enterprise or trial license.

Syntax

void CloseDoc2Email(
    char * from_name_email_id,
    char * subject,
    char * message = NULL,
    char * charSet = NULL,
);

Parameters

from_name_email_id
Specifies the name and email ID of the sender, in "sender_name <sender_email_id>" format.

subject
Specifies the subject title of the email.

message
Specifies the message in text or HTML format.

charSet
Specifies the character-set name of the email text encoding, the default value is NULL, uses the character-set of US-ASCII. The following character-sets are supported:

<table>
<thead>
<tr>
<th>Character Set Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp1250, Windows-1250</td>
<td>ANSI - Central Europe</td>
</tr>
<tr>
<td>cp1251, Windows-1251, ISO-8859-5</td>
<td>ANSI - Cyrillic</td>
</tr>
<tr>
<td>cp1252, Windows-1252, ascii, us-asccii, ISO-8859-1</td>
<td>ANSI – Latin 1</td>
</tr>
<tr>
<td>cp1253 Windows-1253, ISO-8859-7</td>
<td>ANSI - Greek</td>
</tr>
<tr>
<td>cp1254, Windows-1254</td>
<td>ANSI - Turkish</td>
</tr>
<tr>
<td>cp1255, Windows-1255, ISO-8859-8</td>
<td>ANSI - Hebrew</td>
</tr>
<tr>
<td>cp1256, Windows-1256, ISO-8859-6</td>
<td>ANSI - Arabic</td>
</tr>
<tr>
<td>cp1257, Windows-1257, ISO-8859-4</td>
<td>ANSI - Baltic</td>
</tr>
</tbody>
</table>
Sample

```c
char tmp[256];
char subject[] = "Your Visa Card Statement";
char sender_name_email[] = "IBC Bank Billing Center < billing@ibcbank.com>");
char message[] = "Dear Client, 

Attached is your September Visa " 
"Card Statement",

OpenDoc(NULL); // Specify an empty PDF filename, as we need to 
// output PDF into an email

OpenPage(8.27, 11.67);

sprintf(tmp, "%s <%s>,%s", client_name, client_email_id, account_no);

Bookmark(tmp, pageNo); // Page-group level bookmarks must be put at first 
// page of each page-group, to be used by 
// Close2Email()

ClosePage();

CloseDoc2Email(sender_name_email, subject, message);```

The table contains various character sets used for different languages:

<table>
<thead>
<tr>
<th>Character Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp1258, cp866</td>
<td>ANSI - Vietnamese</td>
</tr>
<tr>
<td>cp874, koi8-r</td>
<td>Russian DOS</td>
</tr>
<tr>
<td>ISO-8859-2</td>
<td>Central and East European charset</td>
</tr>
<tr>
<td>ISO-8859-3</td>
<td>Latin 3 - Esperanto, Maltese, Turkish, replaced by Latin 5</td>
</tr>
<tr>
<td>ISO-8859-9</td>
<td>Latin 5</td>
</tr>
<tr>
<td>ISO-8859-10</td>
<td>Latin 6 - Innu, Nordic</td>
</tr>
<tr>
<td>ISO-8859-15</td>
<td>Latin 9 - Latin 5 with Turkish chars replacing Icelandic</td>
</tr>
<tr>
<td>koi8-u</td>
<td>Ukrainian</td>
</tr>
<tr>
<td>BIG5</td>
<td>Traditional Chinese Windows</td>
</tr>
<tr>
<td>GB2312</td>
<td>Simplified Chinese</td>
</tr>
<tr>
<td>Utf-8</td>
<td>Unicode UTF-8</td>
</tr>
</tbody>
</table>
Close Document – Split PDF Pages

Function

Closes the PDF document previously started with the OpenDoc() function call and split PDF pages by per page group.

You must specify an empty PDF filename with the OpenDoc(NULL) function, so that the PDF document can be generated in a memory-block, and then be split by per page group.

Page group level bookmarks must be generated and put on the first page of each page-group, by the function of Bookmark("file_base_name", page_number), to be used by the CloseDoc2Split() function to split PDF pages. You may use your client's account number to form the base name of each PDF file.

Syntax

void CloseDoc2Split(void);

Parameters

None.

Sample

OpenDoc(NULL); // Specify an empty PDF filename, as we need to
// split PDF pages by per page-group

OpenPage(8.27, 11.67);

Bookmark(account_no, pageNo); // Page-group level bookmarks must be put at
// first page of each page-group, to be used
// by Close2Split() function

ClosePage();

CloseDoc2Split();
Close Page

Function

Closes a PDF page previously opened with an OpenPage() function call, once the page formatting is completed, you need to close the page with the ClosePage() function.

With EditPage() function, you could reopen a closed page if needed, for example, to insert dynamic pagination text “Page x of y”, OMR and barcode.

Syntax

void ClosePage()

Parameters

None.

Sample

```c
void main()
{
    Start();
    SetUnit(CM);
    OpenDoc();
    OpenPage(21, 29.7); // A4 paper size, 21cm x 29.7cm
    ClosePage(); // Close PDF page
    OpenPage(21, 29.7); // A4 paper size, 21cm x 29.7cm
    ClosePage(); // Close PDF page

    CloseDoc();
}
```
Color of Fill

Functions

Specify the fill color for the subsequent text data and graphic color background. FastPDF Formatter handles color values conversion according to your current color space automatically, for instance, if your current color space is DEVICE-CMYK defined by ColorSpace() function, your RGB values specified will be auto-converted to CMYK internally.

Syntax

By a predefined RGB color alias name:

```c
void Color(
    Color_RGB       rgb_color_name = BLACK
);
```

By RGB color values:

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

By CYMK color values:

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

By a 32-bit color value:

```c
void SetColor(
    UI32  32bit_color_value
);
```

For converting RGB values to a 32-bit color value:

```c
UI32 rgb(
    UCHAR  red_color,
    UCHAR  green_color,
    UCHAR  blue_color
);
```

For convert CMYK values to a 32-bit color value:

```c
UI32 cmyk(
    UCHAR  cyan_color,
    UCHAR  magenta_color,
    UCHAR  yellow_color,
    UCHAR  black_color
);
```
Parameters

**rgb_color_name**
Any of the predefined color names, such as BLUE, RED, MAGENTA/PINK, GREEN, CYAN, YELLOW, BLACK, BROWN, etc, default value is BLACK. See herder file MakePDF.h for details, you can add your own color alias names in the MakePDF.h header file.

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

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

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

**32-bit color value**
Defines a 32-bit color value. It is strongly recommended to construct a 32-bit CMYK value by cmyk() function or a 32-bit RGB value by rgb() function, to make sure FastPDF Formatter could handle color values conversion according to your current color space properly.

Sample

```plaintext
SetUnit(MM);
OpenDoc();
OpenPage(210,297);

Font("Arial", 10);
Pos(5,5); // current position at (5,5) mm
ColorRGB(255,0,0); // red color by RGB values
Ltxt("RGB Red Color Text");
Pos(5,10);

ColorCMYK(0,0,0,100); // black color by CMYK values
Ltxt("CYMK Black Color Text");
Pos(5,15);

Color(CYAN); // color by a color alias name
Ltxt("PDF CYAN Color Text");
Pos(5,20);

Color(rgb(255,255,0)) // forms a 32-bit color value for yellow color
Ltxt("PDF Yellow Color Text");

ClosePage();
CloseDoc();
```
Color of Stroke

Functions

Specify the stroke color for the stroked object, such as graphic line, graphic object boundary, and text stroking. FastPDF Formatter handles color values conversion according to your current color space automatically, for instance, if your current color space is DEVICE-CMYK defined by ColorSpace() function, your RGB values specified will be auto-converted to CMYK internally.

Syntax

By a predefined RGB color alias name:

```c
void StrokeColor(
    Color_RGB     rgb_color_name = BLACK
);
```

By RGB color values:

```c
void StrokeColorRGB(
    UCHAR  red_color,
    UCHAR   green_color,
    UCHAR  blue_color
);
```

By CYMK color values:

```c
void StrokeColorCMYK(
    UCHAR  cyan_color_percentage,
    UCHAR  magenta_color_percentage,
    UCHAR  yellow_color_percentage,
    UCHAR  black_color_percentage
);
```

By a 32-bit color value:

```c
void SetStrokeColor(
    UI32  32bit_color_value
);
```

For converting RGB values to a 32-bit color value:

```c
UI32 rgb(
    UCHAR  red_color,
    UCHAR   green_color,
    UCHAR  blue_color
);
```

For convert CMYK values to a 32-bit color value:

```c
UI32 cmyk(
    UCHAR  cyan_color,
    UCHAR  magenta_color,
    UCHAR  yellow_color,
    UCHAR  black_color
);
```
Parameters

rgb_color_name
Any of the predefined color names, such as BLUE, RED, MAGENTA/PINK, GREEN, CYAN, YELLOW, BLACK, BROWN, etc, default value is BLACK. See header file MakePDF.h for details, you can add your color alias names in the MakePDF.h header file.

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

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

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

32-bit color value
Defines a 32-bit color value. It is strongly recommended to construct a 32-bit CMYK value by cmyk() function or a 32-bit RGB value by rgb() function, to make sure FastPDF Formatter could handle color values conversion according to your current color space properly.

Sample

OpenDoc();
SetUnit(PT);  
OpenPage(DIN_A4);

:  
GLineWidth(1); // Set line width to 1 point
ColorRGB(220,220,220); // Set RGB gray color as the fill color
StrokeColor(BLACK); // Set black for stroking circle line
Circle(90, 90, 40, FILL_STROKE); // Draw circle at (90, 90), radius 40

:  
ClosePage();
CloseDoc();

Output:
Color Space Management

Function

Specifies the color space. FastPDF Formatter default color space is DEVICE_RGB.

Syntax

```c
void ColorSpace (PDFColorSpace  device_color_space);
void ColorSpaceICC (char *  ICC_filename);
```

Parameters

device_color_space
Supported device color space values are:

- DEVICE_RGB: RGB color space
- DEVICE_CMYK: CMYK color space
- DEVICE_GRAY: GRAY color space

ICC_filename
Specifies the filename of an ICC profile-based color space.

Sample

```c
OpenDoc();
SetUnit(PT);
ColorSpace(DEVICE_CMYK); // Defines current color space to CMYK
OpenPage(DIN_A4);
:
ColorRGB(220,220,220); // RGB color values will be converted to CMYK colors
Color(YELLOW); // RGB color values will be converted to CMYK values
ColorCMYK(30, 50, 30, 100); // CMYK color values
:
ClosePage();
CloseDoc();
```
Compression Filter

Function

Defines the compression filter to be used to compress images in PDF output.

The 1-bit and 4-bit images are always be compressed with compression filter FLATE independent of the current active compression filter.

Syntax

```c
void CompressionFilter(
    CompressingFilter   filter
);
```

Parameters

- **level_value**
  
  Allowed compression filters are:

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLATE</td>
<td>FLATE compression filter</td>
</tr>
<tr>
<td>JPEG</td>
<td>JPEG compression filter, default value</td>
</tr>
<tr>
<td>JPEG2000</td>
<td>JPEG2000 compression filter</td>
</tr>
</tbody>
</table>

Sample

```c
SetUnit(MM);
OpenDoc();
ImageResolution(300);
CompressionFilter(JPEG);
CompressionLevel(MAX_COMPRESS);
JpegQuality(80);
OpenPage();

InclImage("mafp1.jpg", 0.3, 0.35);

ClosePage();
CloseDoc();
```
Compression Level

Function

Defines the PDF compression level.

Syntax

void CompressionLevel(
    CompressingLevel   level_value
);

Parameters

level_value
Valid compression levels are:

<table>
<thead>
<tr>
<th>Level Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_COMPRESS</td>
<td>Default compression level</td>
</tr>
<tr>
<td>FAST_COMPRESS</td>
<td>Faster formatting speed but Less compression ratio</td>
</tr>
<tr>
<td>MAX_COMPRESS</td>
<td>Maximum compression ratio but slow</td>
</tr>
</tbody>
</table>

Sample

SetUnit(MM);
OpenDoc();
ImageResolution(150);
CompressionFilter(JPEG);
CompressionLevel(MAX_COMPRESS);
JpegQuality(80);
OpenPage();
  :
  :
InclImage("mafp1.jpg", 0.3, 0.35);
  :
  :
ClosePage();
CloseDoc();
Decrypt Input PDF Document

Function

Defines the PDF open password and owner password to decrypt the input PDF document.

This function must be called before Start() function if you want to override the command-line flag parameters -U in_open_password and -P in_owner_password.

Syntax

```c
void DecryptDoc(
    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.

Sample

```c
SetUnit(PT);
```
Default Input Data Encoding

**Function**

Defines the default encoding names for your input data.

Make sure you have defined a correct encoding name before calling of text and paragraph functions and other data encoding conversion 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.*

**Sample**

```c
SetUnit(PT);
DefaultCode("gb18030"); // texts are in Chinese GB18030 encoding
OpenDoc();
OpenPage();
Font("Arial", 14, ITALIC + UNDERSCORE);
Color(RED);
Pos(20, 50); Ltxt("Italic UNDERSCORE ASCII Character String");
Color(BLUE);
Font("SimSun", 14, BOLDITALIC, cpUnicode); // uses UTF16LE with font
Pos(20, 75);
Ltxt("Bold & Italic Emulation 仿汉字粗斜体"); // GB18030 will be converted to UTF16LE internally
ClosePage();
CloseDoc();
```

**Output:**

*Italic Underscored English ASCII Character String*  
*Bold and Italic Emulation 仿汉字粗斜体*
Default Language Locale

Function

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

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

Syntax

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

Parameters

localeName
The Locale name of your language, FastPDF Formatter default is “en_US” if this function is not called.

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

Sample

```c
char *text = "The Portable Document Format (PDF), undeniably one of the most "
        "commonly used formats for electronic documents is now "
        "accessible as an ISO International Standard - ISO 32000-1.";

DefaultCode("ibm-437");       // Data from codepage 437, USA ASCII

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

OpenDoc();
OpenPage();
Font("Arial", 10);
LPI(5);                        // Set line density to 5 lines per inch

Color(CYAN);
ParTxt(text, 35, 30, 55);

Color(BLUE);
ParTxt(text, 35, 77, 55, -1, RIGHT, true, 10, 5, 5);

Color(RED);
ParTxt(text, 120, 30, 55, -1, CENTER, true, -10);

Color(GRAY);
ParTxt(text, 120, 70, 55, -1, JUSTIFY, true, -10, 5, 5);

ClosePage();
CloseDoc();
```
Define an Inline Overlay Object

Functions

Defines an inline PDF overlay object for repeat-use.

The smaller PDF file can be generated with inline overlay objects, as the same subsets of resource objects, such as embedding fonts can be shared by both inline overlays and PDF document pages, this is very important to Far East Asian CJK languages with smaller PDF output.

A template overlay is a special PDF object that contains vector graphics, texts, images, and all other object types which are supported by the normal PDF page.

The template overlay object can be scaled and used multiple times, yet only be stored in the PDF file once.

After an inline template overlay object is created, it can be included in PDF by the InclOvlyH() function.

Alternatively, you can also include a PDF page from an external PDF file as the template object.

Syntax

```c
UI16 BgnOvly(
    float   width,
    float   height
);

void EndOvly(void);
```

Parameters

**width, height**
The width and height of the overlay object.

Sample

```c
UI16 create_template1(void)
{
    UI16 t1 = BgnOvly(8.27, 0.45);  // keep the template handle number
    GLineWidth(0.005);
    ColorRGB(216,255,216);
    StrokeColor(GRAY);
    Box(0.3, 0.2, 5.7, 0.2, FILL_STROKE);
    Font("Arial", 12, BOLD);
    Pos(2.57,0.15);
    ColorRGB(105,105,0);
    Ltxt("Domestic Calls");
    Font("Arial", 9, BOLD);
    ColorRGB(0, 146, 0);
    Pos(0.73,0.355);
    Ltxt("No. Called");
    Xpos(1.85);
```
int main()
{
    Start();
    SetUnit(INCH);
    OpenDoc();
    UI16 h1 = create_template1();  // Create a template and get its
    // handle number
    OpenPage();
    :
    InclOvlyH(h1, 0, 0.3);  // Include Overlay at (0", 0.3")
    :
    ClosePage();
    :
    OpenPage();
    :
    InclOvlyH(h1, 0, 0.3);
    :
    ClosePage();
    :
    CloseDoc();
#ifdef _DEBUG
    ViewPDF();  // View PDF file only in debug mode
#endif
    return 0;
}

Output:

<table>
<thead>
<tr>
<th></th>
<th>Destination</th>
<th>Date</th>
<th>Time</th>
<th>Duration</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Called</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Delete Page

Function

Deletes a PDF page.

Syntax

```cpp
bool DeletePage(
    UI32   pageNo = 0
);
```

Parameters

**pageNo**
Specifies which PDF page to be deleted. The default value is 0, which means delete the current PDF page, value 1 means delete the closed PDF page 1, and so on.

Sample

```cpp
SetUnit(INCH);
OpenDoc();
GetPage();        // Read-in a PDF page
:
:
if ( BlankPage() ) // If it is a blank page, then delete it
    DeletePage();
:
:
CloseDoc();
```
Document Description

Function

Defines a PDF document description.

Syntax

void DocInfo(
    DocumentInfo doc_info,
    char * text
);

void DocInfoW(
    DocumentInfo doc_info,
    UChar * utf16_text
);

Parameters

doc_info
Valid documents description info entries are:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHOR</td>
<td>The author of PDF document</td>
</tr>
<tr>
<td>CREATOR</td>
<td>The creator of PDF document</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>The keywords of PDF document</td>
</tr>
<tr>
<td>PRODUCER</td>
<td>The producer of PDF software</td>
</tr>
<tr>
<td>SUBJECT</td>
<td>The subject of PDF document</td>
</tr>
<tr>
<td>TITLE</td>
<td>The title of PDF document</td>
</tr>
<tr>
<td>COMPANY</td>
<td>The company of PDF document</td>
</tr>
<tr>
<td>PDF_VER</td>
<td>The version of PDF document</td>
</tr>
<tr>
<td>CUSTOM</td>
<td>The user define custom-entry</td>
</tr>
</tbody>
</table>

text
The text string of description.

Sample

None.
**Edit Page**

**Function**

Edit a previously created and closed page. You have to close the editing page with `ClosePage()` function after the required changes are made.

**Syntax**

```c
void EditPage(
    UI32   page_no
);
```

**Parameters**

`page_no`
Specifies the page number of a previously created and closed PDF page. If that page does not exist, new pages are appended until the number of pages is equal to the specifying page number `page_no`.

**Sample**

```c
SetUnit(INCH);
OpenDoc();

ClosePage();   // close last page of each page-group
groupPages++;   // now got the total pages of a page-group
char tmp[25];
for (int i = 1; i < groupPages + 1; i++)
{
    sprintf(tmp, "Page %d of %d", i, groupPages);
    EditPage(lastGroupPageNo + i); // edit each page of page-group to insert
    // pagination text "Page x of y"
    Font("Arial", 10);
    Pos(8.0,3.93);
    Rtxt(tmp);
    ClosePage();   // close current editing page
}

lastGroupPageNo += groupPages; // keep the last page number of the last
// page-group

groupPages = 0; // reset it for processing next page-group
```
End of Document File of Input PDF (Checking)

**Function**

Tests for end of document of input PDF file.

Returns 1, if the end of PDF document file has been detected, after “Get Page” function is called for a reading of a PDF page from the input PDF 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**

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

**Parameters**

None.

**Sample**

None.
Ellipse Drawing

Function

Plots a graphic ellipse.

The thickness and type of border can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GEllipse(
    float   x_pos,
    float   y_pos,
    float   width,
    float   height,
    float   angle,
    PathFill_mode  fill_mode = STROKE_PATH
);
```

Parameters

- **x_pos, y_pos**
  The top-left (X, Y) coordinate of the bounding rectangle.

- **Width, height**
  The width and height of the bounding rectangle.

- **angle**
  The rotation angle of the bounding rectangle.

- **fill_mode**
  In the graphic filling mode, you may use the following values:

  - **FILL_PATH**
    Fill color into an ellipse does not stroke ellipse boundary
  - **FILL_STROKE**
    Fill color into an ellipse and stroke ellipse boundary
  - **STROKE_PATH**
    Stroke ellipse boundary does not fill color into an ellipse

Sample

```c
SetUnit(PT);
OpenDoc();
OpenPage();
Color(PINK);                     // Set fill color
StrokeColor(DARKBLUE);          // Set stroke color
GEllipse(190, 170, 100, 50, 10, FILL_STROKE);
ClosePage();
CloseDoc();
```

Output:

![Sample Output Image]
Encrypt Output PDF Document

Function

Specifies the encryption to protect the output PDF document.

This function overrides the command-line flag parameters -u out_open_password and -p out_owner_password.

Syntax

```c
void EncryptDoc(
    char * owner_password,
    char * open_password = NULL,
    char * restrictions = "nomodify,noassemble",
    char * certificate_file = NULL,
    char * cert_file_password = NULL
);
```

Parameters

**owner_password**
Specifies the owner password of the PDF restriction controls. Can be empty with a NULL value.

*Note: A FastPDF Formatter license key is required, otherwise FastPDF Formatter always uses its password.

**open_password**
Specifies the password required for the opening of PDF document file. Default is NULL, PDF file can be opened without the open password.

**restrictions**
Specifies the restrictions to the PDF document. Multiple values can be separated by the comma, valid values allowed are:

- **noaccessible** Prevents extracting of text or graphics for accessibility purposes, such as a screen-reader program.
- **noassemble** Prevents inserting, deleting, modifying, rotating pages, and creating bookmarks and thumbnails.
- **nocopy** Prevents copying and extracting text or graphics from PDF documents.
- **nomodify** Prevents any change to the PDF document.
- **noprint** Prevents the printing of PDF documents.

*Note: You need a FastPDF Formatter license key for this feature, otherwise FastPDF Formatter always prevents any change to the PDF document.

**certificate_file**
Optional, specifies the fully qualified filename of the PKCS#12 certificate file with the encryption key lengths from 1024 bits up to 4096 bits, to digitally sign the high confidential PDF document.

**cert_file_password**
Optional, specifies the password required to decrypt the private keys of the PKCS#12 certificate file.

Sample

None.
File Name of Default Input/Output File (Query)

Function

Queries the filename of the default input/output file, specified by the flag parameters "-i" and "-o" of your command-line.

These functions are mainly developed for calling from other programming languages. with Visual C++, you can use FastPDF Formatter special variable directly.

Syntax

DLL char* FileInName(void); // Gets Input data filename
DLL char* PdfInFile(void); // Gets input PDF document filename
DLL char* PdfOutFile(void); // Gets output PDF document filename
DLL UChar* PdfOutFileW(void);
DLL char* PdfOutPath(void); // Gets PDF output path name
DLL UChar* PdfOutPathW(void);

Sample

None.
Font Definition

Function

Defines a font as the currently active font for the subsequent texts.

Syntax

UI16 Font(
    char *   font_family_name,
    float    font_size,
    FontStyles     font_style = REGULAR,
    Codepage    encoding_name = cp1252,
    bool    font_embedding = true
);

UI16 FontW(
    UChar *   utf16_font_family_name,
    float    font_size,
    FontStyles     font_style = REGULAR,
    Codepage    encoding_name = cp1252,
    bool    font_embedding = true
);

Parameters

**font_family_name**
The family name of the font is case-sensitive which must be specified exactly. You could find a
font family name by Windows Control Panel.

FastPDF Formatter always adds Windows system default font path to its font paths searching list.
You can add font path with the command-line flag -f parameter or parameter FONTLIB defining
in your MakePDF definition file.

All versions of Adobe's Acrobat Reader support the standard PostScript fonts of Courier,
Helvetica, Times, Symbol, and ZapfDingbats, which are always included in the Adobe Acrobat
Reader packages.

**font_size**
The height of the font in points (Each point is equal to 1/72 of one inch).

**font_style**
Specifies the font style, valid values are: REGULAR, BOLD, ITALIC, BOLDITALIC, UNDERSCORE, and
STRIKE, you can specify a combination of the font styles, for instance, BOLD + UNDERSCORE.

**encoding_name**
The encoding name of the text string.

With some encoding names you specified, FastPDF Formatter converts the legacy codepage text
string to Unicode internally, for instance, with cpUnicode encoding name, you have to make sure
the default encoding name is specified properly with the DefaultCode() function so that FastPDF
Formatter converts your legacy input data into Unicode first internally.

Valid encoding names are:

Legacy ASCII codepages:
cp1250 Eastern European
cp1251 Cyrillic
cp1252 Latin 1 Western European
cp1253 Greek
cp1254 Turkish
cp1255 Hebrew
cp1256 Arabic
cp1257 Baltic
cp1258 Vietnamese
cp8859_2 Latin 2 Central Europe
cp8859_3 Latin 3 Maltese
cp8859_4 Baltic
cp8859_5 Cyrillic
cp8859_6 Arabic
cp8859_7 Greek
cp8859_8 Hebrew
cp8859_9 Latin 5 Turkish
cp8859_10 Latin 6 Nordic Area
cp8859_13 Latin 7 Baltic Rim
cp8859_14 Latin 8 Celtic
cp8859_15 Latin 9 French
cp8859_16 Latin 10 Hungarian
cpSymbol Symbol
cp437 DOS USA
cp737 DOS Greek
cp775 DOS Baltic Rim
cp850 DOS Multilingual
cp852 DOS Slavic
cp855 DOS Cyrillic
cp857 DOS Turkish
cp860 DOS Portuguese
cp861 DOS Icelandic
cp862 DOS Hebrew
cp863 DOS French (Canada)
cp864 DOS Arabic
cp865 DOS Nordic
cp866 DOS Russian
cp869 DOS Modern Greek
cp874 DOS Thai

Using Unicode UTF16LE encoding, FastPDF Formatter converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() function first:

```plaintext
cpUnicode Unicode UTF16 little-endian
```

For CJK encoding texts to be used with Unicode encoded fonts:

```plaintext
cpBIG5_Uni Traditional Chinese BIG5 to Unicode UTF16LE
cpGBK_Uni Simplified Chinese GB to Unicode UTF16LE
cpKSC_Uni Korean KSC to Unicode UTF16LE
cpJIS_Uni Japanese S-JIS to Unicode UTF16LE
```

CJK texts are being converted to Unicode string by FastPDF Formatter internally if your input data are in legacy CJK codepage, make sure the encoding name is defined properly with DefaultCode() function first.

