AddArcToPath

Description

Adds an arc to the current path.
The arc is drawn around a center point for a specified number of degrees either clockwise or
anti-clockwise.

Syntax

Dylib

```c
int DPLAddArcToPath(int InstanceID, double CenterX, double CenterY,
double TotalAngle);
```

Objective-C class

```objc
- (int)AddArcToPath:(double)CenterX :(double)CenterY :(double)TotalAngle;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CenterX</td>
<td>The horizontal co-ordinate of the center of the arc</td>
</tr>
<tr>
<td>CenterY</td>
<td>The vertical co-ordinate of the center of the arc</td>
</tr>
<tr>
<td>TotalAngle</td>
<td>The angular length of the arc. If this value is positive the arc will be drawn in a clockwise direction. A negative value will result in an arc drawn in an anti-clockwise direction. This value must be greater or less than 0. A value of 360 will result in a full circle being drawn.</td>
</tr>
</tbody>
</table>
**AddBoxToPath**

Vector graphics, Path definition and drawing

**Description**

Adds a rectangle to the current path.

**Syntax**

**Dylib**

```c
int DPLAddBoxToPath(int InstanceID, double Left, double Top, double Width, double Height);
```

**Objective-C class**

```objc
- (int)AddBoxToPath:(double)Left :(double)Top :(double)Width :(double)Height;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The horizontal co-ordinate of the left edge of the box</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The vertical co-ordinate of the top edge of the box</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the box</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the box</td>
</tr>
</tbody>
</table>
AddCJKFont

Text, Fonts

Description

Adds a CJK (Chinese Japanese Korean) font to the PDF document. At present, the only supported CJK fonts are the Japanese "HeiseiKakuGo-W5" font and the Korean "HYGoThic-Medium" font.

Syntax

Dylib

```c
int DPLAddCJKFont(int InstanceID, int CJKFontID);
```

Objective-C class

```c
- (int)AddCJKFont:(int)CJKFontID;
```

Parameters

<table>
<thead>
<tr>
<th>CJKFontID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HeiseiKakuGo-W5</td>
</tr>
<tr>
<td>2</td>
<td>HeiseiKakuGo-W5 (Bold)</td>
</tr>
<tr>
<td>3</td>
<td>HeiseiKakuGo-W5 (Bold Italic)</td>
</tr>
<tr>
<td>4</td>
<td>HeiseiKakuGo-W5 (Italic)</td>
</tr>
<tr>
<td>5</td>
<td>HYGoThic-Medium</td>
</tr>
<tr>
<td>6</td>
<td>HYGoThic-Medium (Bold)</td>
</tr>
<tr>
<td>7</td>
<td>HYGoThic-Medium (Bold Italic)</td>
</tr>
<tr>
<td>8</td>
<td>HYGoThic-Medium (Italic)</td>
</tr>
</tbody>
</table>
AddCurveToPath
Vector graphics, Path definition and drawing

Description

Adds a bezier curve to the current path.
The curve is drawn from the last point to the point defined by (EndX, EndY).
(CtAX, CtAY) and (CtBX, CtBY) define the two bezier control points.

Syntax

Dylib

```c
int DPLAddCurveToPath(int InstanceID, double CtAX, double CtAY,
                        double CtBX, double CtBY, double EndX, double EndY);
```

Objective-C class

```
- (int)AddCurveToPath:(double)CtAX :(double)CtAY :(double)CtBX
               :(double)CtBY :(double)EndX :(double)EndY;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtAX</td>
<td>The horizontal co-ordinate of the first control point</td>
</tr>
<tr>
<td>CtAY</td>
<td>The vertical co-ordinate of the first control point</td>
</tr>
<tr>
<td>CtBX</td>
<td>The horizontal co-ordinate of the second control point</td>
</tr>
<tr>
<td>CtBY</td>
<td>The vertical co-ordinate of the second control point</td>
</tr>
<tr>
<td>EndX</td>
<td>The horizontal co-ordinate of the end point of the bezier curve</td>
</tr>
<tr>
<td>EndY</td>
<td>The vertical co-ordinate of the end point of the bezier curve</td>
</tr>
</tbody>
</table>
AddEmbeddedFile

Document properties

Description

Embeds a file into the PDF but does not link it to any part of the document. The AddFileAttachment function can be used to make the embedded file available as an attachment in the PDF viewer. The PDF viewer must support this functionality (Adobe Reader 7 and later). This process can be done in one step using the EmbedFile function.

The AddLinkToEmbeddedFile function can be used to create a hotspot on a page that links to the embedded file.

Syntax

Dylib

int DPLAddEmbeddedFile(int InstanceID, wchar_t * FileName, wchar_t * MIMEType);

Objective-C class

- (int)AddEmbeddedFile:(NSString *)FileName :(NSString *)MIMEType;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The path and filename of the file to embed into the PDF.</td>
</tr>
<tr>
<td>MIMEType</td>
<td>The MIME type of the embedded file. For example &quot;image/jpeg&quot; for a JPEG image.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file could not be found or there was an error embedding the file into the PDF</td>
</tr>
<tr>
<td>Non-zero</td>
<td>An EmbeddedFileID that can be used with the AddFileAttachment function</td>
</tr>
</tbody>
</table>
AddFileAttachment

Document properties

Description

Makes an embedded file available as an attachment in the PDF viewer, if it supports this functionality. Adobe Reader 7 and later allow the user to work with file attachments. First use the AddEmbeddedFile function to embed the file into the PDF.

Syntax

Dylib

```c
int DPLAddFileAttachment(int InstanceID, wchar_t * Title,
                         int EmbeddedFileID);
```

Objective-C class

```objective-c
-(int)AddFileAttachment:(NSString *)Title :(int)EmbeddedFileID;
```

Parameters

<table>
<thead>
<tr>
<th>Title</th>
<th>The title of the attachment that should appear in the PDF viewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmbeddedFileID</td>
<td>The value returned from the AddEmbeddedFile function</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The EmbeddedFileID parameter was invalid, or the Title was blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The embedded file was made available as an attachment successfully</td>
</tr>
</tbody>
</table>
AddFormFieldChoiceSub

Form fields

Description

Similar to the AddFormFieldSub function but allows a choice field item's export value and display value to be set.

The function returns a temporary form field Index which can be used with the SetFormFieldBounds, SetFormFieldCheckStyle and other functions.

Syntax

Dylib

```c
int DPLAddFormFieldChoiceSub(int InstanceID, int Index, wchar_t * SubName, wchar_t * DisplayName);
```

Objective-C class

```objc
- (int)AddFormFieldChoiceSub:(int)Index :(NSString *)SubName :(NSString *)DisplayName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to work with. The first form field has an index of 1. The form field must be a choice field.</td>
</tr>
<tr>
<td>SubName</td>
<td>The export value of the new sub-field. The value of the form field could be set to this name using the SetFormFieldValue function.</td>
</tr>
<tr>
<td>DisplayName</td>
<td>The display name of the new sub-field.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The sub-field was not added. The specified form field may not have been a choice form field.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A temporary field Index</td>
</tr>
</tbody>
</table>


AddFormFieldSub

Form fields

Description

Adds a sub-field to the specified radio-button or choice form field. The function returns a temporary form field Index which can be used with the SetFormFieldBounds, SetFormFieldCheckStyle and other functions.

To set a choice item's export value and display value use the AddFormFieldChoiceSub function.

Syntax

Dylib

int DPLAddFormFieldSub(int InstanceID, int Index, wchar_t * SubName);

Objective-C class

- (int)AddFormFieldSub:(int)Index :(NSString *)SubName;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to work with. The first form field has an index of 1.</td>
</tr>
<tr>
<td>SubName</td>
<td>The name of the new sub-field. The value of the form field could be set to this name using the SetFormFieldValue function.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The sub-field was not added. The specified form field may not have been a radio-button or choice form field.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A temporary field Index</td>
</tr>
</tbody>
</table>
AddFormFont
Fonts, Form fields

Description

Adds a font to the form.
The font must have been added using one of the Add*Font functions.

Syntax

Dylib

```c
int DPLAddFormFont(int InstanceID, int FontID);
```

Objective-C class

```objective-c
-(int)AddFormFont:(int)FontID;
```

Parameters

| FontID | The FontID returned by one of the Add*Font functions |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>Invalid FontID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>The font was added successfully, the value returned is the number of fonts available for use by form fields</td>
</tr>
</tbody>
</table>
AddFreeTextAnnotation

**Text, Annotations and hotspot links**

**Description**

Adds a free text annotation to the selected page.

**Syntax**

**Dylib**

```c
int DPLAddFreeTextAnnotation(int InstanceID, double Left, double Top,
                               double Width, double Height, wchar_t * Text, int Angle,
                               int Options);
```

**Objective-C class**

```objc
- (int)AddFreeTextAnnotation:(double)Left :(double)Top :(double)Width
  :(double)Height :(NSString *)Text :(int)Angle :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The horizontal coordinate of the left edge of the annotation rectangle</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The vertical coordinate of the left edge of the annotation rectangle</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the annotation rectangle</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the annotation rectangle</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text content of the annotation</td>
</tr>
<tr>
<td><strong>Angle</strong></td>
<td>The angle of the drawn text. Can be 0, 90, 180 or 270.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Default=0</td>
</tr>
</tbody>
</table>
**AddGlobalJavaScript**

**Description**

Adds JavaScript to a global location in the document. For example, this allows functions to be defined which can then be called from JavaScript attached to events.

**Syntax**

**Dylib**

```c
int DPLAddGlobalJavaScript(int InstanceID, wchar_t * PackageManager, wchar_t * JavaScript);
```

**Objective-C class**

```objective-c
-(int)AddGlobalJavaScript:(NSString *)PackageName :(NSString *)JavaScript;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PackageManager</code></td>
<td>The name to store the JavaScript under. If any JavaScript is already stored under this name it will be removed and the new JavaScript will be stored in its place.</td>
</tr>
<tr>
<td><code>JavaScript</code></td>
<td>The JavaScript to store globally under the specified package name.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The <code>PackageManager</code> was empty</td>
</tr>
<tr>
<td>1</td>
<td>The JavaScript was stored successfully</td>
</tr>
</tbody>
</table>
AddImageFromFile

Image handling

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Adds an image from a file to the selected document.

Once an image has been added to the document it can be drawn on any page multiple times without further increasing the size of the PDF file.

Supported image file types are: BMP, TIFF, JPEG, PNG, GIF, WMF and EMF.

For BMP and TIFF images, the CompressImages function can called before calling this function to compress the image data. Other image types are automatically compressed.

Syntax

Dylib

int DPLAddImageFromFile(int InstanceID, wchar_t * FileName, int Options);

Objective-C class

- (int)AddImageFromFile:(NSString *)FileName :(int)Options;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The file name of the image to add.</td>
</tr>
<tr>
<td>Options</td>
<td>For multi-page TIFF images this parameter specifies the page number to load. For PNG images: 0 = Load the image as usual 1 = Load the alpha channel as a greyscale image 2 = Load the image and alpha channel (limit alpha to 8-bit) 3 = Load the image (limit image 8-bit/channel) 4 = Load the alpha channel (limit to 8-bit/channel) 5 = Load the image with alpha channel (limit both to 8-bit/channel) 6 = Load the image and alpha channel 7 = Load the image and ICC color profile For other image types this parameter should be set to 0.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image could not be added. Either it could not be found or it is in an unsupported format.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The image was added successfully. The ImageID is returned which can be passed to functions like SelectImage and DrawImage.</td>
</tr>
</tbody>
</table>
AddImageFromFileOffset

Image handling

Description

Adds an image from a part of a file to the selected document.

For example, if many images have been concatenated into one file this function will allow the individual images to be extracted and added to the document.

Once an image has been added to the document it can be drawn on any page multiple times without further increasing the size of the PDF file.

Supported image file types are: BMP, TIFF, JPEG, PNG, GIF, WMF and EMF.

For BMP and TIFF images, the CompressImages function can called before calling this function to compress the image data. Other image types are automatically compressed.

Syntax

Dylib

```c
int DPLAddImageFromFileOffset(int InstanceID, wchar_t * FileName,
   int Offset, int DataLength, int Options);
```

Objective-C class

```c
- (int)AddImageFromFileOffset:(NSString *)FileName :(int)Offset
   :(int)DataLength :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The name of the file containing the images.</td>
</tr>
<tr>
<td>Offset</td>
<td>The offset into the file where the required image starts. The first byte in the file has an offset of 0.</td>
</tr>
<tr>
<td>DataLength</td>
<td>The length of the image data in bytes</td>
</tr>
<tr>
<td>Options</td>
<td>For multi-page TIFF images this parameter specifies the page number to load. For PNG images: 0 = Load the image as usual 1 = Load the alpha channel as a greyscale image 2 = Load the image and alpha channel (limit alpha to 8-bit) 3 = Load the image (limit image 8-bit/channel) 4 = Load the alpha channel (limit to 8-bit/channel) 5 = Load the image with alpha channel (limit both to 8-bit/channel) 6 = Load the image and alpha channel 7 = Load the image and ICC color profile For other image types this parameter should be set to 0.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image could not be read from the file. This could indicate invalid image data or the file could not be found.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The image was read from the file and successfully added to the document. The value returned is the ID of the image which can be used with the image drawing functions such as DrawImage.</td>
</tr>
</tbody>
</table>
AddImageFromString

Image handling

Description

Adds an image from memory to the selected document. Once an image has been added to the document it can be drawn on any page multiple times without further increasing the size of the PDF file.

Supported image file types are: BMP, TIFF, JPEG, PNG, GIF, WMF and EMF.

For BMP and TIFF images, the **CompressImages** function can called before calling this function to compress the image data. Other image types are automatically compressed.

Syntax

**Dylib**

```c
int DPLAddImageFromString(int InstanceID, char * Source, int Options);
```

**Objective-C class**

```objc
- (int)AddImageFromString:(NSData *)Source :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Source</th>
<th>A string containing the image data. In the ActiveX version of the library this string must contain 16-bit characters, only the lower 8-bits of each character will be used.</th>
</tr>
</thead>
</table>
| Options | For multi-page TIFF images this parameter specifies the page number to load. For PNG images:  
0 = Load the image as usual  
1 = Load the alpha channel as a greyscale image  
2 = Load the image and alpha channel (limit alpha to 8-bit)  
3 = Load the image (limit image 8-bit/channel)  
4 = Load the alpha channel (limit to 8-bit/channel)  
5 = Load the image with alpha channel (limit both to 8-bit/channel)  
6 = Load the image and alpha channel  
7 = Load the image and ICC color profile  
For other image types this parameter should be set to 0. |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The image data was invalid or the image was in an unsupported format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The image was added successfully. The value returned is the ImageID which can be used with functions like <strong>SelectImage</strong> and <strong>DrawImage</strong>.</td>
</tr>
</tbody>
</table>
AddLGIDictToPage

Page properties, Measurement and coordinate units

Description

Adds a new LGIDict object to the selected page. This is used with the GeoPDF system as defined in Open Geospatial Consortium Inc.'s OGC 08-139r2 specification. More than one dictionary can be added to the page.

Syntax

**Dylib**

```c
int DPLAddLGIDictToPage(int InstanceID, wchar_t * DictContent);
```

**Objective-C class**

```objective-c
- (int)AddLGIDictToPage:(NSString *)DictContent;
```

Parameters

- **DictContent** The LGIDict dictionary content to add to the page.

Return values

- **0** The LGI dictionary could not be added to the page. Check that the dictionary content string is a valid PDF dictionary.
- **1** The LGI dictionary was added successfully.
AddLineToPath

Vector graphics, Path definition and drawing

Description

Add a line to the current path.
The line is drawn from the last point to the point defined by (EndX, EndY).

Syntax

**Dylib**

```c
int DPLAddLineToPath(int InstanceID, double EndX, double EndY);
```

**Objective-C class**

```c
- (int)AddLineToPath:(double)EndX :(double)EndY;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndX</td>
<td>The horizontal co-ordinate of the end point of the line to add to the path</td>
</tr>
<tr>
<td>EndY</td>
<td>The vertical co-ordinate of the end point of the line to add to the path</td>
</tr>
</tbody>
</table>
AddLinkToDestination
Annotations and hotspot links, Page properties

Description

Adds a clickable hotspot rectangle to the selected page which links to another page in the same document. The target page, position and zoom level are specified by a destination object which can be created with the NewDestination function.

Use the SetAnnotBorderColor function to change the color of the hotspot border.

Syntax

Dylib

```
int DPLAddLinkToDestination(int InstanceID, double Left, double Top,
                              double Width, double Height, int DestID, int Options);
```

Objective-C class

```
- (int)AddLinkToDestination:(double)Left :(double)Top :(double)Width
                          :(double)Height :(int)DestID :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The left edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The top edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the hotspot rectangle</td>
</tr>
<tr>
<td>DestID</td>
<td>The DestID of a destination object</td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the appearance of the link:</td>
</tr>
<tr>
<td></td>
<td>0 = No border</td>
</tr>
<tr>
<td></td>
<td>1 = Draw a border</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The DestID property was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The link annotation was created successfully</td>
</tr>
</tbody>
</table>
AddLinkToEmbeddedFile

Document properties, Annotations and hotspot links

Description

Adds a clickable hotspot rectangle to the selected page which links to an embedded file.
Files can be embedded into the PDF using the AddEmbeddedFile function.
The function definition was changed in version 9.11 to provide separate parameters for the title/contents and transparency.

Syntax

**Dylib**

```c
int DPLAddLinkToEmbeddedFile(int InstanceID, double Left, double Top,
                               double Width, double Height, int EmbeddedFileID,
                               wchar_t * Title, wchar_t * Contents, int IconType,
                               int Transparency);
```

**Objective-C class**

```objective-c
- (int)AddLinkToEmbeddedFile:(double)Left :(double)Top :(double)Width
                          :(double)Height :(int)EmbeddedFileID :(NSString *)Title
                          :(NSString *)Contents :(int)IconType :(int)Transpareny;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top of the hotspot rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the hotspot rectangle</td>
</tr>
<tr>
<td>EmbeddedFileID</td>
<td>The value returned from the AddEmbeddedFile function</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the attachment that should appear in the PDF viewer</td>
</tr>
<tr>
<td>Contents</td>
<td>The text to use for the contents of the popup</td>
</tr>
</tbody>
</table>
| IconType       | 0 = Standard icon (PushPin)  
|                | 1 = 28x28 disk image  
|                | 2 = No icon  
|                | 3 = Graph  
|                | 4 = Paperclip  
|                | 5 = Tag  
|                | 6 = Solid white rectangle                                                  |

| Transpareny    | The transparency percentage to apply ranging from 0 to 100. A value of 0 indicates 0% transparency which is fully opaque (no transparency). A value of 100 indicates 100% transparency which would make the icon invisible. |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The EmbeddedFileID parameter was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The link was created successfully</td>
</tr>
</tbody>
</table>
AddLinkToFile

Annotations and hotspot links

Description

Adds a clickable hotspot rectangle to the selected page which links to a specific page and position in another PDF document.

Use the `SetAnnotBorderColor` function to change the color of the hotspot border.

Syntax

Dylib

```c
int DPLAddLinkToFile(int InstanceID, double Left, double Top, double Width, double Height, wchar_t * FileName, int Page, double Position, int NewWindow, int Options);
```

Objective-C class

```objc
- (int)AddLinkToFile:(double)Left :(double)Top :(double)Width
  :(double)Height :(NSString *)FileName :(int)Page
  :(double)Position :(int)NewWindow :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the hotspot rectangle</td>
</tr>
<tr>
<td>FileName</td>
<td>The path and file name of the PDF document to link to.</td>
</tr>
<tr>
<td>Page</td>
<td>The page in the destination document to link to.</td>
</tr>
<tr>
<td>Position</td>
<td>The vertical co-ordinate on the destination page to link to.</td>
</tr>
</tbody>
</table>
| NewWindow | 0 = Close the current document and then open the new document  
          1 = Open the current document in a new window |
| Options   | Specifies the appearance of the link:  
          0 = No border  
          1 = Draw a border |
AddLinkToFileEx
Annotations and hotspot links

Description

Adds a clickable hotspot rectangle to the selected page which links to a specific page and position in another PDF document. Use the SetAnnotBorderColor function to change the color of the hotspot border.

The link to the target document is only via the file name. This means the page dimensions of the target document are not known so the DestLeft, DestTop, DestRight and DestBottom parameters are always specified in points measured from the bottom left corner of the destination page's MediaBox.