```plaintext
font_embedding
```
Specifies whether the subset of the font is being embedded in the PDF output document.
All supported font formats can be embedded as long as font embedding is not restricted by the font copyright.

The function call will be failed if the font embedding is not possible due to the licensing restriction. Some TrueType and OpenType fonts contain the restriction of font embedding.

**Sample**

```plaintext
SetUnit(PT);
DefaultCode("gb18030"); // texts are in Chinese GB18030 encoding
OpenDoc();
OpenPage();
Font("Arial", 14, ITALIC + UNDERSCORE);
Color(RED);
Pos(20, 50); Ltxt("Italic UNDERSCORE ASCII Character String");
Color(BLUE);
Font("SimSun", 14, BOLDITALIC, cpGBK_Uni);
Pos(20, 75);
Ltxt("Bold & Italic Emulation 仿汉字粗斜体"); // GB18030 will be converted // to UTF16LE internally
ClosePage();
CloseDoc();
```

**Output:**

*Italic Underscored English ASCII Character String*

*Bold and Italic Emulation 仿汉字粗斜体*
Font Changes

Functions

Change the currently active font, or change the attributes of the currently active font, such as
font size, style, weight, and scaling.

Syntax

void FontH(
    UI16    font_handle_no
);

void FontItalic(
    float   font_angle = 14.0
);

void FontOrigin(
    FontOrigin  font_origin = DOWN_LEFT
);

void FontScale(
    float   font_scale_ratio
);

void FontSize(
    float   font_size
);

void FontStyle(
    FontStyles   font_style
);

void FontWeight(
    UI16   font_weight
);

Parameters

font_handle_no
Changes the currently active font to a previously defined font. The font handle number must be a
valid font handle that was created by the Font() function.

font_italic
Defines an angle for the currently active font to emulate its italic style font. To disable the
emulation set the angle to zero. Recommended angle values are from -30 to 30 degrees, the
default value is 14.0 degrees.

font_size
Changes the font size of the currently active font

font_origin
Defines the font origin that is used to position the subsequence text strings. Valid values are:

    DOWN_LEFT       By font baseline, which is FastPDF Formatter default setting
    TOP_LEFT         By the upper left corner of the font's bounding box.
Changes the font size of the currently active font.

Changes the style of the currently active font. Only the font style flags UNDERSCORE and STRIKED can be set with this function.

Specifies the thickness or boldness of the currently active font to emulate its bold style. To disable the emulation of bold style set the weight to zero. Suggested values for bold a font are:

- 100 - 300 Ultra-light, Light
- 400 - 500 Standard
- 600 - 1000 Bold, Extrabold

**Sample**

```plaintext
SetUnit(MM);
OpenDoc();
OpenPage();

UI16 f0 = Font("Arial", 9.5, BOLD);
Pos(10, 10); Ltxt("Bold Style Text, size 9.5 points");

UI16 f1 = Font("Arial", 9.5);
Pos(10, 15); Ltxt("Normal Style Text, size 9.5 points");

FontH(f0); FontSize(15);
Pos(10, 20); Ltxt("Bold Style Text, size 15 points");

FontH(f1); FontSize(12);
Pos(10, 25); Ltxt("Normal Style Text, size 12 points");

ClosePage();
CloseDoc();
```
Get Page – Getting PDF Page From Input Document

Function

Gets a PDF page from the PDF input file. You can close the page with the “Close Page” function once you have completed the processes to the page.

Syntax

```cpp
bool GetPage(
    UI32 pageNo = 0,
    float page_width = 0,
    float page_height = 0
);
```

Parameters

- **pageNo**
  Specifies which PDF page to be read-in from the PDF input file. Default value 0 means reading PDF page sequentially by the internal control of FastPDF Formatter, value 1 means read-in page 1, and so on.

- **page_width**
  Specifies the page width, default value 0 means the page width is imported from the input PDF document.

- **page_height**
  Specifies the page height, default value 0 means the page height is imported from the input PDF document.

Sample

```cpp
SetUnit(INCH);
Start();                       // Start a FastPDF Formatter session
OpenDoc();

getPage();

// Once a PDF page has been read-in, then
// you can make changes you want

ClosePage();                   // Close PDF page

CloseDoc();                   // Close PDF document
```
Get Input PDF Page Width, Height, and Number of Pages

Function

Get the page height, page width, and the number of pages contained in the input PDF document.

Syntax

float GetInPageHeight(void);
float GetInPageWidth(void);
UI32 GetInPageCount(void);

Parameters

None.

Sample

None.
Get Text Field by a Location

Function

Captures a text field string by the bottom-up coordinate location of the data field on a PDF page.

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

Syntax

char *GetField(
    float x,
    float y,
    char* str
);

UChar *GetFieldU16(
    float x,
    float y,
    UChar* u16_str
);

Parameters

x
Specifies the X bottom-up position of the data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

y
Specifies the Y bottom-up position of the data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

str
Specifies a variable to store the text string of the data field captured. FastPDF Formatter converts the data field from the Unicode UTF-16 encoding to the native ASCII/DBCS-PC encoding. Make sure the DefaultCode() function is called before this function is called.

u16_str
Specifies a variable to store the Unicode UTF-16LE text string of the data field captured.

Sample

int main()
{
    UI32 i, j, k, inPageNo = 0, outPageNo = 0, groupPages = 0;
    bool firstGroup = 1;
    char str[80], str2[80], acNo[10], acNo_keep[10];
    Start();                 // Start session
    DefaultCode("ibm-437"); // Sets default native encoding to USA ASCII
    SetUnit(INCH);           // Set default units to inch
    OpenDoc();               // Open a new PDF document
    do {
        GetPage();             // Read-in a page from input PDF file
        inPageNo++;
        GetField(524.90, 509.04, str); // In this sample, we use GetField() to
// to capture the text field and then compare
// its value whether match with "Page 1" and
// its string length whether is 6, if all are
// true, then it is the first page of page-group

// You may also use TRIGGER functions with your
// own programs to identify an input PDF page

if (!strcmp(str, "Page 1") && strlen(str) == 6)   // if it first page of
{                                                 // page group
    GetField(43.20, 532.80, str);                // Gets Saving Account No
    strcpy(acNo, &str[18]);        // Gets Saving Account No

    if (!firstGroup)               // Now at first page of new page-group
    {
        ClosePage();                 // Need to close current page first
        outPageNo++;

        // Process pagination, page serial numbers and barcode of previous
        // page group
        k = 1;
        j = inPageNo - groupPages;

        for (i = outPageNo-groupPages; i < outPageNo; i++, k++)
        {
            EditPage(i);
            // replace "Page n" with "Page n of nn"
            sprintf(str, "Page %d", k);
            sprintf(str2, "Page %d of %d", k, groupPages);
            TextReplace(str2, src, 524.90, 509.04, 72, 10);

            // Or you can use MaskArea(), Pos() and Ltxt()/Rtxt()
            // function to mask that area, and then put new pagination
            // MaskArea(7.9, 3.6, 1, 0.34);
            // Font("Helvetica", 10); Pos(8.0,3.93);
            // Rtxt(str2);

            // Replace page serial number as the bank pages inserted
            sprintf(str, "%07d", j++);
            sprintf(str2, "%07d", i);
            TextReplace(str2, str, 7.20, 10.80, 72, 10);
            BarCode(CODE128, acNo_keep, 0.22, 1, 1.5, 0.2, 90);
            ClosePage();
        }
    }  

    if (groupPages % 2 != 0)     // If the pages of page group is an odd
    {                            // number, then insert a blank page
        // for the production printing
        MovePage(++outPageNo, outPageNo);
    }

    groupPages = 1;              // reset for new page group
    strcpy(acNo_keep, acNo);     // Keeps new account number for
    // generating of dynamic barcode
}
else                           // It is first page group of input PDF
{ 
    firstGroup = 0;
    strcpy(acNo_keep, acNo);   // Keeps current account number for
    // generating of dynamic barcode
    ClosePage();
    outPageNo++;
    groupPages++;
else
{
  ClosePage();
  outPageNo++;
  groupPages++;
}
} while ($Edt == 0);     // reprocess input PDF until end of PDF file

// Process pagination, page serial numbers and barcode of last page group
k = 1;
j = inPageNo - groupPages + 1;
for (i = outPageNo-groupPages+1; i <= outPageNo; i++, k++)
{
  EditPage(i);
  // replace "Page n" with "Page n of nn"
  sprintf(str, "Page %d", k);
  sprintf(str2, "Page %d of %d", k, groupPages);
  TextReplace(str2, stc, 524.90, 509.04, 72, 10);
  // Replace page serial number as the bank pages inserted
  sprintf(str, "%07d", j++);
  sprintf(str2, "%07d", i);
  TextReplace(str2, str, 7.20, 10.80, 72, 10);
  BarCode(CODE128, acNo_keep, 0.152, 1, 1.5, 0.15, 90);
  ClosePage();
}

CloseDoc();       // End of PDF document, close input & output PDF files

#ifdef _DEBUG
  ViewPDF();        // View output PDF file only in debug mode
#endif

return 0;
}
Get Text Field by a Location Range

Function
Captures a text field string by the bottom-up coordinate location range of the data field on a PDF page.

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

Syntax

```c
char *GetField2(
    float  x1,
    float  x2,
    float  y1,
    float  y2,
    char*  str
);

UChar *GetField2U16(
    float  x1,
    float  x2,
    float  y1,
    float  y2,
    UChar* u16_str
);
```

Parameters

- **x1, x2**
  Specifies the X bottom-up position range of data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **y1, y2**
  Specifies the Y bottom-up position range of data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **str**
  Specifies a variable to store the text string of the data field captured. FastPDF Formatter converts the data field from the Unicode UTF-16 encoding to the native ASCII/DBCS-PC encoding. Make sure the DefaultCode() function is called before this function is called.

- **u16_str**
  Specifies a variable to store the Unicode UTF-16LE text string of the data field captured.

Sample

None.
Get Text Field by a Location Range and a Pattern

Function

Captures a text field string by the bottom-up coordinate location range of the data field on a PDF page and a matching pattern of symbols.

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

Syntax

```c
char *GetField3(
    float  x1,
    float  x2,
    float  y1,
    float  y2,
    char*   str,
    char*   mask
);
```

```c
UChar *GetField3U16(
    float  x1,
    float  x2,
    float  y1,
    float  y2,
    UChar* u16_str,
    UChar* u16_mask
);
```

Parameters

- **x1, x2**
  Specifies the X bottom-up position range of data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **y1, y2**
  Specifies the Y bottom-up position range of data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **str**
  Specifies a variable to store the text string of the data field captured. FastPDF Formatter converts the data field from the Unicode UTF-16 encoding to the native ASCII/DBCS-PC encoding. Make sure the DefaultCode() function is called before this function is called.

- **u16_str**
  Specifies a variable to store the Unicode UTF-16LE text string of the data field captured.

- **mask, u16_mask**
  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 mask 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 Image Information

Functions

Get the information of an image from its files, such as DPI, size, and pages.

Syntax

```c
void GetImageDPI(
    char *  image_file,
    UI32   &x_res,
    UI32   &y_res,
    UI16  index = 0
);

void GetImageDPIW(
    UChar *  utf16_image_file,
    UI32   &x_res,
    UI32   &y_res,
    UI16  index = 0
);

UI32 GetImagePages(
    char *   image_file
);

UI32 GetImagePagesW(
    UChar * utf16_image_file
);

void GetImageSize(
    char *  image_file,
    UI32   &width,
    UI32   &height,
    UI16  index = 0
);
```

Parameters

- **image_file**
The filename of the image.

- **x_res, y_res**
The horizontal resolution and vertical resolution of the image to be retrieved.

- **width, height**
The width and height of the image to be retrieved, in the point measurement unit.

- **index**
The image index number starting at 0 for the multi-page image.

Sample

None.
Graphic Path Drawing

Functions

In PDF, vector graphics are constructed by the paths. A path is the basic element of the PDF vector graphic. A path itself is invisible until it was filled, stroked, or both.

Syntax

Closes an opened path and applies filling mode:

```c
void GClosePath(
    PathFill_mode fill_mode
);
```

Draws a path from the current position to a specifying point:

```c
void GLineTo(
    float to_x,
    float to_y
);
```

Parameters

`fill_mode`

In the graphic filling mode, you may use the following recommended values:

- `FILL_PATH` Fill color into the path does not stroke path
- `FILL_STROKE` Fill color into the path and stroke path
- `FILL_STROKE_NOCLOSE` Fill and stroke path does not close the path
- `STROKE_PATH` Stroke path only
- `STROKE_NOCLOSE` Stroke path only does not close the path

`to_x, to_y`

The coordinate of the position drawing to. The current position must be defined previously with GPos() function before calling GLineTo() function.

Sample

```c
SetUnit(CM);
OpenDoc();
OpenPage();
Color(CYAN);
StrokeColor(BLUE);
GPos(2, 1);            // plots a triangle
GLineTo(3, 2);
GLineTo(1, 2);
GClosePath(FILL_PATH);
GPos(4.2, 1);
GLineTo(5.2, 2);
GLineTo(3.2, 2);
GClosePath(FILL_STROKE);
GPos(2, 2.5);
GLineTo(3.5, 2);
GLineTo(1, 3.5);
GClosePath(STROKE_PATH);
GPos(4.2, 2.5);
```
GLineTo(5.2, 3.5);
GLineTo(3.2, 3.5);
GClosePath(STROKE_NOCLOSE);

GPos(2, 4);
GLineTo(3, 5);
GLineTo(1, 5);
GClosePath(FILL_STROKE_NOCLOSE);

ClosePage();
CloseDoc();

Output:
Graphic Position

Function

Sets the vector graphic current X and Y coordinates.

Syntax

```c
void GPos(
    float  x_pos,
    float  y_pos
);
```

Parameters

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

Sample

```c
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
GLineWidth(0.0075);
GPos(3,4);    // set graphic current position at (3",4.5")
GLineTo(5, 6);  // plot a line from (3",4.5") to (5",6")
GCosePath(STROKE_PATH) // close and stroke the line
ClosePage();
CloseDoc();
```
Horizontal Boxes

Function

Plots multiple graphic boxes horizontally.

The thickness and type of border can be defined by the GLineWidth() and GLineType() functions.

Syntax

void GHboxes(
    float    x_pos,
    float    y_pos,
    float    width,
    float    height,
    UI16     repeat,
    float    gap_space,
    PathFill_mode  fill_mode = STROKE_PATH;
    float    radius = 0
);

Parameters

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

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

width
The width of the box.

height
The height of the box.

repeat
The number of additional boxes to be repeated.

gap_space
The gap space between the boxes.

fill_mode
In the graphic filling mode, you may use the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILL_PATH</td>
<td>Fill color into the box, does not stroke box boundary</td>
</tr>
<tr>
<td>FILL_STROKE</td>
<td>Fill color into the box and stroke box boundary</td>
</tr>
<tr>
<td>STROKE_PATH</td>
<td>Stroke box boundary does not fill color into box</td>
</tr>
</tbody>
</table>

radius
The radius of rounding of box corners.

Sample

SetUnit(MM);
OpenDoc();
OpenPage();

:
StrokeColor(BLACK);

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

ClosePage();

CloseDoc();
Horizontal Line

Function

Plots a graphic horizontal line.

The thickness and type of line can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GHline(
    float    x_pos,
    float    y_pos,
    float    length
);
```

Parameters

- **x_pos**: The X starting position of the line.
- **y_pos**: The Y starting position of the line.
- **length**: The length of the line.

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage(DIN_A4);

GLineWidth(0.5); // line width is 0.5 mm

GHline(10,10,80); // plot a horizontal line from (10,10) mm, // 80 mm length

ClosePage();
CloseDoc();
```
Horizontal Lines

Function

Plots multiple horizontal graphic lines.

The thickness and type of line can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GHlines(
    float    x_pos,
    float    y_pos,
    float    length,
    UI16      repeat,
    float    space,
    RepeatDirection  direction = DOWN
);
```

Parameters

- **x_pos**
The X starting position of the line.
- **y_pos**
The Y starting position of the line.
- **length**
The length of the line.
- **repeat**
The number of additional lines to be repeated.
- **space**
The gap space between the lines.
- **direction**
The direction of line repeating, valid values are ACROSS and DOWN, default is DOWN.

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage();
:    
:    GHlines(10,10,80,9,5);   // plot 10 horizontal lines from (10,10)
    :    // length is 80 mm, 5 mm space between
    :    // lines
    
:    
ClosePage();
CloseDoc();
```
Image Resolution

Function

Defines the resolution in DPI (Dots per Inch), in which images are to be outputted by FastPDF Formatter. Monochrome 1-bit images are always stored in the original resolution.

Syntax

```c
void ImageResolution(
    UI16 value_dpi
);
```

Parameters

- **value_dpi**
  The value of image resolution must be 72 to 2048 DPI.

Sample

```c
SetUnit(MM);
OpenDoc();

ImageResolution(150);
CompressionFilter(JPEG);
CompressionLevel(MAX_COMPRESS);
JpegQuality(80);
OpenPage();

: 
:

InclImage("mafp1.jpg", 0.3, 0.35);
:
:

ClosePage();
CloseDoc();
```
Lines (Multiple Connected Lines)

Function

Plots a series of connected fillet lines.

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

Syntax

From specified position:

```c
void GLines(
    float     x_pos,
    float     y_pos,
    float*    x_array,
    float*    y_array,
    ushort    counter,
    PathFill_mode  fill_mode = STROKE_NOCLOSE
);
```

Parameters

x_pos
The X starting position.

y_pos
The Y starting position.

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

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

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

fill_mode
In the graphic filling mode, you may use the following recommended values:

- FILL_PATH: Fill color into the path does not stroke the path
- FILL_STROKE: Fill color into the path and stroke the path
- FILL_STROKE_NOCLOSE: Fill and stroke path does not close the path
- STROKE_PATH: Stroke path only
- STROKE_NOCLOSE: Stroke path only does not close the path

Sample

```c
float x_array[] = { 100, 107, 115, 128, 139};
float y_array[] = { 280, 235, 245, 255, 275};
GLines(75, 275, x_array, y_array, 5);
```
Line Ending Style

Function

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

Syntax

```java
void GLineEndStyle(
    EndStyle  line_end_style
);
```

Parameters

- **line_cap_style**

Valid line ending styles are:

- **FLAT_END**
  - The stroke is squared off at the endpoint of the path. There is no projection beyond the end of the path. It is MakePDF Formatter default value.

- **ROUND_END**
  - A semicircular arc with a diameter equal to the line width is drawn around the endpoint and filled in.

- **SQUARE_END**
  - The stroke continues beyond the endpoint of the path for a distance equal to half the line width and is squared off.

Sample

```java
SetUnit(MM);
OpenDoc();
OpenPage();

GLineEndStyle(ROUND_END);
GLine(10,10,80);

ClosePage();
CloseDoc();
```
Line Join Style

Function

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

Syntax

```c
void GLineJoinStyle(
    JoinStyle line_join_style
);
```

Parameters

- **line_join_style**
- Valid line join styles are:
  - MITER_JOIN
  - ROUND_JOIN
  - BEVEL_JOIN

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

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

  The two segments are finished with butt caps (see LineEndStyle() function) and the resulting notch beyond the ends of the segments is filled with a triangle.

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage();

GLineJoinStyle(ROUND_JOIN);
GBox(10, 10, 80, 20);

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

Function

Sets the type of graphic line/path to be used with the subsequent graphic drawing.

Syntax

```c
void GLineType(
    LinePattern   line_pattern_name
);

void GLineType2(
    char *   dash_pattern_array,
    UI16   dash_phase = 0
);
```

Parameters

`line_pattern_name`
The name of the line pattern. Valid values are:

- SOLID: Solid line.
- DOTTED: Dotted line.
- SHORT_DASH: Short dashed line.
- DASH_DOT: Dash-dot line.
- FLOAT_DOT: Float dotted line.
- LONG_DASH: Long dashed line.
- DASH_FLOAT_DOT: Dash-float-dot line.

`dash_pattern` 
Defines the line dash pattern, it controls the pattern of dashes and gaps used to stroke paths. It is specified by an array of numbers that specify the lengths of alternating dashes and gaps.

`dash_phase`
Dash phase specifies the distance into the dash pattern at which to start the dash.

Before beginning to stroke a path, the dash array is cycled through, adding up the lengths of dashes and gaps. When the accumulated length equals the value specified by the dash phase, the stroking of the path begins, and the dash array is used cyclically from that point onward.

<table>
<thead>
<tr>
<th>DASH ARRAY AND PHASE</th>
<th>APPEARANCE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1] 0</td>
<td></td>
<td>No dash; solid, unbroken lines</td>
</tr>
<tr>
<td>[3] 0</td>
<td></td>
<td>3 units on, 3 units off, …</td>
</tr>
<tr>
<td>[2] 1</td>
<td></td>
<td>1 on, 2 off, 2 on, 2 off, …</td>
</tr>
<tr>
<td>[2 1] 0</td>
<td></td>
<td>2 on, 1 off, 2 on, 1 off, …</td>
</tr>
<tr>
<td>[3 5] 6</td>
<td></td>
<td>2 off, 3 on, 5 off, 3 on, 5 off, …</td>
</tr>
<tr>
<td>[2 3] 11</td>
<td></td>
<td>1 on, 3 off, 2 on, 3 off, 2 on, …</td>
</tr>
</tbody>
</table>
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
StrokeColor(RED);
GLineWidth(0.01);   // set line width to zero.01" for path
GLineType(SHORT_DASH);  // set line type to short dash
GHline(0.5, 1, 2.5);   // horizontal line at (0.5", 1")
StrokeColor(BLUE);
GLineType2("4 4");
GHline(0.5, 1.2, 2.5);
GLineType2("8 8");
GHline(0.5, 1.4, 2.5);
GLineType2("12 4");
GHline(0.5, 2.0, 2.5);
GLineType2("16 3 4 3");
GHline(0.5, 2.2, 2.5);
GLineType2("13 3 2 3 2 3");
GHline(0.5, 2.4, 2.5);

ClosePage();
CloseDoc();

\textbf{Output:}

\begin{verbatim}
+------------------+
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
+------------------+
\end{verbatim}
Line Width

Function

Sets the width of the graphic line/path to be used with the subsequent graphic drawing.

Syntax

```c
void GLineWidth(
    float  line_width
);
```

Parameters

- **line_width**
  Specified the width of the line.

Sample

```c
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
GLineWidth(0.01);
GBox(0.5, 4, 7.5, 1); // box size 7.5" x 1", at (0.5", 4")
ClosePage();
CloseDoc();
```
Inch Value

Function

Specifies a value in inches.

Syntax

float inch(
    float    value
);

Parameters

value
The value in inches.

Sample

SetUnit(MM);
OpenDoc();
OpenPage();

: : :

Pos(10,10);       // set X and Y position to (10,10)mm

: : :

Pos(inch(2), 35); // Set X position to 2" and Y position to 35mm

: : :

ClosePage();
CloseDoc();
**Import Image**

**Function**

Imports an image. FastPDF Formatter supports the most important image formats used by the industry such as image formats of BMP, EMF, GIF, JPEG, JPEG2000, PNG, PSD, TIFF, etc.

Uses InclImage() function instead if an image is to be used as a resource object. Using an object as a resource gives us much higher formatting performance with a smaller PDF data stream when that object appears more than once in a PDF document file.

**Syntax**

```c
void ImportImage(
    char * image_file,
    float   x_pos,
    float   y_pos,
    float   width = 0,
    float   height = 0,
    UI16   dpi = 0,
    float   angle = 0,
    UI16   index = 0
);
```

**Parameters**

- **image_file**
  The filename of the image, can be fully qualified with the pathname, or an image filename without the full pathname if it is being imported from the resource library directory specifying by the command-line flag –r parameter or the RESLIB parameter in the definition file of FastPDF Formatter.

- **x_pos, y_pos**
  The (X, Y) coordinate of the top-left corner of the destination bounding rectangle.

- **width, height**
  The width and height of the destination rectangle.

  A negative value of width/height mirrors the image on the x-axis/y-axis; Uses the original image size if both values are 0; If a non-zero value is specified with width or height, then the 0 value for width or height specifies that its value is calculated proportionally to preserve the aspect ratio relative to its original image size.

- **dpi**
  The image resolution in DPI (dots per inch), the default value is 0, uses the resolution information from the original image, but some image formats or image software may not give or give wrong image resolution information. Make sure your image resolution is properly specified.

- **angle**
  The rotation angle of the destination rectangle.

- **index**
  Defines the image page index starting at 0 for the first page in a multi-page image.

**Sample**

```c
SetUnit(MM);
OpenDoc();
```
OpenPage();

ImportImage("d:\images\policy001.jpg", 10, 10);

ClosePage();

CloseDoc();
Import PDF File

Function

Imports all the PDF pages from a PDF document file.

Using a PDF page as the template resource object is much efficient if that template appears more than once in the output PDF document. Refer to InclOvly() function for more details.

Syntax

```c
void ImportPDF(
    char *    pdf_file,
    UI32      dest_page,
    float     x_scale = 1,
    float     y_scale = 1
);

void ImportPDFW(
    UChar *   utf16_pdf_file,
    UI32      dest_page,
    float     x_scale = 1,
    float     y_scale = 1
);
```

Parameters

- **pdf_file, utf16_pdf_file**
  The name of the PDF file can be fully qualified with the pathname, or a PDF filename without the full pathname if it is being imported from the resource library directory specifying by the command-line flag -r parameter or the RESLIB parameter in the definition file of FastPDF Formatter.

- **dest_page**
  The destination page number at which to insert or overlap(if the pages already exist) with the importing PDF pages.

- **x_scale**
  The Horizontal scaling factor, default is 1 without a scaling.

- **y_scale**
  The vertical scaling factor, default is 1 without a scaling.

Sample

```c
SetUnit(MM);
OpenDoc();

ImportPDF("d:\pdffiles\policy001.pdf", 10); // imports PDF file and at
// destination page 10 to
// insert the importing pages

CloseDoc();
```
Include Image

Function

Include an image as an image resource object. FastPDF Formatter supports the most important image formats used by the industry such as image formats of BMP, EMF, GIF, JPEG, JPEG2000, PNG, PSD, TIFF, etc. FastPDF Formatter allows you to include up to 256 images as the resource objects for each PDF file.

Using an image as the resource object gives us much higher formatting performance with a smaller PDF data stream when that object appears more than once in a PDF document file.

Syntax

void InclImage(
    char *  image_file,
    float   x_pos,
    float   y_pos,
    float   width = 0,
    float   height = 0,
    UI16    dpi = 0,
    float   angle = 0,
    UI16    index = 0
);

void InclImageW(
    UChar *  utf16_image_file,
    float   x_pos,
    float   y_pos,
    float   width = 0,
    float   height = 0,
    UI16    dpi = 0,
    float   angle = 0,
    UI16    index = 0
);

Parameters

image_file, utf16_image_file
The filename of the image can be fully qualified with the pathname, or an image filename without full pathname if it is being included from the resource library directory specifying by the command-line flag –r parameter or the RESLIB parameter in the definition file of FastPDF Formatter.

x_pos, y_pos
The (X, Y) coordinate of the top-left corner of the destination rectangle.

width, height
The width and height of the destination rectangle.

A negative value of width/height mirrors the image on the x-axis/y-axis; Uses the original image size if both values are 0; If a non-zero value is specified with width or height, then the 0 value for width or height specifies that its value is calculated proportionally to preserve the aspect ratio relative to its original image size.

dpi
The image resolution in DPI (dots per inch), the default value is 0, uses the resolution information from the original image, but some image formats or image software may not give or give wrong image resolution information. Make sure your image resolution is properly specified.
angle
The rotation angle of the destination rectangle.

index
Defines the image index starting at 0 for the multi-page image.

Sample

SetUnit(MM);
OpenDoc();
OpenPage();

InclImage("d:\images\policy001.jpg", 10, 10);

ClosePage();
CloseDoc();
Include Overlay

Function

Includes the first page of a PDF file as the overlay/template resource object. FastPDF Formatter allows you to include up to 256 overlays as the resource objects for each PDF file.

Using a PDF page as the overlay resource object is much efficient if that template appears more than once in the output PDF document.

Syntax

```c
UI32 Inc1ovly(
    char * template_file,
    float x_pos,
    float y_pos,
    UI16 pageNo = 1 ;
    float width = 0,
    float height = 0,
    UI16 orientation = 0 ;
);

UI32 Inc1ovlyW(
    UChar * utf16_template_file,
    float x_pos,
    float y_pos,
    UI16 pageNo = 1 ;
    float width = 0,
    float height = 0,
    UI16 orientation = 0 ;
);

void Inc1ovlyH(
    UI32 template_handle,
    float x_pos,
    float y_pos,
    float width = 0,
    float height = 0,
    UI16 orientation = 0 ;
);
```

Parameters

**template_file, utf16_template_file**
The filename of the PDF file can be fully qualified with the pathname, or a PDF filename without the full pathname if it is being included from the resource library directory specifying by the command-line flag –r parameter or the RESLIB parameter in the definition file of FastPDF Formatter.

**x_pos, y_pos**
The (X, Y) coordinate of the top-left corner of the destination rectangle.

**pageNo**
Specifies which PDF page to be imported as a template. The default value is page 1. You may generate all templates into one PDF file by your Windows GUI application, by this approach we can reduce the output PDF file size.
**width, height**
The width and height of the destination rectangle.

Uses the original PDF page size if both values are 0; if a non-zero value is specified with width or height, then the 0 value for width or height specifies that its value is calculated proportionally to preserve the aspect ratio relative to its original PDF page size.

**orientation**
Defines the orientation for the subsequence input PDF pages. Supported values are 0, 90, 180, and 270, which rotates the subsequence input PDF pages clockwise.

**template_handle**
The handle number of a template created previously by either InclOvly() function with a PDF filename or BgnOvly() function.

### Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage();
:
:
InclOvly("TEST01.pdf", 10, 10); // Include an overlay at (10,10) mm
:
:
ClosePage();
CloseDoc();
```
JPEG Image Quality

Function

Defines the quality of output JPEG images in percent if JPEG compression is used. Lower values cause higher compression rates, however with worse image quality.

The value must be changed if required before the image is included or imported.

Syntax

void JpegQuality(
    UI16   percent_value
);

Parameters

percent_value
The value in percent for the output image quality in JPEG format.

Sample

SetUnit(MM);
OpenDoc();
ImageResolution(150);
CompressionFilter(JPEG);
CompressionLevel(MAX_COMPRESS);
JpegQuality(80);
OpenPage();
    :
    :
InclImage("mafp1.jpg", 0.3, 0.35);
    :
    :
ClosePage();
CloseDoc();
Left Align Text

Function

Left aligns a line text string.

Syntax

void Ltxt(
    char * data,
);

void Lu8(
    UChar8 * utf8_data,
);

void Lu16(
    UChar * utf16le_data,
);

Parameters

data
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

utf8_data
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_data
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

Sample

DefaultCode("gb18030");        // input data is Chinese GB18030
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
Font("Arial", 10);               // Define a font for English
    Pos(2,2);                    // Current text position at (2",2")
    Ltxt("text is left aligned");         // Left align text at (2",2")
    Font("SimSun", 10, REGULAR, cpUnicode); // Define a font for Chinese, uses Unicode encoding with this font
    Pos(2,2.5);                   // Current text position at (2",2.5")
    Ltxt("实现 Windows 与 Linux 的双引导"); // Left align Chinese text, Converts Chinese to UTF-16LE
ClosePage();
CloseDoc();
**Linechart**

**Function**

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

**Syntax**

```c
void GLinechart(
    float      width,
    float  height,
    UI32       vertical_scale_unit,
    UI16       vertical_scales,
    float*      values,
    UI16   numData_perLine,
    char*  label_texts[],
    UI16        numLabels,
    char*  legend_texts[],
    UI16       lines = 1,
    float       line_weight = DEFAULT,
    bool   grid_line = true,
    bool   symbol_node = false,
    bool   line_label = false,
    UI132*   line_colors = NULL,
    LinePattern*   line_types = NULL
);
```

**Parameters**

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

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

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

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

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

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

- **label_texts**
  The array of label texts under the horizontal axis.

- **num_labels**
  The number of labels under the horizontal axis.

- **legend_texts**
  The array of texts for the legend.

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

- **Line_weight**
  Defines line weight, default value is about 0.015 inches.
grid_line
Specifies whether gridlines are plotted on the linechart, default is TRUE.

symbol_node
Specifies whether draws the symbol at each line node, default is FALSE.

line_label
Specifies whether draws the legend text along with each graphic line, default is FALSE.

line_colors
The one dimension array defining with the predefined color names, such as BLUE, RED, PINK, GREEN, CYAN, YELLOW, BLACK, BROWN, etc, default value is BLACK. See herder file MakePDF.h for details, you can add your color alias names in the MakePDF.h header file.

line_pattern_name
The name of the line pattern. Valid values are:

SOLID   Solid line.
DOTTED  Dotted line.
SHORT_DASH Short dashed line.
DASH_DOT Dash-dot line.
FLOAT_DOT Float dotted line.
LONG_DASH Long dashed line.
DASH_FLOAT_DOT Dash-float-dot line.

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

char *lbs[] = { "02.2", "02.4", "02.6", "02.8", "02.10",  
 "02.12", "02.14", "02.16", "02.18", "02.20",  
 "02.22", "02.24", "02.26", "02.28", "02.30" };  

char *lgdText[] = { "Power", "Water" };  
USHORT lineColors[] = { RED, DARKBLUE };  

Start();
SetUnit(INCH);  // set default units to inch, 600 dpi
OpenDoc();
OpenPage(8.27, 11.67);

UI16 f1 = Font("Helvetica", 6, BOLD);  // define the default font for  
  // Linechart legend texts
Font("Helvetica", 6);  // define the font for Linechart axis texts
GPos(1.75,3);  // Set position of linechart

GLinechartLegend(3.8,  
  -0.4,  // relative Y position
  0.3,  // line length,
  true, // vertical legend
  f1);  // font handle ID for the legend texts

GLinechart(3.5,  // Linechart width
  2,  // Linechart height
  10, // V-axis unit
  ...
10,              // V-axis scales
data,              // Data values by 1D array
15,               // Number of values per line
lbls,             // H-axis label texts
15,               // Number of H-axis labels
lgdText,          // legend texts
2,                // Draw two lines
0.01,             // Line weight, 0.01"
true,             // Draw grid line
false,            // no symbol nodes
false,            // no line labels
lineColors);      // Override the default colors of lines

ClosePage();
CloseDoc();

Output:
Graphic Linechart Legend

Function

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

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

Syntax

```c
void GLinechartLegend(
    float  x_rel_pos,
    float  y_rel_pos,
    float  line_length,
    bool   vertical_layout = TRUE,
    UI16   legend_texts_fontID = DEFAULT
);
```

Parameters

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

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

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

- **legend_texts_fontID**
  Defines the font ID to be used for the legend texts. The font handle number must be a valid font handle that was created by the Font() function.

Sample

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

char *lbs[] = { "02.2", "02.4", "02.6", "02.8", "02.10", "02.12", "02.14", "02.16", "02.18", "02.20", "02.22", "02.24", "02.26", "02.28", "02.30" }; 

char *lgdText[] = { "Power", "Water" }; 

USHORT lineColors[] = { RED, DARKBLUE }; 

Start(); 
    SetUnit(INCH); // set default units to inch, 600 dpi 
    OpenDoc(); 
```
OpenPage(8.27, 11.67);

UI16 f1 = Font("Helvetica", 6, BOLD);     // define the default font for Linechart legend texts

Font("Helvetica", 6);      // define the font for Linechart axis texts

GLinechartLegend(1, -2, 0.3, false, f1);   // draw horizontal legend

GPos(1.75,6);              // Set position of linechart

GLinechart(3.5,           // Linechart width
2,             // Linechart height
10,            // V-axis unit
10,            // V-axis scales
&data2[0][0],  // Data values by 2D array
15,            // Number of values per line
lblr,          // H-axis label texts
15,            // Number of H-axis labels
lgdText,       // legend texts
2,             // Draw two lines
0.01,          // Line weight, 0.01"
false,         // No grid lines
true,          // draw symbol nodes
true,          // draw line labels along with lines
lineColors);    // Override the default colors of lines

ClosePage();

CloseDoc();

Output:
### Lines Per Inch

**Function**

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

**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);
OpenDoc();
OpenPage();

...

LPI(8);  // subsequent texts will be in 8 LPI
...

LPI(6.5);  // subsequent texts will be in 6.5 LPI
...

ClosePage();
CloseDoc();
```
Line Spacing

Function

Defines the default vertical baseline spacing in terms of the current measurement unit for the subsequent texts.

Syntax

```c
void LineSpace(
    float increment
);
```

Parameters

increment
The baseline increment in terms of the measurement unit for the subsequent text.

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage();

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

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

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

**Function**

Sets the inline left margin for the subsequent text to be positioned with the function of NextLine() or Skip().

**Syntax**

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

**Parameters**

**margin**
The left inline margin for the texts in terms of the current measurement unit.

**Sample**

```c
SetUnit(INCH);
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 a PDF page.

You can hide areas on a PDF page that you do not want to display or print, for instance, you might hide an area that contains OMR lines before you create a new barcode at the same area.

Syntax

```c
void MaskArea(
    float    x,
    float    y,
    float    width,
    float    height
);
```

Parameters

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

- **y**
  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.
**Matrix Text**

**Function**

Shows a text string by using a transformation matrix to calculate the presentation of the text string.

**Syntax**

```c
void MatrixTxt(
    char * text,
    float  a,
    float  b,
    float  c,
    float  d,
    float  x,
    float  y
);
```

```c
void MatrixU8(
    UChar8 * utf8_text,
    float  a,
    float  b,
    float  c,
    float  d,
    float  x,
    float  y
);
```

```c
void MatrixU16(
    UChar * utf16_text,
    float  a,
    float  b,
    float  c,
    float  d,
    float  x,
    float  y
);
```

**Parameters**

**text**
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

**utf8_text**
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

**utf16_text**
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

**a, b, c, d, x, y**
The elements of a transformation CTM (Current Transformation Matrix) are defined by Adobe PDF and PostScript.
Sample

char *text = "The quick brown fox jumps over the lazy dog. ";
Start();
SetUnit(PT);
OpenDoc();
OpenPage();
Circle(210, 190, 145);
Circle(210, 190, 113);
Font("Courier", 30, BOLD);
float r1, r2, a, b, c, d, x, y;
char str[2] = "";
float angle1 = 360 / strlen(text);
float angle2 = 180;
for (UI16 i = 0; i < strlen(text); i++)
{
    r1 = (angle2 - 90) / 180 * 3.141592;
    r2 = angle2 / 180 * 3.141592;
    a = cos(r1);    b = -sin(r1);
    c = -sin(r1);   d = -cos(r1);
    x = 210 + cos(r2) * 122;
    y = 190 - sin(r2) * 122;
    *str = text[i];
    MatrixTxt(str, a, b, c, d, x, y);
    angle2 -= angle1;
}
ClosePage();
CloseDoc();

Output:
Multiple-Line Texts

Functions

Output the multiple-line texts.

Line spacing can be defined by the LPI() or LineSpace() function.

Syntax

void BgnTxt(
    float   x_pos,
    float   y_pos
);

void PutTxt(
    char *  data
);

void PutU8(
    UChar8 *  utf8_data
);

void PutU16(
    UChar *   utf16_data
);

void EndTxt(void);

Parameters

x_pos, y_pos
Defines the baseline of the first text line.

data
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

utf8_data
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_data
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

Sample

SetUnit(INCH);
OpenDoc();
OpenPage(8,11);
Font(“Courier", 10, BOLD);
LPI(6);                 // Sets 6 Lines per inch for linedata
BgnTxt(1,1);
PutTxt("Line Text 1");
PutTxt("Line Text 2");
PutTxt("Line Text 3");
EndTxt();
ClosePage();
CloseDoc();

Output:

    Line Text 1
    Line Text 2
    Line Text 3
Millimeter Value

Function

Specifies a value in millimeters.

Syntax

```c
float mm(
    float     value
);
```

Parameters

- **value**
  - The value in millimeters.

Sample

```c
SetUnit(INCH);
OpenDoc();
OpenPage(8,11);

: :

Pos(2.5,4); // set X and Y position to (2.5",4")

: :

Pos(mm(20),3.5); // Set X position to 20 mm and Y position to 3.5"

: :

ClosePage();
CloseDoc();
```
Move PDF Page

Function

Moves a PDF page to another position in the document.

Syntax

```c
bool MovePage(
    UI32      destPage,
    UI32  sourcePage
);
```

Parameters

- **destPage**
  Specifies the destination page number, value 1 refers to page 1 of the document, and so on.

- **sourcePage**
  Specifies the source page number, value 1 refers to page 1 of the document, and so on.

The `sourcePage` must be a valid page number in a PDF document. Page numbering starts at one. If the destination page number is greater than the number of pages currently available in the PDF document, then the document is being filled with empty pages until the destination page.

Sample

None.
Next Line

Function

Starts a new text line from the left inline margin defined by the Margin() function, it increments the current baseline coordinate position by the amount of baseline increment defined by either the LIP() or LineSpace() function call.

Syntax

void Next(void);
void NextLine(void);

Parameters

No parameter to be specified.

Sample

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

LPI(8);              // Line spacing is 8 lines per inch
Margin(0.8);        // left margin for the text is 0.8"
NextLine();        // Jump to next new line position

ClosePage();
CloseDoc();
Open Default Input Data File

Function

Opens the default input data file specified by the flag parameter “-i input_data_file” of the command-line.

It returns the file pointer pointed to the default input data file opened.

It can read the input data from the standard-in (STDIN) if the default input data file is not specified with the flag -i parameter. For instance, with the following example you can read the compressed data on the fly by the system pipe while decompressing the data stream:

```
unzip -p test1.zip | makepdf_program -d test1.def.def -o test1.pdf
```

Syntax

```
FILE *OpenDataFile();
```

Parameters

None.

Sample

```
FILE *fdin;

void main(void)
{
    Start();

    fdin = OpenDataFile(); /* Open default input data file specified */
    /* by the command-line flag parameter -i */

    OpenDoc(); /* Open PDF document */

    :
    :
    :

}
```
Open Document

**Function**

Opens a PDF document, you must call this function to initialize a PDF document before you open a PDF page, and you must close this PDF document by the CloseDoc() function before you end your program.

**Syntax**

```c
void OpenDoc(
    char * pdf_filename = $PdfOutFile
);
void OpenDocW(
    UChar * pdf_filename_utf16le = $PdfOutFile
);
```

**Parameters**

*Pdf_filename*

Specifies the PDF output filename, default is the PDF filename specifying with the command-line flag parameter –o.

If you specify an empty string with value NULL to PDF filename, then the PDF file is created in memory, you can either get this PDF file memory block by CloseDoc() function or output PDF as a file by specifying a PDF filename with CloseDoc() function, for instance, you may need such PDF filename control for the e-Statement multiple PDF outputs, that needs PDF files to be dynamically generated by the account numbers or customers names.

**Sample**

```c
Sample:

void main( )
{
    Start();
    OpenDoc(); // First PDF filename is token from command-line

    CloseDoc(); // End first PDF document and close file

    OpenDoc("test.pdf"); // Second PDF filename is specified here

    CloseDoc(); // End Second PDF document and close file

    OpenDoc(NULL); // Third PDF filename is Empty

    CloseDoc(account_no) // End third PDF document and close file, whose filename is dynamic generated by the account_no variable, MakePDF adds PDF filename extension /*.pdf if it is not specified with PDF filename
}
```
Open Page

Function

Opens a PDF page. Once the page formatting is completed, you must close the page with the ClosePage() function.

Syntax

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

void OpenPage2(
    PaperName paper_name = DIN_A4
);
```

Parameters

- **page_width**
  Width of the page.

- **page_length**
  Length of the page.

- **paper_name**
  Defines page size by a paper format name. Valid values are:

<table>
<thead>
<tr>
<th>Paper Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIN_A3</td>
<td>420 x 297 mm</td>
</tr>
<tr>
<td>DIN_A4</td>
<td>210 x 297 mm, default value</td>
</tr>
<tr>
<td>DIN_A5</td>
<td>210 x 148 mm</td>
</tr>
<tr>
<td>DIN_B4</td>
<td>250 x 353 mm</td>
</tr>
<tr>
<td>DIN_B5</td>
<td>176 x 250 mm</td>
</tr>
<tr>
<td>DIN_B6</td>
<td>125 x 176 mm</td>
</tr>
<tr>
<td>DIN_C3</td>
<td>324 x 458 mm</td>
</tr>
<tr>
<td>DIN_C4</td>
<td>229 x 324 mm</td>
</tr>
<tr>
<td>DIN_C5</td>
<td>162 x 229 mm</td>
</tr>
<tr>
<td>DIN_C6</td>
<td>114 x 162 mm</td>
</tr>
<tr>
<td>DIN_C65</td>
<td>114 x 229 mm</td>
</tr>
<tr>
<td>DIN_DL</td>
<td>210 x 98 mm</td>
</tr>
<tr>
<td>DIN_E4</td>
<td>280 x 400 mm</td>
</tr>
<tr>
<td>DIN_E5</td>
<td>200 x 280 mm</td>
</tr>
<tr>
<td>DIN_E6</td>
<td>200 x 140 mm</td>
</tr>
<tr>
<td>LEGAL</td>
<td>8.5 x 14 in</td>
</tr>
<tr>
<td>LETTER</td>
<td>8.5 x 11 in</td>
</tr>
</tbody>
</table>

Sample

```c
SetUnit(INCH);
Start();                   // Start a FastPDF Formatter session
OpenDoc();
```
OpenPage(8.27, 11.67);  // A4 size page, 8.27" x 11.67"

ClosePage();

OpenPage2(DIN_A4);

ClosePage();

CloseDoc();
Page Height

Function

Returns the page height of the current PDF page.

Syntax

void PageHeight(void);

Parameters

None.

Sample

None.
Page Width

Function

Returns the page width of the current PDF page.

Syntax

    void PageWidth(void);

Parameters

    None.

Sample

    None.
Pages of Input PDF File

Function

Returns the number of pages of the input PDF file.

Syntax

    void InPages(void);

Parameters

    None.

Sample

    None.
Paragraph of Texts

Functions

Format a paragraph of texts.

You can call the function of LPI() or LineSpace() first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

Paragraph functions support the escape formatting control codes to be inserted in your texts. Refer to the function of ParFlow() for more details.

Syntax

```c
void ParTxt(
    char *     text,
    float      x_pos,
    float      y_pos,
    float      width,
    float      height = -1,
    Alignment     align = LEFT,
    bool      fit = true,
    float     angle = 0,
    float      skew_alpha = 0,
    float      skew_beta = 0
);

void ParU8(
    UChar8 *     utf8_text,
    float      x_pos,
    float      y_pos,
    float      width,
    float      height = -1,
    Alignment     align = LEFT,
    bool      fit = true,
    float     angle = 0,
    float      skew_alpha = 0,
    float      skew_beta = 0
);

void ParU16(
    UChar *     utf16_text,
    float      x_pos,
    float      y_pos,
    float      width,
    float      height = -1,
    Alignment     align = LEFT,
    bool      fit = true,
    float     angle = 0,
    float      skew_alpha = 0,
    float      skew_beta = 0
);```

Parameters

text
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

utf8_text
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_text
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

The escape formatting control codes are allowed to insert into your texts so that some formatting attributes can be dynamically defined. Refer to the function of ParFlow() for more details.

x_pos, y_pos
The (X, Y) coordinate of the top-left corner of the bounding rectangle area.

width, height
The width and height of the bounding rectangle area. The default value for height is -1, which defines that it is auto-determined and no page break.

align
Specifies how the paragraph should be aligned. The valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT</td>
<td>Texts are left aligned</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Texts are right aligned</td>
</tr>
<tr>
<td>CENTER</td>
<td>Texts are center-aligned</td>
</tr>
<tr>
<td>JUSTIFY</td>
<td>Texts are justify aligned</td>
</tr>
</tbody>
</table>

fit
Specifies whether paragraph should be fit into the dimension of the bounding rectangle area. The default value is TRUE.

angle
Specifies an angle to rotate the bounding rectangle area. The default value is 0.

skew_alpha, skew_beta
Defines the alpha and beta angle in the degree to skew the texts. Default values are 0.

Sample

cchar *text = "The Portable Document Format (PDF), undeniably one of the most "
"commonly used formats for electronic documents is now "
"accessible as an ISO International Standard - ISO 32000-1."

DefaultCode("ibm-437"); // Data from codepage 437, USA ASCII

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

OpenDoc();
OpenPage();
Font("Arial", 10);
LPI(5); // Set line density to 5 lines per inch
Color(CYAN);
The Portable Document Format (PDF), undeniably one of the most commonly used formats for electronic documents, is now accessible as an ISO International Standard - ISO 32000-1.
Paragraph of Texts (Multi-Column Texts Flowing)

Functions

Format the multi-column paragraphs with page-breaking across multi-page.

You can call the function of LPI() or LineSpace() first to set the line spacing before you call this function. You must ensure that the paragraph fits on the page.

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

The escape formatting control codes are allowed to insert into your texts so that some formatting attributes can be dynamically defined.

Syntax

```c
void ParFlow(
    char *     text,
    float      x_pos,
    float      y_pos,
    float      width,
    float      height,
    UI16     columns,
    Float      gap_space
    Alignment     align = LEFT,
);
```

```c
void ParFlowU8(
    UChar8 *    utf8_text,
    float      x_pos,
    float      y_pos,
    float      width,
    float      height,
    UI16     columns,
    Float      gap_space
    Alignment     align = LEFT,
);
```

```c
void ParFlowU16(
    UChar *     utf16_text,
    float      x_pos,
    float      y_pos,
    float      width,
    float      height,
    UI16     columns,
    Float      gap_space
    Alignment     align = LEFT,
);
```

Parameters

text
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

The (X, Y) coordinate of the top-left corner of the first bounding rectangle area.

The width and height of a bounding rectangle area.

Specifies how many columns.

Specifies the gap space between the columns.

Specifies how the paragraph should be aligned. The valid values are:

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

### Escape Formatting Control Codes

The following control codes are supported:

- **\LA#**: Left aligns text
- **\RA#**: Right aligns text
- **\CA#**: Center aligns text
- **\JA#**: Justify aligns text
- **\UL#**: Activates the underlined text drawing or deactivates it
- **\ST#**: Activates the strike text drawing or deactivates it
- **\US#**: Activates the underline \ strike text or deactivates it
- **\NP#**: Starts a new page
- **\TL[int]**: Sets the current tabulator length
- **\LS[float]**: Sets the line spacing
- **\CS[float]**: Sets the character spacing
- **\TR[float]**: Sets the text rise
- **\TS[float]**: Sets the text width scaling
- **\FS[float]**: Sets the current font size
- **\FC[UI32]**: Sets the text color
- **\FN[int or char*]**: Sets a font by its font handle number or PostScript name
- **\BM[char*]**: Adds a bookmark
- **\LI[float,int,float]**: Defines the text List, the first parameter defines the left margin of the list character measured from the left side bounding box; the second and third are optional parameters, which define the list character index decimal number that should be used and the distance between the list character and the text. The last character must be a valid character index decimal number of the Windows font Wingdings, you may verify it by the demo version of FontLab Studio, or by MS-Word options Insert-->Symbol…--> Select Wingdings font-->See the value at Character code. Default list character is the one in decimal index 159 of the font Wingdings, whose symbol glyph is (•).
The list character values from 1000 are used to form the numbered list symbol using the current font. FastPDF Formatter subtracts 1000 from the number and then presents the resulting number as list symbol plus the full-stop symbol (.)

\EL#

Ends the text list

Sample 1

Start();
OpenDoc();                   // Begin of PDF document
SetUnit(MM);                // Set default unit to MM
OpenPage();
LPI(8);
char *text = "With the explosive growth of the Internet, &FN[0]&UL#PDF has become one " "of the most common formats for document exchange&UL#, &FN[1]widely used " "in all professional and personal contexts. The format enables 
"
Font("Arial", 9.5, BOLD);                 // First font handle number is 0
Font("Arial", 9.5);                       // Second font handle number is 1
ParFlow(text, 20, 20, 50, 60, 3, 10);
ClosePage();
CloseDoc();

Output:

With the explosive growth of the Internet, PDF has become one of the most common formats for document exchange, widely used in all professional and personal contexts. The format enables:

- Preservation of document fidelity independent of device or platform
- Merging of content from diverse sources
- Collaborative editing of documents using multiple platforms
- Digital signatures for authenticity
- Security and permissions to preserve control over content
- Accessibility of content to those with disabilities
- Extraction and reuse of content for use with other file formats, and
- Gathering data and integrating it with business systems using PDF forms

Major corporations, government agencies and educational institutions use PDF to streamline their operations by replacing paper documentation with electronic exchange. Already, over 2 000 PDF product developers use this standard for their products and billions of PDF files are in existence today.
Sample 2

Formatting the texts read from a text file