Syntax

Dylib

int DPLAddLinkToFileEx(int InstanceID, double Left, double Top, double Width, double Height, wchar_t * FileName, int DestPage, int NewWindow, int Options, int Zoom, int DestType, double DestLeft, double DestTop, double DestRight, double DestBottom);

Objective-C class

- (int)AddLinkToFileEx:(double)Left :(double)Top :(double)Width :(double)Height :(NSString *)FileName :(int)DestPage :(int)NewWindow :(int)Options :(int)Zoom :(int)DestType :(double)DestLeft :(double)DestTop :(double)DestRight :(double)DestBottom;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the hotspot rectangle</td>
</tr>
<tr>
<td>FileName</td>
<td>The path and file name of the PDF document to link to.</td>
</tr>
<tr>
<td>DestPage</td>
<td>The page in the destination document to link to</td>
</tr>
<tr>
<td>NewWindow</td>
<td>0 = Close the current document and then open the new document</td>
</tr>
<tr>
<td></td>
<td>1 = Open the current document in a new window</td>
</tr>
<tr>
<td>Options</td>
<td>Specifies the appearance of the link:</td>
</tr>
<tr>
<td></td>
<td>0 = No border</td>
</tr>
<tr>
<td></td>
<td>1 = Draw a border</td>
</tr>
<tr>
<td>Zoom</td>
<td>The zoom percentage to use for the destination object, valid values from 0 to 6400. Only used for DestType = 1, should be set to 0 for other DestTypes.</td>
</tr>
<tr>
<td>DestType</td>
<td>1 = &quot;XYZ&quot; - the target page is positioned at the point specified by the Left and Top parameters. The Zoom parameter specifies the zoom percentage. 2 = &quot;Fit&quot; - the entire page is zoomed to fit the window. None of the other parameters are used and should be set to zero. 3 = &quot;FitH&quot; - the page is zoomed so that the entire width of the page is visible. The height of the page may be greater or less than the height of the window. The page is positioned at the vertical position specified by the Top parameter. 4 = &quot;FitV&quot; - the page is zoomed so that the entire height of the page can be seen. The width of the page may be greater or less than the width of the window. The page is positioned at the horizontal position specified by the Left parameter. 5 = &quot;FitR&quot; - the page is zoomed so that a certain rectangle on the page is visible. The Left, Top, Right and Bottom parameters define the rectangular area on the page. 6 = &quot;FitB&quot; - the page is zoomed so that its bounding box is visible. 7 = &quot;FitBH&quot; - the page is positioned vertically at the position specified by the Top parameter. The page is zoomed so that the entire width of the page's bounding box is visible. 8 = &quot;FitBV&quot; - the page is positioned at the horizontal position specified by the Left parameter. The page is zoomed just enough to fit the entire height of the bounding box into the window.</td>
</tr>
<tr>
<td>DestLeft</td>
<td>The horizontal position used by DestType = 1, 4, 5 and 8</td>
</tr>
<tr>
<td>DestTop</td>
<td>The vertical position used by DestType = 1, 3, 5 and 7</td>
</tr>
<tr>
<td>DestRight</td>
<td>The horizontal position of the righthand edge of the rectangle. Used by DestType = 5</td>
</tr>
<tr>
<td>DestBottom</td>
<td>The horizontal position of the bottom of the rectangle. Used by DestType = 5</td>
</tr>
</tbody>
</table>
AddLinkToJavaScript

JavaScript, Annotations and hotspot links

Description

Adds a clickable hotspot rectangle to the selected page which links to a JavaScript action. Use the SetAnnotBorderColor function to change the color of the hotspot border.

Syntax

Dylib

```c
int DPLAddLinkToJavaScript(int InstanceID, double Left, double Top, double Width, double Height, wchar_t * JavaScript, int Options);
```

Objective-C class

```objective-c
- (int)AddLinkToJavaScript:(double)Left :(double)Top :(double)Width :(double)Height :(NSString *)JavaScript :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the hotspot rectangle</td>
</tr>
<tr>
<td>JavaScript</td>
<td>The JavaScript to execute.</td>
</tr>
</tbody>
</table>
| Options     | Specifies the appearance of the link:  
0 = No border  
1 = Draw a border |
**AddLinkToLocalFile**

**Annotations and hotspot links**

**Description**

Adds a clickable hotspot rectangle to the selected page which links to a local file. The file doesn't have to exist when the PDF is created but should exist when the PDF is viewed for the link to work.

Use the **SetAnnotBorderColor** function to change the color of the hotspot border.

**Syntax**

**Dylib**

```c
int DPLAddLinkToLocalFile(int InstanceID, double Left, double Top, double Width, double Height, wchar_t * FileName, int Options);
```

**Objective-C class**

```c
- (int)AddLinkToLocalFile:(double)Left :(double)Top :(double)Width :(double)Height :(NSString *)FileName :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The left edge of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The top edge of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>FileName</strong></td>
<td>The relative or absolute path to the local file.</td>
</tr>
</tbody>
</table>
| **Options** | Specifies the appearance of the link and whether the target is opened in a new window or the same window:
  0 = No border, same window
  1 = Draw a border, same window
  2 = No border, new window
  3 = Draw a border, new window |
AddLinkToPage
Annotations and hotspot links, Page properties

Description

Adds a clickable hotspot rectangle to the selected page which links to another page in the same document.

Use the SetAnnotBorderColor function to change the color of the hotspot border.

Syntax

Dylib

```c
int DPLAddLinkToPage(int InstanceID, double Left, double Top,
                      double Width, double Height, int Page, double Position,
                      int Options);
```

Objective-C class

```c
- (int)AddLinkToPage:(double)Left :(double)Top :(double)Width
               :(double)Height :(int)Page :(double)Position :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The left edge of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The top edge of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the hotspot rectangle</td>
</tr>
<tr>
<td><strong>Page</strong></td>
<td>The destination page number to link to</td>
</tr>
<tr>
<td><strong>Position</strong></td>
<td>The vertical position on the destination page to link to</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Specifies the appearance of the link:</td>
</tr>
<tr>
<td></td>
<td>0 = No border</td>
</tr>
<tr>
<td></td>
<td>1 = Draw a border</td>
</tr>
</tbody>
</table>
AddLinkToWeb

Annotations and hotspot links

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Adds a clickable hotspot rectangle to the selected page which links to a URL on the internet. This can also be used to link to an e-mail address.

Use the *SetAnnotBorderColor* function to change the color of the hotspot border.

**Syntax**

**Dylib**

```c
int DPLAddLinkToWeb(int InstanceID, double Left, double Top, double Width,
                     double Height, wchar_t * Link, int Options);
```

**Objective-C class**

```objective-c
- (int)AddLinkToWeb:(double)Left :(double)Top :(double)Width
            :(double)Height :(NSString *)Link :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The left edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The top edge of the hotspot rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the hotspot rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the hotspot rectangle</td>
</tr>
</tbody>
</table>
| Link      | The URL to link to. Some examples: "http://www.example.com/"
          | "mailto:info@example.com" |
| Options   | Specifies the appearance of the link: 0 = No border 1 = Draw a border |
AddNoteAnnotation

Annotations and hotspot links

Description

Adds a note annotation to the selected page. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLAddNoteAnnotation(int InstanceID, double Left, double Top,
int AnnotType, double PopupLeft, double PopupTop,
double PopupWidth, double PopupHeight, wchar_t * Title,
wchar_t * Contents, double Red, double Green, double Blue,
int Open);
```

Objective-C class

```objc
- (int)AddNoteAnnotation:(double)Left :(double)Top :(int)AnnotType
 :(double)PopupLeft :(double)PopupTop :(double)PopupWidth
 :(double)PopupHeight :(NSString *)Title :(NSString *)Contents
 :(double)Red :(double)Green :(double)Blue :(int)Open;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the anchor for the annotation</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the anchor for the annotation</td>
</tr>
<tr>
<td>AnnotType</td>
<td>The annotation type: 0 = Note, 1 = Comment, 2 = Help, 3 = Insert, 4 = Key, 5 = New paragraph, 6 = Paragraph. Add 100 to any of the above values to suppress the date shown in the popup annotation's title</td>
</tr>
<tr>
<td>PopupLeft</td>
<td>The horizontal co-ordinate of the left edge of the popup window</td>
</tr>
<tr>
<td>PopupTop</td>
<td>The vertical co-ordinate of the left edge of the popup window</td>
</tr>
<tr>
<td>PopupWidth</td>
<td>The width of the popup window</td>
</tr>
<tr>
<td>PopupHeight</td>
<td>The height of the popup window</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the annotation</td>
</tr>
<tr>
<td>Contents</td>
<td>The body of the popup annotation</td>
</tr>
<tr>
<td>Red</td>
<td>The red component of the color of the annotation</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color of the annotation</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color of the annotation</td>
</tr>
<tr>
<td>Open</td>
<td>Specifies whether to show the annotation when the document is opened: 0 = hide, 1 = show</td>
</tr>
</tbody>
</table>
AddOpenTypeFontFromFile

Text, Fonts

Description

This function is identical to AddTrueTypeFontFromFile. Both functions allow a TrueType, OpenType/TrueType or OpenType/CFF font to be added from a file. This version of the function provides an Options parameter which may be expanded in future to support advanced OpenType features.

Syntax

Dylib

    int DPLAddOpenTypeFontFromFile(int InstanceID, wchar_t * FileName,
        int Options);

Objective-C class

    - (int)AddOpenTypeFontFromFile:(NSString *)FileName :(int)Options;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The font file name.</td>
</tr>
<tr>
<td>Options</td>
<td>Should be set to 0.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The font could not be embedded</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the font that was successfully added. This ID can be used with the SelectFont function to select the font</td>
</tr>
</tbody>
</table>
AddPageLabels

**Description**

Adds a range of page labels to the selected document. A range starting from page 1 must be present in the document for the page labels to display correctly.

**Syntax**

**Dylib**

```c
int DPLAddPageLabels(int InstanceID, int Start, int Style, int Offset, wchar_t * Prefix);
```

**Objective-C class**

```objective-c
- (int)AddPageLabels:(int)Start :(int)Style :(int)Offset :(NSString *)Prefix;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start</strong></td>
<td>The starting page for the range of page labels</td>
</tr>
</tbody>
</table>
| **Style** | 0 = No numbers  
| | 1 = Decimal arabic numerals  
| | 2 = Uppercase roman numerals  
| | 3 = Lowercase roman numerals  
| | 4 = Uppercase letters (A to Z for first 26 pages, AA to ZZ for next 26, etc.)  
| | 5 = Lowercase letters (a to z for first 26 pages, aa to zz for next 26, etc.) |
| **Offset** | The value of the numeric portion for the first page label in the range. Subsequent values will be numbered sequentially from this value, which must be greater than or equal to 1. |
| **Prefix** | The prefix for the page labels in this range. |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Style parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>The page label range was added successfully</td>
</tr>
</tbody>
</table>
AddSVGAnnotationFromFile

Vector graphics, Image handling, Annotations and hotspot links, Page layout

Description
Add an SVG file as an annotation to the current page. This is only supported if the PDF is viewed using Adobe Acrobat 6 or Adobe Reader 6. Earlier and later versions will not show the SVG annotation.

Syntax

Dylib

```c
int DPLAddSVGAnnotationFromFile(int InstanceID, double Left, double Top,
                                double Width, double Height, wchar_t * FileName, int Options);
```

Objective-C class

```objective-c
- (int)AddSVGAnnotationFromFile:(double)Left :(double)Top :(double)Width
                          :(double)Height :NSString *)FileName :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the annotation rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the annotation rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the annotation rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the annotation rectangle</td>
</tr>
<tr>
<td>FileName</td>
<td>The path and name of the file containing the SVG image.</td>
</tr>
<tr>
<td>Options</td>
<td>This parameter is ignored and should be set to 0</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The SVG file could not be opened</td>
</tr>
<tr>
<td>1</td>
<td>The SVG annotation was added successfully</td>
</tr>
</tbody>
</table>
AddSWFAnnotationFromFile
Vector graphics, Image handling, Annotations and hotspot links, Page layout

Description
Adds a Flash SWF file as an annotation to the current page.

Syntax
Dylib

```c
int DPLAddSWFAnnotationFromFile(int InstanceID, double Left, double Top,
                              double Width, double Height, wchar_t * FileName,
                              wchar_t * Title, int Options);
```

Objective-C class

```c
-(int)AddSWFAnnotationFromFile:(double)Left :(double)Top :(double)Width
 :(double)Height :(NSString *)FileName :(NSString *)Title
 :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the annotation rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the annotation rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the annotation rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the annotation rectangle</td>
</tr>
<tr>
<td>FileName</td>
<td>The path and name of the SWF file</td>
</tr>
<tr>
<td>Title</td>
<td>The annotation title</td>
</tr>
<tr>
<td>Options</td>
<td>Annotation event to activate SWF: 0 = Page visible 1 = Mouse enter 2 = Mouse button click</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified file could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The SWF was successfully added as an annotation</td>
</tr>
</tbody>
</table>
AddSeparationColor

Vector graphics, Color

Description

Adds a separation color to the document.
A separation color has a name and an equivalent color in the CMYK color space. If the document is viewed, the CMYK color will be used. If the document is printed to an image setter, a separation with the specified name will be generated.
The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLAddSeparationColor(int InstanceID, wchar_t * ColorName, double C, double M, double Y, double K, int Options);
```

Objective-C class

```objective-c
- (int)AddSeparationColor:(NSString *)ColorName :(double)C :(double)M :(double)Y :(double)K :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorName</td>
<td>The name of the separation color, for example &quot;PANTONE 403 EC&quot;. This can be any name you want, but is usually set to the name of a specific spot color that your printing press will know what to do with.</td>
</tr>
<tr>
<td>C</td>
<td>The cyan component of the color equivalent to the spot color</td>
</tr>
<tr>
<td>M</td>
<td>The magenta component of the color equivalent to the spot color</td>
</tr>
<tr>
<td>Y</td>
<td>The yellow component of the color equivalent to the spot color</td>
</tr>
<tr>
<td>K</td>
<td>The black component of the color equivalent to the spot color</td>
</tr>
<tr>
<td>Options</td>
<td>This parameter is ignored and should be set to 0</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The separation color could not be added. The color name may already have been used.</td>
</tr>
<tr>
<td>1</td>
<td>The separation color was added successfully</td>
</tr>
</tbody>
</table>
AddStampAnnotation

Annotations and hotspot links

Description

Adds a stamp annotation to the selected page. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLAddStampAnnotation(int InstanceID, double Left, double Top, double Width, double Height, int StampType, wchar_t * Title, wchar_t * Contents, double Red, double Green, double Blue, int Options);
```

Objective-C class

```objective-c
-(int)AddStampAnnotation:(double)Left :(double)Top :(double)Width
:(double)Height :(int)StampType :(NSString *)Title
:(NSString *)Contents :(double)Red :(double)Green
:(double)Blue :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal coordinate of the left edge of the stamp annotation</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical coordinate of the top edge of the stamp annotation</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the annotation</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the annotation</td>
</tr>
<tr>
<td>StampType</td>
<td>0 = Approved 1 = Experimental 2 = NotApproved 3 = AsIs 4 = Expired 5 = NotForPublicRelease 6 = Confidential 7 = Final 8 = Sold 9 = Departmental 10 = ForComment 11 = TopSecret 12 = Draft 13 = ForPublicRelease</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the popup annotation</td>
</tr>
<tr>
<td>Contents</td>
<td>The contents of the popup annotation</td>
</tr>
<tr>
<td>Red</td>
<td>The red component of the popup annotation's background color</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the popup annotation's background color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the popup annotation's background color</td>
</tr>
<tr>
<td>Options</td>
<td>Reserved for future use. Should always be set to 0.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The stamp annotation could not be added to the page</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
AddStandardFont

Text, Fonts

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Adds a standard font to the document. These standard fonts will always be available on all PDF viewers.

Syntax

**Dylib**

```c
int DPLAddStandardFont(int InstanceID, int StandardFontID);
```

**Objective-C class**

```objc
-(int)AddStandardFont:(int)StandardFontID;
```

Parameters

<table>
<thead>
<tr>
<th>StandardFontID</th>
<th>The ID of the font to add:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Courier</td>
</tr>
<tr>
<td>1</td>
<td>CourierBold</td>
</tr>
<tr>
<td>2</td>
<td>CourierBoldOblique</td>
</tr>
<tr>
<td>3</td>
<td>CourierOblique</td>
</tr>
<tr>
<td>4</td>
<td>Helvetica</td>
</tr>
<tr>
<td>5</td>
<td>HelveticaBold</td>
</tr>
<tr>
<td>6</td>
<td>HelveticaBoldOblique</td>
</tr>
<tr>
<td>7</td>
<td>HelveticaOblique</td>
</tr>
<tr>
<td>8</td>
<td>TimesRoman</td>
</tr>
<tr>
<td>9</td>
<td>TimesBold</td>
</tr>
<tr>
<td>10</td>
<td>TimesItalic</td>
</tr>
<tr>
<td>11</td>
<td>TimesBoldItalic</td>
</tr>
<tr>
<td>12</td>
<td>Symbol</td>
</tr>
<tr>
<td>13</td>
<td>ZapfDingbats</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The font could not be added</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>The ID of the font that was successfully added</td>
</tr>
</tbody>
</table>
AddSubsettedFont

Text, Fonts

Description

This function is used to embed a "subset" of a font. This means that only the font information for specified characters is embedded, reducing the size of the document. This function also allows any Unicode character to be embedded which means that characters from Chinese, Japanese, Korean and other languages can be used.

The newer AddTrueTypeSubsettedFont function provides more advanced font subsetting functionality.

Syntax

Dylib

```c
int DPLAddSubsettedFont(int InstanceID, wchar_t * FontName,
                        int CharSetIndex, wchar_t * SubsetChars);
```

Objective-C class

```c
- (int)AddSubsettedFont:(NSString *)FontName :(int)CharsetIndex
    :(NSString *)SubsetChars;
```

Parameters

<table>
<thead>
<tr>
<th>FontName</th>
<th>The name of the TrueType font to install. This can either be the name of the font as shown in the Windows\Fonts folder (for example &quot;Times New Roman Bold&quot;) or it can be the font family name with an optional style specifier in square brackets (for example &quot;Times New Roman [BoldItalic]&quot;). Possible optional specifiers are: [Bold], [Italic] or [BoldItalic].</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharSetIndex</td>
<td>You must specify a character set containing the characters you want to subset: 1 = ANSI 2 = Default 3 = Symbol 4 = Shift JIS 5 = Hangeul 6 = GB2312 7 = Chinese Big 5 8 = OEM 9 = Johab 10 = Hebrew 11 = Arabic 12 = Greek 13 = Turkish 14 = Vietnamese 15 = Thai 16 = East Europe 17 = Russian 18 = Mac 19 = Baltic</td>
</tr>
<tr>
<td>SubsetChars</td>
<td>A string containing the characters you would like to subset. Repeated characters are ignored. A maximum of 255 characters can be placed in any font subset. Any Unicode character can be embedded, but you must ensure that the character is available in the specified character set.</td>
</tr>
</tbody>
</table>

Return values

| 0 | The subsetted font could not be added or the CharSet parameter was out of range |
| Non-zero | The FontID of the added font. This ID can be used with the SelectFont function to select the font. |
AddTextMarkupAnnotation

Annotations and hotspot links

Description

Adds a text markup annotation to the current page.
By default the annotation will consist of a single rectangular area matching the annotation's bounding box. This area can be edited and other areas can be added using the GetAnnotQuadCount, GetAnnotQuadPoints and SetAnnotQuadPoints functions.

Syntax

Dylib

int DPLAddTextMarkupAnnotation(int InstanceID, int MarkupType, double Left, double Top, double Width, double Height);

Objective-C class

- (int)AddTextMarkupAnnotation:(int)MarkupType :(double)Left :(double)Top :(double)Width :(double)Height;

Parameters

<table>
<thead>
<tr>
<th>MarkupType</th>
<th>0 = Highlight</th>
<th>1 = Underline</th>
<th>2 = Squiggly</th>
<th>3 = Strike out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the annotation bounding box</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the annotation bounding box</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Width</td>
<td>The width of the annotation bounding box</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td>The height of the annotation bounding box</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return values

| 0 | The MarkupType parameter was not between 1 and 4. |
| 1 | The text markup annotation was added successfully. |
AddToBuffer

Miscellaneous functions

Description

Adds a block of data to the buffer created with the CreateBuffer function. This function can be called multiple times until the buffer is full. The return value is the number of bytes remaining in the buffer.

Syntax

Dylib

```c
int DPLAddToBuffer(int InstanceID, char * Buffer, char * Source, int SourceLength);
```

Parameters

<table>
<thead>
<tr>
<th>Buffer</th>
<th>A value returned from the CreateBuffer function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>A pointer to the first byte of data to add</td>
</tr>
<tr>
<td>SourceLength</td>
<td>The total number of bytes to copy from the source</td>
</tr>
</tbody>
</table>
AddToFileList

Miscellaneous functions

Description

Adds a file to a named file list. This file list can later be used with functions that will operate on all the files in the list.

Syntax

Dylib

```c
int DPLAddToFileList(int InstanceID, wchar_t * ListName, wchar_t * FileName);
```

Objective-C class

```objc
- (int)AddToFileList:(NSString *)ListName :(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>ListName</th>
<th>The name of the file list to work with</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The file name to add to the list.</td>
</tr>
</tbody>
</table>
AddTrueTypeFont

Text, Fonts

Description

Adds a TrueType font to the document. The font must be installed on the system. If the font is not embedded, then the reader of the PDF document must have the font installed on their system too. If the font is embedded, then the reader does not need the font installed on their system. Embedding a font makes the PDF file much larger. Some fonts are not licensed to be embedded.

Syntax

Dylib

```
int DPLAddTrueTypeFont(int InstanceID, wchar_t * FontName, int Embed);
```

Objective-C class

```
-(int)AddTrueTypeFont:(NSString *)FontName :(int)Embed;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FontName</strong></td>
<td>The name of the TrueType font to install. This can either be the name of the font as shown in the Windows\Fonts folder (for example &quot;Times New Roman&quot;) or it can be the font family name with an optional style specifier in square brackets (for example &quot;Times New Roman [BoldItalic]&quot;). Possible optional specifiers are: [Bold], [Italic] or [BoldItalic]. A codepage can also be specified (for example &quot;Arial [Bold] {1250}&quot;) which allows other encodings to be used. Possible code pages are: {0} Direct mapping {437} OEM_CHARSET {850} OEM_CHARSET {852} OEM_CHARSET {874} THAI_CHARSET {1250} EASTEUROPE_CHARSET {1251} RUSSIAN_CHARSET {1252} ANSI_CHARSET {1253} GREEK_CHARSET {1254} TURKISH_CHARSET {1255} HEBREW_CHARSET {1256} ARABIC_CHARSET {1257} BALTIC_CHARSET {1258} VIETNAMESE_CHARSET {1361} JOHAB_CHARSET Note: {932}, {936}, {949} and {950} are not supported from version 8.11</td>
</tr>
<tr>
<td><strong>Embed</strong></td>
<td>Specifies whether to embed the font or not: 0 = Don't embed the font 1 = Embed the font</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The font could not be added. This may mean that the font is not licensed to be embedded, or that the font could not be found.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the font that was successfully added. This ID can be used with the SelectFont function to select the font</td>
</tr>
</tbody>
</table>
AddTrueTypeFontFromFile

Description

Embeds a TrueType, OpenType/TrueType or OpenType/CFF font into the document. The TrueType font is specified by the file name and does not have to be installed as a system font. This function is functionally identical to AddOpenTypeFontFromFile.

Syntax

**Dylib**

```c
int DPLAddTrueTypeFontFromFile(int InstanceID, wchar_t * FileName);
```

**Objective-C class**

```objc
- (int)AddTrueTypeFontFromFile:(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The full path and file name of the TrueType font file to embed.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The font could not be embedded</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the font that was successfully added. This ID can be used with the SelectFont function to select the font</td>
</tr>
</tbody>
</table>
AddTrueTypeSubsettedFont

Text, Fonts

Description

Adds a subsetted TrueType font to the document.
For Options 0 and 1 the font subset is fixed and cannot be changed.
For Options 2 and 3 the font subset can be changed Similar to 0 but subset can be updated using UpdateTrueTypeSubsettedFont

Syntax

Dylib

```c
int DPLAddTrueTypeSubsettedFont(int InstanceID, wchar_t * FontName, wchar_t * SubsetChars, int Options);
```

Objective-C class

```objc
- (int)AddTrueTypeSubsettedFont:(NSString *)FontName :(NSString *)SubsetChars :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>FontName</th>
<th>The name of the TrueType font that must be subsetted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubsetChars</td>
<td>A string containing the characters that should be included in the font subset.</td>
</tr>
<tr>
<td>Options</td>
<td>0=MS PlatformID, Unicode charset 1=Unicode PlatformID, &quot;don't care&quot; charset 2=Similar to 0 but subset can be updated using UpdateTrueTypeSubsettedFont 3=Similar to 1 but subset can be updated using UpdateTrueTypeSubsettedFont 4=Similar to 2 but subset is automatically updated 5=Similar to 3 but subset is automatically updated</td>
</tr>
</tbody>
</table>

Return values

| 0 | The subsetted font could not be added or the CharSet parameter was out of range |
| Non-zero | The ID of the font that was successfully added. This ID can be used with the SelectFont function to select the font |
**AddType1Font**

**Text, Fonts**

**Description**

Adds a PostScript Type1 font to the document. The font must be supplied as two files, a .pfm and a .pfb file. The full path to the .pfm file must be supplied. The font is embedded in the document.

**Syntax**

**Dylib**

```plaintext
int DPLAddType1Font(int InstanceID, wchar_t * FileName);
```

**Objective-C class**

```plaintext
- (int)AddType1Font:(NSString *)FileName;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The full path to the .pfm file. A .pfb file with the same name should exist in the same directory as the .pfm file.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The font could not be added. Either the font files are in the wrong format, or they cannot be found.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the font that was successfully added. This ID can be used with the <strong>SelectFont</strong> function to select the font</td>
</tr>
</tbody>
</table>
AddU3DAnnotationFromFile
Vector graphics, Image handling, Annotations and hotspot links, Page layout

Description

Adds an SVG file as an annotation to the current page. The SVG annotation will only be visible if the PDF is viewed with Adobe Acrobat 7 or higher.

Syntax

Dylib

```
int DPLAddU3DAnnotationFromFile(int InstanceID, double Left, double Top, 
double Width, double Height, wchar_t * FileName, int Options);
```

Objective-C class

```
- (int)AddU3DAnnotationFromFile:(double)Left :(double)Top :(double)Width 
:(double)Height :(NSString *)FileName :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the annotation rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the annotation rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the annotation rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the annotation rectangle</td>
</tr>
<tr>
<td>FileName</td>
<td>The path and name of the file containing the U3D model.</td>
</tr>
<tr>
<td>Options</td>
<td>0 = the 3D annotation is static, 1 = the 3D annotation is interactive</td>
</tr>
</tbody>
</table>
**AnalyseFile**

**Document properties**

**Description**

Analyses a file on disk. The entire file is not loaded into memory so huge files can be examined. Use the `GetAnalysisInfo` function to retrieve the individual analysis results. Call `DeleteAnalysis` to remove the results from memory when you are finished.

**Syntax**

**Dylib**

```c
int DPLAnalyseFile(int InstanceID, wchar_t * InputFileName, wchar_t * Password);
```

**Objective-C class**

```objective-c
- (int)AnalyseFile:(NSString *)InputFileName :(NSString *)Password;
```

**Parameters**

- **InputFileName**
  The path and name of the file to analyse.

- **Password**
  The password to use when opening the file. This can be either the owner or the user password. This parameter can be left blank if the file does not require a password to be opened.

**Return values**

- **0**
  The file could not be analysed. Check the result of the `LastErrorCode` function to determine the reason for the failure.

- **Non-zero**
  The analysis results ID. Pass this to the `GetAnalysisInfo` function.
AnnotationCount
Annotations and hotspot links

Description

Returns the number of annotations on the selected page.

Syntax

Dylib

```c
int DPLAnnotationCount(int InstanceID);
```

Objective-C class

```objc
-(int)AnnotationCount
```
AnsiStringResultLength

Miscellaneous functions

Description

Returns the length of the most recent string returned from the library by all functions that return 8-bit strings.

Syntax

Dylib

```c
int DPLAnsiStringResultLength(int InstanceID);
```
AppendSpace

**Text, Page layout**

**Description**

Moves the current text position horizontally by a percentage of the height of the text.

**Syntax**

**Dylib**

```c
int DPLAppendSpace(int InstanceID, double RelativeSpace);
```

**Objective-C class**

```objective-c
- (int)AppendSpace:(double)RelativeSpace;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>RelativeSpace</strong></th>
<th>A value of 1 moves the horizontal position by a value equal to the height of the text at the present font size, also known as an EM space. A value of 0.5 moves the horizontal position by half the height of the text at the present font size, also known as an EN space.</th>
</tr>
</thead>
</table>


AppendTableColumns

**Description**

Adds columns to the right of the specified table

**Syntax**

**Dylib**

```c
int DPLAppendTableColumns(int InstanceID, int TableID, int NewColumnCount);
```

**Objective-C class**

```objective-c
-(int)AppendTableColumns:(int)TableID :(int)NewColumnCount;
```

**Parameters**

<table>
<thead>
<tr>
<th>TableID</th>
<th>A TableID returned by the <strong>CreateTable</strong> function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewColumnCount</td>
<td>The number of columns to add to the table</td>
</tr>
</tbody>
</table>

**Return values**

| 0 | Columns could not be added. Check the TableID parameter and make sure NewColumnCount is greater than or equal to 1. |
| Non-zero | The total number of columns in the table after adding the new columns. |
AppendTableRows

Description

Adds rows to the bottom of the specified table.

Syntax

Dylib

```c
int DPLAppendTableRows(int InstanceID, int TableID, int NewRowCount);
```

Objective-C class

```c
-(int)AppendTableRows:(int)TableID :(int)NewRowCount;
```

Parameters

<table>
<thead>
<tr>
<th>TableID</th>
<th>A TableID returned by the CreateTable function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewRowCount</td>
<td>The number of rows to add to the table</td>
</tr>
</tbody>
</table>

Return values

| 0 | Rows could not be added. Check the TableID parameter and make sure NewRowCount is greater than or equal to 1. |
| Non-zero | The total number of rows in the table after adding the new rows. |
AppendText

Text, Page layout

Description

Draws text immediately following text previously drawn with DrawText or AppendText.

Syntax

**Dylib**

```
int DPLAppendText(int InstanceID, wchar_t * Text);
```

**Objective-C class**

```
-(int)AppendText:(NSString *)Text;
```

Parameters

| Text               | The text to append to the previously drawn text |
AppendToFile
Document management

Description

Appends the changed objects to the specified file in an incremental update.
The file name specified should be the same file that was the source of the document in the earlier call to LoadFromFile, LoadFromString or LoadFromStream.
Appending to a different file will result in a corrupt PDF.

Syntax

**Dylib**

```c
int DPLAppendToFile(int InstanceID, wchar_t * FileName);
```

**Objective-C class**

```objc
- (int)AppendToFile:(NSString *)FileName;
```

Parameters

| FileName | The name of the file to create |

Return values

| 0 | The incremental update could not be appended to the specified file |
| 1 | Success |
ApplyStyle

Text, Page layout

Description

Applies a style that was previously saved using the SaveStyle function. The style name is case sensitive, it must exactly match the style name used with the SaveStyle function.

Syntax

Dylib

```c
int DPLApplyStyle(int InstanceID, wchar_t * StyleName);
```

Objective-C class

```objc
- (int)ApplyStyle:(NSString *)StyleName;
```

Parameters

| **StyleName** | The name to associate with the style. This name is case sensitive. |

Return values

| **0** | The specified StyleName could not be found |
| **1** | The style was applied successfully |
AttachAnnotToForm

Form fields, Annotations and hotspot links

Description

This function attaches an annotation to the document form.
Use the IsAnnotFormField function to check if the specified annotation can be attached to the document form and whether it is currently attached or not.

Syntax

Dylib

```c
int DPLAttachAnnotToForm(int InstanceID, int Index);
```

Objective-C class

```c
-(int)AttachAnnotToForm:(int)Index;
```

Parameters

| Index     | The index of the annotation. The first annotation on the page has an index of 1. |

Return values

| 0          | The specified annotation could not be attached to the document form. |
| 1          | The specified annotation was attached successfully to the document form. |
BalanceContentStream

Content Streams and Optional Content Groups, Page manipulation

Description

This function combines the content stream parts and surrounds the content stream with "save graphics state" and "restore graphics state" operators. If the page contains unbalanced "save graphics state" and "restore graphics state" commands this function will add extra "restore graphics state" commands at the end of the page to balance the graphics state stack.

Syntax

**Dylib**

```
int DPLBalanceContentStream(int InstanceID);
```

**Objective-C class**

```
-(int)BalanceContentStream
```
BalancePageTree

Document management, Page properties

Description

Arranges the selected document's internal page structure into a balanced tree for faster random access to pages in the document.

Syntax

Dylib

```
int DPLBalancePageTree(int InstanceID, int Options);
```

Objective-C class

```
-(int)BalancePageTree:(int)Options;
```

Parameters

| Options | Reserved for future use, should be set to zero. |

Return values

| 0 | The page tree could not be balanced |
| 1 | Success |
**BeginPageUpdate**

**Page layout**

**Description**

For detailed page layouts this function can be called before a group of drawing commands. The page layout commands will then be buffered until a matching call to the `EndPageUpdate` function.

**Syntax**

**Dylib**

```c
int DPLBeginPageUpdate(int InstanceID);
```

**Objective-C class**

```objc
- (int)BeginPageUpdate
```
CapturePage

Page manipulation

Description

This function "captures" a page. Once the page has been captured it can be drawn onto other pages. This is useful for combining different pages or for placing more than one original page onto another page (imposition). Once a page has been captured it is removed from the document. If you would like the page to remain in the document you must create a blank page and draw the captured page onto the blank page.

Also, because a document must have at least one page at all times it is not possible to capture a page if it is the only page in the document. In this case, you must add a new blank page before the existing page can be captured.

The "media box" for the page is used as the bounding rectangle for the captured page. The CapturePageEx function can be used in cases where the "crop box" for the page should be used instead.

Syntax

Dylib

int DPLCapturePage(int InstanceID, int Page);

Objective-C class

- (int)CapturePage:(int)Page;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>The page number to capture. The first page in the document is page 1.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified page does not exist, or it is the only page in the document</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the capture process. This ID must be supplied to the DrawCapturedPage function.</td>
</tr>
</tbody>
</table>


**CapturePageEx**

**Page manipulation**

**Description**

This function "captures" a page. Once the page has been captured it can be drawn onto other pages. This is useful for combining different pages or for placing more than one original page onto another page (imposition). Once a page has been captured it is removed from the document. If you would like the page to remain in the document you must create a blank page and draw the captured page onto the blank page.

Also, because a document must have at least one page at all times it is not possible to capture a page if it is the only page in the document. In this case, you must add a new blank page before the existing page can be captured.

**Syntax**

**Dylib**

```c
int DPLCapturePageEx(int InstanceID, int Page, int Options);
```

**Objective-C class**

```c
-(int)CapturePageEx:(int)Page :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>Page</strong></th>
<th>The page number to capture. The first page in the document is page 1.</th>
</tr>
</thead>
</table>
| **Options** | 0 = Use the page's media box for the bounding rectangle  
|           | 1 = Use the page's crop box for the bounding rectangle if it has one, otherwise use the media box  
|           | 2 = Use the page's bleed box for the bounding rectangle if it has one, otherwise use the crop box  
|           | 3 = Use the page's trim box for the bounding rectangle if it has one, otherwise use the crop box  
|           | 4 = Use the page's art box for the bounding rectangle if it has one, otherwise use the crop box |

**Return values**

| **0** | The specified page does not exist, or it is the only page in the document |
| **Non-zero** | The ID of the capture process. This ID must be supplied to the `DrawCapturedPage` function. |
**CharWidth**

**Text, Fonts**

**Description**

Returns the width of a character for the selected font. This width is returned as a ratio to the text size. For example, if this function returns 750 for a certain character, then the width of the character for a 12 point font will be (750 / 1000) * 12.

**Syntax**

**Dylib**

```c
int DPLCharWidth(int InstanceID, int CharCode);
```

**Objective-C class**

```c
- (int)CharWidth:(int)CharCode;
```

**Parameters**

- **CharCode**
  
The character to determine the width for. For example, 65 is the character A.

**Return values**

The width of the specified character. Divide this value by 1000, and multiply by the text size in points to get the width of the character.
CheckFileCompliance

**Document manipulation**

**Description**

This function tests a PDF document against various standards to determine compliance with the standard.

This function is currently under development and currently runs only a small subset of possible tests.

**Syntax**

**Dylib**

```c
int DPLCheckFileCompliance(int InstanceID, wchar_t * InputFileName, wchar_t * Password, int ComplianceTest, int Options);
```

**Objective-C class**

```objective-c
- (int)CheckFileCompliance:(NSString *)InputFileName :(NSString *)Password :(int)ComplianceTest :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputFileName</strong></td>
<td>The file to check</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>The password to open the file. If there is no password an empty string should be used.</td>
</tr>
<tr>
<td><strong>ComplianceTest</strong></td>
<td>1 = PDF/A compliance test</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>For PDF/A compliance test: 0 = Show all errors 1 = Stop after the first error</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file passed the compliance test.</td>
</tr>
<tr>
<td><strong>Non-zero</strong></td>
<td>A StringListID that can be used with the getStringListCount and GetStringListItem functions.</td>
</tr>
</tbody>
</table>
CheckObjects

Miscellaneous functions

Description

Checks the file to ensure all objects are valid. This may take some time with large files and consume large amounts of memory.

Syntax

**Dylib**

```
int DPLCheckObjects(int InstanceID);
```

**Objective-C class**

```
 -(int)CheckObjects
```
CheckPageAnnots
Annotations and hotspot links, Miscellaneous functions

Description
Checks all the annotations on the selected page and ensures that they are all valid. Invalid annotations are removed from the page.

Syntax

**Dylib**
```
int DPLCheckPageAnnots(int InstanceID);
```

**Objective-C class**
```
- (int)CheckPageAnnots
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No annotations were found to be in an incorrect format</td>
</tr>
<tr>
<td>1</td>
<td>One or more annotations were not in the correct format and were unlinked from the page</td>
</tr>
</tbody>
</table>
CheckPassword

Security and Signatures

Description
Determines if a password is a valid password for the selected document. This is useful when the document has been opened with the user password but confirmation should be obtained from the user before changing security settings.

Syntax

Dylib

```c
int DPLCheckPassword(int instanceID, wchar_t * Password);
```

Objective-C class

```c
-(int)CheckPassword:(NSString *)Password;
```

Parameters

| Password | The password to check |

Return values

| 0 | The document is not encrypted or the supplied password is not a valid owner or user password |
| 1 | Valid user password |
| 2 | Valid owner password |
| 3 | Valid owner and user password |
ClearFileList

Miscellaneous functions

Description

Clears a named file list.

Syntax

**Dylib**

```c
int DPLClearFileList(int InstanceID, wchar_t * ListName);
```

**Objective-C class**

```objc
- (int)ClearFileList:(NSString *)ListName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ListName</strong></td>
<td>The name of the file list to clear</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The named list could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The named list was cleared successfully</td>
</tr>
</tbody>
</table>
ClearImage

Image handling

Description

Clears the specified image.

To prevent the corruption of existing links to the image it will not be deleted from the document. The image will be converted into a 24-bit RGB format consisting of a single transparent pixel.

Syntax

Dylib

```c
int DPLClearImage(int InstanceID, int ImageID);
```

Objective-C class

```objective-c
- (int)ClearImage:(int)ImageID;
```

Parameters

| ImageID | The ImageID of the image to be cleared |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified ImageID was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The image was cleared</td>
</tr>
</tbody>
</table>
ClearPageLabels

Page properties

Description

Removes all the page labels from the selected document.

Syntax

Dylib

```c
int DPLClearPageLabels(int InstanceID);
```

Objective-C class

```objc
-(int)ClearPageLabels
```
ClearTextFormatting

Description

Clears any formatting that has been applied. Subsequently drawn text will be drawn left aligned in black with all highlighting, underlining, character spacing, word spacing, horizontal scaling and vertical spacing removed.

Syntax

**Dylib**

```c
int DPLClearTextFormatting(int InstanceID);
```

**Objective-C class**

```objc
- (int)ClearTextFormatting
```
CloneOutlineAction
Annotations and hotspot links, Outlines

Description
Calling this function will clone the action dictionary of the specified outline. This is useful when an
outline and an annotation share the same action dictionary and the actions must be set
individually.

Syntax
Dylib

```c
int DPLCloneOutlineAction(int InstanceID, int OutlineID);
```

Objective-C class

```objective-c
- (int)CloneOutlineAction:(int)OutlineID;
```

Parameters

| OutlineID  | The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID. |
ClonePages
Page manipulation

Description
Copies pages from the document multiple times, with only a negligible increase in file size. Note that only the first "layer" of the page is cloned. Unless you specifically want to take part of the page you should call CombineContentStreams for all the pages you want to clone before calling this function.

Syntax
Dylib

    int DPLClonePages(int InstanceID, int StartPage, int EndPage,
                        int RepeatCount);

Objective-C class

    -(int)ClonePages:(int)StartPage :(int)EndPage :(int)RepeatCount;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPage</td>
<td>The first page to clone</td>
</tr>
<tr>
<td>EndPage</td>
<td>The last page to clone</td>
</tr>
<tr>
<td>RepeatCount</td>
<td>The number of times to clone the pages</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The parameters were out of range</td>
</tr>
<tr>
<td>1</td>
<td>The function was successful</td>
</tr>
</tbody>
</table>
CloseOutline
Outlines

Description

Collapses an outline item (bookmark).

Syntax

Dylib

int DPLCloseOutline(int InstanceID, int OutlineID);

Objective-C class

-(int)CloseOutline:(int)OutlineID;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutlineID</td>
<td>The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Outline ID provided was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The outline item was collapsed</td>
</tr>
</tbody>
</table>
**ClosePath**

*Vector graphics, Path definition and drawing*

**Description**

Closes the path defined by calls to **StartPath**, **AddLineToPath**, and **AddCurveToPath**. A line is drawn from the last point to the first point.

**Syntax**

**Dylib**

```c
int DPLClosePath(int InstanceID);
```

**Objective-C class**

```objective-c
- (int)ClosePath
```
**CombineContentStreams**

**Content Streams and Optional Content Groups**

**Description**

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function combines all the content stream parts of the selected page into a single content stream.