```c
char* ReadFileToBuffer(const char* fileName)
{
    FILE* f = fopen(fileName, "rb");
    if (f == NULL)
        printf( "Could not open input text file" );
    char* buffer;
    UI32 bufSize;
    fseek(f, 0, SEEK_END);
    bufSize = ftell(f);
    fseek(f, 0, SEEK_SET);
    // allocate one more character for the null-terminator
    buffer = (char*)malloc(bufSize +1);
    if (!buffer) return NULL;
    fread(buffer, 1, bufSize, f);
    fclose(f);
    buffer[bufSize] = NULL;  // add NULL to end the text string
    return buffer;
}

void main( )
{
    DefaultCode("ibm-437");       // Input data and text are USA ASCII PC
    DefaultLocale("en_US");       // language locale is USA English
    Start();
    SetUnit(MM);                  // Set default units to MM
    OpenDoc();
    OpenPage();
    Color(BLACK);
    LPI(7);
    Font("Arial", 9.5);
    char *p = ReadFileToBuffer("d:\sample.txt");     // read texts from a file
    ParFlow(p, 20, 20, 50, 50, 2, 10);
    ClosePage();
    CloseDoc();
}
```

Output:

The Portable Document Format (PDF), undeniably one of the most commonly used formats for electronic documents, is now accessible as an ISO International Standard - ISO 32000-1. This move follows a decision by Adobe Systems Incorporated, original developer and copyright owner of the format, to relinquish control to ISO, who is now in charge of publishing the specifications for the current version (1.7) and for updating and developing future versions.

PDF, a digital form used to represent electronic documents, allows users to exchange and view the documents easily and reliably, independent of the environments in which they are created, viewed and printed, while preserving their content and visual appearance.
Pattern of Graphic Tiling

Function

Creates a tiling pattern. Six types of patterns can be created by this function.

Syntax

```c
void GPattern(
    TilePattern   tile_pattern,
    float    line_width,
    float     line_length,
    UI32   line_color,
    UI32    background_color,
    ColorMode   fill_mode
);
```

Parameters

**tile_pattern**
Supported tile patterns are:

- HORIZONTAL /* ----------- */
- VERTICAL /* ||||||||||| */
- RDIAGONAL /* \\\\\\\ */
- LDIAGONAL /* /////////// */
- CROSS /* +++++++++++ */
- DIACROSS /* xxxxxxxxxxx */

**line_width**
The width of line to draw pattern.

**line_length**
The length of line to draw pattern.

**line_color**
Specifies the color of the line.

**background_color**
Specifies the color of the background.

**color_mode**
Valid color modes are:

- FILL_MODE Use the tile pattern as fill color
- STROKE_MODE Use the tile pattern as stroke color
- FILL_STROKE_MODE Use the tile pattern as fill and stroke color

Sample

```c
SetUnit(PT);
OpenDoc();
OpenPage();
GLineType(SOLID);
GLineWidth(15);

GPattern(DIACROSS, 1, 4, BLUE, YELLOW, STROKE_MODE);
GPattern(CROSS, 2, 8, CYAN, WHITE, FILL_MODE);
GBox(125, 580, 75, 75, FILL_STROKE);
```
GPattern(DIACROSS, 1, 4, RED, WHITE, STROKE_MODE);
GPattern(CROSS, 2, 8, GREEN, WHITE, FILL_MODE);
GBox(225, 580, 75, 75, FILL_STROKE, 5);

OpenPage();
CloseDoc();

Output:
**Piechart**

**Function**

Plots a graphic piechart with its center of the pie at the current vector graphic position.

With the GChartColors() function, you can override the default RGB color values predefined for the graphic bar chart and pie chart.

**Syntax**

```c
void GPiechart(
    float width,
    float height,
    float shadow_depth,
    UI16 sectors,
    float* values,
    char* label_texts[],
    float start_angle = 0,
    label label_layout = CIRCLE_LABEL,
    bool plot_boundary = FALSE
);
```

**Parameters**

- **width**
  The width of the pie chart.

- **height**
  The height of the pie chart.

- **shadow_depth**
  The shadow depth of the 3D pie chart, specify value 0 for the 1D pie chart.

- **sectors**
  The number of sectors of the pie chart.

- **values**
  The array of the data values.

- **label_texts**
  The array of the label texts.

- **start_angle**
  The starting angle of pie chart in degree, default is 0.

- **label_layout**
  Defines the layout of the pie chart labels. Default is CIRCLE, labels are plotted around the pie chart. Valid values are:
  - CIRCLE_LABEL: Plots the labels around the pie chart
  - LEGEND_LABEL: Plots the percentages with label texts by the legend

- **plot_boundary**
  Defines whether the boundaries of the pie chart is plotted, default is FALSE, does not plot boundary.

**Sample**

```c
float data[]  = {45, 65, 35, 70, 40, 50};
char *label[] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
```
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
Font(1);                   // select font 1 for piechart
LPI(8);                     // Set line space to 8 LPI for label texts
PiechartExplode(1, 0.1);    // Explode sector 1, with distant 0.1"
PiechartLabel(0.10, "{percent}%\n{label}");  // change defaults of label
GPos(2, 2);                 // Set current Graphic position
GPiechart(1.6,             // Piechart width
  1,                // Piechart height
  0.13,             // Piechart shadow depth
  6,                // Slices of piechart
  data,             // Input data array
  label,            // Label text array
  0,                // Starting angle
  CIRCLE_LABEL,     // Shows labels around the piechart
  TRUE);             // Plots boundary
ClosePage();
CloseDoc();

Output:
Piechart Explode

Function

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

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

Syntax

```c
void PiechartExplode(
    int sectorNo,
    float distance = DEFAULT
);
```

Parameters

- **sectorNo**
  The sector number of the sector to be exploded from the piechart, starting from 1.

- **distance**
  The explosion distance from the perimeter of the piechart. The default value is DEFAULT; the distance is determined automatically.

Sample

See sample for the Graphic Piechart function.
Piechart Label

Function

Optional functions for the graphic piechart, which defines the position and label format of the circular labels, must be called before the PieChart() function.

Syntax

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

void PiechartLabel2(
    float   *lbl_distances,
    UI16    num_of_lbl_distances,
    char *  lbl_fmt = "\{label\}\n\{percent\}\%",
    bool    draw_label_line = FALSE
);
```

Parameters

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

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

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

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

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

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

**draw_label_line**
Defines whether draw the join lines to connect the piechart labels to the pie sectors, default is FALSE.

Sample

```c
int main( )
{
    float data[]  = {10, 35, 40, 25, 30, 45, 55};
    float label_dist_array[7] = {0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1};
```
char *label[] = {
    "Monday", "Tuesday", "Wednesday", "Thursday",
    "Friday", "Saturday", "Sunday"};

Start();
SetUnit(INCH); // Set default unit to inch
OpenDoc();
OpenPage(8.5,11);
Font(1);
PiechartLabel2(label_dist_array, 6, "{label}\n\{percent\} %", TRUE);
LPI(8); // Set line space to 8 LPI for label texts
GPos(2, 2); // Set piechart position – center of piechart
GPiechart(1.6, // Piechart width
    1.6, // Piechart height
    0, // No piechart shadow depth, it is 1D piechart
    6, // Slices of piechart
    data, // Input data array
    label); // Label text array
ClosePage();
CloseDoc();

Output:
Piechart Legend

Function

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

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

Syntax

```cpp
void PiechartLegend(
    float x_rel_pos = DEFAULT,
    float y_rel_pos = DEFAULT,
    float legend_box_size = DEFAULT
);
```

Parameters

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

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

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

Sample

None.
Point Value

Function

Specifies a value in points.

Syntax

float pt(
    float value
);

Parameters

value
    The value in points.

Sample

SetUnit(INCH);
OpenDoc();
OpenPage(8,11);
Pos(2.5,4);                   // set X and Y position to (2.5",4")
Pos(pt(20),3.5);              // Set X position to 20 points and Y position to 3.5"
ClosePage();
CloseDoc();
Polarchart

Function

Plots a graphic polarchart with its center of the circle at the current graphic position, and the current default font is to be used for its label texts.

Syntax

```c
void GPolarchart(
    float  radius,
    float  scale_unit,
    UI16      scales,
    float*      values,
    char*  label_texts[],
    UI16        num_labels,
    UI16      axis_labels_fontID,
    char*  axis_label_format = "%0.0f",
    UI16  num_groups = 1,
    float  line_weight = 2.0,
    UI32*  line_colors = NULL,
    UI32   background_RGB_color = CYAN,
    char*  legend_texts[] = NULL,
);
```

Parameters

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

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

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

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

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

- **num_labels**
The number of labels around the outer circle.

- **axis_lables_font_ID**
The font ID to be used to format the radial axis label texts. This font handle number must be a valid font handle that was created by the Font() function.

- **axis_label_format**
Specifies the format-control string to format the axis labels, refer to the MSDN library for more details about the format control string used in C standard function fprintf(). Default is "%0.0f".

- **num_groups**
The number of the line groups.

- **line_weight**
Defines line weight, default value is about 0.02 inches.

- **line_colors**
The one dimension array defining with the predefined color names, such as BLUE, RED, PINK, GREEN, CYAN, YELLOW, BLACK, BROWN, etc, default value is BLACK. See header file MakePDF.h for details, you can add your color alias names in the MakePDF.h header file.
**background_RGB_color**
Defines the circular background color in RGB value, which must be defined by format of 0x[red][green][blue], valid RGB intensity range values for each component are 0 through 255 (hex value x'FF').

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

**Sample**

```c
int main()
{
    float data2[2][6] = {{78, 80, 90, 80, 70, 75}, {35, 40, 55, 60, 40, 55}};
    char *lbs[] = {"Speed", "Reliability", "Comfort", "Safety",
                  "Efficiency", "Economy"};
    char *legend[] = {"Car-1", "Car-2"};
    Start();
    SetUnit(INCH);           // set default units to inch
    OpenDoc();
    OpenPage(8.27, 11.67);
    UI16 f1 = Font("Helvetica", 5.5);     // define the default font for
                                           // Polarchart axis texts
                                           // Polarchart axis texts
    UI16 f2 = Font("Helvetica", 8, BOLD);
    GPos(2.13, 1.7);              // Set position of polar chart
                                  // Set position of polar chart
    GPolarchartLegend(0.9, -1.1, 0.15, true);
    GPolarchart(1,            // radius of polat chart
                20,           // size per unit of radial axis
                5,            // 5 units
                data,         // data by a 1D array
                lbs,          // labels' texts
                6,            // 6 labels
                f1,           // font for the radial axis texts
                "%0.0f",      // format of axis label texts
                2,            // draw 2 groups of lines
                0.02,         // line weight 0.02"
                NULL,         // define line colors
                0xc1ffc1,     // circular background color
                legend);      // legend texts
    ClosePage();
    CloseDoc();
}
```

**Output:**

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

Function

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

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

Syntax

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

Parameters

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

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

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

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

Sample

None.
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) top-left corner.

Syntax

```c
void Pos(
    float  x_position,
    float  y_position
);
```

Parameters

- **x_position**
  The value of the absolute horizontal position from the page origin. Negative values are invalid.

- **y_position**
  The value of the absolute vertical position from the page origin. Negative values are invalid.

Sample

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

Pos(5, 10);  // Set X position to 5 mm and
             // y position to 10 mm

ClosePage();
CloseDoc();
```
Right Align Text

Function

Right aligns a line text string.

Syntax

```c
void Rtxt(
    char * data,
);
void Ru8(
    UChar8 * utf8_data,
);
void Ru16(
    UChar * utf16le_data,
);
```

Parameters

data
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

utf8_data
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_data
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

Sample

```c
DefaultCode("gb18030");       // input data is Chinese GB18030
SetUnit(INCH);
OpenDoc();
OpenPage(8.5,11);
Font("Arial", 10);  // Define a font for English
    :
Pos(2,2);                            // Current text position at (2",2")
Rtxt("text is right aligned");    // Right align text at (2",2")
    :
Font("SimSun", 10, REGULAR, cpUnicode);  // Define a font for Chinese, uses
    // Unicode encoding with this font
    :
Pos(2,2.5);                          // Current text position at (2",2.5")
Rtxt("实现 Windows 与 Linux 的双引导");  // Right align Chinese text,
    // Converts Chinese to UTF-16LE
ClosePage();
CloseDoc();
```

- 146 -
Rounded Corners Box

Function

Plots a rounded corners graphic box.

The thickness and type of border can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GRbox(
    float   x_pos,
    float   y_pos,
    float   width,
    float   height,
    float   top_left_corner,
    float   top_right_corner,
    float   bottom_left_corner,
    float   bottom_right_corner,
    PathFill_mode  fill_mode = STROKE_PATH
);
```

Parameters

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

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

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

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

- **top_left_corner, top_right_corner, bottom_left_corner, bottom_right_corner**
The radius values to be used to round the box’s corners.

- **fill_mode**
In the graphic filling mode, you may use the following values:

  - **FILL_PATH**
    Fill color into the box does not stroke box boundary
  - **FILL_STROKE**
    Fill color into the box and stroke box boundary
  - **STROKE_PATH**
    Stroke box boundary does not fill color into box

Sample

```c
SetUnit(PT);
OpenDoc();
OpenPage(8.5,11);
GLineWidth(0.01); // set box border width to zero.01"
Color(YELLOW);    // Set fill color to yellow
StrokeColor(GRAY); // Set stroke color to gray for box border
GRbox(370, 170, 150, 80, 15, 0, 0, 15, FILL_STROKE); // Box size is 150 x 80,
                                                         // point, two corners
                                                         // radius are 15 point:
```
Font("Arial", 18, BOLD);

Pos(445, 215);  // Set position at (445, 220) for text data
Color(RED);  // Set RED fill color for text
Ctxt("Rounded Box");  // Center put the text

Output:

**Rounded Box**
Set Coordinate System and Page Orientation

Function

Set and change PDF coordinate system and page orientation for the subsequence pages.

Syntax

Sets orientation of page:

DLL bool PageOrientation(SI32 degree = 0);

Sets coordinate system:

DLL bool PageCoords(Page_Coords page_cords = TOP_DOWN);

Parameters

degree
Changes the orientation of a page, including page width, page height, and the coordinate system. Allowed values are 0, 90, 180, 270, which rotates the page clockwise. FastPDF Formatter default value is zero.

The zero points of the coordinate system is set to the upper left or down left point depending on the current page coordinate system (top-down or bottom-up). FastPDF Formatter default page coordinate system is top-down, all functions are adjusted and tuned according to the top-down coordinate system.

page_cords
Defines the coordinate system. Allowed values are TOP_DOWN and BOTTOM_UP.

FastPDF Formatter default page coordinate system is top-down and the font coordinate origin is a bottom-left corner, all FastPDF Formatter functions are adjusted and tuned according to its default systems. Font origin can be changed by the FontOrigin() function.

Top-down coordinates are then converted to bottom-up coordinates by FastPDF Formatter internally, as the actual PDF coordinate system defined by PDF specifications is bottom-up.

Sample

None.
## Set Default Unit

### Function

Sets default measurement unit, it can be called before calling of OpenDoc() or OpenPage() function, or at any time if you want to change the default measurement unit.

FastPDF Formatter default is POINT if this function is not called.

### Syntax

```c
void SetUnit(
    Units    makepdf_unit
);
```

### Parameters

`makepdf_unit`

Valid values are:

- CM: Centimeter
- MM: Millimeter
- INCH: Inch
- POINT: Point

### Sample

```c
void main( )
{
    Start();
    SetUnit(INCH);  // Set default units to INCH
    OpenDoc();
    OpenPage(8.5,11);  // LETTER page size, 8.5" x 11"
    ::
    :
    ::
    CloseDoc();
}
```
Set Attributes (Position, Scale, and Orientation) to Input PDF Pages

Functions

Sets attributes offset position, scale, and orientation to the subsequence input PDF pages.

Syntax

```c
Void SetInPage(
    float  x_offset = 0,
    float  y_offset = 0,
    float  x_scale_ratio = 100,
    float  y_scale_ratio = 100,
    SI16   orientation = 0
);
```

Parameters

- **x_offset, y_offset**
  Defines the (x, y) offset position to place the subsequence input PDF pages.

- **x_scale_ratio, y_scale_ratio**
  Defines the scaling ratios in percent to the subsequence input PDF pages. These parameters are disabled if your input PDF page orientation is not 0 degree.

- **orientation**
  Defines the orientation for the subsequence input PDF pages. Supported values are 0, 90, 180 and 270, rotates the subsequence input PDF pages clockwise.

Sample

```c
int main()
{
    Start();                 // Start session
    OpenDoc();               // Open a new PDF document
    SetInPage(0.2, -0.1, 99.5, 100, 0);
    do {
        GetPage();             // Read-in a page from input PDF file
        ClosePage();           // Close PDF page
    } while ($Edt == 0);     // reprocess input PDF until end of PDF file
    CloseDoc();       // End of PDF document, close output PDF file
    #ifdef _DEBUG
    ViewPDF();        // View output PDF file only in debug mode
    #endif
    return 0;
}
```
Shaded Graphical Text

Functions

Plots the shaded graphical texts.

Syntax

void ShadeTxt(
    char * text,
    float  x,
    float  y,
    float  shading_center,
    UI32   start_color,
    UI32   end_color,
    bool   hShading = true,
    bool   stroke = false
);

void ShadeU8(
    UChar8 * utf9_text,
    float  x_pos,
    float  y_pos,
    float  shading_center,
    UI32   start_color,
    UI32   end_color,
    bool   hShading = true,
    bool   stroking = false
);

void ShadeU16(
    UChar * utf16_text,
    float  x_pos,
    float  y_pos,
    float  shading_center,
    UI32   start_color,
    UI32   end_color,
    bool   hShading = true,
    bool   stroke = false
);

Parameters

text
The text string to be shaded.

utf8_data
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_data
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

x_pos, y_pos
The (X, Y) coordinate of the text.
**start_color, end_color**
The starting and ending colors.

**shade_center**
Defines the point from where the first color will blend into the other. A value of 1 specifies the exact center between the starting and ending point of the shading. Smaller values shift the shading center in direction to the start point, greater values in the direction to the endpoint.

**hShading**
Specifies the shading direction, TRUE specifies shading is changing horizontally, FALSE specifies shading is changing vertically.

**stroking**
Specifies whether the text is being stroked, default is FALSE.

**Sample**
```
SetUnit(MM);
OpenDoc();
OpenPage();

Font("Times New Roman", 42, BOLD);
ShadeTxt("Graphic Text", 30, 220, 1, BLUE, RED);

ClosePage();
CloseDoc();
```

**Output:**

Graphic Text
Shaded Graphical Drawing

Functions

Plots the graphical shading.

Axial shadings define a color blend that varies along a linear axis between two endpoints and extends indefinitely perpendicular to that axis.

Radial shadings define a color blend that varies between two circles.

Syntax

Axial Shading:

```cpp
void GShade(
    float  x_pos,
    float  y_pos,
    float  width,
    float  height,
    float  shade_center,
    UI32   start_color,
    UI32   end_color,
    bool   hShading
);
```

Radial Shading:

```cpp
void GRshade(
    float  x_inner,
    float  y_inner,
    float  r_inner,
    float  x_outer,
    float  y_outer,
    float  r_outer,
    float  shade_center,
    UI32   start_color,
    UI32   end_color
);
```

Parameters

- **x_pos, y_pos**
The (X, Y) coordinate of the top-left starting point of axial shading.

- **Width, height**
The width and height of the axial shading.

- **start_color, end_color**
The starting and ending colors.

- **shade_center**
Defines the point from where the first color will blend into the other. A value of 1 specifies the exact center between the starting and ending point of the shading. Smaller values shift the shading center in direction to the start point, greater values in the direction to the endpoint.

- **hShading**
Specifies the shading direction, TRUE specifies shading is changing horizontally, FALSE specifies shading is changing vertically.
**x_inner, y_inner, r_inner**  
Specifies the (X, Y) coordinate of the inner circle and its radius of radial shading.

**x_outer, y_outer, r_outer**  
Specifies the (X, Y) coordinate of the outer circle and its radius of radial shading.

**Sample**

```plaintext
SetUnit(PT);
OpenDoc();
OpenPage();
:
GShade(75, 420, 445, 50, 1.5, WHITE, RED, true);
GShade(75, 490, 445, 50, 1.5, WHITE, BLUE, false);
GRshade(300, 465, 5, 285, 480, 55, 1, GREEN, WHITE);
:
ClosePage();
CloseDoc();
```

**Output:**

![Image of radial shading sample]
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 LPI() or LineSpace() function.

Syntax

```c
void Skip(
    float lines
);
```

Parameters

- `lines`
  The number of lines to skip.

Sample

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

LineSpace(0.25); // Line spacing is 0.25", 4 LPI
Margin(0.8);    // left margin for the text is 0.8"
Skip(10.5);     // Skip 10.5 lines, baseline increment is
                // 0.25" x 11.5 = 2.625"

ClosePage();
CloseDoc();
```
Skew Line

Function

Plots a graphic skew line.

The thickness and type of line can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GSline(
    float from_x,
    float from_y,
    float to_x,
    float to_y,
);
```

Parameters

- **from_x, from_y**
The (X, Y) coordinate of the line starting point.

- **to_x, to_y**
The (X, Y) coordinate of the line ending point.

Sample

```c
SetUnit(MM);

OpenDoc();

OpenPage(DIN_A4);


GLLineWidth(0.5);

GSline(10, 10, 80, 80);

ClosePage();

CloseDoc();
```
Start Session

Function

Starts a FastPDF Formatter session before calling any of the other FastPDF Formatter functions.

This function starts and establishes initiation of a FastPDF Formatter session, opens a MakePDF definition file if any, parses the parameters defined in FastPDF Formatter definition file, and command-line flag arguments.

Syntax

\[
\begin{align*}
\text{char* Start(} & \text{char* command\_line\_arguments = NULL); } \\
\text{char* StartW(} & \text{UChar* command\_line\_arguments\_utf16le = NULL); }
\end{align*}
\]

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 a command.

Refer to Chapter 2. Running Your FastPDF Formatter Program, FastPDF Formatter Users' Guide, for more details about command-line flag-arguments supported by FastPDF Formatter.

Sample

```c
void main( )
{
  Start(); // Start initiation, open default input, 
  // output and optional definition files
  :
  :
  :
  :
}
```
Text Mode and Baseline Changes

Functions

Make changes to the text drawing.

Syntax

```cpp
void TextMode(
    TextDrawMode drawing_mode = TEXT_REGULAR
);
void TextRise(
    float rise_value
);
```

Parameters

drawing_mode

Specifies how text should be rendered as the vector graphics. Valid values are:

- **TEXT_REGULAR**: Render the text in the normal method. The fill color is used as the text’s color.
- **TEXT_STROKE**: Tread the glyph outlines as the path to be stroked. The current line width and stroke color are used to paint the path.
- **TEXT_FILL_STROKE**: Tread the glyph outlines as the path to be stroked and filled. The current line width, fill color, and stroke color are used to paint the path.
- **TEXT_INVISIBLE**: Neither fill nor stroke the text, the text is invisible.
- **TEXT_FILL_CLIP**: Tread the glyph outlines as the path to be filled and add this path to the current clipping path. The settings of the current line width and fill color are used to paint the path.
- **TEXT_STROKE_CLIP**: Tread the glyph outlines as the path to be stroked and add this path to the current clipping path. The settings of the current line width and stroke color are used to paint the path.
- **TEXT_FILL_STROKE_CLIP**: Tread the glyph outlines as the path to be stroked and filled and add this path to the current clipping path. The settings of the current line width, fill color, and stroke color are used to paint the path.
- **TEXT_CLIPPING**: Tread the glyph outlines as the path and add this path to the current clipping path.

rise_value

Specifies a distance to move the baseline up or down from its default location. Positive values of text rise move the baseline up; negative values of text rise move the baseline down. The default location of the baseline can be restored by setting the text rise to 0.

Adjustments to the baseline are useful for drawing superscripts or subscripts.

Sample

```cpp
SetUnit(MM);
OpenDoc();
OpenPage();
```
Font("Arial", 26);

TextMode(TEXT_STROKE);

StrokeColor(BLUE);

ParTxt(text2, 35, 20, 65, -1, LEFT, true, 10);

Color(GREEN);
StrokeColor(RED);

TextMode(TEXT_FILL_STROKE);

ParTxt(text2, 120, 20, 65, -1, RIGHT, true, -10);

ClosePage();
CloseDoc();

Output:

The Portable Document Format (PDF), undeniably one of the most commonly used formats for electronic documents. The Portable Document Format (PDF), undeniably one of the most commonly used formats for electronic documents.
Text Replace or Delete

**Function**

Replaces or deletes a text string on the currently opened PDF page.

**Syntax**

```c
void TextReplace(
    char*     dst_text,
    char*     src_text,
    float     x_bottomLeft,
    float     y_bottomLeft,
    float     width,
    float     height,
);

void TextReplaceU16(
    UChar*    dsttext,
    UChar*     src_text,
    float     x_bottomLeft,
    float     y_bottomLeft,
    float     width,
    float     height,
);
```

**Parameters**

- **dst_text**
  The new text string. Deletes the text if the new text string is empty.

- **src_text**
  The text string to be replaced or deleted.

- **x_bottomLeft**
  The X position of the bottom-left corner of the area, in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **y_bottomLeft**
  The Y position of the bottom-left corner of the area, in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **width**
  The width of area, in points.

- **height**
  The height of the area in points

**Sample**

None.
Text Width

Function

Measures and then returns the width of the text string with the currently active font. The return value is in the default unit you defined with the SetUnit() function.