**Syntax**

**Dylib**

```c
int DPLCombineContentStreams(int InstanceID);
```

**Objective-C class**

```c
- (int)CombineContentStreams
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The content stream could not be combined</td>
</tr>
<tr>
<td>1</td>
<td>The content stream was combined successfully</td>
</tr>
</tbody>
</table>
## CompareOutlines

### Description

Compares two OutlineID values.

### Syntax

**Dylib**

```c
int DPLCompareOutlines(int InstanceID, int FirstOutlineID,
                        int SecondOutlineID);
```

**Objective-C class**

```objective-c
-(int)CompareOutlines:(int)FirstOutlineID :(int)SecondOutlineID;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstOutlineID</td>
<td>The first OutlineID to compare</td>
</tr>
<tr>
<td>SecondOutlineID</td>
<td>The second OutlineID to compare</td>
</tr>
</tbody>
</table>

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>One or both of the OutlineID values were not valid or there is no relationship between the two outlines.</td>
</tr>
<tr>
<td>1</td>
<td>The OutlineID values refer to the same outline item.</td>
</tr>
</tbody>
</table>
CompressContent

Document properties

Description
Compresses the content of the selected document. The Flate algorithm is used to compress the content.

Syntax

Dylib

```c
int DPLCompressContent(int InstanceID);
```

Objective-C class

```c
- (int)CompressContent
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The content could not be compressed</td>
</tr>
<tr>
<td>1</td>
<td>The content was compressed successfully</td>
</tr>
</tbody>
</table>
**CompressFonts**

**Fonts, Document properties**

**Description**

Specifies whether or not to compress TrueType, Packaged and Type1 fonts subsequently added to the document.

**Syntax**

**Dylib**

```c
int DPLCompressFonts(int InstanceID, int Compress);
```

**Objective-C class**

```c
-(int)CompressFonts:(int)Compress;
```

**Parameters**

<table>
<thead>
<tr>
<th>Compress</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Don't compress fonts</td>
</tr>
<tr>
<td>1</td>
<td>Compress all subsequently added fonts</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Compress parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>The font compression setting was changed successfully</td>
</tr>
</tbody>
</table>
### CompressImages

**Image handling, Document properties**

**Description**

Specifies the compression to use for images added to the document.

**Syntax**

**Dylib**

```c
int DPLCompressImages(int InstanceID, int Compress);
```

**Objective-C class**

```objective-c
-(int)CompressImages:(int)Compress;
```

**Parameters**

<table>
<thead>
<tr>
<th>Compress</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No compression</td>
</tr>
<tr>
<td>1</td>
<td>Flate compression</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Compress parameter was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The image compression was set successfully</td>
</tr>
</tbody>
</table>
CompressPage

Description

This function is similar to the CompressContent function, however it only compresses the selected page. Looping through all the pages using this function will have the same effect as CompressContent, however it will be possible to provide feedback to the user.

Syntax

Dylib

```c
int DPLCompressPage(int InstanceID);
```

Objective-C class

```objective-c
-(int)CompressPage
```
ContentStreamCount

Content Streams and Optional Content Groups

**Description**

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function returns the total number of content stream parts for the selected page.

**Syntax**

**Dylib**

```c
int DPLContentStreamCount(int InstanceID);
```

**Objective-C class**

```objective-c
- (int)ContentStreamCount
```

**Return values**

The number of content stream parts on the selected page
ContentStreamSafe

Content Streams and Optional Content Groups

Description

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function determines if the content stream part that was selected using the SelectContentStream function was created by Quick PDF Library or not.

Only content stream parts created by Quick PDF Library should be considered "safe" to drawn on. If a content stream part is not safe it would be best to combine all the content stream parts using the CombineContentStreams function before drawing on the page to prevent later errors in the document.

Syntax

Dylib

int DPLContentStreamSafe(int InstanceID);

Objective-C class

- (int)ContentStreamSafe

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The layer was not created by Quick PDF Library and care should be taken when drawing onto this layer</td>
</tr>
<tr>
<td>1</td>
<td>The layer was created by Quick PDF Library and is safe to draw on</td>
</tr>
</tbody>
</table>
CopyPageRanges

Description

Use this function to copy one or more pages from one document to another. The pages are copied in sequential order and duplicates are not allowed. To extract pages in a different order to the source document or with duplicate pages the CopyPageRangesEx function can be used.

Syntax

Dylib

```c
int DPLCopyPageRanges(int InstanceID, int DocumentID, wchar_t * Rangelist);
```

Objective-C class

```c
-(int)CopyPageRanges:(int)DocumentID :(NSString *)RangeList;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocumentID</td>
<td>The ID of the document to copy the pages from</td>
</tr>
<tr>
<td>Rangelist</td>
<td>The pages to extract, for example &quot;10,15,18-20,25-35&quot;. Invalid characters and duplicate page numbers in the string will be ignored. Reversed page ranges such as &quot;5-1&quot; will be accepted. The list of pages will be sorted resulting in the pages being extracted in numerical order.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified DocumentID was not valid or was the same as the selected document, or the Rangelist was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The pages were successfully copied from the specified document to the selected document</td>
</tr>
</tbody>
</table>
CopyPageRangesEx

Description

Use this function to copy one or more pages from one document to another. It is functionality identical to the CopyPageRanges function but adds an option to allow the page list to contain duplicate page numbers and a different page order to the original document.

Syntax

Dylib

```c
int DPLCopyPageRangesEx(int InstanceID, int DocumentID, wchar_t * RangeList, int Options);
```

Objective-C class

```objective-c
- (int)CopyPageRangesEx:(int)DocumentID :(NSString *)RangeList :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocumentID</td>
<td>The ID of the document to copy the pages from</td>
</tr>
<tr>
<td>RangeList</td>
<td>The pages to extract, for example &quot;10,15,18-20,25-35&quot;. Invalid characters in the string will be ignored.</td>
</tr>
<tr>
<td>Options</td>
<td>0 = Identical behaviour to the CopyPageRanges function. The page list is sorted and duplicate page numbers are ignored. 1 = Do not sort the page list and allow duplicate page numbers</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified DocumentID was not valid or was the same as the selected document, or the RangeList was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The pages were successfully copied from the specified document to the selected document</td>
</tr>
</tbody>
</table>
CreateBuffer

Miscellaneous functions

Description

Creates a buffer that can be used to send strings to Quick PDF Library DLL containing null characters.

Once the buffer has been created, use the AddToBuffer function to add data to the buffer. The data can be added to the buffer in one call, or chunks of data can be sent one at a time until the buffer is full.

When you are finished with the buffer, call the ReleaseBuffer function to release it.

Syntax

Dylib

char * DPLCreateBuffer(int InstanceID, int BufferLength);

Parameters

| BufferLength | The size in bytes of the buffer that must be created |

Return values

| 0            | The BufferLength value was less than 1, or the InstanceID was invalid |
| Non-zero     | A PChar that can be passed as any string parameter to other functions |
CreateLibrary

Miscellaneous functions

Description

Call this function to create an instance of Quick PDF Library in the DLL. The value returned is used as the InstanceID parameter of all the other functions.

Call the ReleaseLibrary function to free the instance when you are finished with it.

Syntax

Dylib

int DPLCreateLibrary(int InstanceID);

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An instance of Quick PDF Library could not be created</td>
</tr>
<tr>
<td>Non-zero</td>
<td>An InstanceID value that can be used with other functions</td>
</tr>
</tbody>
</table>
CreateNewObject

Miscellaneous functions

Description

Adds a new PDF object to the document. The contents of the object can be set using the `SetObjectFromString` function.

Syntax

**Dylib**

```c
int DPLCreateNewObject(int InstanceID);
```

**Objective-C class**

```objective-c
- (int)CreateNewObject
```

Return values

<table>
<thead>
<tr>
<th>Non-zero</th>
<th>The object number of the newly created object</th>
</tr>
</thead>
</table>
CreateTable

Page layout

Description

Creates a table with the specified number of rows and columns. Use the other table functions to set up the table and then use DrawTableRows to draw the table onto the page.

Syntax

Dylib

```c
int DPLCreateTable(int InstanceID, int RowCount, int ColumnCount);
```

Objective-C class

```c
- (int)CreateTable:(int)RowCount :(int)ColumnCount;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowCount</td>
<td>The number of rows that the new table should have</td>
</tr>
<tr>
<td>ColumnCount</td>
<td>The number of columns that the new table should have</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The table could not be created. Row and column count must be greater or equal to 1.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A TableID that can be used with the other table functions.</td>
</tr>
</tbody>
</table>
**DAAppendFile**

**Document management, Direct access functionality**

**Description**

Appends any changes made to a document originally opened using the `DAOpenFile` function. This is a fast operation because only the changed objects must be added to the end of the original file. The file is closed after this operation and the file handle will no longer be valid.

This function will not work if the source file was opened in read only mode or if the document was loaded from a malformed file for example where whitespace was added to the start of the file. In these cases the `DASaveAsFile` function should be used instead.

**Syntax**

**Dylib**

```c
int DPLDAAppendFile(int InstanceID, int FileHandle);
```

**Objective-C class**

```objective-c
- (int)DAAppendFile:(int)FileHandle;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The changes to the file were appended successfully</td>
</tr>
<tr>
<td>2</td>
<td>The file was opened in read only mode and the update cannot be written. Use <code>DASaveAsFile</code> instead.</td>
</tr>
<tr>
<td>3</td>
<td>The document was opened from a malformed file and an append operation is not possible. See the <code>DAShiftedHeader</code> function.</td>
</tr>
</tbody>
</table>
DACapturePage
Direct access functionality, Page manipulation

Description
This function "captures" the specified page from a document originally opened with DAOpenFile. The captured page can then be drawn onto any other page using the DADrawCapturedPage function. This is useful for combining different pages or for placing more than one original page onto another page (imposition).

Once a page has been captured it is removed from the document. If you would like the page to remain in the document you must create a blank page and draw the captured page onto the blank page.

The "media box" for the page is used as the bounding rectangle for the capture page. The DACapturePageEx function can be used in cases where the "crop box" for the page should be used instead.

Syntax
Dylib

int DPLDACapturePage(int InstanceID, int FileHandle, int PageRef);

Objective-C class

- (int)DACapturePage:(int)FileHandle :(int)PageRef;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the DAFindPage or DANewPage functions</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle or PageRef were not valid</td>
</tr>
<tr>
<td>Non-zero</td>
<td>An ID that can be used with the DADrawCapturedPage function</td>
</tr>
</tbody>
</table>
**DACapturePageEx**

Direct access functionality, Page manipulation

**Description**

Captures the specified page from a document originally opened with **DAOpenFile**. The captured page is hidden, but can then be drawn onto any other page using the **DADrawCapturedPage** function. The "media box" for the page is used as the bounding rectangle for the capture page. The **DACapturePageEx** function can be used in cases where the "crop box" for the page should be used instead.

**Syntax**

**Dylib**

```c
int DPLDACapturePageEx(int InstanceID, int FileHandle, int PageRef, int Options);
```

**Objective-C class**

```c
- (int)DACapturePageEx:(int)FileHandle :(int)PageRef :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <strong>DAFindPage</strong> or <strong>DANewPage</strong> functions</td>
</tr>
</tbody>
</table>
| Options | 0 = Use the page's media box for the bounding rectangle  
1 = Use the page's crop box for the bounding rectangle if it has one, otherwise use the media box  
2 = Use the page's bleed box for the bounding rectangle if it has one, otherwise use the crop box  
3 = Use the page's trim box for the bounding rectangle if it has one, otherwise use the crop box  
4 = Use the page's art box for the bounding rectangle if it has one, otherwise use the crop box |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle or PageRef were not valid, or the specified page was the only page in the document</td>
</tr>
<tr>
<td>Non-zero</td>
<td>An ID that can be used with the <strong>DADrawCapturedPage</strong> function</td>
</tr>
</tbody>
</table>
**DACloseFile**

**Direct access functionality**

**Description**

Closes a file that was originally opened using the **DAOpenFile** function. Any changes made to the file are lost. If you would like to keep your changes you must use either the **DASaveAsFile** function or the **DAAppendFile** function before closing the file.

**Syntax**

**Dylib**

```c
int DPLDACloseFile(int InstanceID, int FileHandle);
```

**Objective-C class**

```objc
-(int)DACloseFile:(int)FileHandle;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle was not valid, the file may already have been closed</td>
</tr>
<tr>
<td>1</td>
<td>The file was closed successfully</td>
</tr>
</tbody>
</table>
**DADrawCapturedPage**

**Direct access functionality, Page layout**

**Description**

Draws a page originally captured using the `DrawCapturedPage` function onto the specified page. The original page must have been captured from the same document (having the same `FileHandle`).

**Syntax**

_Dylib_

```c
int DPLDADrawCapturedPage(int InstanceID, int FileHandle, int DACaptureID,
                          int DestPageRef, double PntLeft, double PntBottom,
                          double PntWidth, double PntHeight);
```

_Objective-C class_

```c
-(int)DADrawCapturedPage:(int)FileHandle :(int)DACaptureID
          :(int)DestPageRef :(double)PntLeft :(double)PntBottom
          :(double)PntWidth :(double)PntHeight;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>FileHandle</strong></th>
<th>A handle returned by the <strong>DAOpenFile, DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DACaptureID</strong></td>
<td>A capture ID returned by the <strong>DACapturePage</strong> function</td>
</tr>
<tr>
<td><strong>DestPageRef</strong></td>
<td>A page reference returned by the <strong>DAFindPage or DANewPage</strong> functions</td>
</tr>
<tr>
<td><strong>PntLeft</strong></td>
<td>The horizontal co-ordinate of the left edge of the destination rectangle, measured in points from the left edge of the page</td>
</tr>
<tr>
<td><strong>PntBottom</strong></td>
<td>The vertical co-ordinate of the bottom edge of the destination rectangle, measured in points from the bottom edge of the page</td>
</tr>
<tr>
<td><strong>PntWidth</strong></td>
<td>The width of the destination rectangle, measured in points</td>
</tr>
<tr>
<td><strong>PntHeight</strong></td>
<td>The height of the destination rectangle, measured in points</td>
</tr>
</tbody>
</table>

**Return values**

| **0** | The specified FileHandle, PageRef or DACaptureID were not valid                                 |
| **1** | The captured page was drawn successfully                                                      |
DADrawRotatedCapturedPage

Direct access functionality, Page layout

Description

Similar to the DADrawCapturedPage function but allows the captured page to be drawn at any angle.

Syntax

Dylib

```c
int DPLDADrawRotatedCapturedPage(int InstanceID, int FileHandle,
   int DACaptureID, int DestPageRef, double PntLeft,
   double PntBottom, double PntWidth, double PntHeight,
   double Angle);
```

Objective-C class

```c
- (int)DADrawRotatedCapturedPage:(int)FileHandle :(int)DACaptureID
   :(int)DestPageRef :(double)PntLeft :(double)PntBottom
   :(double)PntWidth :(double)PntHeight :(double)Angle;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td>DACaptureID</td>
<td>A capture ID returned by the DACapturePage function</td>
</tr>
<tr>
<td>DestPageRef</td>
<td>A page reference returned by the DAFindPage or DANewPage functions</td>
</tr>
<tr>
<td>PntLeft</td>
<td>The horizontal co-ordinate of the left edge of the destination rectangle, measured in points from the left edge of the page</td>
</tr>
<tr>
<td>PntBottom</td>
<td>The vertical co-ordinate of the bottom edge of the destination rectangle, measured in points from the bottom edge of the page</td>
</tr>
<tr>
<td>PntWidth</td>
<td>The width of the destination rectangle, measured in points</td>
</tr>
<tr>
<td>PntHeight</td>
<td>The height of the destination rectangle, measured in points</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle to rotate the captured page by, measured anti-clockwise in degrees from the baseline</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle, PageRef or DACaptureID were not valid</td>
</tr>
<tr>
<td>1</td>
<td>The captured page was drawn successfully</td>
</tr>
</tbody>
</table>
DAEmbedFileStreams

Document manipulation, Direct access functionality

Description

Converts any stream object where the data is stored in an external file into a regular embedded stream object.

Syntax

Dylib

```c
int DPLDAEmbedFileStreams(int InstanceID, int FileHandle, wchar_t * RootPath);
```

Objective-C class

```objective-c
- (int)DAEmbedFileStreams:(int)FileHandle :(NSString *)RootPath;
```

Parameters

<table>
<thead>
<tr>
<th>FileHandle</th>
<th>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RootPath</td>
<td>The directory to use as the root for relative paths.</td>
</tr>
</tbody>
</table>
Description

This function provides two different methods for extracting text from the selected page, and presents the results in a variety of formats.

The **DASetTextExtractionWordGap, DASetTextExtractionOptions** and **DASetTextExtractionArea** functions can be used to adjust the text extraction process.

Syntax

**Dylib**

```c
wchar_t * DPLDAExtractPageText(int InstanceID, int FileHandle, int PageRef, int Options);
```

**Objective-C class**

```objective-c
- (NSString *)DAExtractPageText:(int)FileHandle :(int)PageRef :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile, DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <strong>DAFindPage</strong> or <strong>DANewPage</strong> functions</td>
</tr>
<tr>
<td>Options</td>
<td>Using the standard text extraction algorithm:</td>
</tr>
<tr>
<td></td>
<td>- 0 = Extract text in human readable format</td>
</tr>
<tr>
<td></td>
<td>- 1 = Deprecated</td>
</tr>
<tr>
<td></td>
<td>- 2 = Return a CSV string including font, color, size and position of each piece of text on the page</td>
</tr>
<tr>
<td></td>
<td>Using the more accurate but slower text extraction algorithm:</td>
</tr>
<tr>
<td></td>
<td>- 3 = Return a CSV string for each piece of text on the page with the following format:</td>
</tr>
<tr>
<td></td>
<td>Font Name, Text Color, Text Size, X1, Y1, X2, Y2, X3, Y3, X4, Y4, Text</td>
</tr>
<tr>
<td></td>
<td>The co-ordinates are the four points bounding the text, measured using the units set with the <strong>SetMeasurementUnits</strong> function and the origin set with the <strong>SetOrigin</strong> function. Co-ordinate order is anti-clockwise with the bottom left corner first.</td>
</tr>
<tr>
<td></td>
<td>- 4 = Similar to option 3, but individual words are returned, making searching for words easier</td>
</tr>
<tr>
<td></td>
<td>- 5 = Similar to option 3 but character widths are output after each block of text</td>
</tr>
<tr>
<td></td>
<td>- 6 = Similar to option 4 but character widths are output after each line of text</td>
</tr>
<tr>
<td></td>
<td>- 7 = Extract text in human readable format with improved accuracy compared to option 0</td>
</tr>
<tr>
<td></td>
<td>- 8 = Similar output format as option 0 but using the more accurate algorithm. Returns unformatted lines.</td>
</tr>
</tbody>
</table>
DAExtractPageTextBlocks

Text, Extraction, Direct access functionality

Description

Similar to the DAExtractPageText function but the results are stored in a text block list rather than returned as a CSV string.

Once the results are in the text block list, functions such as DAGetTextBlockCount, DAGetTextBlockText and DAGetTextBlockColor can be used to retrieve the properties of each block of text.

Syntax

Dylib

```c
int DPLDAExtractPageTextBlocks(int InstanceID, int FileHandle, int PageRef, int ExtractOptions);
```

Objective-C class

```objc
- (int)DAExtractPageTextBlocks:(int)FileHandle :(int)PageRef :(int)ExtractOptions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the DAFindPage or DANewPage functions</td>
</tr>
</tbody>
</table>
| ExtractOptions | 3 = Normal extraction  
                        4 = Split words |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Text could not be extracted from the page</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A TextBlockListID value</td>
</tr>
</tbody>
</table>
DAFindPage
Direct access functionality

Description
Use this function to obtain a page reference for use with other Direct Access functions. This page reference will remain constant even if other pages are added to or removed from the document.

Syntax

**Dylib**

```c
int DPLDAFindPage(int InstanceID, int FileHandle, int Page);
```

**Objective-C class**

```c
-(int)DAFindPage:(int)FileHandle :(int)Page;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileHandle</strong></td>
<td>A handle returned by the <a href="#">DAOpenFile</a>, <a href="#">DAOpenFileReadOnly</a> or <a href="#">DAOpenFromStream</a> functions</td>
</tr>
<tr>
<td><strong>Page</strong></td>
<td>The page number. The first page in the document has a page number of 1.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle was not valid or the Page parameter was out of range</td>
</tr>
<tr>
<td>Non-zero</td>
<td>An ID that can be used as the PageRef parameter for any of the direct access functions</td>
</tr>
</tbody>
</table>
DAGetAnnotationCount

Direct access functionality

Description

Returns the number of annotations on the specified page.

Syntax

Dylib

```c
int DPLDAGetAnnotationCount(int InstanceID, int FileHandle, int PageRef);
```

Objective-C class

```objc
-(int)DAGetAnnotationCount:(int)FileHandle :(int)PageRef;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <strong>DAFindPage</strong> or <strong>DANewPage</strong> functions</td>
</tr>
</tbody>
</table>
**DAGetFormFieldCount**

**Form fields, Direct access functionality**

**Description**

Returns the number of form fields in the document.

**Syntax**

**Dylib**

```c
int DPLDAGetFormFieldCount(int InstanceID, int FileHandle);
```

**Objective-C class**

```c
- (int)DAGetFormFieldCount:(int)FileHandle;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>FileHandle</strong></th>
<th>A handle returned by the <strong>DAOpenFile, DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</th>
</tr>
</thead>
</table>
**DAGetFormFieldTitle**

**Form fields, Direct access functionality**

**Description**

Returns the title of the specified form field.

**Syntax**

**Dylib**

```c
wchar_t * DPLDAGetFormFieldTitle(int InstanceID, int FileHandle, int FieldIndex);
```

**Objective-C class**

```objective-c
- (NSString *)DAGetFormFieldTitle:(int)FileHandle :(int)FieldIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <a href="#">DAOpenFile</a>, <a href="#">DAOpenFileReadOnly</a> or <a href="#">DAOpenFromStream</a> functions</td>
</tr>
<tr>
<td>FieldIndex</td>
<td>The index of the form field to work with. The first form field has an index of 1.</td>
</tr>
</tbody>
</table>
DAGetFormFieldValue
Form fields, Direct access functionality

**Description**

Returns the value of the specified form field.

**Syntax**

**Dylib**

```c
wchar_t * DPLDAGetFormFieldValue(int InstanceID, int FileHandle, int FieldIndex);
```

**Objective-C class**

```objective-c
-(NSString *)DAGetFormFieldValue:(int)FileHandle :(int)FieldIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileHandle</strong></td>
<td>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</td>
</tr>
<tr>
<td><strong>FieldIndex</strong></td>
<td>The index of the form field to work with. The first form field has an index of 1.</td>
</tr>
</tbody>
</table>
**Description**

Returns the image data of an image in an image list. The format of the data depends on the type of the image. The `DAGetImageIntProperty` function can be used to determine the image type.

**Syntax**

**Dylib**

```c
char * DPLDAGetImageDataToString(int InstanceID, int FileHandle, int ImageListID, int ImageIndex);
```

**Objective-C class**

```objective-c
-(NSData *)DAGetImageDataToString:(int)FileHandle :(int)ImageListID :(int)ImageIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</td>
</tr>
<tr>
<td>ImageListID</td>
<td>A value returned by the <code>DAGetPageImageList</code> function</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image. The first image in the list has an index of 1. Use the <code>DAGetImageListCount</code> function to determine the number of images in the list.</td>
</tr>
</tbody>
</table>
DAGetImageDblProperty

Image handling, Direct access functionality

Description

Returns certain properties of an image in an image list.

Syntax

C

double DPLDAGetImageDblProperty(int InstanceID, int FileHandle,
int ImageListID, int ImageIndex, int PropertyID);

Objective-C class

- (double)DAGetImageDblProperty:(int)FileHandle :(int)ImageListID
 :(int)ImageIndex :(int)PropertyID;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td>ImageListID</td>
<td>A value returned by the DAGetPageImageList function</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image. The first image in the list has an index of 1. Use the DAGeImageListCount function to determine the number of images in the list.</td>
</tr>
</tbody>
</table>
| PropertyID   | 501 = Horizontal co-ordinate of top-left corner  
502 = Vertical co-ordinate of top-left corner  
503 = Horizontal co-ordinate of top-right corner  
504 = Vertical co-ordinate of top-right corner  
505 = Horizontal co-ordinate of bottom-right corner  
506 = Vertical co-ordinate of bottom-right corner  
507 = Horizontal co-ordinate of bottom-left corner  
508 = Vertical co-ordinate of bottom-left corner |
DAGetImageIntProperty

Image handling, Direct access functionality

Description

Returns certain properties of an image in an image list.

Syntax

**Dylib**

```c
int DPLDAGetImageIntProperty(int InstanceID, int FileHandle,
                          int ImageListID, int ImageIndex, int PropertyID);
```

**Objective-C class**

```objective-c
-(int)DAGetImageIntProperty:(int)FileHandle :(int)ImageListID
            :(int)ImageIndex :(int)PropertyID;
```

Parameters

- **FileHandle**
  A handle returned by the **DAOpenFile**, **DAOpenFileReadOnly** or **DAOpenFromStream** functions

- **ImageListID**
  A value returned by the **DAGetPageImageList** function

- **ImageIndex**
  The index of the image. The first image in the list has an index of 1. Use the **DAGetImageListCount** function to determine the number of images in the list.

- **PropertyID**
  400 = Image type (see **ImageType**)
  401 = Width in pixels
  402 = Height in pixels
**DAGetImageListCount**

*Image handling, Direct access functionality*

**Description**

Returns the number of images in an image list.

**Syntax**

Dylib

```c
int DPLDAGetImageListCount(int InstanceID, int FileHandle, int ImageListID);
```

Objective-C class

```objc
- (int)DAGetImageListCount:(int)FileHandle :(int)ImageListID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileHandle</strong></td>
<td>A handle returned by the <a href="#">DAOpenFile</a>, <a href="#">DAOpenFileReadOnly</a> or <a href="#">DAOpenFromStream</a> functions</td>
</tr>
<tr>
<td><strong>ImageListID</strong></td>
<td>A value returned by the <a href="#">DAGetPageImageList</a> function</td>
</tr>
</tbody>
</table>
### Description

Retrieves information from the document information section. This could be standard information such as Author and Subject, or custom information.

### Syntax

**Dylib**

```c
wchar_t * DPLDAGetInformation(int InstanceID, int FileHandle, wchar_t * Key);
```

**Objective-C class**

```objc
-(NSString *)DAGetInformation:(int)FileHandle :(NSString *)Key;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileHandle</strong></td>
<td>A handle returned by the <a href="https://www.docushare.com/">DAOpenFile</a>, <a href="https://www.docushare.com/">DAOpenFileReadOnly</a> or <a href="https://www.docushare.com/">DAOpenFromStream</a> functions</td>
</tr>
<tr>
<td><strong>Key</strong></td>
<td>For standard information use &quot;Author&quot;, &quot;Title&quot;, &quot;Subject&quot;, &quot;Keywords&quot;, &quot;Creator&quot;, or &quot;Producer&quot;. For custom information any other string can be used.</td>
</tr>
</tbody>
</table>
**DAGetObjectCount**

**Miscellaneous functions, Direct access functionality**

**Description**

Returns the number of raw PDF objects in the document.

**Syntax**

**Dylib**

```c
int DPLDAGetObjectCount(int InstanceID, int FileHandle);
```

**Objective-C class**

```objc
-(int)DAGetObjectCount:(int)FileHandle;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>FileHandle</strong></th>
<th>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</th>
</tr>
</thead>
</table>
**DAGetObjectToString**

*Miscellaneous functions, Direct access functionality*

**Description**

Returns the raw PDF object data for the specified object number. This is for advanced use only.

**Syntax**

**Dylib**

```c
char * DPLDAGetObjectToString(int InstanceID, int FileHandle, int ObjectNumber);
```

**Objective-C class**

```objc
- (NSData *)DAGetObjectToString:(int)FileHandle :(int)ObjectNumber;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileHandle</strong></td>
<td>A handle returned by the <a href="#">DAOpenFile</a>, <a href="#">DAOpenFileReadOnly</a> or <a href="#">DAOpenFromStream</a> functions</td>
</tr>
<tr>
<td><strong>ObjectNumber</strong></td>
<td>The number of the object to retrieve. The first object is numbered 1 and the last object has an object number equal to the result of the <a href="#">GetObjectCount</a> function.</td>
</tr>
</tbody>
</table>
**DAGetPageBox**

**Direct access functionality, Page properties**

**Description**

Returns a dimension of the specified page boundary rectangle.

Returned values are points measured from the bottom left corner of the page.

**Syntax**

**Dylib**

```c
double DPLDAGetPageBox(int InstanceID, int FileHandle, int PageRef, int BoxIndex, int Dimension);
```

**Objective-C class**

```c
- (double)DAGetPageBox:(int)FileHandle :(int)PageRef :(int)BoxIndex :(int)Dimension;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileHandle</strong></td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td><strong>PageRef</strong></td>
<td>A page reference returned by the DAFindPage or DANewPage functions</td>
</tr>
</tbody>
</table>
| **BoxIndex** | 1 = MediaBox  
2 = CropBox  
3 = BleedBox  
4 = TrimBox  
5 = ArtBox |
| **Dimension** | 0 = Left  
1 = Top  
2 = Width  
3 = Height  
4 = Right  
5 = Bottom |
**Description**

Retrieves the graphics commands and operators that make up the specified page.

**Syntax**

**Dylib**

```c
char * DPLDAGetPageContentToString(int InstanceID, int FileHandle, int PageRef);
```

**Objective-C class**

- `(NSData *)DAGetPageContentToString:(int)FileHandle :(int)PageRef;

**Parameters**

<table>
<thead>
<tr>
<th>FileHandle</th>
<th>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <code>DAFindPage</code> or <code>DANewPage</code> functions</td>
</tr>
</tbody>
</table>
DAGetPageCount

Document properties, Direct access functionality

Description

Returns the number of pages in a document opened with the DAOpenFile function.

Syntax

Dylib

int DPLDAGetPageCount(int InstanceID, int FileHandle);

Objective-C class

- (int)DAGetPageCount:(int)FileHandle;

Parameters

| FileHandle | A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions |

Return values

| 0           | The specified FileHandle was not valid |
| Non-zero    | The number of pages in the document |
**DAGetPageHeight**

**Direct access functionality, Page properties**

**Description**

Returns the height of the specified page in a document opened with the *DAOpenFile* function.

**Syntax**

**Dylib**

```c
double DPLDAGetPageHeight(int InstanceID, int FileHandle, int PageRef);
```

**Objective-C class**

```c
-(double)DAGetPageHeight:(int)FileHandle :(int)PageRef;
```

**Parameters**

- **FileHandle**
  A handle returned by the *DAOpenFile, DAOpenFileReadOnly* or *DAOpenFromStream* functions

- **PageRef**
  A page reference returned by the *DAFindPage* or *DANewPage* functions
**DAGetPageImageList**

*Image handling, Direct access functionality, Page properties*

**Description**

Finds all the images on the specified page and stores them in an image list.

**Syntax**

**Dylib**

```c
int DPLDAGetPageImageList(int InstanceID, int FileHandle, int PageRef);
```

**Objective-C class**

```objc
- (int)DAGetPageImageList:(int)FileHandle :(int)PageRef;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <code>DAFindPage</code> function</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The FileHandle or PageRef parameters were invalid</td>
</tr>
<tr>
<td>Non-zero</td>
<td>An ImageListID value that can be used with the other direct access image list functions</td>
</tr>
</tbody>
</table>
**DAGetPageWidth**

**Direct access functionality, Page properties**

**Description**

Returns the width of the specified page in a document opened with the *DAOpenFile* function.

**Syntax**

**Dylib**

```c
double DPLDAGetPageWidth(int InstanceID, int FileHandle, int PageRef);
```

**Objective-C class**

```c
- (double)DAGetPageWidth:(int)FileHandle :(int)PageRef;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <em>DAOpenFile</em>, <em>DAOpenFileReadOnly</em> or <em>DAOpenFromStream</em> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <em>DAFindPage</em> or <em>DANewPage</em> functions</td>
</tr>
</tbody>
</table>
**DAGetTextBlockBound**

**Text, Extraction, Direct access functionality**

**Description**

Returns one of the bounds of the specified text block.

**Syntax**

**Dylib**

```c
double DPLDAGetTextBlockBound(int InstanceID, int TextBlockListID, int Index, int BoundIndex);
```

**Objective-C class**

```objective-c
- (double)DAGetTextBlockBound:(int)TextBlockListID :(int)Index :(int)BoundIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the <strong>DAExtractPageTextBlocks</strong> or <strong>ExtractFilePageTextBlocks</strong> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
| BoundIndex      | 1 = Bottom left horizontal coordinate  
                              2 = Bottom left vertical coordinate  
                              3 = Top left horizontal coordinate  
                              4 = Top left vertical coordinate  
                              5 = Top right horizontal coordinate  
                              6 = Top right vertical coordinate  
                              7 = Bottom right horizontal coordinate  
                              8 = Bottom right vertical coordinate |
**DAGetTextBlockCharWidth**

**Text, Fonts, Extraction, Direct access functionality**

**Description**

Returns the width of a particular character within the specified text block.

**Syntax**

**Dylib**

```c
double DPLDAGetTextBlockCharWidth(int InstanceID, int TextBlockListID, int Index, int CharIndex);
```

**Objective-C class**

```objective-c
-(double)DAGetTextBlockCharWidth:(int)TextBlockListID :(int)Index :(int)CharIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TextBlockListID</strong></td>
<td>A value returned by the <a href="#">DAExtractPageTextBlocks</a> or <a href="#">ExtractFilePageTextBlocks</a> functions</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
<tr>
<td><strong>CharIndex</strong></td>
<td>The index of the character to retrieve the width of. The first character has an index of 1.</td>
</tr>
</tbody>
</table>
**DAGetTextBlockColor**

**Text, Extraction, Color, Direct access functionality**

**Description**

Returns one component of the color of the text in the specified text block. The color component value is returned as a value between 0 and 1.

**Syntax**

**Dylib**

```c
double DPLDAGetTextBlockColor(int InstanceID, int TextBlockListID, int Index, int ColorComponent);
```

**Objective-C class**

```objective-c
- (double)DAGetTextBlockColor:(int)TextBlockListID :(int)Index :(int)ColorComponent;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>TextBlockListID</strong></th>
<th>A value returned by the <strong>DAExtractPageTextBlocks</strong> or <strong>ExtractFilePageTextBlocks</strong> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
| **ColorComponent**  | For RGB: 1 = Red  
|                     | 2 = Green  
|                     | 3 = Blue  
|                     | For CMYK: 1 = Cyan  
|                     | 2 = Magenta  
|                     | 3 = Yellow  
|                     | 4 = Black |


**DAGetTextColorType**

**Text, Extraction, Color, Direct access functionality**

**Description**

Returns the type of color of the text in the specified text block.

**Syntax**

**Dylib**

```c
int DPLDAGetTextColorType(int InstanceID, int TextBlockListID, int Index);
```

**Objective-C class**

```objc
- (int)DAGetTextColorType:(int)TextBlockListID :(int)Index;
```

**Parameters**

- **TextBlockListID**  
  A value returned by the [DAExtractPageTextBlocks](#) or [ExtractFilePageTextBlocks](#) functions

- **Index**  
  The index of the text block. The first text block in the list has an index of 1.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>RGB</td>
</tr>
<tr>
<td>4</td>
<td>CMYK</td>
</tr>
</tbody>
</table>
**DAGetTextBlockCount**

*Text, Extraction, Direct access functionality*

**Description**

Returns the number of text blocks in the specified text block list.

**Syntax**

**Dylib**

```c
int DPLDAGetTextBlockCount(int InstanceID, int TextBlockListID);
```

**Objective-C class**

```objc
- (int)DAGetTextBlockCount:(int)TextBlockListID;
```

**Parameters**

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the <a href="#">DAExtractPageTextBlocks</a> or <a href="#">ExtractFilePageTextBlocks</a> functions</th>
</tr>
</thead>
</table>
**Description**

Returns the font name of the text in the specified text block.

**Syntax**

**Dylib**

```c
wchar_t * DPLDAGetTextBlockFontName(int InstanceID, int TextBlockListID, int Index);
```

**Objective-C class**

```objc
- (NSString *)DAGetTextBlockFontName:(int)TextBlockListID :(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the DAExtractPageTextBlocks or ExtractFilePageTextBlocks functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
DAGetTextBlockFontSize

Description

Returns the font size of the text in the specified text block.

Syntax

**Dylib**

```c
double DPLDAGetTextBlockFontSize(int InstanceID, int TextBlockListID, int Index);
```

**Objective-C class**

```objective-c
- (double)DAGetTextBlockFontSize:(int)TextBlockListID :(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the <a href="#">DAExtractPageTextBlocks</a> or <a href="#">ExtractFilePageTextBlocks</a> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
**DAGetTextBlockText**

**Text, Extraction, Direct access functionality**

**Description**

Returns the text in the specified text block.

**Syntax**

**Dylib**

```c
wchar_t * DPLDAGetTextBlockText(int InstanceID, int TextBlockListID, int Index);
```

**Objective-C class**

```objc
- (NSString *)DAGetTextBlockText:(int)TextBlockListID :(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextBlockListID</td>
<td>A value returned by the <a href="#">DAExtractPageTextBlocks</a> or <a href="#">ExtractFilePageTextBlocks</a> functions</td>
</tr>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
DAHasPageBox
Direct access functionality, Page properties

Description
Determines if a page has a particular page boundary rectangle.

Syntax
Dylib

```c
int DPLDAHasPageBox(int InstanceID, int FileHandle, int PageRef,
                     int BoxIndex);
```

Objective-C class

```objective-c
-(int)DAHasPageBox:(int)FileHandle :(int)PageRef :(int)BoxIndex;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <code>DAFindPage</code> or <code>DANewPage</code> functions</td>
</tr>
</tbody>
</table>
| BoxIndex    | 1 = MediaBox  
              2 = CropBox  
              3 = BleedBox  
              4 = TrimBox  
              5 = ArtBox |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page does not have the specified page boundary rectangle</td>
</tr>
<tr>
<td>1</td>
<td>The page has the specified page boundary rectangle</td>
</tr>
</tbody>
</table>
## DAHidePage

**Direct access functionality, Page manipulation**

**Description**

Hides the specified page from a document originally opened with **DAOpenFile**. The content of the page is still in the document, but the page will not be visible.

**Syntax**

**Dylib**

```c
int DPLDAHidePage(int InstanceID, int FileHandle, int PageRef);
```

**Objective-C class**

```objc
- (int)DAHidePage:(int)FileHandle :(int)PageRef;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FileHandle</code></td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td><code>PageRef</code></td>
<td>A page reference returned by the <strong>DAFindPage</strong> or <strong>DANewPage</strong> functions</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle or PageRef were not valid</td>
</tr>
<tr>
<td>1</td>
<td>The page was hidden successfully</td>
</tr>
</tbody>
</table>
DAMovePage
Direct access functionality, Page manipulation

Description

Moves a page to a new location in the document.

Syntax

Dylib

    int DPLDAMovePage(int InstanceID, int FileHandle, int PageRef,
                      int TargetPageRef, int Options);

Objective-C class

    - (int)DAMovePage:(int)FileHandle :(int)PageRef :(int)TargetPageRef
                     :(int)Options;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the DAFindPage or DANewPage functions. This is the page that will be moved.</td>
</tr>
<tr>
<td>TargetPageRef</td>
<td>A page reference returned by the DAFindPage or DANewPage functions. The page will be moved before or after this page.</td>
</tr>
</tbody>
</table>
| Options        | 0 = Move before target page  
                    1 = Move after target page                                      |

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page could not be moved. Check that the FileHandle, PageRef and TargetPageRef values are correct.</td>
</tr>
<tr>
<td>1</td>
<td>The page was moved successfully</td>
</tr>
</tbody>
</table>
**DANewPage**

Direct access functionality, Page manipulation

**Description**

Adds a new blank page to the end of the document. The page will have a standard size of 612x792 points.

**Syntax**

**Dylib**

```c
int DPLDANewPage(int InstanceID, int FileHandle);
```

**Objective-C class**

```objective-c
-(int)DANewPage:(int)FileHandle;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle was not valid</td>
</tr>
<tr>
<td>Non-zero</td>
<td>An ID that can be used as the PageRef parameter for any of the direct access functions</td>
</tr>
</tbody>
</table>
DANewPages
Direct access functionality, Page manipulation

Description

Adds a number of new pages to the end of the document. All new pages have a standard size of 612x792 points.

Syntax

Dylib

```c
int DPLDANewPages(int InstanceID, int FileHandle, int PageCount);
```

Objective-C class

```objc
- (int)DANewPages:(int)FileHandle :(int)PageCount;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</td>
</tr>
<tr>
<td>PageCount</td>
<td>The number of pages to add to the document</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle was not valid</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The total number of pages in the document after the new pages were added</td>
</tr>
</tbody>
</table>
**DAOpenFile**

**Document management, Direct access functionality**

**Description**

Opens a file in direct access mode. This allows large files to be processed. The file will not be accessible by other processes until the file is closed using the one of the following functions: **DACloseFile**, **DAAappendFile** or **DASaveAsFile**. Read only files can be opened and all other direct access functions will work but **DAAappendFile** will not work as the file cannot be written.