Syntax

```c
void TextWidth(
    char * text_data,
);
void U8Width(
    UChar8 * utf8_data,
);
void U16Width(
    UChar * utf16le_data,
);
```

Parameters

text
The NULL-terminated text string. MakePDF converts legacy codepage text to Unicode little-endian if cpUnicode encoding is specified with current active font, make sure encoding name is defined properly with DefaultCode() first.

utf8_text
The NULL-terminated UTF-8 string. Your current active font must be defined with cpUnicode encoding.

utf16_text
The NULL-terminated UTF16 little-endian string. Your current active font must be defined with cpUnicode encoding.

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage(210,297);
:
Font("Arial", 10, BOLD);
:
float w = TextWidth("Get a text string width");
:
/* UTF-16LE string, "test测试" */
UChar  utf16data[] = {0x0074, 0x0065, 0x0073, 0x0074, 0x6d4b, 0x8bd5};
:
Font("SimSun", 10, REGULAR, cpUnicode);
W = TextWidth(utf16data);
ClosePage();
CloseDoc();
```
Trigger by a Location and a Pattern

Function

Defines a location and a string or a pattern of symbols to uniquely identify a specific page, or to be used with the “Get Field” function to identify a text string to be captured from the currently opened PDF page. It returns a TRUE bool if the trigger is found.

Syntax

```c
bool Trigger(
    float x,
    float y,
    char* mask
);

bool TriggerU16(
    float x,
    float y,
    UChar* u16_mask
);
```

Parameters

- **x**
  Specifies the X bottom-up position of the data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **y**
  Specifies the Y bottom-up position of the data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

- **mask, u16_mask**
  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 mask 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 the text string.

Sample

```c
int main()
{
    UI32 i, j, inPageNo = 0, outPageNo = 0, groupPages = 0;
    bool firstGroup = 1;
    char str[80], str2[80], acNo[10], acNo_keep[10];
    Start(); // Start session
```
DefaultCode("ibm-437"); // Sets default native encoding to USA ASCII
SetUnit(INCH); // Set default units to inch
OpenDoc(); // Open a new PDF document
do {
    GetPage(); // Read-in a page from input PDF file
    inPageNo++;
    if (Trigger(524.90, 509.04, "Page 1 of")) // if it first page of page
        GetField(43.20, 532.80, str);
    strcpy(acNo, &str[18]); // Gets Saving Account No
    if (!firstGroup) // Now at first page of new page-group
        ClosePage(); // Need to close current page first
        outPageNo++;
    // Process pagination, page serial numbers and barcode of previous
    // page group
    j = inPageNo - groupPages;
    for (i = outPageNo-groupPages; i < outPageNo; i++)
        { 
            EditPage(i);
            // Replace page serial number as the bank pages inserted
            sprintf(str, "%07d", j++);
            sprintf(str2, "%07d", i);
            TextReplace(str2, str, 7.20, 10.80, 72, 10);
            BarCode(CODE128, acNo_keep, 0.22, 1, 1.5, 0.2, 90);
            ClosePage();
        }
    if (groupPages % 2 != 0) // If the pages of page group is an odd
        // number, then insert a blank page
        { // for the production printing
            MovePage(++outPageNo, outPageNo);
        }
    groupPages = 1; // reset for new page group
    strcpy(acNo_keep, acNo); // Keeps new account number for
    // generating of dynamic barcode
} else // It is first page group of input PDF
    { 
        firstGroup = 0;
        strcpy(acNo_keep, acNo); // Keeps current account number for
        // generating of dynamic barcode
        ClosePage();
        outPageNo++;
        groupPages++;
    }
else
    { 
        ClosePage();
        outPageNo++;
        groupPages++;
    }
} while ($Edt == 0); // reprocess input PDF until end of PDF file
// Process pagination, page serial numbers and barcode of last page group
j = inPageNo - groupPages + 1;

for (i = outPageNo-groupPages+1; i <= outPageNo; i++)
{
    EditPage(i);

    // Replace page serial number as the bank pages inserted
    sprintf(str, "%07d", j++);
    sprintf(str2, "%07d", i);
    TextReplace(str2, str, 7.20, 10.80, 72, 10);
    BarCode(CODE128, acNo_keep, 0.152, 1, 1.5, 0.15, 90);
    ClosePage();
}

CloseDoc();       // End of PDF document, close input & output PDF files

#ifdef _DEBUG
    ViewPDF();        // View output PDF file only in debug mode
#endif

return 0;
Trigger by a Location Ranger and a Pattern

Function

Defines a location range and a string or a pattern of symbols to uniquely identify a specific page, or to be used with the “Get Field” function to identify a text string to be captured from the currently opened PDF page. It returns a TRUE bool if the trigger is found.

Syntax

```c
bool Trigger2(
    ushort    x1,
    ushort    x2,
    ushort    y1,
    ushort    y2,
    char*     mask
);  
```

```c
bool Trigger2U16(
    ushort    x1,
    ushort    x2,
    ushort    y1,
    ushort    y2,
    UChar*    u16_mask
);  
```

Parameters

**x1, x2**
Specifies the X bottom-up position range of data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

**y1, y2**
Specifies the Y bottom-up position range of data field in points. With MPDFTEXT utility, you can dump out the data fields and their bottom-up coordinate locations.

**mask, u16_mask**
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 mask 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.
Transparency of Color Image

Functions

Specify whether images should get a transparent background. If true, the transparent
the color defined by the TransparentColor() function is used as the color background to mask the
image.

Syntax

void Transparency(
    bool   use_transparency
);  

void TransparentColor(
    UI32    transparent_color
);  

Parameters

use_transparency
Specifies whether images should get a transparent background. The default value of FastPDF
Formatter is TRUE.

transparent_color
Specifies a transparent color that is to be used for the images. The default value of FastPDF
Formatter is WHITE.

Sample

None.
The version of PDF Format

Function

Specifies the PDF format version. THE Default PDF output format version is PDF 1.5.

Syntax

```c
void Version(
    PDFVersion version
);
```

Parameters

version
The version of the PDF format, valid values are:

- PDF_1.0
- PDF_1.1
- PDF_1.2
- PDF_1.3
- PDF_1.4
- PDF_1.5
- PDF_1.6
- PDF_1.7
- PDF_1.8
- PDFX1a_2001
- PDFX1a_2003
- PDFX3_2002
- PDFX3_2003
- PDFA_2005

Sample

None.
Vertical Boxes

Function

Plots multiple graphic boxes vertically.

The thickness and type of line can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GVboxes(
    float    x_pos,
    float    y_pos,
    float    width,
    float    height,
    float    thickness,
    uint   repeat,
    float    gap_space,
    PathFill_mode  fill_mode = STROKE_PATH;
    float    radius = 0
);
```

Parameters

- **x_pos**
  The X position of the top left corner of the first box, specify CP if you want to use the current position.

- **y_pos**
  The Y position of the top left corner of the first box, specify CP if you want to use the current position.

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

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

- **thickness**
  The thickness of the box border.

- **repeat**
  The number of boxes to be repeated if you want to draw more than one box of the same size and border thickness.

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

- **fill_mode**
  In the graphic filling mode, you may use the following values:
  - **FILL_PATH**: Fill color into the box does not stroke box boundary
  - **FILL_STROKE**: Fill color into the box and stroke box boundary
  - **STROKE_PATH**: Stroke box boundary does not fill color into box

- **radius**
  The radius of rounding of box corners.

Sample

```c
SetUnit(MM);
OpenDoc();
```
OpenPage();
:
StrokeColor(BLACK);

Vboxes(10,10,20,5,9,2);  // plots 10 boxes vertically from (10,10)mm, box width is 20mm, height is 5mm, 2mm space between boxes:

ClosePage();
CloseDoc();
Vertical Line

Function

Draws a vertical graphic line.

The thickness and type of line can be defined by the GLineWidth() and GLineType() functions.

Syntax

```c
void GVline( 
    float   x_pos, 
    float     y_pos, 
    float   length 
);
```

Parameters

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

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

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

Sample

```c
SetUnit(MM);
OpenDoc();
OpenPage();

: 
:  

GVline(10,10,100); // draw a vertical blue line from 
:  // (10,10)mm, its length is 100 mm, 
:  

ClosePage();
CloseDoc();
```
Vertical Lines

**Function**

Repeats drawing vertical lines.

The thickness and type of line can be defined by the GLineWidth() and GLineType() functions.

**Syntax**

```c
void GVlines(
    float    x_pos,
    float    y_pos,
    float    length,
    UI16   repeat,
    float    space,
    RepeatDirection  direction = ACROSS
);
```

**Parameters**

- **x_pos**
The X starting position of the line.
- **y_pos**
The Y starting position of the line.
- **length**
The length of the line.
- **repeat**
The number of additional lines to be repeated.
- **space**
The gap space between the lines.
- **direction**
The direction of line repeating, valid values are ACROSS and DOWN, default is ACROSS.

**Sample**

```c
SetUnit(MM);
OpenDoc();
OpenPage();
GVlines(10,10,80,9,5);  // plot 10 vertical lines from (10,10)
                        // length is 80 mm, 5 mm space between lines
ClosePage();
CloseDoc();
```
View PDF File

Function

Views the generated PDF file, it must be specified after the CloseDoc() function.

You need a PDF reader installed on your Windows.

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 MakePDF definition file, and then define “Program Arguments” in syntax of -d definition_file -i input_file -o output_PDF_file.

Syntax

    void ViewPDF();

Parameters

None.

Sample

    Start();
    SetUnit(INCH);
    OpenDoc();
    
    CloseDoc();  // ViewPDF() must be called after PDF document
                 // file is closed by the CloseDoc() function

#ifdef DEBUG
    ViewPDF();  // Only view PDF in the debug mode
#endif
Web Link

Functions

Add a web hypertext link.

Syntax

```c
void WebLink(
    char * website_linkage,
    float x_pos,
    float y_pos,
    float width,
    float height
);
```

```c
bool WebLinkHiliteMode(
    HiliteMode hilite_mode
);
```

Parameters

**website_linkage**
Defined a web page linkage/location in ASCII 7-bit string.

**x_pos, y_pos**
The (X, Y) coordinate of the top-left corner of the bounding rectangle.

**width, height**
The width and height of the bounding rectangle.

**hilite_mode**
The highlight mode is used by the link annotation visual effect while moving the mouse over it. Valid highlight mode is:

- **HILITE_NONE** No highlight effect
- **HILITE_INVERT** Invert the contents of the bounding rectangle
- **HILITE_OUTLINE** Invert the bounding border
- **HILITE_PUSH** Simulate a push button effect
- **HILITE_PUSHUP** Update the appearance stream on changes

Sample

None.
Word Space

Function

Sets the word spacing.

Syntax

    void WordSpace(
        float  word_spacing = 0
    );

Parameters

word_spacing
    The amount of space between words. Default value 0 indicates that the word spacing is
determined by the font default.

Sample

    SetUnit(MM);
    OpenDoc();
    OpenPage(210,297);
    :
    :
    WordSpace(3);  // Word space is 3 mm
    Ctxt("Extra Intercharacter spacing");
    WordSpace();   // reset word space back to font
                    // default
    :
    :
    ClosePage();
    CloseDoc();
**X Absolute Position of Text**

**Function**

Sets the new horizontal absolute X position for the output text on the page. The origin position on the page is at (0, 0) top-left corner.

**Syntax**

```c
void Xpos(
    float x_position
);
```

**Parameters**

- **x_position**
  
The value of the absolute horizontal position from the page origin. Negative values are invalid.

**Sample**

```c
SetUnit(MM);
OpenDoc();
OpenPage(210,297);
:
:
Xpos(5);                // Set text X position to 5 mm
:
:
ClosePage();
CloseDoc();
```
X Current Position of Text (Query)

Function
Queries the current text horizontal X position on the page returns a value is in the current measurement unit.

Syntax
```c
float GetXpos( );
```

Sample
```c
SetUnit(INCH);
OpenDoc();
OpenPage(8.27, 11.67);
if (GetXpos() > 5.5 ) { // if current X position is more than // 5.5"
    :
} else
{
    :
}
ClosePage();
CloseDoc();
```
X Relative Move Position of Text

Function

Moves the text horizontal X position relative to the current text horizontal coordinate position.

Syntax

void Xmove(
    float  x_move
);

Parameters

x_move
The value of the text's horizontal movement relative to the text's current horizontal position (X). A positive value moves the text position to the right; a negative value moves the text position to the left.

Sample

SetUnit(MM);
OpenDoc();
OpenPage(210,297);
    :
    :
Xmove(25);  // Move text 25 mm to the right
    :
    :
Xmove(-10);  // Move text 10 mm to the left
    :
ClosePage();
CloseDoc();
Y Absolute Position of Text

**Function**

Sets the new vertical absolute Y position for the output text on the page. The origin position on the page is at (0, 0) top-left corner.

**Syntax**

```c
void Ypos(
    float y_position
);
```

**Parameters**

* y_position
  The value of the text vertical position absolute from the page origin. Negative values are invalid.

**Sample**

```c
SetUnit(MM);
OpenDoc();
OpenPage(210,297);
    :
    :
Ypos(15);  // Set text Y position to 15 mm
    :
    :
ClosePage();
CloseDoc();
```
Function

Queries the current text vertical Y position on the page, returns a value in the current measurement unit.

Syntax

float GetYpos();

Sample

SetUnit(INCH);
OpenDoc();
OpenPage(8.27, 11.67);
:
:
if ( GetYpos() > 11.2 )  // if current Y position is more than
// 11.2", may need to do a page-breaking
{
    :
    :
}
else
{
    :
    :
}
ClosePage();
CloseDoc();
Y Relative Move Position of Text

**Function**

Moves the text vertical Y position relative to the current text vertical coordinate position.

**Syntax**

```c
void Ymove(
    float    y_move
);
```

**Parameters**

- **y_move**
  
The value of the text vertical movement relative to the current text vertical position (Y). Positive values move the position down; negative values move the position up.

**Sample**

```c
SetUnit(MM);
OpenDoc();
OpenPage(210,297);
:
:
Ymove(25);             // Move text 25 mm down
:
:
Ymove(-10);           // Move text 10 mm up
:
ClosePage();
CloseDoc();
```
Chapter 2. FastPDF Formatter Parameters

This chapter describes the FastPDF Formatter parameters to be defined in the MakePDF definition file, including the syntax rules and values.

FONTLIB – Specifies the Library Path of Fonts

Function

Specifies a path where your own OpenType/TrueType/PostScript fonts are stored.

Syntax

FONTLIB = path_name

Parameter

Path_name
Any valid Windows fully-qualified pathname specifies an additional font path to the Windows system font path, which is always be searched by FastPDF Formatter.

Alternatively, you can specify a font pathname with your command-line flag parameter “-f font_path”. If both are specified, the value specified with the command-line overrides the value specified in the MakePDF definition file.

Sample

fontlib=e:\pdf_jobs\fonts

OPENPW – Specifies the PDF Open Password

Function

Specifies a password of up to 32 characters to protect your PDF file, user is required to enter this password to open the protected confidential PDF document.

Syntax

OPENPW = password

Parameter

password
Specifies a password string up to 32 characters.

Sample

openpw=tiger168@#
OWNERPW – Specifies the PDF Open Password

Function

Specifies the PDF owner password up to 32 characters and the permissions to the PDF document.

Syntax


Parameter

password
Specifies a PDF owner password string up to 32 characters.

noaccessible
Prevents extracting of text or graphics for accessibility purposes, such as a screen-reader program.

noassemble
Prevents inserting, deleting, modifying, rotating pages, and creating bookmarks and thumbnails.

copy
Prevents copying and extracting text or graphics from PDF documents.

nomodify
Prevents any change to PDF document.

noprint
Prevents printing of PDF document.

Sample

ownerpw=elephant!@#,nocopy,nomodify

RESLIB – Specifies the Library Path of Images and PDF Templates

Function

Specifies a path where your images and PDF templates are stored.

Syntax

RESLIB = path_name

 Parameter

Path_name
Any valid Windows fully-qualified pathname, where your images and PDF templates are stored.

Alternatively, you can specify it with your command-line flag parameter “-r res_path”. If both are specified, the value specified with the command-line overrides the value specified in the MakePDF definition file.

Sample

reslib = e:\pdf_jobs\abc_bank\resources
Chapter 3. FastPDF Formatter Variables

In addition to the FastPDF Formatter functions described in Chapter 1, there are a few variables are maintained by FastPDF Formatter for internal use or exchanging of information during the data formatting. The FastPDF Formatter variables described below are accessible from your program.

$Edt – End of PDF Document
Indicates whether the “End of PDF Document” has been detected after the “Get Page” function is called for a reading of a PDF page from the input PDF file. The “Get Page” function reads a PDF page from PDF input file and also sets the $Edt variable TRUE if the “End of Document” boundary is detected.

$FileInName – Filename of the Default Input Data File
Contains the filename of the default input data file specified by the flag parameter “-i in_data_file” entered with your command in batch mode.

This variable gives you a choice to open and access the input data file by your programming completely.

$PdfOutFile – Filename of the Output PDF File
Contains the filename of PDF output file specified by the flag parameter “-o pdf_filename.pdf” entered with your command in batch mode.

This variable gives you a choice to access the PDF output file once it is created after CloseDoc() function is called.

$PdfOutPath – Path Name of the PDF Output
Contains the name of PDF output path specified by the flag parameter “-o pdf_output_path” entered with your command in batch mode.
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 MakePDF’s complementary functions specially developed for data formatting requirements to assist your MakePDF 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 FastPDF Formatter functions for string manipulation are listed in alphabetic order. The description of Each function includes the following sections:

**Function**  
A description of the major purpose of the function.

**Syntax**  
A diagram showing the function parameters.

**Parameters**  
Explanation of each parameter.

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

**Default Values**  
When calling these functions, every parameter must be specified in the order shown in this chapter. MakePDF 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,  
    uint fraction_digits  
);  
```
Parameters

**float_value**
Source float value.

**fraction_digits**
The number of significant fractional digits.

Sample

double 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**

```c
char *CommaInt(_int64 integer_value);
```

**Parameters**

**integer_value**
Source 64-bit integer value.

**fraction_digits**
The number of significant fractional digits.

Sample

```c
_int64 total = 1298945698123;
printf("Total Amount: %s", CommaInt(total));
Output: Total Amount: 1,298,945,698,123
```

---

**Comma Digital String**

**Function**

Formats a digital string using commas as the thousandth separators and a specified number of significant fractional digits, removing the leading zeros.

**Syntax**

```c
char *CommaDigit(str* digital_string, uint fraction_digits);
char *CommaDigit2(str* digital_string);
```

**Parameters**

**digital_string**
Source digital data string.
fraction_digits
The number of significant fractional digits.

Sample

```c
char data1[20] = "001298945698123";
printf("Total Amount: %s", CommaDigit(data1,2));
Output: Total Amount: 12,989,456,981.23

char data2[10] = "12989.49";
printf("Total Amount: %s", CommaDigit(data2);
Output: Total Amount: 12,989.49
```

---

Delete Characters

**Function**

Deletes a range of characters from the string.

**Syntax**

```c
char *Delete(
    char*  string,
    uint  start_col,
    uint  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**

```c
Extract once by a delimiter:
char *Extract1(
    char*  srcStr,
    uint  order_pos,
    char*  delimiter
);
```
Extract multiple time by a delimiter:

```c
void Extract(
    char* dstStr_array[],
    char* srcStr,
    char delimiter,
    char qualifier
);
```

```c
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 qualifier character.

**Sample**

```c
char src1[256] = "substring1;substring2;substring3;substring4";
char src2[256] = "field1:,123,456,000.00:,field3:,12,341.00:,field5";
char src3[256] = "test1,'168,456,000.00',test3,'88,666.00'";
char *dst[20];

printf("Extracted 3rd substring is: %s, Extract1(src1, 3,";"));
Extract2(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
```c
int Find1(
    char* str,
    char* search,
    int  start_pos
)
```
```c
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
```c
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
```c
int FirstChar(
    char* string
);
```

Parameters
- **string**: Source data string.
Sample

cchar 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

cchar *payment [] [2] = { {"001", "Cash Payment"},
{"005", "Master Payment"},
{"003", "Visa Payment"},
{"007", "Check Payment"},
{"011", "GORO payment"},
{"\0", "\0"} };  // End of table

int intab = InSubst(payment, "002");
intab = InSubst(payment, "007");
Return:
     0
     1

Insert String

Function
Inserts a character string after the specified position.

Syntax
char *Insert(
    char*  srcStr,
    uint   pos,
    uint   insertStr
);

Parameters

string
Source data string.
**pos**
Character position where you insert a string after.

**insertStr**
The string to be inserted in the destination string.

**Sample**
char src[256] = "This is data string";
printf("%s", Insert(src, 8, "inserted ");
Output:
   This is inserted data string

---

**Is Empty**

**Function**
Checks if a string is either of 0-byte length or contains white-space characters only. Returns 1 if it is empty otherwise returns 0.

**Syntax**
int IsEmpty(
    char*    string
);

**Parameters**
- **string**
data string.

**Sample**
char str1[80] = "This is data string";
char str2[80] = "        ";
char str3[80] = NULL;
int empty = IsEmpty(str1);
empty = IsEmpty(str2);
empty = IsEmpty(str3);
Return:
   0
   1
   1

---

**Last Character**

**Function**
Returns the position of the last non-white-space character.

**Syntax**
int LastChar(
    char*    string
);

**Parameters**
- **string**
Source data string.

**Sample**
char str[80] = "This is data string    ";
```
int pos = LastChar(str);
Return:
19
```

**Left Copy**

**Function**

Copies characters from the left of the source string to destination string with null-terminated.

**Syntax**

```c
char *Lcp(
    char* dstStr,
    char* srcStr,
    uint 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 the source string to destination string which may be padded with the pad character if the length of the source string is less than the specified length.

**Syntax**

```c
char *LcpPad(
    char* dstStr,
    char* srcStr,
    uint length,
    char pad
);
```

**Parameters**

- `dstStr`
  Destination string.
- `srcStr`
  Source string.
- `length`
  The length of the destination string.
pad
The character to be used to pad destination string.

Sample

```
char src[] = "This is padded string";
char dst[26];
printf("Padded String: %s", LcpPad(dst, src, 25, '.");
Output:
    Padded String: This is padded string....
```

---

**Left Copy and Right Trim**

**Function**

Copies characters from the left of the source string to the destination string. The white-space, carriage return, newline control codes on the right side of the destination will be trimmed before being terminated with NULL.

**Syntax**

```
char *LcpRtrim(
    char* dstStr,
    char* srcStr,
    uint  length
);
```

**Parameters**

- dstStr
  Destination string.
- srcStr
  Source string.
- length
  Number of characters to be copied.

**Sample**

```
char src[] = "This is a string testing          The second string";
char dst[100];
printf("'Left Copied: %s'", LcpRtrim(dst, src, 28));
Output:
    'Left Copied: This is a string testing'
```

---

**Left Trim**

**Function**

Trims white-space characters from the left side of the source string.

**Syntax**

```
char *Ltrim(
    char* string
);
```

**Parameters**

- string
  Data string to be left trim.

**Sample**

```
```
char str[] = "      This is a string testing";
printf("Left Trimmed: %s", Ltrim(str));

Output:

Left Trimmed: This is a string testing

---

**Left Trim for EBCDIC**

**Function**

Trims EBCDIC white-space characters from the left side of the source EBCDIC string.

**Syntax**

```c
char *E_Ltrim(
    char* string
);
```

**Parameters**

- `string`
  EBCDIC data string to be left trim.

**Sample**

Refer to the sample for the Ltrim() function.

---

**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**

- `separator`
  characters to be inserted between concatenated strings.

- `string,...`
  variable-argument lists of multiple strings.

**Sample**

```c
char str1[] = "      This is a string testing.";
char str2[] = "    The second string.";
printf("After Concatenation: %s", LtrimCat(" ", str1, str2));
```

Output:

After Concatenation: This is 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**

```c
char str1[] = "This is data string";
char str2[] = "TX 20890";
char str3[] = "Answer?";

int rc = Match(str1, "*.*", 0);
rc = Match(str2, "[A-Z][A-Z] [0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]", 0);
rc = Match(str3, "*\?", 0);
```

Return:  
0 1 1

Pattern Searching

**Function**

Searches a string for a set of characters that match a specified pattern, returns the characters if it is finding a match.

**Syntax**

```
Char *Pattern(
    char* string,
    ...
);
```
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

cchar 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 string.

Syntax

Remove once:
char *Remove1(
    char* srcStr,
    char* rmStr
);

Remove all:
char *Remove(
    char* srcStr,
    char* rmStr
);
Parameters

srcStr
Source string.

rmStr
The string to be removed from the source string.

Sample

char str[] = "This is a string testing string."
printf("After Removed once: %s\n", Remove1(str," string"));
printf("After Removed all: %s\n", Remove(str," string"));
Output: After Removed once: This is a testing string.
After Removed all: This is 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
A new string to be used to replace the target string.

Sample

char str[] = "This is a string testing"
printf("After Replaced: %s", Replace(str,"string", "replaced")
Output: After Replaced: This is a replaced testing
Reverse Find String

**Function**

Checks whether a substring is in the data string reversely (from right to left), returns its position if found, otherwise, return 0.

**Syntax**

```c
int Rfind(  
    char* str,  
    char* search  
)
```

Or

```c
int Rfind1(  
    char* str,  
    char* search,  
    int start_pos  
)
```

Or

```c
int Rfind2(  
    char* str,  
    char* search,  
    int start_pos,  
    int stop_pos  
)
```

**Parameters**

- **str**  
  Source data string.
- **search**  
  Search string.
- **start_pos**  
  The position to start the search.
- **stop_pos**  
  The position to stop the search.

**Sample**

```c
char str[80] = "This is data string search testing";
int pos = Rfind(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,  
    int pos  
)
```
char* srcStr,
    uint from_pos)
);

Parameters

    dstStr
    Destination string.

    srcStr
    Source string.

    from_pos
    The starting position to be copied from.

Sample

char str[] = "This is a string testing";
char dst[20];
printf("Right Copied: %s", Rcp(dst, str, 11));
Output:

    Right Copied: string testing

---

Right Copy and Left Trim

Function

Copies characters from the specified position until the end of the source string to the destination string where the white-space characters will be trimmed from the left side before being terminated with NULL.

Syntax

    char *RcpLtrim(
        char* dstStr,
        char* srcStr,
        uint from_pos
    );

Parameters

    dstStr
    Destination string.

    srcStr
    Source string.

    from_pos
    The starting position to be copied from.

Sample

    char str[] = "This is a string testing";
    char dst[20];
    printf("Result: %s", RcpLtrim(dst, str, 11));
Output:

    Result: string testing
Right Copy and Pad

Function
Copies the most right characters of a specified length from the source string, and pad to the left of the destination string with the pad character if appropriate.

Syntax
```c
char *RcpPad(
    char* dstStr,
    char* srcStr,
    uint length,
    char pad
);
```

Parameters
- **dstStr**: Destination string.
- **srcStr**: Source string.
- **length**: The length of the most right characters to be copied from the source string.
- **pad**: The character to be padded to the left of the destination string if the length of the source string is less than the specified length.

Sample
```c
char str[] = "The string testing";
char dst[25];
printf("Result: %s", RcpPad(dst, str, 25, '.'));
```
Output:
```
Result:........The string testing
```

Right Copy and Right Trim

Function
Copies characters from the specified position until the end of the source string to the destination string where the white-space, carriage return, newline control codes would be trimmed from the right before being terminated with NULL.

Syntax
```c
char *RcpRtrim(
    char* dstStr,
    char* srcStr,
    uint from_pos
);
```

Parameters
- **dstStr**: Destination string.
**Right Trim**

**Function**

Trims white-space, carriage return, newline 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, newline 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 strings.

Sample

char str1[] = "This is a string testing. ";
char str2[] = "The second string. ";
printf("After Concatenation: %s", RtrimCat(" ", str1, str2));
Output:
     After Concatenation: This is string testing. The second string.

Right Trim for EBCDIC

Function

Trims EBCDIC white-space, carriage return, newline control codes, from the right side of the source EBCDIC string.

Syntax

char *E_Rtrim(
    char*    string
);

Parameters

string
Data string to be right trimmed.

Sample

Refer to the sample for the Rtrim function.

Strings Concatenate

Function

Concatenates multiple strings into a string.

Syntax

char *Strcat(
    char*    string,...
);

Parameters

string,...
Variable-argument lists of multiple strings.

Sample

char str1[] = "This is a string testing. ";
char str2[] = "The second string. ";
printf("After Concatenation: %s", Strcat(str1, str2));
Output:

After Concatenation: This is string testing. The second string.

---

**String Pad**

**Function**

Pads a character to the right side of the string.

**Syntax**

```c
char *StrPad(
    char    *srcStr,
    uint    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**

```c
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**

```c
char *Subst(
    char*    subst_tbl[][2],
    char*    srtcStr
);
```

**Parameters**

- **subst_tbl**
  Substitution table.
- **srtcStr**
  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 specified length of the substring from the specified position of the source string.

**Syntax**

```c
char *SubStr(
    char* dstStr,
    char* srcStr,
    uint from_pos,
    uint length);
```
char *SubStr2(
    char*  srcStr,
    uint  from_pos,
    uint  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", 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,
    uint  from_pos,
    uint  length
);

char *SubStrLtrim2(
    char*  srcStr,
    uint  from_pos,
    uint  length
);
```

Parameters

dstStr
Destination string.

srcStr
Source string.

from_pos
The starting position to start from.

length
Length of the substring.
Sample
char str[] = "This is a substring testing";
char dst[20];
printf("Result: %s", SubStrLtrim(dst,str,11,14));
Output:
 Result: substring

Substring Pad

Function
Gets the specified length of the substring from the specified position of the source string, and pads with pad character to the right of the destination string if appropriate.