**Syntax**

**Dylib**

```c
int DPLDAOpenFile(int InstanceID, wchar_t * InputFileName, wchar_t * Password);
```

**Objective-C class**

```objc
-(int)DAOpenFile:(NSString *)InputFileName :(NSString *)Password;
```

**Parameters**

<table>
<thead>
<tr>
<th>InputFileName</th>
<th>The path and name of the document to open in direct access mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>The password to use when opening the document. This can be the owner or user password. If the user password is used certain functionality may be restricted depending on the permissions of the document.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The file could not be opened. Use the <strong>LastErrorCode</strong> function to determine the cause of the failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>A FileHandle that can be used with the other Direct Access functions</td>
</tr>
</tbody>
</table>
**DAOpenFileReadOnly**

**Document management, Direct access functionality**

**Description**

Opens a file in direct access mode. This allows large files to be processed. The file is opened with read only access so other processes will also be able to open the file in read only mode. **DASaveAsFile** should be used to save any changes to a new file as **DAAppendFile** cannot update read only files.

**Syntax**

**Dylib**

```c
int DPLDAOpenFileReadOnly(int InstanceID, wchar_t * InputFileName, wchar_t * Password);
```

**Objective-C class**

```objective-c
-(int)DAOpenFileReadOnly:(NSString *)InputFileName :(NSString *)Password;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputFileName</strong></td>
<td>The path and name of the document to open in direct access mode with read only access.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>The password to use when opening the document. This can be the owner or user password. If the user password is used certain functionality may be restricted depending on the permissions of the document.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file could not be opened. Use the <strong>LastErrorCode</strong> function to determine the cause of the failure.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A FileHandle that can be used with the other Direct Access functions</td>
</tr>
</tbody>
</table>
**DAPageRotation**

**Direct access functionality, Page properties**

**Description**

Returns the rotation of the specified page.

**Syntax**

**Dylib**

```c
int DPLDAPageRotation(int InstanceID, int FileHandle, int PageRef);
```

**Objective-C class**

```c
-(int)DAPageRotation:(int)FileHandle :(int)PageRef;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <a href="#">DAOpenFile</a>, <a href="#">DAOpenFileReadOnly</a> or <a href="#">DAOpenFromStream</a> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <a href="#">DAFindPage</a> or <a href="#">DANewPage</a> functions</td>
</tr>
</tbody>
</table>
**DAResleaseImageList**

**Image handling, Direct access functionality, Page properties**

**Description**

Releases the specified image list including all the image data extracted from the images in the list. Releasing the image list does not affect the original images.

**Syntax**

**Dylib**

```c
int DPLDAResleaseImageList(int InstanceID, int FileHandle, int ImageListID);
```

**Objective-C class**

```c
-(int)DAResleaseImageList:(int)FileHandle :(int)ImageListID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>ImageListID</td>
<td>A value returned by the <strong>DAGetPageImageList</strong> function</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image list could not be released. The ImageListID parameter might be invalid or does not refer to an image list within the specified document.</td>
</tr>
<tr>
<td>1</td>
<td>The image list was released successfully.</td>
</tr>
</tbody>
</table>
**DAReleaseTextBlocks**

**Description**

Releases the memory used by a text block list.

**Syntax**

**Dylib**

```c
int DPLDAReleaseTextBlocks(int InstanceID, int TextBlockListID);
```

**Objective-C class**

```objc
- (int)DAReleaseTextBlocks:(int)TextBlockListID;
```

**Parameters**

| TextBlockListID | A value returned by the **DAExtractPageTextBlocks** or **ExtractFilePageTextBlocks** functions |
**DARemoveUsageRights**

**Document manipulation, Direct access functionality**

**Description**

Removes any usage rights from the document.

**Syntax**

**Dylib**

```c
int DPLDARemoveUsageRights(int InstanceID, int FileHandle);
```

**Objective-C class**

```swift
-(int)DARemoveUsageRights:(int)FileHandle;
```

**Parameters**

<table>
<thead>
<tr>
<th>FileHandle</th>
<th>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</th>
</tr>
</thead>
</table>

**Return values**

| 0 | The document did not have any usage rights |
| 1 | Success |
**DARenderPageToDC**

Direct access functionality, Rendering and printing

**Description**

Renders the specified page from the specified document directly onto a graphics surface.

**Syntax**

**Dylib**

```c
int DPLDARenderPageToDC(int InstanceID, int FileHandle, int PageRef, double DPI, int DC);
```

**Objective-C class**

```c
- (int)DARenderPageToDC:(int)FileHandle :(int)PageRef :(double)DPI :(int)DC;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <strong>DAFindPage</strong> or <strong>DANewPage</strong> functions</td>
</tr>
<tr>
<td>DPI</td>
<td>The DPI to use when rendering the page</td>
</tr>
<tr>
<td>DC</td>
<td>The device context handle</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page could not be rendered</td>
</tr>
<tr>
<td>1</td>
<td>The page was rendered successfully</td>
</tr>
</tbody>
</table>
**DARenderPageToFile**

Direct access functionality, Rendering and printing

**Description**

Renders the specified page from the specified document to an image and saves the image data as a file on disk.

**Syntax**

**Dylib**

```c
int DPLDARenderPageToFile(int InstanceID, int FileHandle, int PageRef, int Options, double DPI, wchar_t * FileName);
```

**Objective-C class**

```c
-(int)DARenderPageToFile:(int)FileHandle :(int)PageRef :(int)Options :(double)DPI :(NSString *)FileName;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <a href="#">DAOpenFile</a>, <a href="#">DAOpenFileReadOnly</a> or <a href="#">DAOpenFromStream</a> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <a href="#">DAFindPage</a> or <a href="#">DANewPage</a> functions</td>
</tr>
</tbody>
</table>
| Options     | 0 = BMP output  
1 = JPEG output  
2 = WMF output  
3 = EMF output  
4 = EPS output  
5 = PNG output  
6 = GIF output  
7 = TIFF output  
8 = EMF+ output  
9 = HTML5 output  
10 = G4 TIFF output |
| DPI         | The DPI to use when rendering the page. Values over 300 will cause excessive memory usage.                                                    |
| FileName    | The path and file name of the file to create to store the rendered page image data in.                                                        |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page could not be rendered. Check the value of the FileHandle and PageRef parameters.</td>
</tr>
<tr>
<td>1</td>
<td>The page was rendered correctly and the image file was saved to disk</td>
</tr>
<tr>
<td>2</td>
<td>The file could not be written to disk</td>
</tr>
</tbody>
</table>
**DARenderPageToString**

**Direct access functionality, Rendering and printing**

**Description**

It renders the specified page from the specified document to an image and returns the image data as a string.

**Syntax**

**Dylib**

```c
char * DPLDARenderPageToString(int InstanceID, int FileHandle,
                               int PageRef, int Options, double DPI);
```

**Objective-C class**

```c
- (NSData *)DARenderPageToString:(int)FileHandle :(int)PageRef
  :(int)Options :(double)DPI;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the <strong>DAFindPage</strong> or <strong>DANewPage</strong> functions</td>
</tr>
</tbody>
</table>
| Options | 0 = BMP output  
1 = JPEG output  
2 = WMF output  
3 = EMF output  
4 = EPS output  
5 = PNG output  
6 = GIF output  
7 = TIFF output  
8 = EMF+ output  
9 = HTML5 output  
10 = G4 TIFF output |
| DPI | The DPI to use when rendering the page. Values over 300 will cause excessive memory usage. |
**DARotatePage**

**Direct access functionality, Page properties**

**Description**

Sets the rotation of the selected page. The rotation is only applicable to the viewed page, the co-ordinate system rotates with the page.

**Syntax**

**Dylib**

```
int DPLDARotatePage(int InstanceID, int FileHandle, int PageRef, int Angle, int Options);
```

**Objective-C class**

```
-(int)DARotatePage:(int)FileHandle :(int)PageRef :(int)Angle :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileHandle</strong></td>
<td>A handle returned by the <code>DAOpenFile</code>, <code>DAOpenFileReadOnly</code> or <code>DAOpenFromStream</code> functions</td>
</tr>
<tr>
<td><strong>PageRef</strong></td>
<td>A page reference returned by the <code>DAFindPage</code> or <code>DANewPage</code> functions</td>
</tr>
<tr>
<td><strong>Angle</strong></td>
<td>The clockwise angle in degrees to rotate the page by, must be a multiple of 90</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Reserved for future use. Must be set to 0.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page rotation could not be set. Check that the FileHandle and PageRef parameters are correct, and ensure that the Angle parameter is a multiple of 90.</td>
</tr>
<tr>
<td>1</td>
<td>The page rotation was set successfully</td>
</tr>
</tbody>
</table>
**DASaveAsFile**

**Document management, Direct access functionality**

**Description**

Rewrites the entire file, including all changes, to a new file. This operation may take some time with large files or files with many objects. The original file is closed after this operation and the file handle will no longer be valid. The original file cannot be overwritten. Use **DAAppendFile** if you want to append changes to original file.

**Syntax**

**Dylib**

```c
int DPLDASaveAsFile(int InstanceID, int FileHandle, wchar_t * OutputFileName);
```

**Objective-C class**

```objective-c
- (int)DASaveAsFile:(int)FileHandle :(NSString *)OutputFileName;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>The path and name of the new document to create.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The new file could not be created</td>
</tr>
<tr>
<td>1</td>
<td>The document was saved to the new file successfully</td>
</tr>
</tbody>
</table>
DASaveImageDataToFile

Image handling, Direct access functionality

Description

Saves an image in an image list to a file on disk. The type of image file depends on the type of the image. The **DAGetImageIntProperty** function can be used to determine the image type.

Syntax

**Dylib**

```c
int DPLDASaveImageDataToFile(int InstanceID, int FileHandle,
                                int ImageListID, int ImageIndex, wchar_t * ImageFileName);
```

**Objective-C class**

```c
- (int)DASaveImageDataToFile:(int)FileHandle :(int)ImageListID
                         :(int)ImageIndex :(NSString *)ImageFileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the <strong>DAOpenFile</strong>, <strong>DAOpenFileReadOnly</strong> or <strong>DAOpenFromStream</strong> functions</td>
</tr>
<tr>
<td>ImageListID</td>
<td>A value returned by the <strong>DAGetPageImageList</strong> function</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image. The first image in the list has an index of 1. Use the <strong>DAGetImageListCount</strong> function to determine the number of images in the list.</td>
</tr>
<tr>
<td>ImageFileName</td>
<td>The path and file name of the file to create to store the image data in.</td>
</tr>
</tbody>
</table>
DASetInformation

Document properties, Direct access functionality

Description

Sets values in the document information section. This could be standard information such as Author and Subject, or custom information.

For CreationDate and ModDate (modification date), the format of the date should be:

D:YYYYMMDDHhmSSOHH'mm'

where

YYYY shall be the year
MM shall be the month (01?12)
DD shall be the day (01?31)
HH shall be the hour (00?23)
mm shall be the minute (00?59)
SS shall be the second (00?59)
O shall be the relationship of local time to Universal Time (UT) using a +, - or Z character
HH followed by APOSTROPHE (U+0027) (') shall be the absolute value of the offset from UT in hours (00?23)
mm followed by an optional APOSTROPHE (U+0027) (') shall be the absolute value of the offset from UT in minutes (00?59)

Syntax

Dylib

```c
int DPLDASetInformation(int InstanceID, int FileHandle, wchar_t * Key, wchar_t * NewValue);
```

Objective-C class

```objc
- (int)DASetInformation:(int)FileHandle :(NSString *)Key : (NSString *)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td>Key</td>
<td>For standard information use &quot;Author&quot;, &quot;Title&quot;, &quot;Subject&quot;, &quot;Keywords&quot;, &quot;Creator&quot;, &quot;Producer&quot;, &quot;CreationDate&quot; or &quot;ModDate&quot;. For custom information any other string can be used.</td>
</tr>
<tr>
<td>NewValue</td>
<td>The new value for the specified key.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The information key was set or updated successfully</td>
</tr>
</tbody>
</table>
**Description**

Sets the dimensions of the specified page's boundary rectangles.

- The **MediaBox** represents the physical medium of the page.
- The **CropBox** represents the visible region of the page, the contents will be clipped to this region.
- The **BleedBox** is similar to the **CropBox**, but is the rectangle used in a production environment.
- The **TrimBox** indicates the intended dimensions of the finished page after trimming, and the **ArtBox** defines the extent of the page's meaningful content as intended by the page's creator.

**Syntax**

**Dylib**

```c
int DPLDASetPageBox(int InstanceID, int FileHandle, int PageRef,
                     int BoxIndex, double X1, double Y1, double X2, double Y2);
```

**Objective-C class**

```objc
-(int)DASetPageBox:(int)FileHandle :(int)PageRef :(int)BoxIndex
:(double)X1 :(double)Y1 :(double)X2 :(double)Y2;
```

**Parameters**

- **FileHandle** A handle returned by the **DAOpenFile, DAOpenFileReadOnly** or **DAOpenFromStream** functions
- **PageRef** A page reference returned by the **DAFindPage** or **DANewPage** functions
- **BoxIndex**
  - 1 = **MediaBox**
  - 2 = **CropBox**
  - 3 = **BleedBox**
  - 4 = **TrimBox**
  - 5 = **ArtBox**
- **X1** The horizontal coordinate of the bottom left corner of the box measured in points from the left edge of the page
- **Y1** The vertical coordinate of the bottom left corner of the box measured in points from the bottom of the page
- **X2** The horizontal coordinate of the top right corner of the box measured in points from the bottom of the page
- **Y2** The vertical coordinate of the top right corner of the box measured in points from the bottom of the page

**Return values**

- 0 The FileHandle or PageRef parameters were invalid
- 1 Success
**DASetPageSize**

**Direct access functionality, Page properties**

**Description**

Sets the specified page to have a certain width and height.

**Syntax**

**Dylib**

```c
int DPLDASetPageSize(int InstanceID, int FileHandle, int PageRef,
                      double PntWidth, double PntHeight);
```

**Objective-C class**

```objc
- (int)DASetPageSize:(int)FileHandle :(int)PageRef :(double)PntWidth
                    :(double)PntHeight;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileHandle</td>
<td>A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions</td>
</tr>
<tr>
<td>PageRef</td>
<td>A page reference returned by the DAFindPage or DANewPage functions</td>
</tr>
<tr>
<td>PntWidth</td>
<td>The new width of the page, measured in points</td>
</tr>
<tr>
<td>PntHeight</td>
<td>The new height of the page, measured in points</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FileHandle or PageRef were not valid</td>
</tr>
<tr>
<td>1</td>
<td>The page size was set successfully</td>
</tr>
</tbody>
</table>
**DASetTextExtractionArea**  
**Text, Extraction, Direct access functionality**

**Description**
Sets the area for certain modes of text extraction. Any text that appears outside this area will be excluded from the results. This function has no effect on text extraction using modes 0 to 2.

This function affects the results of the `ExtractFilePageText` and `DAExtractPageText` functions only.

The coordinate values passed into this function are specified in points with the bottom left corner of the page as the origin.

The area limitation can be removed by calling this function with a value of zero for both the Width and Height parameters.

**Syntax**

**Dylib**

```c
int DPLDASetTextExtractionArea(int InstanceID, double Left, double Top, double Width, double Height);
```

**Objective-C class**

```c
-(int)DASetTextExtractionArea:(double)Left :(double)Top :(double)Width :(double)Height;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal coordinate of the left edge of the area</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical coordinate of the top edge of the area</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the area</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the area</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The text extraction area was set successfully</td>
</tr>
<tr>
<td>2</td>
<td>The text extraction area was cleared</td>
</tr>
</tbody>
</table>
**Description**

Sets various options that affect the text extraction functionality.
This function affects the results of the `ExtractFilePageText` and `DAExtractPageText` functions only.

**Syntax**

**Dylib**

```
int DPLDASetTextExtractionOptions(int InstanceID, int OptionID,
int NewValue);
```

**Objective-C class**

```
-(int)DASetTextExtractionOptions:(int)OptionID :(int)NewValue;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>OptionID</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use Font information matching when grouping to separate text blocks</td>
</tr>
<tr>
<td>2</td>
<td>Use Color information matching when grouping to separate text blocks</td>
</tr>
<tr>
<td>3</td>
<td>Use Block information matching when grouping to separate text blocks</td>
</tr>
<tr>
<td>4</td>
<td>Output CMYK color values</td>
</tr>
<tr>
<td>5</td>
<td>Sort text blocks based on top left position</td>
</tr>
<tr>
<td>6</td>
<td>Descenders from font metrics</td>
</tr>
<tr>
<td>7</td>
<td>Ignore overlaps</td>
</tr>
<tr>
<td>8</td>
<td>Ignore duplicates</td>
</tr>
<tr>
<td>9</td>
<td>Split on double space</td>
</tr>
<tr>
<td>10</td>
<td>Trim characters outside area</td>
</tr>
<tr>
<td>11</td>
<td>Alternative block matching</td>
</tr>
<tr>
<td>12</td>
<td>Ignore rotated text blocks</td>
</tr>
<tr>
<td>13</td>
<td>Trim leading and trailing whitespace from text blocks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>NewValue</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For OptionID = 1, 2, 3 and 6:</td>
<td>0 = Use, 1 = Ignore</td>
</tr>
<tr>
<td>For OptionID = 4:</td>
<td>0 = Show as RGB (default), 1 = Show as CMYK</td>
</tr>
<tr>
<td>For OptionID = 5:</td>
<td>0 = Do not sort blocks (default), 1 = Sort blocks</td>
</tr>
<tr>
<td>For OptionID = 7, 8 and 12:</td>
<td>0 = Do not ignore, 1 = Ignore</td>
</tr>
<tr>
<td>OptionID = 9:</td>
<td>0 = Do not split on double space (default)</td>
</tr>
<tr>
<td>OptionID = 10:</td>
<td>0 = Do not trim characters outside area (default)</td>
</tr>
<tr>
<td>OptionID = 11:</td>
<td>0 = Regular block matching</td>
</tr>
<tr>
<td>OptionID = 13:</td>
<td>0 = Do not trim leading or trailing whitespace</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The OptionID or NewValue parameter was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The text extraction option was set successfully</td>
</tr>
</tbody>
</table>
**DASetTextExtractionScaling**

**Text, Extraction, Direct access functionality**

**Description**

Sets the scaling to use for text extraction in Mode 7. This controls the number of rows and columns in the monospaced text output.

This function affects the results of the `ExtractFilePageText` and `DAExtractPageText` functions only.

**Syntax**

**Dylib**

```c
int DPLDASetTextExtractionScaling(int InstanceID, int Options, double Horizontal, double Vertical);
```

**Objective-C class**

```objc
- (int)DASetTextExtractionScaling:(int)Options :(double)Horizontal : (double)Vertical;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>Options</strong></th>
<th>Should always be set to 0. This indicates a scaling factor will be set for the Horizontal and Vertical parameters, with a default value of 5 for horizontal and 8 for vertical. Smaller values stretch the text out into more rows/columns.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Horizontal</strong></td>
<td>The scaling to use for the horizontal axis in units defined by the Options parameter.</td>
</tr>
<tr>
<td><strong>Vertical</strong></td>
<td>The scaling to use for the vertical axis in units defined by the Options parameter.</td>
</tr>
</tbody>
</table>

**Return values**

| **0** | The Options parameter was not valid or a value less than 1 was used for the Horizontal or Vertical parameters. |
| **1** | Text extraction scaling was set successfully. |
**Description**

Sets the word gap ratio for the text extraction functionality. This function affects the results of the `ExtractFilePageText` and `DAExtractPageText` functions only.

**Syntax**

**Dylib**

```c
int DPLDASetTextExtractionWordGap(int InstanceID, double NewWordGap);
```

**Objective-C class**

```c
- (int)DASetTextExtractionWordGap:(double)NewWordGap;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NewWordGap</strong></td>
<td>The new WordGap ratio</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The word gap ratio was set successfully.</td>
</tr>
</tbody>
</table>
DAShiftedHeader

Document management, Direct access functionality

Description

Returns a value to determine if the source PDF was malformed due to byte shifting. For example, leading whitespace added to the file.

In such a case the file will be loaded taking this offset into account. This function will return a non-zero number indicating the number of bytes the file was shifted by.

Note that if the file is loaded this way it will not be possible to use the DAAppendFile function to add an incremental update.

Syntax

Dylib

```c
int DPLDAShiftedHeader(int InstanceID, int FileHandle);
```

Objective-C class

```objective-c
- (int)DAShiftedHeader:(int)FileHandle;
```

Parameters

| FileHandle | A handle returned by the DAOpenFile, DAOpenFileReadOnly or DAOpenFromStream functions |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The file was loaded as usual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>The number of bytes the file was shifted by</td>
</tr>
</tbody>
</table>
Decrypt

Document properties, Security and Signatures

Description

This function attempts to remove the encryption setting from the selected document using the password provided when originally opening the document.

This function will succeed even if the user password was used (including an valid blank password) rather than the master password. Developers are advised that they should respect the security wishes of the document’s author.

Syntax

**Dylib**

```c
int DPLDecrypt(int InstanceID);
```

**Objective-C class**

```c
- (int)Decrypt
```

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document could not be decrypted</td>
</tr>
<tr>
<td>1</td>
<td>The document was decrypted successfully</td>
</tr>
</tbody>
</table>
**DecryptFile**

**Document management, Security and Signatures**

**Description**

This function attempts to remove the encryption from a file on disk, saving the decrypted document to a new file.

This function will succeed even if the user password is supplied (including an valid blank password) rather than the master password. Developers are advised that they should respect the security wishes of the document's author.

**Syntax**

**Dylib**

```c
int DPLDecryptFile(int InstanceID, wchar_t * InputFileName,
       wchar_t * OutputFileName, wchar_t * Password);
```

**Objective-C class**

```objective-c
- (int)DecryptFile:(NSString *)InputFileName :(NSString *)OutputFileName
       :(NSString *)Password;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputFileName</strong></td>
<td>The name of the file to decrypt.</td>
</tr>
<tr>
<td><strong>OutputFileName</strong></td>
<td>The name of the destination file to create. If this file already exists it will be overwritten.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>The password to use when decrypting the file.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document could not be decrypted. Check the result of the <strong>LastErrorCode</strong> function to determine the cause of the failure.</td>
</tr>
<tr>
<td>1</td>
<td>The document was decrypted successfully</td>
</tr>
</tbody>
</table>
DeleteAnalysis

Document properties

Description

Removes a set of analysis results from memory. Call this function after calling AnalyseFile and GetAnalysisInfo when you no longer need the information.