Syntax
char *SubStrPad(
      char*   dstStr,
      char*   srcStr,
      uint    from_pos,
      uint    length,
      char    pad_char
    );

char *SubStrPad2(
      char*   srcStr,
      uint    from_pos,
      uint    length,
      char    pad_char
    );

Parameters
dstStr
Destination string.
srcStr
Source string.
from_pos
The starting position to start from.
length
Length of the substring.
Pad_char
The character to use to pad destination string.

Sample
char str[] = "This is testing";
char dst[20];
SubStrPad(dst,str,9,12,'*');
printf("Result: %s", dst);
Output:
 Result: testing*****
### Substring Right Trim

**Function**

Gets the specified length of the substring from the specified position of the source string, and trims white-space characters from the right of the destination string.

**Syntax**

```c
char *SubStrRtrim(
    char* dstStr,
    char* srcStr,
    uint from_pos,
    uint length
);
```

```c
char *SubStrRtrim2(
    char* srcStr,
    uint from_pos,
    uint 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];
SubStrRtrim(dst1,str1, 11,16);
printf("Result: %s", dst1);
```

Output:

```
Result: substring
```

### Substring Trim Both Sides

**Function**

Gets the specified length of the substring from the specified position of the source string, and trims white-space, carriage return, newline control codes from both sides of the destination string.

**Syntax**

```c
char *SubStrTrim(
    char* dstStr,
    char* srcStr,
    ...)
```

```c
char *SubStrTrim2(
    char* srcStr,
    ...)
```


```
uint from_pos,
uint length
);

c char *SubStrTrim2(
    char* srcStr,
    uint from_pos,
    uint 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";
c char dst1[20];
printf("Result: %s", SubStrRtrim(dst1,str1, 11,20));
Output:
        Result: substring
```

---

**System Date and Time**

**Function**

Returns a formatted time and date string.

**Syntax**

```
char *SysTime(
    char* format
);
```

**Parameters**

format
The format argument consists of one or more codes. The formatting codes are preceded by a percent sign (%). Characters that do not begin with % are copied unchanged.

The formatting codes are listed below:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>Abbreviated weekday name</td>
</tr>
<tr>
<td>%A</td>
<td>Full weekday name</td>
</tr>
<tr>
<td>%b</td>
<td>Abbreviated month name</td>
</tr>
<tr>
<td>%B</td>
<td>Full month name</td>
</tr>
<tr>
<td>%c</td>
<td>Date and time representation appropriate for the locale</td>
</tr>
<tr>
<td>%d</td>
<td>Day of the month as decimal number (01 - 31)</td>
</tr>
<tr>
<td>%H</td>
<td>Hour in 24-hour format (00 - 23)</td>
</tr>
<tr>
<td>%I</td>
<td>Hour in 12-hour format (01 - 12)</td>
</tr>
</tbody>
</table>
Day of year as a decimal number (001 - 366)
%m    Month as a decimal number (01 - 12)
%M    Minute as a decimal number (00 - 59)
%p    Current locale's A.M./P.M. indicator for 12-hour clock
%S    Second as a decimal number (00 - 59)
%U    Week of the year as a decimal number, with Sunday as the first day of the week (00 - 53)
%W    Weekday as a decimal number (0 - 6; Sunday is 0)
%W    Week of the year as a decimal number, with Monday as the first day of the week (00 - 53)
%x    Date representation for the current locale
%X    Time representation for the current locale
%y    Year without century, as a decimal number (00 - 99)
%Y    Year with century, as a decimal number
%z, %Z Time-zone name or abbreviation; no characters if the time zone is unknown

Sample
printf("Current date is: ", SysTime("%Y-%m-%d"));
Output:
The current date is: 2008-10-22

Title String

Function
Returns a string with the first character of each word in uppercase.

Syntax
char *Title(
    char* string
);

Parameters
string    Data string to be processed.

Sample
char str[] = "This is a string testing";
printf("Title Text: %s", Title(str));
Output:
Title Text: This Is A String Testing

Translate Digits to Simplified Chinese Figures

Function
Translate ASCII digits to Simplified Chinese figures in GB18030 encoding.

Syntax
char *DigitGBK(
    char* dst,
    char* src
);

Parameters
**dst**
Destination of Simplified Chinese figures string encoding in GD18030.

**src**
Source of digits string encoded in ASCII.

**Sample**

```c
printf("Chinese figures: %s", DigitGBK("123.45"));
```

Output:

```
Chinese figures: 壹佰 貳拾 叁元 肆角 伍分
```

---

**Translate Digits to Traditional Chinese Figures**

**Function**
Translate ASCII digits to Traditional Chinese figures in BIG5 encoding.

**Syntax**

```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**

```c
char *E_Trim(
    char* string
);
```

**Parameters**

- `string`
  EBCDIC data string to be trimmed.

**Sample**

Refer to the sample for 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**

```c
char *TrimCat(
    char *separators,
    char *srcStr,...
);
```

**Parameters**

- `separators`
  Characters to be inserted between concatenated strings.

- `srcStr,...`
  Variable-argument lists of multiple strings.

**Sample**

```c
char str1[] = "      This is a string testing.    ";
char str2[] = "   This is 2nd string.    ";
printf("Result: %s", TrimCat(" ***** ", str1, str2));
```

Output:

```
Result: This is 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 returns the length of the UTF-16 output.

Syntax

```c
int32_t  ChartoU16(
    UChar   *target,
    int32_t  *targetCapacity,
    char    *source,
    int32_t  sourceLen = -1,
    char  fromCode = NULL
);
```

Parameters

target
Point to the targeted UTF-16 output buffer.

targetCapacity
The maximum size of the targeted UTF-16 buffer.

source
Pointer to the input source buffer, in bytes.

sourceLen
Length of the input source, or default -1 for NULL-terminated input.

fromCode
The name of the source encoding. Default is NULL, uses the encoding name pre-defined and loaded by the DefaultCode() function. Refer to MakePDF 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 ChartoUB(
    UChar8 *target,
    int32_t *targetCapacity,
    char *source,
    int32_t sourceLen = -1,
    char fromCode = NULL
);
```

**Parameters**

**target**
Point to the targeted UTF-8 output buffer.

**targetCapacity**
The maximum size of the targeted UTF-8 buffer.

**source**
Pointer to the input source buffer, in bytes.

**sourceLen**
Length of the input source, or default -1 for NULL-terminated input.

**fromCode**
The name of the source encoding. Default is NULL, uses the encoding name pre-defined and loaded by the DefaultCode() function. Refer to MakePDF 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 of 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 MakePDF 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 MakePDF document *Encoding Names* for more details about the available names.

- **fromCode**
  The name of the source encoding. Refer to MakePDF 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.
source
Pointer to the UTF-16 input source buffer.

sourceLen
Length of the UTF-16 input source, or default -1 for NULL-terminated input.

toCode
The name of the target encoding. Default is NULL, uses the encoding names pre-defined and loaded by DefaultCode() function. Refer to MakePDF document Encoding Names for more details about the available names.

UTF-16 to UTF-8 Conversion

Function
Converts from Unicode UTF-16 to UTF-8 and returns the length of the complete UTF-8 target output.

Syntax
```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-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.
Parameters

**target**
Point to the target UTF-16 output buffer.

**targetCapacity**
The maximum size of the UTF-16 target buffer.

**source**
Pointer to the UTF-8 input source buffer.

**sourceLen**
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

**UTF-8 to Codepage/Charset Conversion**

**Function**
Converts from Unicode UTF-8 to a codepage/charset stream returns the length of the complete target output.

**Syntax**

```
int32_t U8toChar(  
    char   *target,  
    int32_t  *targetCapacity,  
    UCha8    *source,  
    int32_t   sourceLen = -1,  
    char  toCode = NULL);  
```

**Parameters**

**target**
Point to the target output buffer.

**targetCapacity**
The maximum size of the target buffer, in bytes.

**source**
Pointer to the UTF-8 input source buffer.

**sourceLen**
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

**toCode**
The name of the target encoding. Default is NULL, uses the encoding names pre-defined and loaded by the DefaultCode() function. Refer to MakePDF document *Encoding Names for more details about the available names.*

**Syntax**

```
int32_t U8toChar(  
    UChar   *target,  
    int32_t  *targetCapacity,  
    UChar8   *source,  
    int32_t   sourceLen = -1  
);  
```
Parameters

**target**
Point to the target UTF-16 output buffer.

**targetCapacity**
The maximum size of the UTF-16 target buffer.

**source**
Pointer to the UTF-8 input source buffer.

**sourceLen**
Length of the UTF-8 input source, or default -1 for NULL-terminated input.

---

**Vietnamese Codepage/Charset Codepage/Charset Conversion**

**Function**
Converts Vietnamese from one external charset to another, like conversion between legacy PC formats, between Unicode and legacy text data, or between Unicode encodings. External string used as source or target for the conversion is always treated as a byte stream. It returns the length of the complete target output.