Syntax

Dylib

```c
int DPLDeleteAnalysis(int InstanceID, int AnalysisID);
```

Objective-C class

```objc
- (int)DeleteAnalysis:(int)AnalysisID;
```

Parameters

| AnalysisID | The ID of the set of analysis results to delete, as returned by the AnalyseFile function |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The specified analysis ID was not valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The set of analysis results with the specified ID was deleted successfully</td>
</tr>
</tbody>
</table>
DeleteAnnotation
Annotations and hotspot links

Description
Removes an annotation from the selected page.

Syntax

Dylib

```c
int DPLDeleteAnnotation(int InstanceID, int Index);
```

Objective-C class

```c
- (int)DeleteAnnotation:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation to delete. The first annotation on the page has an index of 1. The <code>AnnotationCount</code> function returns the total number of annotations on the selected page.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified annotation could not be deleted. Check the value of the Index parameter is between 1 and the value returned by the <code>AnnotationCount</code> function.</td>
</tr>
<tr>
<td>1</td>
<td>The specified annotation was deleted from the page successfully.</td>
</tr>
</tbody>
</table>
DeleteContentStream
Content Streams and Optional Content Groups

Description
A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.
This function removes the specified content stream part that was selected with the SelectContentStream function.

Syntax
Dylib
`int DPLDeleteContentStream(int InstanceID);`

Objective-C class
```
- (int)DeleteContentStream
```

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The content stream part could not be deleted</td>
</tr>
<tr>
<td>1</td>
<td>The content stream part was deleted successfully</td>
</tr>
</tbody>
</table>
DeleteFormField

Form fields

Description

Deletes the specified form field. If the field is deleted successfully the field index of subsequent form fields will be decreased by 1.

Syntax

Dylib

```
int DPLDeleteFormField(int InstanceID, int Index);
```

Objective-C class

```
- (int)DeleteFormField:(int)Index;
```

Parameters

| Index       | The index of the form field to delete |

Return values

| 0           | The form field was not found          |
| 1           | The form field was deleted successfully |
DeleteOptionalContentGroup

Description

Deletes an optional content group.

Syntax

Dylib

```c
int DPLDeleteOptionalContentGroup(int InstanceID, int OptionalContentGroupID);
```

Objective-C class

```objective-c
- (int)DeleteOptionalContentGroup:(int)OptionalContentGroupID;
```

Parameters

| OptionalContentGroupID | An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions |
DeletePageLGIDict

Page properties, Measurement and coordinate units

Description

Deletes the specified LGIDict dictionary from the selected page.

Syntax

Dylib

```c
int DPLDeletePageLGIDict(int InstanceID, int DictIndex);
```

Objective-C class

```objective-c
-(int)DeletePageLGIDICT:(int)DictIndex;
```

Parameters

| DictIndex | The index of the LGIDict dictionary to delete. The first dictionary has an index of 1. Use the `GetPageLGIDictCount` function to determine the number of LGIDict dictionaries attached to the selected page. |

Return values

| 0 | The dictionary could not be deleted. Check that the DictIndex parameter is in range. |
| 1 | The specified dictionary was deleted successfully. |
**DeletePages**

**Page manipulation**

**Description**

Removes one or more pages from the document. The document will always have at least one page.

**Syntax**

**Dylib**

```c
int DPLDeletePages(int InstanceID, int StartPage, int PageCount);
```

**Objective-C class**

```
-(int)DeletePages:(int)StartPage :(int)PageCount;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPage</td>
<td>The page number of the first page to delete</td>
</tr>
<tr>
<td>PageCount</td>
<td>The total number of pages to delete. The value will be automatically adjusted if necessary so that the document is left with at least one page.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The PageCount parameter was 0 or there was only a single page in the document.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of pages remaining in the original document.</td>
</tr>
</tbody>
</table>
DocJavaScriptAction

Document properties, JavaScript

Description

This function is used to add JavaScript to document events. This JavaScript will be executed when, for example, the document is closed or printed.

Syntax

Dylib

int DPLDocJavaScriptAction(int InstanceID, wchar_t * ActionType, wchar_t * JavaScript);

Objective-C class

- (int)DocJavaScriptAction:(NSString *)ActionType :(NSString *)JavaScript;

Parameters

<table>
<thead>
<tr>
<th>ActionType</th>
<th>The event to attach the JavaScript to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;WC&quot;</td>
<td>Will close</td>
</tr>
<tr>
<td>&quot;WS&quot;</td>
<td>Will save</td>
</tr>
<tr>
<td>&quot;DS&quot;</td>
<td>Did save</td>
</tr>
<tr>
<td>&quot;WP&quot;</td>
<td>Will print</td>
</tr>
<tr>
<td>&quot;DP&quot;</td>
<td>Did print</td>
</tr>
</tbody>
</table>

| JavaScript | The JavaScript to attach to the event. |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified ActionType was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The JavaScript was added successfully</td>
</tr>
</tbody>
</table>
DocumentCount

Document management

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Returns the total number of documents.

When an instance of Quick PDF Library is first created a blank one page document is automatically created so the document count will be 1. Each time a new document is created or loaded the document count will be increased. The RemoveDocument function will only succeed if there are at least two documents loaded, so the document count will always be at least 1.

Syntax

Dylib

```c
int DPLDocumentCount(int InstanceID);
```

Objective-C class

```c
-(int)DocumentCount
```
**DrawArc**

**Vector graphics**

**Description**

Draw a circular arc on the selected page. The arc is drawn in a clockwise direction from StartAngle to EndAngle.

**Syntax**

**Dylib**

```c
int DPLDrawArc(int InstanceID, double XPos, double YPos, double Radius,
               double StartAngle, double EndAngle, int Pie, int DrawOptions);
```

**Objective-C class**

```objective-c
-(int)DrawArc:(double)XPos :(double)YPos :(double)Radius
:(double)StartAngle :(double)EndAngle :(int)Pie
:(int)DrawOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XPos</strong></td>
<td>Horizontal co-ordinate of the center of the arc</td>
</tr>
<tr>
<td><strong>YPos</strong></td>
<td>Vertical co-ordinate of center of the arc</td>
</tr>
<tr>
<td><strong>Radius</strong></td>
<td>Radius of the arc</td>
</tr>
<tr>
<td><strong>StartAngle</strong></td>
<td>Angle to start drawing from</td>
</tr>
<tr>
<td><strong>EndAngle</strong></td>
<td>Angle to end drawing at</td>
</tr>
<tr>
<td><strong>Pie</strong></td>
<td>Draw the arms of the arc:</td>
</tr>
<tr>
<td></td>
<td>0 = No</td>
</tr>
<tr>
<td></td>
<td>1 = Yes</td>
</tr>
<tr>
<td><strong>DrawOptions</strong></td>
<td>0 = Outline</td>
</tr>
<tr>
<td></td>
<td>1 = Fill</td>
</tr>
<tr>
<td></td>
<td>2 = Fill and Outline</td>
</tr>
<tr>
<td></td>
<td>3 = Close, Fill and Outline (if Pie = 1)</td>
</tr>
</tbody>
</table>
Draw a barcode on the selected page.
For Code128, the barcode is a combination of the "B" and "C" character sets resulting in the most compact representation.
GS1-128 barcodes (also known as EAN-128) can be drawn by setting the Barcode parameter to 3 (Code128) and using the string "[FNC1]" in the appropriate place. For example:
"[FNC1]21ABC123[FNC1]2013"
The previous example indicates a serial number (AI 21) of "ABC123" and a product variant (AI 20) of "13".

**Syntax**

Dylib

```c
int DPLDrawBarcode(int InstanceID, double Left, double Top, double Width,
                    double Height, wchar_t * Text, int Barcode, int Options);
```

Objective-C class

```c
- (int)DrawBarcode:(double)Left :(double)Top :(double)Width
            :(double)Height :(NSString *)Text :(int)Barcode :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>Horizontal co-ordinate of left edge of the barcode</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>Vertical co-ordinate of top edge of the barcode</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>Width of the barcode</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>Height of the barcode</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The barcode data. The barcode can be rotated by appending the following to the barcode data string:</td>
</tr>
<tr>
<td></td>
<td>/RC = Rotate clockwise</td>
</tr>
<tr>
<td></td>
<td>/RA = Rotate anti-clockwise</td>
</tr>
<tr>
<td></td>
<td>/RU = Rotate 180 degrees</td>
</tr>
<tr>
<td><strong>Barcode</strong></td>
<td>1 = Code39 (or Code 3 of 9)</td>
</tr>
<tr>
<td></td>
<td>2 = EAN-13</td>
</tr>
<tr>
<td></td>
<td>3 = Code128</td>
</tr>
<tr>
<td></td>
<td>4 = PostNet</td>
</tr>
<tr>
<td></td>
<td>5 = Interleaved 2 of 5</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Code39: 0 = Default drawing</td>
</tr>
<tr>
<td></td>
<td>EAN-13: 0 = Only draw the barcode</td>
</tr>
<tr>
<td></td>
<td>1 = Extend the guard bars</td>
</tr>
<tr>
<td></td>
<td>2 = Draw the human-readable numbers</td>
</tr>
<tr>
<td></td>
<td>3 = Draw the human-readable numbers, with right spacer</td>
</tr>
<tr>
<td></td>
<td>Code128: 0 = Default drawing</td>
</tr>
<tr>
<td></td>
<td>PostNet: 0 = Default drawing</td>
</tr>
<tr>
<td></td>
<td>Interleaved 2 of 5: 0 = Do not add a checksum, no bearer bars</td>
</tr>
<tr>
<td></td>
<td>1 = Add a checksum character, no bearer bars</td>
</tr>
<tr>
<td></td>
<td>2 = Do not add a checksum, draw bearer bars</td>
</tr>
<tr>
<td></td>
<td>3 = Add a checksum character, draw bearer bars</td>
</tr>
</tbody>
</table>

To apply 10% bar width reduction to the barcode, increase the value of the Options parameter by 10

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The barcode could not be drawn. Invalid Barcode or Options parameters.</td>
</tr>
<tr>
<td>1</td>
<td>The barcode was drawn successfully</td>
</tr>
</tbody>
</table>
**DrawBox**

**Vector graphics, Page manipulation**

**Description**

Draw a rectangle on the selected page.

**Syntax**

**Dylib**

```c
int DPLDrawBox(int InstanceID, double Left, double Top, double Width, double Height, int DrawOptions);
```

**Objective-C class**

```c
-(int)DrawBox:(double)Left :(double)Top :(double)Width :(double)Height :(int)DrawOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>Horizontal co-ordinate of left edge of rectangle</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>Vertical co-ordinate of top edge of rectangle</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>Rectangle width</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>Rectangle height</td>
</tr>
<tr>
<td><strong>DrawOptions</strong></td>
<td>0 = Outline 1 = Fill 2 = Fill and Outline</td>
</tr>
</tbody>
</table>
**DrawCapturedPage**

**Page layout**

**Description**

This function draws a page previously captured with the *CapturePage* function onto the current page. It can be drawn at any size and position, allowing for imposition of pages.

**Syntax**

**Dylib**

```c
int DPLDrawCapturedPage(int InstanceID, int CaptureID, double Left, double Top, double Width, double Height);
```

**Objective-C class**

```c
-(int)DrawCapturedPage:(int)CaptureID :(double)Left :(double)Top :(double)Width :(double)Height;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CaptureID</strong></td>
<td>The ID returned by the <em>CapturePage</em> function when a page was previously captured</td>
</tr>
<tr>
<td><strong>Left</strong></td>
<td>The co-ordinate of the left edge of the destination area</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The co-ordinate of the top edge of the destination area</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the destination area</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the destination area</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An invalid CaptureID was specified</td>
</tr>
<tr>
<td>1</td>
<td>The captured page was drawn successfully</td>
</tr>
</tbody>
</table>
DrawCapturedPageMatrix

Description

This function draws a page previously captured with the CapturePage function onto the current page. The size/position/rotation is specified using a transformation matrix, allowing for advanced imposition of pages.

Syntax

**Dylib**

```c
int DPLDrawCapturedPageMatrix(int InstanceID, int CaptureID, double M11, double M12, double M21, double M22, double MDX, double MDY);
```

**Objective-C class**

```objective-c
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaptureID</td>
<td>The ID returned by the CapturePage function when a page was previously captured</td>
</tr>
<tr>
<td>M11</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M12</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M21</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M22</td>
<td>Matrix component</td>
</tr>
<tr>
<td>MDX</td>
<td>Matrix component</td>
</tr>
<tr>
<td>MDY</td>
<td>Matrix component</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An invalid CaptureID was specified</td>
</tr>
<tr>
<td>1</td>
<td>The captured page was drawn successfully</td>
</tr>
</tbody>
</table>
**DrawCircle**

**Vector graphics**

**Description**

Draw a circle on the selected page.

**Syntax**

**Dylib**

```c
int DPLDrawCircle(int InstanceID, double XPos, double YPos, double Radius, int DrawOptions);
```

**Objective-C class**

```objective-c
-(int)DrawCircle:(double)XPos :(double)YPos :(double)Radius :
:(int)DrawOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>Horizontal co-ordinate of the center of the circle</td>
</tr>
<tr>
<td>YPos</td>
<td>Vertical co-ordinate of center of the circle</td>
</tr>
<tr>
<td>Radius</td>
<td>Size of the circle</td>
</tr>
<tr>
<td>DrawOptions</td>
<td>0 = Outline 1 = Fill 2 = Fill and Outline</td>
</tr>
</tbody>
</table>
**DrawDataMatrixSymbol**

*Vector graphics, Barcodes*

**Description**

This function draws a Data Matrix symbol onto the page. Data Matrix is a 2D barcode symbology allowing large amounts of data to be stored.

**Syntax**

**Dylib**

```c
int DPLDrawDataMatrixSymbol(int InstanceID, double Left, double Top, double ModuleSize, wchar_t * Text, int Encoding, int SymbolSize, int Options);
```

**Objective-C class**

```objective-c
- (int)DrawDataMatrixSymbol:(double)Left :(double)Top :(double)ModuleSize :(NSString *)Text :(int)Encoding :(int)SymbolSize :(int)Options;
```

**Parameters**

- **Left** The horizontal co-ordinate of the left edge of the symbol
- **Top** The vertical co-ordinate of the top edge of the symbol
- **ModuleSize** This value is used for the width and height of the dots which make up the symbol
- **Text** The text/data to store in the symbol
- **Encoding** 1 = ASCII encoding. See the Data Matrix specification for details.
- **SymbolSize** 0 = Auto size
  - 1 = 10x10
  - 2 = 12x12
  - 3 = 8x18
  - 4 = 14x14
  - 5 = 8x32
  - 6 = 16x16
  - 7 = 12x26
  - 8 = 18x18
  - 9 = 20x20
  - 10 = 12x36
  - 11 = 22x22
  - 12 = 16x36
  - 13 = 24x24
  - 14 = 26x26
  - 15 = 16x48
  - 16 = 32x32
  - 17 = 36x36
  - 18 = 40x40
  - 19 = 44x44
  - 20 = 48x48
  - 21 = 52x52
  - 22 = 64x64
  - 23 = 72x72
  - 24 = 80x80
  - 25 = 88x88
  - 26 = 96x96
  - 27 = 104x104
  - 28 = 120x120
  - 29 = 132x132
- **Options** 0 = Normal
  - 1 = Rotate 90 degrees counter clockwise
  - 2 = Rotate 180 degrees
  - 3 = Rotate 90 degrees clockwise
  - Add 100 to for 1 unit quiet zone (white border) - (default)
  - Add 200 to for 2 units quiet zone
  - Add 300 to for 3 units quiet zone
  - Add 400 to for 4 units quiet zone

**Return values**

- **0** The Encoding, SymbolSize or Options parameter was invalid
- **1** The Data Matrix symbol was drawn successfully
### DrawEllipse

**Vector graphics**

**Description**

Draws an ellipse centered at a certain point which fits into the specified size box.

**Syntax**

**Dylib**

```c
int DPLDrawEllipse(int InstanceID, double XPos, double YPos, double Width, double Height, int DrawOptions);
```

**Objective-C class**

```c
- (int)DrawEllipse:(double)XPos :(double)YPos :(double)Width :(double)Height :(int)DrawOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>The horizontal co-ordinate of the center of the ellipse</td>
</tr>
<tr>
<td>YPos</td>
<td>The vertical co-ordinate of the center of the ellipse</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the ellipse</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the ellipse</td>
</tr>
</tbody>
</table>
| DrawOptions | 0 = Outline  
|             | 1 = Fill  
|             | 2 = Fill and Outline                            |
**DrawEllipticArc**

**Vector graphics**

**Description**

Draws an arc which is the result of cutting an ellipse between the start angle and the end angle. The angles are measured clockwise with 0 being at the top of the ellipse.

**Syntax**

**Dylib**

```c
int DPLDrawEllipticArc(int InstanceID, double XPos, double YPos,
                        double Width, double Height, double StartAngle,
                        double EndAngle, int Pie, int DrawOptions);
```

**Objective-C class**

```c
-(int)DrawEllipticArc:(double)XPos :(double)YPos :(double)Width
:(double)Height :(double)StartAngle :(double)EndAngle
:(int)Pie :(int)DrawOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>The horizontal co-ordinate of the center of the ellipse</td>
</tr>
<tr>
<td>YPos</td>
<td>The vertical co-ordinate of the center of the ellipse</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the ellipse</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the ellipse</td>
</tr>
<tr>
<td>StartAngle</td>
<td>The angle to start the curve at</td>
</tr>
<tr>
<td>EndAngle</td>
<td>The angle to end the curve at</td>
</tr>
<tr>
<td>Pie</td>
<td>Draw the arms of the arc: 0 = No 1 = Yes</td>
</tr>
<tr>
<td>DrawOptions</td>
<td>0 = Outline 1 = Fill 2 = Fill and Outline 3 = Close, Fill and Outline (if Pie = 1)</td>
</tr>
</tbody>
</table>
DrawHTMLText
Text, HTML text, Page layout

Description

Draws HTML text onto the page. See Appendix A for details of the supported HTML tags.

Syntax

Dylib

```c
int DPLDrawHTMLText(int InstanceID, double Left, double Top, double Width, wchar_t * HTMLText);
```

Objective-C class

```Objective-C
-(int)DrawHTMLText:(double)Left :(double)Top :(double)Width :
:(NSString *)HTMLText;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The left edge of the area to draw the text into</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The top edge of the area to draw the text into</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the area to draw the text into</td>
</tr>
<tr>
<td><strong>HTMLText</strong></td>
<td>The HTML text to draw</td>
</tr>
</tbody>
</table>
**DrawHTMLTextBox**

**Text, HTML text, Page layout**

**Description**

Similar to the **DrawHTMLText** function, but the text drawn is limited to a specific area. The remaining HTML text is returned, which can be passed to this function again (perhaps on a different page or location) until the function returns an empty string. See **Appendix A** for details of the supported HTML tags.

**Syntax**

**Dylib**

```c
wchar_t * DPLDrawHTMLTextBox(int InstanceID, double Left, double Top, double Width, double Height, wchar_t * HTMLText);
```

**Objective-C class**

```objc
-(NSString *)DrawHTMLTextBox:(double)Left :(double)Top :(double)Width :(double)Height :(NSString *)HTMLText;
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Horizontal co-ordinate of the left edge of the drawing area</td>
</tr>
<tr>
<td>Top</td>
<td>Vertical co-ordinate of the top edge of the drawing area</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the drawing area</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the drawing area</td>
</tr>
<tr>
<td>HTMLText</td>
<td>The HTML text to draw</td>
</tr>
</tbody>
</table>
**DrawHTMLTextBoxMatrix**

**Text, HTML text, Page layout**

**Description**

Similar to the `DrawHTMLTextBox` function but the position/scaling/rotation is specified using a transformation matrix.

The remaining HTML text is returned, which can be passed to this function again (perhaps on a different page or location) until the function returns an empty string. See Appendix A for details of the supported HTML tags.