**Syntax**
```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. 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-G8-2312</td>
<td>Simplified Chinese, International, and national Standard</td>
<td></td>
</tr>
<tr>
<td>ibm-37, IBM037, ibm-037, eabd037, cp-us, eabd037-ca, eabd037-wt, eabd037-nl, csIBM037, cp037, 037, cpibm37, cp37, T1V10037</td>
<td>USA/Canada – CECP, EBCDIC</td>
<td>037</td>
</tr>
<tr>
<td>ibm-259, IBM-Symbols, csIBMSymbols</td>
<td>Symbols, Set 7, EBCDIC</td>
<td></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, eabd037, cp-dk, csIBM278, ebcdic-sv, cpibm278, 278, T1V10278</td>
<td>Finland/Sweden- CECP, EBCDIC</td>
<td>278</td>
</tr>
<tr>
<td>ibm-280, IBM280, CP280, eabd037, cp-it, csIBM280, cpibm280, 280, T1V10280</td>
<td>ITALY- CECP, EBCDIC</td>
<td>280</td>
</tr>
<tr>
<td>ibm-284, IBM284, CP284, eabd037, 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, csEBCDICATDEA, 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, eabd037, cp-fr, csIBM297, cpibm297, 297, T1V10297</td>
<td>France – CECP, EBCDIC</td>
<td>297</td>
</tr>
<tr>
<td>ibm-367, US-ASCII, ASCII, ANSI_X3.4-1968, ANSI_X3.4-1986,</td>
<td>ASCII</td>
<td>367</td>
</tr>
<tr>
<td>Code</td>
<td>Name Description</td>
<td>Code</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>ibm-420, IBM420, cp420, ebcdic-cp-ar1, cslIBM420, 420, T1000420</td>
<td>Arabi Bilingual, EBCDIC</td>
<td>420</td>
</tr>
<tr>
<td>ibm-424, IBM424, cp424, ebcdic-cp-he, cslIBM424, 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, cslIBM500, 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, cslSOLatin1, iso-ir-100, ISO_8859, ISO_8859-1:1987, 819, T1000819</td>
<td>Latin-1, West European, ASCII</td>
<td>819</td>
</tr>
<tr>
<td>ibm-838, IBM-Thai, cslIBMThai, 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, cslIBM855, 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, cslIBM857, 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, cslIBM860, T1000860</td>
<td>Portugal - Personal Computer, ASCII</td>
<td>860</td>
</tr>
<tr>
<td>ibm-861, IBM861, cp861, 861, cp-is, cslIBM861, windows-861, T1000861</td>
<td>Iceland - Personal Computer, ASCII</td>
<td>861</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Language</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>ibm-862</td>
<td>IBM862, cp862, 862, DOS-862, cspc862LatinHebrew, windows-862, T1000862</td>
<td>Hebrew - PC, ASCII</td>
</tr>
<tr>
<td>ibm-863</td>
<td>IBM863, cp863, 863, csIBM863, T1000863</td>
<td>Canadian French - PC, ASCII</td>
</tr>
<tr>
<td>ibm-864</td>
<td>IBM864, cp864, csIBM864, T1000864</td>
<td>Arabic - PC, ASCII</td>
</tr>
<tr>
<td>ibm-865</td>
<td>IBM865, cp865, 865, csIBM865, T1000865</td>
<td>Nordic - PC, ASCII</td>
</tr>
<tr>
<td>ibm-866</td>
<td>IBM866, cp866, 866, csIBM866, windows-866, T1000866</td>
<td>Cyrillic #2 - PC, ASCII</td>
</tr>
<tr>
<td>ibm-867</td>
<td>cp867, T1000867</td>
<td>Israel - PC, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-868</td>
<td>IBM868, CP868, 868, csIBM868, cp-ar, T1000868</td>
<td>Urdu, ASCII</td>
</tr>
<tr>
<td>ibm-869</td>
<td>IBM869, cp869, 869, cp-gr, csIBM869, windows-869, T1000869</td>
<td>Greek - PC, ASCII</td>
</tr>
<tr>
<td>ibm-870</td>
<td>IBM870, CP870, ebcdic-cp-roece, ebcdic-cp-yu, csIBM870, T1000870</td>
<td>Latin2 Multilingual, EBCDIC</td>
</tr>
<tr>
<td>ibm-871</td>
<td>IBM871, ebcdic-cp-is, csIBM871, CP871, ebcdic-is, cpibm871, 871, T1V10871</td>
<td>Iceland – CECP, EBCDIC</td>
</tr>
<tr>
<td>ibm-872</td>
<td>T1000872</td>
<td>Cyrillic PC with Euro, ASCII</td>
</tr>
<tr>
<td>ibm-874</td>
<td>ibm-9066, cp874, TIS-620, tis620.2533, eucTH, cp9066, ibm874, MS874, windows-874, T1000874</td>
<td>Thai - PC, ASCII</td>
</tr>
<tr>
<td>ibm-875</td>
<td>IBM875, cp875, 875, T1000875</td>
<td>Greece, EBCDIC</td>
</tr>
<tr>
<td>ibm-877</td>
<td>KOI8-R, koi8, csKOI8R, cp878, windows-20866, T1000878</td>
<td>Russian internet, ASCII</td>
</tr>
<tr>
<td>ibm-880</td>
<td>IBM880, cp880, EBCDIC-Cyrillic, csIBM880, windows-20880, T1000880</td>
<td>Cyrillic Multilingual, ASCII</td>
</tr>
<tr>
<td>ibm-896</td>
<td>T1000896</td>
<td>Japanese Katakana, ASCII</td>
</tr>
<tr>
<td>ibm-897</td>
<td>JIS_X0201, X0201, csHalfWidthKatakana, T1000897</td>
<td>Japanese, Half Width Katakana, ASCII</td>
</tr>
<tr>
<td>ibm-901</td>
<td>T1000901</td>
<td>Baltic – PC Multilingual, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-902</td>
<td>T1000902</td>
<td>Estonia, ASCII with Euro sign</td>
</tr>
<tr>
<td>ibm-905</td>
<td>IBM905, CP905, EBCDIC-cp-tr, csIBM905, windows-20905, T1000905</td>
<td>Latin3 Multilingual, EBCDIC</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>ibm-915</td>
<td>Cyrillic ISO/ASCII 8-Bit, ASCII</td>
<td></td>
</tr>
<tr>
<td>ibm-916</td>
<td>Hebrew ISO/ASCII 8-Bit, ASCII</td>
<td></td>
</tr>
<tr>
<td>ibm-918</td>
<td>Urdu, EBCDIC</td>
<td></td>
</tr>
<tr>
<td>ibm-920</td>
<td>Latin5 ISO/ANSI 8-BIT, ASCII</td>
<td></td>
</tr>
<tr>
<td>ibm-921</td>
<td>Baltic – PC Multilingual, ASCII</td>
<td></td>
</tr>
<tr>
<td>ibm-922</td>
<td>Estonian, ASCII</td>
<td></td>
</tr>
<tr>
<td>ibm-923</td>
<td>Latin 9, ASCII</td>
<td></td>
</tr>
<tr>
<td>ibm-924</td>
<td>Latin 9 EBCDIC</td>
<td></td>
</tr>
<tr>
<td>ibm-930</td>
<td>Japanese, mixed EBCDIC/DBCS-HOST</td>
<td></td>
</tr>
<tr>
<td>ibm-931</td>
<td>Japanese, EBCDIC</td>
<td></td>
</tr>
<tr>
<td>ibm-932</td>
<td>Japanese, SJIS without MS/IBM extensions</td>
<td></td>
</tr>
<tr>
<td>ibm-933</td>
<td>Korean, mixed EBCDIC/DBCS-HOST</td>
<td></td>
</tr>
<tr>
<td>ibm-935</td>
<td>Chinese (simplified), mixed EBCDIC/DBCS-HOST</td>
<td></td>
</tr>
<tr>
<td>ibm-936</td>
<td>Windows Chinese(simplified), GBK</td>
<td></td>
</tr>
<tr>
<td>ibm-937</td>
<td>Chinese (traditional), mixed EBCDIC/DBCS-HOST</td>
<td></td>
</tr>
<tr>
<td>ibm-943</td>
<td>Japanese, Shift-JIS standard with extension</td>
<td></td>
</tr>
<tr>
<td>ibm-949</td>
<td>Windows Korean, (old) encoding</td>
<td></td>
</tr>
<tr>
<td>ibm-950</td>
<td>Windows Chinese(Traditional), ASCII BIG5</td>
<td></td>
</tr>
<tr>
<td>ibm-954</td>
<td>Extended Unix Code Packed Format for Japanese</td>
<td></td>
</tr>
<tr>
<td>ibm-964</td>
<td>Extended Unix Code Packed Format for Chinese (Traditional)</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Identifier</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</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>Cyrilic 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>Cyrilic, Ukraine EBCDIC</td>
<td>1123</td>
</tr>
<tr>
<td>Ibm-1124, cp1124, 1124, T1001124</td>
<td>ISO Cyrilic Ukraine, ASCII</td>
<td>1124</td>
</tr>
<tr>
<td>Ibm-1125, cp1125, 1125, T1001125</td>
<td>Cyrilic 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, T1001131</td>
<td>Cyrilic 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, cpibm1140, 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, cpibm1141, 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, cpibm1142, 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, cpibm1143, 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, cpibm1144, 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, cpibm1145, ebcdic-es-284+euro, T1001145</td>
<td>Spain, Latin America ECECP, EBCDIC with the Euro sign</td>
<td>1145</td>
</tr>
<tr>
<td>Ibm-1146, IBM01146, CCSID01146, UK ECECP, EBCDIC with the Euro sign</td>
<td>UK ECECP, EBCDIC with the Euro sign</td>
<td>1146</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>CP01146, cp1146, cpibm1146, ebc dic-gb-285+euro, T1001146</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ibm-1147, IBM01147, CCSID01147, CP01147, cp1147, cpibm1147, ebc dic-fr-297+euro, T1001147</td>
<td>France ECECP, EBCDIC with the Euro sign</td>
<td>1147</td>
</tr>
<tr>
<td>ibm-1148, IBM01148, CCSID01148, CP01148, cp1148, cpibm1148, ebc dic-international-500+euro, T1001148</td>
<td>International Latin 1, EBCDIC with the Euro sign</td>
<td>1148</td>
</tr>
<tr>
<td>ibm-1149, IBM01149, CCSID01149, CP01149, cp1149, cpibm1149, ebc dic-is-871+euro, T1001149</td>
<td>Iceland, EBCDIC with the Euro sign</td>
<td>1149</td>
</tr>
<tr>
<td>ibm-1153, cpibm1153, T1001153</td>
<td>Latin2 Multilingual with Euro, EBCDIC</td>
<td>1153</td>
</tr>
<tr>
<td>ibm-1154, cpibm1154, T1001154</td>
<td>Cyrillic Multilingual with Euro, EBCDIC</td>
<td>1154</td>
</tr>
<tr>
<td>ibm-1155, cpibm1155, T1001155</td>
<td>EBCDIC Turkey with Euro, EBCDIC</td>
<td>1155</td>
</tr>
<tr>
<td>ibm-1156, cpibm1156, T1001156</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>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</td>
<td>1236</td>
</tr>
<tr>
<td>IBM Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></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>IBM-1252, 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 Arabic, with the Euro sign</td>
<td>1256</td>
</tr>
<tr>
<td>IBM-1257, windows-1257, CP1257, 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, Big5HKSCS</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, EUCCN, 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>IBM-1390, CPIBM1390</td>
<td>Japanese, mixed EBCDIC/DBCS-HOST with the Euro sign</td>
<td>1390</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>ibm-1392</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</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</td>
<td>EBCDIC Greek, with the Euro sign</td>
<td>4971</td>
</tr>
<tr>
<td>ibm-5012</td>
<td>Hebrew, ASCII</td>
<td>5012</td>
</tr>
<tr>
<td>ibm-5471</td>
<td>Chinese (traditional) Big5-HKSCS-2001 Hong Kong with Unicode 3.0 mappings</td>
<td>5471</td>
</tr>
<tr>
<td>ibm-5478</td>
<td>Extended Unix Code Packed Format for Chinese (Simplified)</td>
<td>5478</td>
</tr>
<tr>
<td>ibm-9005</td>
<td>ISO Greek (with euro update), ASCII</td>
<td>9005</td>
</tr>
<tr>
<td>ibm-12712</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</td>
<td>Arabic, EBCDIC with the Euro sign</td>
<td>16804</td>
</tr>
<tr>
<td>ibm-33722</td>
<td>Extended Unix Code Packed Format for Japanese</td>
<td>33722</td>
</tr>
<tr>
<td>x iscii-de</td>
<td>Windows Devanagari</td>
<td></td>
</tr>
<tr>
<td>x iscii-be</td>
<td>Windows Bengali</td>
<td></td>
</tr>
<tr>
<td>x iscii-pa</td>
<td>Windows Punjabi</td>
<td></td>
</tr>
<tr>
<td>x iscii-gu</td>
<td>Windows Gujarati</td>
<td></td>
</tr>
<tr>
<td>x iscii-or</td>
<td>Windows Oriya</td>
<td></td>
</tr>
<tr>
<td>x iscii-ta</td>
<td>Windows Tamil</td>
<td></td>
</tr>
<tr>
<td>x iscii-te</td>
<td>Windows Telugu</td>
<td></td>
</tr>
<tr>
<td>x iscii-ka</td>
<td>Windows Kannada</td>
<td></td>
</tr>
<tr>
<td>x iscii-ma</td>
<td>Windows Malayalam</td>
<td></td>
</tr>
<tr>
<td>macos-0_2-10.2</td>
<td>macintosh, mac, csMacintosh, windows-10000</td>
<td></td>
</tr>
<tr>
<td>macos-2566-10.2</td>
<td>Big5-HKSCS, bigShk, HKSCS-BIG5</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The table contains various code sets and their associated descriptions. The codes and their descriptions are used to represent different character sets and encoding standards, particularly for languages like Chinese, Japanese, Greek, and Hebrew, among others. Each code set is linked to specific national and international standards, ensuring compatibility and proper representation of text in different locales and languages.
<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>macos-29-10.2, x-mac-centraleurroman, windows-10029, x-mac-ce, macce</td>
<td>Macintosh, Central Euro, ASCII</td>
</tr>
<tr>
<td>macos-35-10.2, x-mac-turkish, windows-10081, mactr</td>
<td>Macintosh, Turkish, ASCII</td>
</tr>
<tr>
<td>macos-6-10.2, x-mac-greek, windows-10006, macgr</td>
<td>Macintosh, Greek, ASCII</td>
</tr>
<tr>
<td>macos-7_3-10.2, x-mac-cyrillic, windows-10007, maccy</td>
<td>Macintosh, Cyrillic, ASCII</td>
</tr>
<tr>
<td>UTF-7, windows-65000</td>
<td>Unicode UTF-7, a Unicode character set with 7-bit characters; used primarily for email headers</td>
</tr>
</tbody>
</table>
Appendix B. 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 components 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>
<tbody>
<tr>
<td>Language</td>
<td>Language_Country</td>
</tr>
</tbody>
</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>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>Afar</td>
<td>cy</td>
<td>Welsh</td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td>Abkhazian</td>
<td>da</td>
<td>Danish</td>
<td></td>
</tr>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td>de</td>
<td>German</td>
<td></td>
</tr>
<tr>
<td>am</td>
<td>Amharic</td>
<td>dz</td>
<td>Bhutani</td>
<td></td>
</tr>
<tr>
<td>ar</td>
<td>Arabic</td>
<td>el</td>
<td>Greek</td>
<td></td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
<td>en</td>
<td>English</td>
<td></td>
</tr>
<tr>
<td>ay</td>
<td>Aymara</td>
<td>eo</td>
<td>Esperanto</td>
<td></td>
</tr>
<tr>
<td>az</td>
<td>Azerbaijani</td>
<td>es</td>
<td>Spanish</td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td>Bashkir</td>
<td>et</td>
<td>Estonian</td>
<td></td>
</tr>
<tr>
<td>be</td>
<td>Byelorussian</td>
<td>eu</td>
<td>Basque</td>
<td></td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarian</td>
<td>fa</td>
<td>Persian</td>
<td></td>
</tr>
<tr>
<td>bh</td>
<td>Bihari</td>
<td>fo</td>
<td>Faroese</td>
<td></td>
</tr>
<tr>
<td>bi</td>
<td>Bislama</td>
<td>fr</td>
<td>French</td>
<td></td>
</tr>
<tr>
<td>bn</td>
<td>Bengali; Bangla</td>
<td>fy</td>
<td>Frisian</td>
<td></td>
</tr>
<tr>
<td>bo</td>
<td>Tibetan</td>
<td>ga</td>
<td>Irish</td>
<td></td>
</tr>
<tr>
<td>br</td>
<td>Breton</td>
<td>gd</td>
<td>Scots Gaelic</td>
<td></td>
</tr>
<tr>
<td>ca</td>
<td>Catalan</td>
<td>gl</td>
<td>Galician</td>
<td></td>
</tr>
<tr>
<td>co</td>
<td>Corsican</td>
<td>gn</td>
<td>Guarani</td>
<td></td>
</tr>
<tr>
<td>cs</td>
<td>Czech</td>
<td>gu</td>
<td>Gujarati</td>
<td></td>
</tr>
<tr>
<td>ha</td>
<td>Hausa</td>
<td>m</td>
<td>Kirundi</td>
<td></td>
</tr>
<tr>
<td>he</td>
<td>Hebrew</td>
<td>ro</td>
<td>Romanian</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
<td>Code</td>
<td>Language</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>hi</td>
<td>Hindi</td>
<td>ru</td>
<td>Russian</td>
<td></td>
</tr>
<tr>
<td>hr</td>
<td>Croatian</td>
<td>rw</td>
<td>Kinyarwanda</td>
<td></td>
</tr>
<tr>
<td>hu</td>
<td>Hungarian</td>
<td>sa</td>
<td>Sanskrit</td>
<td></td>
</tr>
<tr>
<td>hy</td>
<td>Armenian</td>
<td>sd</td>
<td>Sindhi</td>
<td></td>
</tr>
<tr>
<td>ia</td>
<td>Interlingua</td>
<td>sg</td>
<td>Sangho</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>Indonesian</td>
<td>sh</td>
<td>Serbo-Croatian</td>
<td></td>
</tr>
<tr>
<td>ie</td>
<td>Interlingue</td>
<td>si</td>
<td>Sinhalese</td>
<td></td>
</tr>
<tr>
<td>ik</td>
<td>Inupiak</td>
<td>sk</td>
<td>Slovak</td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>Icelandic</td>
<td>sl</td>
<td>Slovenian</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>Italian</td>
<td>sm</td>
<td>Samoan</td>
<td></td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
<td>so</td>
<td>Somali</td>
<td></td>
</tr>
<tr>
<td>jw</td>
<td>Javanese</td>
<td>sq</td>
<td>Albanian</td>
<td></td>
</tr>
<tr>
<td>ka</td>
<td>Georgian</td>
<td>sr</td>
<td>Serbian</td>
<td></td>
</tr>
<tr>
<td>kk</td>
<td>Kazakh</td>
<td>ss</td>
<td>Siswati</td>
<td></td>
</tr>
<tr>
<td>kl</td>
<td>Greenlandic</td>
<td>st</td>
<td>Sesotho</td>
<td></td>
</tr>
<tr>
<td>km</td>
<td>Cambodian</td>
<td>su</td>
<td>Sundanese</td>
<td></td>
</tr>
<tr>
<td>kn</td>
<td>Kannada</td>
<td>sv</td>
<td>Swedish</td>
<td></td>
</tr>
<tr>
<td>ko</td>
<td>Korean</td>
<td>sw</td>
<td>Swahili</td>
<td></td>
</tr>
<tr>
<td>ks</td>
<td>Kashmiri</td>
<td>ta</td>
<td>Tamil</td>
<td></td>
</tr>
<tr>
<td>ku</td>
<td>Kurdish</td>
<td>te</td>
<td>Telugu</td>
<td></td>
</tr>
<tr>
<td>ky</td>
<td>Kirghiz</td>
<td>tg</td>
<td>Tajik</td>
<td></td>
</tr>
<tr>
<td>la</td>
<td>Latin</td>
<td>th</td>
<td>Thai</td>
<td></td>
</tr>
<tr>
<td>ln</td>
<td>Lingala</td>
<td>ti</td>
<td>Tigrinya</td>
<td></td>
</tr>
<tr>
<td>lo</td>
<td>Laotian</td>
<td>tk</td>
<td>Turkmen</td>
<td></td>
</tr>
<tr>
<td>lt</td>
<td>Lithuanian</td>
<td>tl</td>
<td>Tagalog</td>
<td></td>
</tr>
<tr>
<td>lv</td>
<td>Latvian, Lettish</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mg</td>
<td>Malagasy</td>
<td>to</td>
<td>Tonga</td>
<td></td>
</tr>
<tr>
<td>mi</td>
<td>Maori</td>
<td>tr</td>
<td>Turkish</td>
<td></td>
</tr>
<tr>
<td>mk</td>
<td>Macedonian</td>
<td>ts</td>
<td>Tsonga</td>
<td></td>
</tr>
<tr>
<td>ml</td>
<td>Malayalam</td>
<td>tt</td>
<td>Tatar</td>
<td></td>
</tr>
<tr>
<td>mn</td>
<td>Mongolian</td>
<td>tw</td>
<td>Taiwan</td>
<td></td>
</tr>
<tr>
<td>mo</td>
<td>Moldavian</td>
<td>ug</td>
<td>Uighur</td>
<td></td>
</tr>
<tr>
<td>mr</td>
<td>Marathi</td>
<td>uk</td>
<td>Ukrainian</td>
<td></td>
</tr>
<tr>
<td>ms</td>
<td>Malay</td>
<td>ur</td>
<td>Urdu</td>
<td></td>
</tr>
<tr>
<td>mt</td>
<td>Maltese</td>
<td>uz</td>
<td>Uzbek</td>
<td></td>
</tr>
<tr>
<td>my</td>
<td>Burmese</td>
<td>vi</td>
<td>Vietnamese</td>
<td></td>
</tr>
<tr>
<td>na</td>
<td>Nauru</td>
<td>vo</td>
<td>Volapuk</td>
<td></td>
</tr>
<tr>
<td>ne</td>
<td>Nepali</td>
<td>wo</td>
<td>Wolof</td>
<td></td>
</tr>
<tr>
<td>nl</td>
<td>Dutch</td>
<td>xh</td>
<td>Xhosa</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>Norwegian</td>
<td>yi</td>
<td>Yiddish</td>
<td></td>
</tr>
<tr>
<td>oc</td>
<td>Occitan</td>
<td>yo</td>
<td>Yoruba</td>
<td></td>
</tr>
<tr>
<td>om</td>
<td>(Afan) Oromo</td>
<td>za</td>
<td>Zhuang</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>Oriya</td>
<td>zh</td>
<td>Chinese</td>
<td></td>
</tr>
<tr>
<td>pa</td>
<td>Punjabi</td>
<td>zu</td>
<td>Zulu</td>
<td></td>
</tr>
<tr>
<td>pl</td>
<td>Polish</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ps</td>
<td>Pashto, Pushto</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qu</td>
<td>Quechua</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rm</td>
<td>Rhaeto-Romance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Country/Area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>AFGHANISTAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AX</td>
<td>ALAND ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AL</td>
<td>ALBANIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DZ</td>
<td>ALGERIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AS</td>
<td>AMERICAN SAMOA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>ANDORRA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AO</td>
<td>ANGOLA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AI</td>
<td>ANGUILLA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AQ</td>
<td>ANTARCTICA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG</td>
<td>ANTIGUA AND BARBUDA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR</td>
<td>ARGENTINA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td>ARMENIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AW</td>
<td>ARUBA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>AUSTRALIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT</td>
<td>AUSTRIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AZ</td>
<td>AZERBAIJAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>BAHAMAS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BH</td>
<td>BAHRAIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BD</td>
<td>BANGLADESH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>BARBADOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BY</td>
<td>BELARUS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>BELGIUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BZ</td>
<td>BELIZE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BJ</td>
<td>BENIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>BERMUDA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>BHUTAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BO</td>
<td>BOLIVIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td>BOSNIA AND HERZEGOVINA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW</td>
<td>BOTSWANA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BV</td>
<td>BOUVET ISLAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>BRAZIL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>BRITISH INDIAN OCEAN TERRITORY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BN</td>
<td>BRUNEI DARUSSALAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BG</td>
<td>BULGARIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>BURKINA FASO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BI</td>
<td>BURUNDI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KH</td>
<td>CAMBODIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM</td>
<td>CAMEROON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>CANADA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV</td>
<td>CAPE VERDE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KY</td>
<td>CAYMAN ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF</td>
<td>CENTRAL AFRICAN REPUBLIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TD</td>
<td>CHAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CL</td>
<td>CHILE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>CHINA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CX</td>
<td>CHRISTMAS ISLAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>COCOS (KEELING) ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO</td>
<td>COLOMBIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KM</td>
<td>COMOROS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Country Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td>CONGO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>CONGO, THE DEMOCRATIC REPUBLIC OF THE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK</td>
<td>COOK ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>COSTA RICA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CI</td>
<td>COTE D'IVOIRE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HR</td>
<td>CROATIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CU</td>
<td>CUBA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CY</td>
<td>CYPRUS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CZ</td>
<td>CZECH REPUBLIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DK</td>
<td>DENMARK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DJ</td>
<td>DJIBOUTI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DM</td>
<td>DOMINICA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td>DOMINICAN REPUBLIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td>ECUADOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EG</td>
<td>EGYPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV</td>
<td>EL SALVADOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GQ</td>
<td>EQUATORIAL GUINEA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ER</td>
<td>ERITREA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>ESTONIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ET</td>
<td>ETHIOPIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FK</td>
<td>FALKLAND ISLANDS (MALVINAS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FO</td>
<td>FAROE ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FJ</td>
<td>FUJISI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td>FINLAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td>FRANCE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GF</td>
<td>FRENCH GUIANA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PF</td>
<td>FRENCH POLYNESIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TF</td>
<td>FRENCH SOUTHERN TERRITORIES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>GABON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM</td>
<td>GAMBIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>GEORGIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>GERMANY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GH</td>
<td>GHANA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GI</td>
<td>GIBRALTAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GR</td>
<td>GREECE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GL</td>
<td>GREENLAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD</td>
<td>GRENADA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GP</td>
<td>GUADELOUPE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GU</td>
<td>GUAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GT</td>
<td>GUATEMALA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GG</td>
<td>GUERNSEY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GN</td>
<td>GUINEA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GW</td>
<td>GUINEA-BISSAU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GY</td>
<td>GUYANA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HT</td>
<td>HAITI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HM</td>
<td>HEARD ISLAND AND MCDONALD ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VA</td>
<td>HOLY SEE (VATICAN CITY STATE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HN</td>
<td>HONDURAS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK</td>
<td>HONG KONG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HU</td>
<td>HUNGARY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>ICELAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>INDIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>INDONESIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>--------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR</td>
<td>IRAN (ISLAMIC REPUBLIC OF)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IQ</td>
<td>IRAQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td>IRELAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IM</td>
<td>ISLE OF MAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>ISRAEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IT</td>
<td>ITALY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JM</td>
<td>JAMAICA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JP</td>
<td>JAPAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JE</td>
<td>JERSEY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JO</td>
<td>JORDAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KZ</td>
<td>KAZAKHSTAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KE</td>
<td>KENYA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KI</td>
<td>KIRIBATI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KP</td>
<td>KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KR</td>
<td>KOREA, REPUBLIC OF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KW</td>
<td>KUWAIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KG</td>
<td>KYRGYZSTAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>LAO PEOPLE'S DEMOCRATIC REPUBLIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LV</td>
<td>LATVIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB</td>
<td>LEBANON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS</td>
<td>LESOTHO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR</td>
<td>LIBERIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LY</td>
<td>LIBYAN ARAB JAMAHIRIYA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LI</td>
<td>LIECHTENSTEIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT</td>
<td>LITHUANIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td>LUXEMBOURG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MO</td>
<td>MACAO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MK</td>
<td>MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td>MADAGASCAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW</td>
<td>MALAWI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MY</td>
<td>MALAYSIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MV</td>
<td>MALDIVES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>MALI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MT</td>
<td>MALTA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MH</td>
<td>MARSHALL ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQ</td>
<td>MARTINIQUE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MR</td>
<td>MAURITANIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MU</td>
<td>MAURITIUS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YT</td>
<td>MAYOTTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MX</td>
<td>MEXICO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FM</td>
<td>MICRONESIA, FEDERATED STATES OF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD</td>
<td>MOLDOVA, REPUBLIC OF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC</td>
<td>MONACO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MN</td>
<td>MONGOLIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ME</td>
<td>MONTENEGRO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS</td>
<td>MONTSESRAT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MA</td>
<td>MOROCCO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MZ</td>
<td>MOZAMBIQUE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>MYANMAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NA</td>
<td>NAMIBIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NR</td>
<td>NAURU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Country</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>NEPAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NL</td>
<td>NETHERLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AN</td>
<td>NETHERLANDS ANTILLES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td>NEW CALEDONIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NZ</td>
<td>NEW ZEALAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NI</td>
<td>NICARAGUA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>NIGER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NG</td>
<td>NIGERIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NU</td>
<td>NIUE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>NORFOLK ISLAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>NORTHERN MARIANA ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td>NORWAY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OM</td>
<td>OMAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK</td>
<td>PAKISTAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW</td>
<td>PALAU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>PALESTINIAN TERRITORY, OCCUPIED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PA</td>
<td>PANAMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG</td>
<td>PAPUA NEW GUINEA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PY</td>
<td>PARAGUAY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE</td>
<td>PERU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PH</td>
<td>PHILIPPINES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PN</td>
<td>PITCAIRN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL</td>
<td>POLAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT</td>
<td>PORTUGAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>PUERTO RICO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QA</td>
<td>QATAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RE</td>
<td>REUNION</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RO</td>
<td>ROMANIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td>RUSSIAN FEDERATION</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RW</td>
<td>RWANDA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td>SAINT HELENA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KN</td>
<td>SAINT KITTS AND NEVIS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC</td>
<td>SAINT LUCIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM</td>
<td>SAINT PIERRE AND MIQUELON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td>SAINT VINCENT AND THE GRENADINES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td>SAMOA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM</td>
<td>SAN MARINO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>SAO TOME AND PRINCIPE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA</td>
<td>SAUDI ARABIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SN</td>
<td>SENEGAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RS</td>
<td>SERBIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>SEYCHELLES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SL</td>
<td>SIERRA LEONE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SG</td>
<td>SINGAPORE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SK</td>
<td>SLOVAKIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>SLOVENIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB</td>
<td>SOLOMON ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SO</td>
<td>SOMALIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZA</td>
<td>SOUTH AFRICA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GS</td>
<td>SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ES</td>
<td>SPAIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LK</td>
<td>SRI LANKA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Country</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>Sudan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR</td>
<td>Suriname</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SJ</td>
<td>Svalbard and Jan Mayen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td>St. Helena</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM</td>
<td>St. Pierre and Miquelon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SZ</td>
<td>Swaziland</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>Sweden</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>Switzerland</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY</td>
<td>Syrian Arab Republic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TW</td>
<td>Taiwan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TJ</td>
<td>Tajikistan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TZ</td>
<td>Tanzania, United Republic of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TH</td>
<td>Thailand</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TL</td>
<td>Timor-Leste</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TG</td>
<td>Togo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TK</td>
<td>Tokelau</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO</td>
<td>Tonga</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TT</td>
<td>Trinidad and Tobago</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TN</td>
<td>Tunisia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR</td>
<td>Turkey</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td>Turkmenistan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TC</td>
<td>Turks and Caicos Islands</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TV</td>
<td>Tuvalu</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UG</td>
<td>Uganda</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UA</td>
<td>Ukraine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>United Arab Emirates</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GB</td>
<td>United Kingdom</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UM</td>
<td>United States Minor Outlying Islands</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UY</td>
<td>Uruguay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UZ</td>
<td>Uzbekistan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VU</td>
<td>Vanuatu</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VA</td>
<td>Vatican City State (Holy See)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VE</td>
<td>Venezuela</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VN</td>
<td>Viet Nam</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VG</td>
<td>Virgin Islands (British)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI</td>
<td>Virgin Islands (U.S.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WF</td>
<td>Wallis and Futuna</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EH</td>
<td>Western Sahara</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YE</td>
<td>Yemen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YU</td>
<td>Yugoslavia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZM</td>
<td>Zambia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZW</td>
<td>Zimbabwe</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C. Visual Studio C++ Run-Time Routines

For reference convenience, some Visual Studio C++ routines you may use are listed here by the categories. Refer to Microsoft MSDN Library for more details.

### Buffer-Manipulation Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_memcpy</td>
<td>Copy characters from one buffer to another until a given character or given number of characters has been copied</td>
</tr>
<tr>
<td>memchr</td>
<td>Return a pointer to the first occurrence, within a specified number of characters, of a given character in the buffer</td>
</tr>
<tr>
<td>memcmp</td>
<td>Compare the specified number of characters from two buffers</td>
</tr>
<tr>
<td>memcpy</td>
<td>Copy specified number of characters from one buffer to another</td>
</tr>
<tr>
<td>_memicmp</td>
<td>Compare the specified number of characters from two buffers without regard to case</td>
</tr>
<tr>
<td>memmove</td>
<td>Copy specified number of characters from one buffer to another</td>
</tr>
<tr>
<td>memset</td>
<td>Use given character to initialize a specified number of bytes in the buffer</td>
</tr>
<tr>
<td>_swab</td>
<td>Swap bytes of data and store them at the specified location</td>
</tr>
</tbody>
</table>

### Character-Classification Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Character test condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum, iswalnum, _ismbcalnum</td>
<td>True if alphanumeric</td>
</tr>
<tr>
<td>isalpha, iswalpha, _ismbcalpha</td>
<td>True if alphabetic</td>
</tr>
<tr>
<td>_isascii, iswascii</td>
<td>True if ASCII</td>
</tr>
<tr>
<td>iscntrl, iswcntrl</td>
<td>True if control character</td>
</tr>
<tr>
<td>_iscsym</td>
<td>True if the letter, underline, or digit</td>
</tr>
<tr>
<td>_iscsymf</td>
<td>True if the letter or underline</td>
</tr>
<tr>
<td>isdigit, iswdigit, _ismbcdigit</td>
<td>True if the decimal digit</td>
</tr>
<tr>
<td>isgraph, iswgraph, _ismbcgraph</td>
<td>True if printable other than space</td>
</tr>
<tr>
<td>islower, iswlower, _ismbclower</td>
<td>True if lowercase</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>_ismbchira</td>
<td>True if Hiragana</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, _ismbcpunct</td>
<td>True if the 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, _ismbcspace</td>
<td>True if uppercase</td>
</tr>
<tr>
<td>iswctype</td>
<td>The property specified by desc argument</td>
</tr>
<tr>
<td>isxdigit, iswxdigit</td>
<td>True if the hexadecimal digit</td>
</tr>
<tr>
<td>mblen</td>
<td>Return the length of valid multibyte character; result depends on LC_CTYPE category setting of the 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 the console</td>
</tr>
<tr>
<td>_cprintf, _cwprintf</td>
<td>Write formatted data to the console</td>
</tr>
<tr>
<td>_cputs</td>
<td>Write a string to console</td>
</tr>
<tr>
<td>_cscanf, _cwscanf</td>
<td>Read formatted data from the console</td>
</tr>
<tr>
<td>_getch, _getwch</td>
<td>Read a character from the console</td>
</tr>
<tr>
<td>_getche, _getwche</td>
<td>Read a character from the console and echo it</td>
</tr>
<tr>
<td>_inp</td>
<td>Read one byte from the specified I/O port</td>
</tr>
<tr>
<td>_inpd</td>
<td>Read float word from specified I/O port</td>
</tr>
<tr>
<td>_inpw</td>
<td>Read 2-byte word from specified I/O port</td>
</tr>
<tr>
<td>_kbhit</td>
<td>Check for keystroke at the console; use before attempting to read from the console</td>
</tr>
<tr>
<td>_outp</td>
<td>Write one byte to specified I/O port</td>
</tr>
<tr>
<td>_outpd</td>
<td>Write float word to specified I/O port</td>
</tr>
<tr>
<td>_outpw</td>
<td>Write the word to the specified I/O port</td>
</tr>
<tr>
<td>_putch, _putwch</td>
<td>Write character to console</td>
</tr>
<tr>
<td>_ungetch, _ungetwch</td>
<td>&quot;Unget&quot; last character read from the console so it becomes the next character read</td>
</tr>
</tbody>
</table>
## Data-Conversion Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Find the absolute value of the integer</td>
</tr>
<tr>
<td>atof</td>
<td>Convert string to float</td>
</tr>
<tr>
<td>atoi, _atoi64</td>
<td>Convert string to int</td>
</tr>
<tr>
<td>atol</td>
<td>Convert string to long</td>
</tr>
<tr>
<td>_ecvt</td>
<td>Convert float to the string of specified length</td>
</tr>
<tr>
<td>_fcvt</td>
<td>Convert float to string with a specified number of digits following the decimal point</td>
</tr>
<tr>
<td>_gcvt</td>
<td>Convert float number to string; store string in the buffer</td>
</tr>
<tr>
<td>_itoa, _i64toa, _itow, _i64tow</td>
<td>Convert int to string</td>
</tr>
<tr>
<td>labs</td>
<td>Find the absolute value of the long integer</td>
</tr>
<tr>
<td>_itoa, _itow</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>_mbcjjstojms</td>
<td>Convert Japan Industry Standard (JIS) character to Japan Microsoft (JMS) character</td>
</tr>
<tr>
<td>_mbcjmsstojis</td>
<td>Convert JMS character to JIS character</td>
</tr>
<tr>
<td>_mbctohira</td>
<td>Convert multibyte character to 1-byte hiragana code</td>
</tr>
<tr>
<td>_mbctokata</td>
<td>Convert multibyte character to 1-byte katakana code</td>
</tr>
<tr>
<td>_mbctombb</td>
<td>Convert 2-byte multibyte character to corresponding 1-byte multibyte character</td>
</tr>
<tr>
<td>mbstowcs</td>
<td>Convert sequence of multibyte characters to the corresponding sequence of wide characters</td>
</tr>
<tr>
<td>mbtowc</td>
<td>Convert multibyte character to corresponding wide character</td>
</tr>
<tr>
<td>strtod, wcstod</td>
<td>Convert string to a float</td>
</tr>
<tr>
<td>strtol, wcstol</td>
<td>Convert string to a long integer</td>
</tr>
<tr>
<td>strtoul, wcstoul</td>
<td>Convert string to an unsigned long integer</td>
</tr>
<tr>
<td>strxfrm, wcsxfrm</td>
<td>Transform string into collated form based on locale-specific information</td>
</tr>
<tr>
<td>_toascii</td>
<td>Convert character to ASCII code</td>
</tr>
<tr>
<td>tolower, towlower, _mbctolower</td>
<td>Test character and convert to lowercase if currently uppercase</td>
</tr>
<tr>
<td>_tolower</td>
<td>Convert character to lowercase unconditionally</td>
</tr>
<tr>
<td>toupper, towupper, _mbctoupper</td>
<td>Test character and convert to uppercase if currently lowercase</td>
</tr>
<tr>
<td>_toupper</td>
<td>Convert character to uppercase unconditionally</td>
</tr>
<tr>
<td>_ultoa, _ultow</td>
<td>Convert unsigned long to string</td>
</tr>
<tr>
<td>wcstombs</td>
<td>Convert sequence of wide characters to the corresponding sequence of multibyte characters</td>
</tr>
</tbody>
</table>
Directory-Control Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_chdir, _wchdir</td>
<td>Change current working directory</td>
</tr>
<tr>
<td>_chdrive</td>
<td>Change current drive</td>
</tr>
<tr>
<td>_getcwd, _wgetcwd</td>
<td>Get current working directory for default drive</td>
</tr>
<tr>
<td>_getdcwd, _wgetdcwd</td>
<td>Get current working directory for the specified drive</td>
</tr>
<tr>
<td>_getdiskfree</td>
<td>Populates a _diskfree_t structure with information about a disk drive.</td>
</tr>
<tr>
<td>_getdrive</td>
<td>Get current (default) drive</td>
</tr>
<tr>
<td>_getdrives</td>
<td>Returns a bitmask representing the currently available disk drives.</td>
</tr>
<tr>
<td>_mkdir, _wmkdir</td>
<td>Make new directory</td>
</tr>
<tr>
<td>_rmdir, _wrmdir</td>
<td>Remove directory</td>
</tr>
<tr>
<td>_searchenv,</td>
<td>Search for given file on specified paths</td>
</tr>
<tr>
<td>_wsearchenv</td>
<td></td>
</tr>
</tbody>
</table>

File-Handling Routines (File Descriptor)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_chsize</td>
<td>Change file size</td>
</tr>
<tr>
<td>_filelength</td>
<td>Get file length</td>
</tr>
<tr>
<td>_fstat, _fstat64, _fstat64</td>
<td>Get file-status information on the descriptor</td>
</tr>
<tr>
<td>_isatty</td>
<td>Check for character device</td>
</tr>
<tr>
<td>_locking</td>
<td>Lock areas of the file</td>
</tr>
<tr>
<td>_setmode</td>
<td>Set file-translation mode</td>
</tr>
</tbody>
</table>

File-Handling Routines (Path or Filename)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_access, _waccess</td>
<td>Check file-permission setting</td>
</tr>
<tr>
<td>_chmod, _wchmod</td>
<td>Change file-permission setting</td>
</tr>
<tr>
<td>_fullpath, _wfullpath</td>
<td>Expand a relative path to its absolute pathname</td>
</tr>
<tr>
<td>_get_osfhandle</td>
<td>Return operating-system file handle associated with existing stream</td>
</tr>
<tr>
<td>_makepath, _wmakepath</td>
<td>Merge path components into the single, full path</td>
</tr>
<tr>
<td></td>
<td>FILE pointer</td>
</tr>
<tr>
<td>Function</td>
<td>Use</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>_mktemp, _wmktemp</td>
<td>Create unique filename</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, _wremove</td>
<td>Delete file</td>
</tr>
<tr>
<td>rename, _wrename</td>
<td>Rename file</td>
</tr>
<tr>
<td>_splitpath, _wsplitpath</td>
<td>Parse path into components</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, _wunlink</td>
<td>Delete file</td>
</tr>
</tbody>
</table>

### File-Handling Routines (Open File)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>fopen</td>
<td>Opens a file and returns a pointer to the open file.</td>
</tr>
<tr>
<td>_fsopen</td>
<td>Open a stream with file sharing and returns a pointer to the open file.</td>
</tr>
<tr>
<td>_open</td>
<td>Opens a file and returns a file descriptor to the opened file.</td>
</tr>
<tr>
<td>_sopen</td>
<td>Open a file with file sharing and returns a file descriptor to the open file.</td>
</tr>
<tr>
<td>_fdopen</td>
<td>Associates a stream with a file that was previously opened for low-level I/O and returns a pointer to the open stream.</td>
</tr>
<tr>
<td>_fileno</td>
<td>Gets the file descriptor associated with a stream.</td>
</tr>
<tr>
<td>_open_osfhandle</td>
<td>Associates C run-time file descriptor with an existing operating-system file handle.</td>
</tr>
<tr>
<td>_pipe</td>
<td>Creates a pipe for reading and writing.</td>
</tr>
<tr>
<td>freopen</td>
<td>Reassign a file pointer.</td>
</tr>
</tbody>
</table>

### Low-Level I/O Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_close</td>
<td>Close file</td>
</tr>
<tr>
<td>_commit</td>
<td>Flush file to disk</td>
</tr>
<tr>
<td>_creat, _wcreat</td>
<td>Create file</td>
</tr>
<tr>
<td>_dup</td>
<td>Return next available file descriptor for a given file</td>
</tr>
<tr>
<td>_dup2</td>
<td>Create the second descriptor for a given file</td>
</tr>
<tr>
<td>_eof</td>
<td>Test for end of file</td>
</tr>
<tr>
<td>_lseek, _lseeki64</td>
<td>Reposition file pointer to a given location</td>
</tr>
<tr>
<td>_open, _wopen</td>
<td>Open file</td>
</tr>
<tr>
<td>_read</td>
<td>Read data from the file</td>
</tr>
<tr>
<td>_sopen, _wsopen</td>
<td>Open file for file sharing</td>
</tr>
<tr>
<td>Routine</td>
<td>Use</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>_tell, _telli64</td>
<td>Get current file-pointer position</td>
</tr>
<tr>
<td>_umask</td>
<td>Set file-permission mask</td>
</tr>
<tr>
<td>_write</td>
<td>Write data to file</td>
</tr>
</tbody>
</table>

## Stream I/O Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearerr</td>
<td>Clear error indicator for the stream</td>
</tr>
<tr>
<td>fclose</td>
<td>Close stream</td>
</tr>
<tr>
<td>_fcloseall</td>
<td>Close all open streams except stdin, stdout, and stderr</td>
</tr>
<tr>
<td>_fdopen, wfdopen</td>
<td>Associate stream with file descriptor of open file</td>
</tr>
<tr>
<td>feof</td>
<td>Test for end of file on stream</td>
</tr>
<tr>
<td>ferror</td>
<td>Test for error on stream</td>
</tr>
<tr>
<td>fflush</td>
<td>Flush stream to buffer or storage device</td>
</tr>
<tr>
<td>fgetc, fgetwc</td>
<td>Read a character from the stream (function versions of <code>getc</code> and <code>getwc</code>)</td>
</tr>
<tr>
<td>_fgetchar, _fgetwchar</td>
<td>Read a character from stdin (function versions of <code>getchar</code> and <code>getwchar</code>)</td>
</tr>
<tr>
<td>fgetpos</td>
<td>Get position indicator of stream</td>
</tr>
<tr>
<td>fgets, fgetws</td>
<td>Read string from stream</td>
</tr>
<tr>
<td>_fileno</td>
<td>Get file descriptor associated with the stream</td>
</tr>
<tr>
<td>_flushall</td>
<td>Flush all streams to buffer or storage device</td>
</tr>
<tr>
<td>fopen, _wfopen</td>
<td>Open stream</td>
</tr>
<tr>
<td>fprintf, fwprintf</td>
<td>Write formatted data to stream</td>
</tr>
<tr>
<td>fputc, fputwc</td>
<td>Write a character to a stream (function versions of <code>putc</code> and <code>putwc</code>)</td>
</tr>
<tr>
<td>_fputchar, _fputwchar</td>
<td>Write character to stdout (function versions of <code>putchar</code> and <code>putwchar</code>)</td>
</tr>
<tr>
<td>fputs, fputws</td>
<td>Write a string to the stream</td>
</tr>
<tr>
<td>fread</td>
<td>Read unformatted data from the stream</td>
</tr>
<tr>
<td>freopen, _wfreopen</td>
<td>Reassign FILE stream pointer to new file or device</td>
</tr>
<tr>
<td>fscanf, fwscanf</td>
<td>Read formatted data from the stream</td>
</tr>
<tr>
<td>fseek</td>
<td>Move file position to the given location</td>
</tr>
<tr>
<td>fsetpos</td>
<td>Set position indicator of stream</td>
</tr>
<tr>
<td>_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 a character from the stream (macro versions of <code>getc</code> and <code>fgetwc</code>)</td>
</tr>
<tr>
<td>getchar, getwchar</td>
<td>Read a character from stdin (macro versions of <code>getchar</code> and <code>fgetwchar</code>)</td>
</tr>
<tr>
<td>_getmaxstdio</td>
<td>Returns the number of simultaneously open files permitted at the stream I/O level.</td>
</tr>
<tr>
<td>gets, getws</td>
<td>Read line from stdin</td>
</tr>
</tbody>
</table>
String-Manipulation Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>_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 the multibyte-character string; dependent upon OEM code page</td>
</tr>
<tr>
<td>_mbsnbcat</td>
<td>Append, at most, the 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 the 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 the 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 the next character in the string</td>
</tr>
<tr>
<td>_mbssninc, _strnincc, _wcsnincc</td>
<td>Advance string pointer by n characters</td>
</tr>
<tr>
<td>_mbssspnp, _strspnp, _wcsspnp</td>
<td>Return pointer to the first character in the given string that is not in another given string</td>
</tr>
<tr>
<td>_mbstrlen</td>
<td>Get number of multibyte characters in the 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 the first occurrence of a specified character in the string</td>
</tr>
<tr>
<td>_strdup, _wcsdup, _mbsdup</td>
<td>Duplicate string</td>
</tr>
<tr>
<td>_stricmp, _wcsicmp, _mbsicmp</td>
<td>Compare two strings without regard to case</td>
</tr>
<tr>
<td>_strcoll, _wscoll, _stricoll, _wscoll, _stricoll, _wscoll, _stricoll, _wscoll</td>
<td>Compare two strings using current locale code page information (_stricoll, _wscoll, _stricoll, and _wscoll are case-insensitive)</td>
</tr>
<tr>
<td>strftime, wcsftime</td>
<td>Format date-and-time string</td>
</tr>
<tr>
<td>_strlwr, _wcslwr, _mbslwr</td>
<td>Convert string to lowercase</td>
</tr>
<tr>
<td>_strlwr, _wcslwr, _mbslwr</td>
<td>Convert string to lowercase</td>
</tr>
<tr>
<td>_strlwr, _wcslwr, _mbslwr</td>
<td>Convert string to lowercase</td>
</tr>
<tr>
<td>strncat, wcsncat, _mbsncat</td>
<td>Append characters of the string</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>strncmp, wcsncmp, _mbsncmp</td>
<td>Compare characters of two strings</td>
</tr>
<tr>
<td>strncpy, wcsncpy, _mbsncpy</td>
<td>Copy characters of one string to another</td>
</tr>
<tr>
<td>_strnicmp, _wcsnicmp, _mbsnicmp</td>
<td>Compare characters of two strings without regard to case</td>
</tr>
<tr>
<td>_strncpy, _wcsncpy, _mbsncpy</td>
<td>Copy characters of one string to another</td>
</tr>
<tr>
<td>strpbrk, wcspbrk, _mbspbrk</td>
<td>Find the first occurrence of character from one string in another string</td>
</tr>
<tr>
<td>strchr, wcschr, _mbschr</td>
<td>Find the last occurrence of a given character in the string</td>
</tr>
<tr>
<td>_strrev, _wcsrev, _mbsrev</td>
<td>Reverse string</td>
</tr>
<tr>
<td>_strset, _wcsset, _mbsset</td>
<td>Set all characters of the string to the specified character</td>
</tr>
<tr>
<td>strspn, wcsspnp, _mbsspnp</td>
<td>Find the first substring from one string in another string</td>
</tr>
<tr>
<td>strstr, wcsstr, _mbstrstr</td>
<td>Find the first occurrence of the specified string in another string</td>
</tr>
<tr>
<td>strtok, wcstok, _mbstok</td>
<td>Find the next token in the string</td>
</tr>
<tr>
<td>_strupr, _wcsupr, _mbsupr</td>
<td>Convert string to uppercase</td>
</tr>
<tr>
<td>strxfrm, wcstrxfrm</td>
<td>Transform string into collated form based on locale-specific information</td>
</tr>
<tr>
<td>vsprintf, _vstprintf</td>
<td>Write formatted output using a pointer to a list of arguments</td>
</tr>
</tbody>
</table>

### Time Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>asctime, _wasctime</td>
<td>Convert time from type <code>struct tm</code> to character string</td>
</tr>
<tr>
<td>clock</td>
<td>Return elapsed CPU time for the process</td>
</tr>
<tr>
<td>ctime, _ctime64, _wctime, _wctime64</td>
<td>Convert from type <code>time_t</code> or <code>_time64_t</code> to a character string</td>
</tr>
<tr>
<td>difftime</td>
<td>Compute the difference between two times</td>
</tr>
<tr>
<td>_ftime, _ftime64</td>
<td>Store current system time in the variable of type <code>struct _timeb</code> or type <code>struct _timeb64</code></td>
</tr>
<tr>
<td>_futime, _futime64</td>
<td>Set modification time on open file</td>
</tr>
<tr>
<td>gmttime, _gmtime64</td>
<td>Convert from type <code>time_t</code> to <code>struct tm</code> or from type <code>_time64_t</code> to <code>struct tm</code></td>
</tr>
<tr>
<td>localtime, _localtime64</td>
<td>Convert from type <code>time_t</code> to <code>struct tm</code> or from type <code>_time64_t</code> to <code>struct tm</code> with local correction</td>
</tr>
<tr>
<td>mktime, _mktime64</td>
<td>Convert time to calendar value</td>
</tr>
<tr>
<td>_strdate, _wstrdate</td>
<td>Return current system date as a string</td>
</tr>
<tr>
<td>strftime, wcsftime</td>
<td>Format date-and-time string for international use</td>
</tr>
<tr>
<td>_strftime, _wstrtime</td>
<td>Return current system time as a string</td>
</tr>
</tbody>
</table>
time, _time64
_gettime
_tzset
_wutime, _wutime64

Get 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 the specified file using either current time or time value stored

Memory and Other Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>Stops the program</td>
</tr>
<tr>
<td>assert</td>
<td>Stops the program if an expression isn’t true</td>
</tr>
<tr>
<td>atexit</td>
<td>Sets a function to be called when the program exits</td>
</tr>
<tr>
<td>bsearch</td>
<td>Perform a binary search</td>
</tr>
<tr>
<td>calloc</td>
<td>Allocates a two-dimensional chunk of memory</td>
</tr>
<tr>
<td>exit</td>
<td>Stop the program</td>
</tr>
<tr>
<td>free</td>
<td>Frees memory available for future allocation</td>
</tr>
<tr>
<td>getenv</td>
<td>Get environment information about a variable</td>
</tr>
<tr>
<td>longjmp</td>
<td>Start execution at a certain point in the program</td>
</tr>
<tr>
<td>qsort</td>
<td>Perform a quicksort</td>
</tr>
<tr>
<td>malloc</td>
<td>Allocates memory</td>
</tr>
<tr>
<td>raise</td>
<td>Send a signal to the program</td>
</tr>
<tr>
<td>rand</td>
<td>Returns a pseudorandom number</td>
</tr>
<tr>
<td>realloc</td>
<td>Changes the size of previously allocated memory</td>
</tr>
<tr>
<td>setjmp</td>
<td>Set execution to start at a certain point</td>
</tr>
<tr>
<td>signal</td>
<td>Register a function as a signal handler</td>
</tr>
<tr>
<td>srand</td>
<td>Initialize the random number generator</td>
</tr>
<tr>
<td>system</td>
<td>Perform a system call</td>
</tr>
<tr>
<td>va_arg</td>
<td>Use variable-length parameter lists</td>
</tr>
</tbody>
</table>
## Appendix B. RGB Color Values

For reference convenience, some RGB color values you may use are listed here by the color names with their RGB values.

<table>
<thead>
<tr>
<th>Color Name</th>
<th>RGB Values</th>
<th>Truncate Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>aliceblue</td>
<td>rgb(240, 248, 255)</td>
<td>rgb(255, 20, 147)</td>
</tr>
<tr>
<td>antiquewhite</td>
<td>rgb(250, 235, 215)</td>
<td>rgb(0, 191, 255)</td>
</tr>
<tr>
<td>aqua</td>
<td>rgb(0, 255, 255)</td>
<td>rgb(105, 105, 105)</td>
</tr>
<tr>
<td>aquamarine</td>
<td>rgb(127, 255, 212)</td>
<td>rgb(105, 105, 105)</td>
</tr>
<tr>
<td>azure</td>
<td>rgb(240, 255, 255)</td>
<td>rgb(30, 144, 255)</td>
</tr>
<tr>
<td>beige</td>
<td>rgb(245, 245, 220)</td>
<td>rgb(178, 34, 34)</td>
</tr>
<tr>
<td>bisque</td>
<td>rgb(255, 228, 196)</td>
<td>rgb(255, 250, 240)</td>
</tr>
<tr>
<td>black</td>
<td>rgb(0, 0, 0)</td>
<td>rgb(34, 139, 34)</td>
</tr>
<tr>
<td>blanchedalmond</td>
<td>rgb(255, 235, 205)</td>
<td>rgb(255, 0, 255)</td>
</tr>
<tr>
<td>blue</td>
<td>rgb(0, 0, 255)</td>
<td>rgb(220, 220, 220)</td>
</tr>
<tr>
<td>blueviolet</td>
<td>rgb(138, 43, 226)</td>
<td>rgb(248, 248, 255)</td>
</tr>
<tr>
<td>brown</td>
<td>rgb(165, 42, 42)</td>
<td>rgb(255, 215, 0)</td>
</tr>
<tr>
<td>burlywood</td>
<td>rgb(222, 184, 135)</td>
<td>rgb(218, 165, 32)</td>
</tr>
<tr>
<td>cadetblue</td>
<td>rgb(95, 158, 160)</td>
<td>rgb(128, 128, 128)</td>
</tr>
<tr>
<td>chartreuse</td>
<td>rgb(127, 255, 0)</td>
<td>rgb(128, 128, 128)</td>
</tr>
<tr>
<td>chocolate</td>
<td>rgb(210, 105, 30)</td>
<td>rgb(0, 128, 0)</td>
</tr>
<tr>
<td>coral</td>
<td>rgb(255, 127, 80)</td>
<td>rgb(173, 255, 47)</td>
</tr>
<tr>
<td>cornflowerblue</td>
<td>rgb(100, 149, 237)</td>
<td>rgb(240, 255, 240)</td>
</tr>
<tr>
<td>cornsilk</td>
<td>rgb(255, 248, 220)</td>
<td>rgb(255, 105, 180)</td>
</tr>
<tr>
<td>crimson</td>
<td>rgb(220, 20, 60)</td>
<td>rgb(205, 92, 92)</td>
</tr>
<tr>
<td>cyan</td>
<td>rgb(0, 255, 255)</td>
<td>rgb(75, 0, 130)</td>
</tr>
<tr>
<td>darkblue</td>
<td>rgb(0, 0, 139)</td>
<td>rgb(255, 255, 240)</td>
</tr>
<tr>
<td>darkcyan</td>
<td>rgb(0, 139, 139)</td>
<td>rgb(240, 230, 140)</td>
</tr>
<tr>
<td>darkgoldenrod</td>
<td>rgb(184, 134, 11)</td>
<td>rgb(230, 230, 250)</td>
</tr>
<tr>
<td>darkgray</td>
<td>rgb(169, 169, 169)</td>
<td>rgb(255, 240, 245)</td>
</tr>
<tr>
<td>darkgreen</td>
<td>rgb(0, 100, 0)</td>
<td>rgb(124, 252, 0)</td>
</tr>
<tr>
<td>darkgrey</td>
<td>rgb(169, 169, 169)</td>
<td>rgb(255, 250, 205)</td>
</tr>
<tr>
<td>darkkhaki</td>
<td>rgb(189, 183, 107)</td>
<td>rgb(173, 216, 230)</td>
</tr>
<tr>
<td>darkmagenta</td>
<td>rgb(139, 0, 139)</td>
<td>rgb(240, 128, 128)</td>
</tr>
<tr>
<td>darkolivegreen</td>
<td>rgb(85, 107, 47)</td>
<td>rgb(224, 255, 255)</td>
</tr>
<tr>
<td>darkorange</td>
<td>rgb(255, 140, 0)</td>
<td>rgb(250, 250, 210)</td>
</tr>
<tr>
<td>darkorchid</td>
<td>rgb(153, 50, 204)</td>
<td>rgb(211, 211, 211)</td>
</tr>
<tr>
<td>darkred</td>
<td>rgb(139, 0, 0)</td>
<td>rgb(144, 238, 144)</td>
</tr>
<tr>
<td>darksalmon</td>
<td>rgb(233, 150, 122)</td>
<td>rgb(211, 211, 211)</td>
</tr>
<tr>
<td>darkseagreen</td>
<td>rgb(143, 188, 143)</td>
<td>rgb(255, 182, 193)</td>
</tr>
<tr>
<td>darkslateblue</td>
<td>rgb(72, 61, 139)</td>
<td>rgb(255, 160, 122)</td>
</tr>
<tr>
<td>darkslategray</td>
<td>rgb(47, 79, 79)</td>
<td>rgb(32, 178, 170)</td>
</tr>
<tr>
<td>darkslategrey</td>
<td>rgb(47, 79, 79)</td>
<td>rgb(135, 206, 250)</td>
</tr>
<tr>
<td>darkturquoise</td>
<td>rgb(0, 206, 209)</td>
<td>rgb(119, 136, 153)</td>
</tr>
<tr>
<td>darkviolet</td>
<td>rgb(148, 0, 211)</td>
<td>rgb(119, 136, 153)</td>
</tr>
<tr>
<td>Color Name</td>
<td>RGB Values</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>lightsteelblue</td>
<td>rgb(176, 196, 222)</td>
<td></td>
</tr>
<tr>
<td>lightyellow</td>
<td>rgb(255, 255, 224)</td>
<td></td>
</tr>
<tr>
<td>lime</td>
<td>rgb(247, 247, 247)</td>
<td></td>
</tr>
<tr>
<td>seagreen</td>
<td>rgb(66, 139, 202)</td>
<td></td>
</tr>
<tr>
<td>limegreen</td>
<td>rgb(50, 205, 50)</td>
<td></td>
</tr>
<tr>
<td>sienna</td>
<td>rgb(160, 82, 45)</td>
<td></td>
</tr>
<tr>
<td>linen</td>
<td>rgb(250, 240, 230)</td>
<td></td>
</tr>
<tr>
<td>silver</td>
<td>rgb(192, 192, 192)</td>
<td></td>
</tr>
<tr>
<td>magenta</td>
<td>rgb(255, 0, 255)</td>
<td></td>
</tr>
<tr>
<td>skyblue</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>maroon</td>
<td>rgb(128, 0, 0)</td>
<td></td>
</tr>
<tr>
<td>slateblue</td>
<td>rgb(106, 90, 205)</td>
<td></td>
</tr>
<tr>
<td>mediumaquamarine</td>
<td>rgb(102, 205, 170)</td>
<td></td>
</tr>
<tr>
<td>slategray</td>
<td>rgb(112, 128, 144)</td>
<td></td>
</tr>
<tr>
<td>mediumblue</td>
<td>rgb(0, 0, 205)</td>
<td></td>
</tr>
<tr>
<td>slategrey</td>
<td>rgb(112, 128, 144)</td>
<td></td>
</tr>
<tr>
<td>mediumorchid</td>
<td>rgb(186, 85, 211)</td>
<td></td>
</tr>
<tr>
<td>snow</td>
<td>rgb(255, 250, 250)</td>
<td></td>
</tr>
<tr>
<td>mediumpurple</td>
<td>rgb(147, 112, 219)</td>
<td></td>
</tr>
<tr>
<td>springgreen</td>
<td>rgb(255, 255, 127)</td>
<td></td>
</tr>
<tr>
<td>mediumseagreen</td>
<td>rgb(60, 179, 113)</td>
<td></td>
</tr>
<tr>
<td>steelblue</td>
<td>rgb(70, 130, 180)</td>
<td></td>
</tr>
<tr>
<td>mediumslateblue</td>
<td>rgb(123, 104, 238)</td>
<td></td>
</tr>
<tr>
<td>tan</td>
<td>rgb(210, 180, 140)</td>
<td></td>
</tr>
<tr>
<td>mediumspringgreen</td>
<td>rgb(0, 250, 154)</td>
<td></td>
</tr>
<tr>
<td>teal</td>
<td>rgb(0, 128, 255)</td>
<td></td>
</tr>
<tr>
<td>mediumturquoise</td>
<td>rgb(72, 209, 204)</td>
<td></td>
</tr>
<tr>
<td>thistle</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>mediumvioletred</td>
<td>rgb(199, 21, 133)</td>
<td></td>
</tr>
<tr>
<td>tomato</td>
<td>rgb(255, 99, 71)</td>
<td></td>
</tr>
<tr>
<td>midnightblue</td>
<td>rgb(25, 25, 112)</td>
<td></td>
</tr>
<tr>
<td>turquoise</td>
<td>rgb(64, 224, 208)</td>
<td></td>
</tr>
<tr>
<td>mintcream</td>
<td>rgb(245, 255, 250)</td>
<td></td>
</tr>
<tr>
<td>violet</td>
<td>rgb(238, 130, 238)</td>
<td></td>
</tr>
<tr>
<td>mistyrose</td>
<td>rgb(255, 228, 225)</td>
<td></td>
</tr>
<tr>
<td>wheat</td>
<td>rgb(245, 222, 179)</td>
<td></td>
</tr>
<tr>
<td>moccasin</td>
<td>rgb(255, 228, 181)</td>
<td></td>
</tr>
<tr>
<td>white</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>navajowhite</td>
<td>rgb(255, 222, 173)</td>
<td></td>
</tr>
<tr>
<td>whitesmoke</td>
<td>rgb(245, 245, 245)</td>
<td></td>
</tr>
<tr>
<td>navy</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>olive</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>olive</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>olivedrab</td>
<td>rgb(107, 142, 35)</td>
<td></td>
</tr>
<tr>
<td>orange</td>
<td>rgb(255, 165, 0)</td>
<td></td>
</tr>
<tr>
<td>orangered</td>
<td>rgb(255, 69, 0)</td>
<td></td>
</tr>
<tr>
<td>orchid</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>palegoldenrod</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>palegreen</td>
<td>rgb(152, 251, 152)</td>
<td></td>
</tr>
<tr>
<td>paleturquoise</td>
<td>rgb(175, 238, 238)</td>
<td></td>
</tr>
<tr>
<td>palevioletred</td>
<td>rgb(219, 112, 147)</td>
<td></td>
</tr>
<tr>
<td>papayawhip</td>
<td>rgb(255, 239, 213)</td>
<td></td>
</tr>
<tr>
<td>peachpuff</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>peru</td>
<td>rgb(255, 255, 255)</td>
<td></td>
</tr>
<tr>
<td>pink</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>plum</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>powderblue</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>purple</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>red</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>rosybrown</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>royalblue</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
<tr>
<td>saddlebrown</td>
<td>rgb(128, 128, 0)</td>
<td></td>
</tr>
</tbody>
</table>