**Syntax**

**Dylib**

```c
wchar_t * DPLDrawHTMLTextBoxMatrix(int InstanceID, double Width, double Height, wchar_t * HTMLText, double M11, double M12, double M21, double M22, double MDX, double MDY);
```

**Objective-C class**

```objective-c
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>The width of the drawing area</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the drawing area</td>
</tr>
<tr>
<td><strong>HTMLText</strong></td>
<td>The HTML text to draw</td>
</tr>
<tr>
<td><strong>M11</strong></td>
<td>Matrix component</td>
</tr>
<tr>
<td><strong>M12</strong></td>
<td>Matrix component</td>
</tr>
<tr>
<td><strong>M21</strong></td>
<td>Matrix component</td>
</tr>
<tr>
<td><strong>M22</strong></td>
<td>Matrix component</td>
</tr>
<tr>
<td><strong>MDX</strong></td>
<td>Matrix component</td>
</tr>
<tr>
<td><strong>MDY</strong></td>
<td>Matrix component</td>
</tr>
</tbody>
</table>
**DrawHTMLTextMatrix**

**HTML text, Page layout**

**Description**
Similar to the **DrawHTMLText** function but the position/scaling/rotation is specified using a transformation matrix. 
See **Appendix A** for details of the supported HTML tags.

**Syntax**

**Dylib**

```c
int DPLDrawHTMLTextMatrix(int InstanceID, double Width,
 wchar_t * HTMLText, double M11, double M12, double M21,
 double M22, double MDX, double MDY);
```

**Objective-C class**

```objc
- (int)DrawHTMLTextMatrix:(double)Width :(NSString *)HTMLText :(double)M11
 :(double)M12 :(double)M21 :(double)M22 :(double)MDX
 :(double)MDY;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>The width of the area to draw the text into</td>
</tr>
<tr>
<td>HTMLText</td>
<td>The HTML text to draw</td>
</tr>
<tr>
<td>M11</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M12</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M21</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M22</td>
<td>Matrix component</td>
</tr>
<tr>
<td>MDX</td>
<td>Matrix component</td>
</tr>
<tr>
<td>MDY</td>
<td>Matrix component</td>
</tr>
</tbody>
</table>
DrawImage

Image handling, Page layout

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Draw the selected image on the page.

Syntax

Dylib

int DPLDrawImage(int InstanceID, double Left, double Top, double Width, double Height);

Objective-C class

- (int)DrawImage:(double)Left :(double)Top :(double)Width :(double)Height;

Parameters

<table>
<thead>
<tr>
<th>Left</th>
<th>Horizontal co-ordinate of the left edge of the image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>Vertical co-ordinate of the top edge of the image</td>
</tr>
<tr>
<td>Width</td>
<td>Width of the image</td>
</tr>
<tr>
<td>Height</td>
<td>Height of the image</td>
</tr>
</tbody>
</table>

Return values

| 0        | An image has not been selected                      |
| 1        | The image was drawn successfully                    |
**DrawImageMatrix**

**Image handling, Page layout**

**Description**

Draws the selected image on the page using a transformation matrix.

**Syntax**

**Dylib**

```c
int DPLDrawImageMatrix(int InstanceID, double M11, double M12, double M21,
                       double M22, double MDX, double MDY);
```

**Objective-C class**

```objc
                 :(double)MDX :(double)MDY;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M11</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M12</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M21</td>
<td>Matrix component</td>
</tr>
<tr>
<td>M22</td>
<td>Matrix component</td>
</tr>
<tr>
<td>MDX</td>
<td>Matrix component</td>
</tr>
<tr>
<td>MDY</td>
<td>Matrix component</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An image has not been selected</td>
</tr>
<tr>
<td>1</td>
<td>The image was drawn successfully</td>
</tr>
</tbody>
</table>
DrawIntelligentMailBarcode

Vector graphics, Barcodes

Description

This function draws a USPS Intelligent Mail (also known as OneCode) barcode onto the page.

Syntax

Dylib

```c
int DPLDrawIntelligentMailBarcode(int InstanceID, double Left, double Top,
    double BarWidth, double FullBarHeight, double TrackerHeight,
    double SpaceWidth, wchar_t * BarcodeData, int Options);
```

Objective-C class

```c
- (int)DrawIntelligentMailBarcode:(double)Left :(double)Top
    :(double)BarWidth :(double)FullBarHeight
    :(double)TrackerHeight :(double)SpaceWidth
    :(NSString *)BarcodeData :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Horizontal co-ordinate of the left edge of the barcode</td>
</tr>
<tr>
<td>Top</td>
<td>Vertical co-ordinate of the top edge of the barcode</td>
</tr>
<tr>
<td>BarWidth</td>
<td>The width of the bars</td>
</tr>
<tr>
<td>FullBarHeight</td>
<td>The height of a full bar</td>
</tr>
<tr>
<td>TrackerHeight</td>
<td>The height of a tracker bar</td>
</tr>
<tr>
<td>SpaceWidth</td>
<td>The width of the spaces between the bars</td>
</tr>
<tr>
<td>BarcodeData</td>
<td>The barcode data to encode. This should be a 20, 25, 29 or 31 character string containing only the digits 0 to 9. No spaces or any other non-numeric characters will be accepted. The second digit has a maximum value of 4.</td>
</tr>
<tr>
<td>Options</td>
<td>0 = Normal</td>
</tr>
<tr>
<td></td>
<td>10 = Bar width reduction</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The barcode could not be drawn</td>
</tr>
<tr>
<td>1</td>
<td>The barcode was drawn successfully</td>
</tr>
</tbody>
</table>
**DrawLine**

Vector graphics

**Description**

Draws a line between two points.

**Syntax**

**Dylib**

```c
int DPLDrawLine(int InstanceID, double StartX, double StartY, double EndX, double EndY);
```

**Objective-C class**

```objc
- (int)DrawLine:(double)StartX :(double)StartY :(double)EndX :(double)EndY;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartX</td>
<td>Horizontal co-ordinate of start point</td>
</tr>
<tr>
<td>StartY</td>
<td>Vertical co-ordinate of start point</td>
</tr>
<tr>
<td>EndX</td>
<td>Horizontal co-ordinate of end point</td>
</tr>
<tr>
<td>EndY</td>
<td>Vertical co-ordinate of end point</td>
</tr>
</tbody>
</table>
DrawMultiLineText

Text, Page layout

Description

Draw text which is wrapped at a specific delimiter. The SetTextAlign function can be used to change the alignment of the text.

Syntax

**Dylib**

```
int DPLDrawMultiLineText(int InstanceID, double XPos, double YPos,
wchar_t * Delimiter, wchar_t * Text);
```

**Objective-C class**

```
- (int)DrawMultiLineText:(double)XPos :(double)YPos :(NSString *)Delimiter
 :(NSString *)Text;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>The horizontal reference point of the text block</td>
</tr>
<tr>
<td>YPos</td>
<td>The baseline of the first line of text</td>
</tr>
<tr>
<td>Delimiter</td>
<td>The delimiter to use when splitting the text into lines. The only valid characters to use as the delimiter are characters which have a &quot;width&quot;, as well as the CR and LF characters (ASCII values 13 and 10).</td>
</tr>
<tr>
<td>Text</td>
<td>The text to draw</td>
</tr>
</tbody>
</table>
**DrawPDF417Symbol**

**Vector graphics, Barcodes**

**Description**

Draws a PDF417 symbol onto the selected page.
From version 9.15 the `DrawPDF417SymbolEx` function can be used for extra functionality.

**Syntax**

**Dylib**

```c
int DPLDrawPDF417Symbol(int InstanceID, double Left, double Top, wchar_t * Text, int Options);
```

**Objective-C class**

```c
- (int)DrawPDF417Symbol:(double)Left :(double)Top :(NSString *)Text :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The horizontal coordinate of the left edge of the PDF417 symbol</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The vertical coordinate of the top edge of the PDF417 symbol</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text to store in the symbol</td>
</tr>
</tbody>
</table>
| **Options** | 0 = Normal  
1 = Rotate 90 degrees counter clockwise  
2 = Rotate 180 degrees  
3 = Rotate 90 degrees clockwise |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Options parameter was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The PDF417 symbol was drawn successfully</td>
</tr>
</tbody>
</table>
DrawPDF417SymbolEx

Vector graphics, Barcodes

Description

Draws a PDF417 symbol onto the selected page. Similar to DrawPDF417Symbol but providing extra functionality.

Syntax

Dylib

```c
int DPLDrawPDF417SymbolEx(int InstanceID, double Left, double Top,
                          wchar_t * Text, int Options, int FixedColumns, int FixedRows,
                          int ErrorLevel, double ModuleSize, double HeightWidthRatio);
```

Objective-C class

```
- (int)DrawPDF417SymbolEx:(double)Left :(double)Top :(NSString *)Text
                     :(int)Options :(int)FixedColumns :(int)FixedRows
                     :(int)ErrorLevel :(double)ModuleSize :(double)HeightWidthRatio;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal coordinate of the left edge of the PDF417 symbol</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical coordinate of the top edge of the PDF417 symbol</td>
</tr>
<tr>
<td>Text</td>
<td>The text to store in the symbol</td>
</tr>
</tbody>
</table>
| Options     | 0 = Normal
             | 1 = Rotate 90 degrees counter clockwise
             | 2 = Rotate 180 degrees
             | 3 = Rotate 90 degrees clockwise                                             |
| FixedColumns| 0 = Auto
             | Non-zero = fixed number of columns                                          |
| FixedRows   | 0 = Auto
             | Non-zero = fixed number of rows                                             |
| ErrorLevel  | -1 = Auto
             | 0 to 8 = User error level                                                   |
| ModuleSize  | The width of the smallest element in units defined by a call to
             | SetMeasurementUnits                                                         |
| HeightWidthRatio | The ratio of the needed module height to the module width                   |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>One of the parameters was invalid or the text was too big for the symbol site.</td>
</tr>
<tr>
<td>1</td>
<td>The PDF417 symbol was drawn successfully</td>
</tr>
</tbody>
</table>
DrawPath

Vector graphics, Path definition and drawing

Description

Draws the path defined by calls to `StartPath`, `AddLineToPath`, `AddCurveToPath` and/or `ClosePath`.

Syntax

**Dylib**

```c
int DPLDrawPath(int InstanceID, int PathOptions);
```

**Objective-C class**

```c
-(int)DrawPath:(int)PathOptions;
```

Parameters

<table>
<thead>
<tr>
<th>PathOptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Outline</td>
</tr>
<tr>
<td>1</td>
<td>Fill</td>
</tr>
<tr>
<td>2</td>
<td>Fill and Outline</td>
</tr>
</tbody>
</table>
### DrawPathEvenOdd

**Vector graphics, Path definition and drawing**

**Description**

Similar to the `DrawPath` function, but draws the path using the "even odd" method. This is important when different parts of the path overlap.

**Syntax**

**Dylib**

```c
int DPLDrawPathEvenOdd(int InstanceID, int PathOptions);
```

**Objective-C class**

```objective-c
-(int)DrawPathEvenOdd:(int)PathOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>PathOptions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Outline</td>
</tr>
<tr>
<td>1</td>
<td>Fill</td>
</tr>
<tr>
<td>2</td>
<td>Fill and outline</td>
</tr>
</tbody>
</table>
**DrawPostScriptXObject**

**Annotations and hotspot links, Page layout**

**Description**

Adds a reference to a PostScript XObject at the current position in the page contents. This function is for specific advanced use and will not be useful to the majority of users. For historical reasons, the PDF specification allows raw PostScript language commands to be embedded inside a document.

When the document is printed (using certain PDF software tools) on a PostScript printer, these raw PostScript commands will be sent directly to the printer.

Most PDF viewers are not able to display this embedded PostScript because this would require a full PostScript language interpreter.

**Syntax**

**Dylib**

```c
int DPLDrawPostScriptXObject(int InstanceID, int PSRef);
```

**Objective-C class**

```c
-(int)DrawPostScriptXObject:(int)PSRef;
```

**Parameters**

| PSRef | A value that was returned by the **NewPostScriptXObject** function |

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The PostScript XObject could not be drawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The PostScript XObject was drawn successfully</td>
</tr>
</tbody>
</table>
DrawQRCode

Vector graphics, Barcodes

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Draws a QR Code onto the selected page.

Syntax

**Dylib**

```c
int DPLDrawQRCode(int InstanceID, double Left, double Top, double SymbolSize, wchar_t * Text, int EncodeOptions, int DrawOptions);
```

**Objective-C class**

```objc
-(int)DrawQRCode:(double)Left :(double)Top :(double)SymbolSize :(NSString *)Text :(int)EncodeOptions :(int)DrawOptions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal coordinate of the left edge of the QR Code</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical coordinate of the top edge of the QR Code</td>
</tr>
<tr>
<td>SymbolSize</td>
<td>The width and height of the QR Code</td>
</tr>
<tr>
<td>Text</td>
<td>The text to encode in the QR Code</td>
</tr>
</tbody>
</table>
| EncodeOptions | 0=A Auto  
1=N Numeric  
2=A=Alphanumeric  
3=ISO-8859-1  
4=UTF-8 with BOM  
5=UTF-8 without BOM |
| DrawOptions | 0 = Normal  
1 = Rotate 90 degrees counter clockwise  
2 = Rotate 180 degrees  
3 = Rotate 90 degrees clockwise |

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The QR Code could not be drawn, check for an out of range value for the EncodeOptions or DrawOptions parameter.</td>
</tr>
<tr>
<td>1</td>
<td>The QR Code was drawn successfully.</td>
</tr>
</tbody>
</table>
**DrawRotatedBox**

**Vector graphics, Page manipulation**

**Description**

Draws a rotated rectangle on the selected page.

**Syntax**

**Dylib**

```c
int DPLDrawRotatedBox(int InstanceID, double Left, double Bottom, 
double Width, double Height, double Angle, int DrawOptions);
```

**Objective-C class**

```objective-c
- (int)DrawRotatedBox:(double)Left :(double)Bottom :(double)Width 
:(double)Height :(double)Angle :(int)DrawOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the anchor point</td>
</tr>
<tr>
<td>Bottom</td>
<td>The vertical co-ordinate of the anchor point</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the rectangle</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle to rotate the rectangle, measured anti-clockwise in degrees from the baseline, around the anchor point (bottom-left of the rectangle)</td>
</tr>
</tbody>
</table>
| DrawOptions | 0 = Outline  
1 = Fill  
2 = Fill and Outline |
**DrawRotatedCapturedPage**

*Page layout, Page manipulation*

**Description**

Similar to the **DrawCapturedPage** function, but allows the captured page to be drawn at any angle. Note that the anchor point is the bottom-left corner, not the top-left corner as with the **DrawCapturedPage** function.

**Syntax**

**Dylib**

```c
int DPLDrawRotatedCapturedPage(int InstanceID, int CaptureID, double Left, double Bottom, double Width, double Height, double Angle);
```

**Objective-C class**

```objective-c
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaptureID</td>
<td>The ID returned by the <strong>CapturePage</strong> function</td>
</tr>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the anchor point</td>
</tr>
<tr>
<td>Bottom</td>
<td>The vertical co-ordinate of the anchor point</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the rectangle to place the captured page in</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the rectangle to place the captured page in</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle to rotate the captured page by, measured anti-clockwise in degrees from the baseline</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The CaptureID was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The captured page was drawn successfully</td>
</tr>
</tbody>
</table>
DrawRotatedImage
Image handling, Page layout

Description
Similar to the DrawImage function but the image can be rotated at any angle. Note that the anchor point is the bottom left corner of the image, not the top-left as in the DrawImage function.

Syntax
Dylib

```c
int DPLDrawRotatedImage(int InstanceID, double Left, double Bottom, double Width, double Height, double Angle);
```

Objective-C class

```c
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the anchor point</td>
</tr>
<tr>
<td>Bottom</td>
<td>The vertical co-ordinate of the anchor point</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the image</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the image</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle to rotate the image, measured anti-clockwise in degrees from the baseline, around the anchor point (bottom-left of the image)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image has been selected</td>
</tr>
<tr>
<td>1</td>
<td>The image was drawn successfully</td>
</tr>
</tbody>
</table>
DrawRotatedMultiLineText

Text, Page layout

Description

Draws rotated text which is wrapped at a specific delimiter.

The **SetTextAlign** function can be used to change the alignment of the text.

The first line of text will start with the baseline at the anchor point used for rotation.

Syntax

Dylib

```c
int DPLDrawRotatedMultiLineText(int InstanceID, double XPos, double YPos,
                                 double Angle, wchar_t * Delimiter, wchar_t * Text);
```

Objective-C class

```objective-c
- (int)DrawRotatedMultiLineText:(double)XPos :(double)YPos :(double)Angle
  :(NSString *)Delimiter :(NSString *)Text;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>The horizontal coordinate of the anchor point</td>
</tr>
<tr>
<td>YPos</td>
<td>The vertical coordinate of the anchor point</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle to rotate the text, measured anti-clockwise in degrees from the baseline, around the anchor point</td>
</tr>
<tr>
<td>Delimiter</td>
<td>The delimiter to use when splitting the text into lines. The only valid characters to use as the delimiter are characters which have a &quot;width&quot;, as well as the CR and LF characters (ASCII values 13 and 10).</td>
</tr>
<tr>
<td>Text</td>
<td>The text to draw</td>
</tr>
</tbody>
</table>
DrawRotatedText

Text, Page layout

Description

Draws text on the selected page, using the selected font at the predetermined font size. If no fonts have been added, then the standard font Helvetica will automatically be added, selected and set to 12pt. The alignment of the text is determined by the previous call to the SetTextAlign function.

Syntax

Dylib

```c
int DPLDrawRotatedText(int InstanceID, double XPos, double YPos,
double Angle, wchar_t * Text);
```

Objective-C class

```objective-c
- (int)DrawRotatedText:(double)XPos :(double)YPos :(double)Angle
:(NSString *)Text;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>The horizontal position of where to draw the text</td>
</tr>
<tr>
<td>YPos</td>
<td>The vertical position of where to draw the text. The reference point is the text baseline.</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle to draw the text, measured clockwise from the horizontal. Must be between 0 and 360, inclusive.</td>
</tr>
<tr>
<td>Text</td>
<td>The text to draw on the page</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Angle parameter was less than 0 or greater than 360, or the Text parameter was blank</td>
</tr>
<tr>
<td>1</td>
<td>The text was drawn successfully</td>
</tr>
</tbody>
</table>
**DrawRotatedTextBox**

**Text, Page layout**

**Description**

Similar to the `DrawTextBox` function, but allows the text box to be rotated at any angle.

**Syntax**

**Dylib**

```c
int DPLDrawRotatedTextBox(int InstanceID, double Left, double Top, double Width, double Height, double Angle, wchar_t * Text, int Options);
```

**Objective-C class**

```
- (int)DrawRotatedTextBox:(double)Left :(double)Top :(double)Width :(double)Height :(double)Angle :(NSString *)Text :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the top-left corner of the text box</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top-left corner of the text box</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the box</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the box</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle the box should be rotated around the top-left corner, measured anti-clockwise in degrees</td>
</tr>
<tr>
<td>Text</td>
<td>The text to place in the box</td>
</tr>
</tbody>
</table>
| Options   | 0 = Center vertical alignment  
            1 = Top vertical alignment  
            2 = Bottom vertical alignment  
            3 = Center vertical alignment, no wrapping  
            4 = Top vertical alignment, no wrapping  
            5 = Bottom vertical alignment, no wrapping |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Options parameter was out of range, or the Width parameter was too small to contain any text</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of lines of text actually drawn</td>
</tr>
</tbody>
</table>
DrawRoundedBox
Vector graphics, Page layout

Description

Draw a rectangle with rounded corners on the selected page.

Syntax

Dylib

```c
int DPLDrawRoundedBox(int InstanceID, double Left, double Top,
                        double Width, double Height, double Radius, int DrawOptions);
```

Objective-C class

```c
- (int)DrawRoundedBox:(double)Left :(double)Top :(double)Width
                    :(double)Height :(double)Radius :(int)DrawOptions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Horizontal co-ordinate of left edge of rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>Vertical co-ordinate of top edge of rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>Rectangle width</td>
</tr>
<tr>
<td>Height</td>
<td>Rectangle height</td>
</tr>
<tr>
<td>Radius</td>
<td>Radius of the corner arcs</td>
</tr>
<tr>
<td>DrawOptions</td>
<td>0 = Outline</td>
</tr>
<tr>
<td></td>
<td>1 = Fill</td>
</tr>
<tr>
<td></td>
<td>2 = Fill and outline</td>
</tr>
</tbody>
</table>
DrawScaledImage
Image handling, Page layout

Description

Draw the selected image on the page. The image is drawn at the scale specified, assuming 72 DPI for both the horizontal and vertical resolution.

Syntax

Dylib

```c
int DPLDrawScaledImage(int InstanceID, double Left, double Top, double Scale);
```

Objective-C class

```objc
- (int)DrawScaledImage:(double)Left :(double)Top :(double)Scale;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>Horizontal co-ordinate of the left edge of the image</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>Vertical co-ordinate of the top edge of the image</td>
</tr>
<tr>
<td><strong>Scale</strong></td>
<td>The scale to use, for example: &lt;br&gt; 0.5 = 50% &lt;br&gt; 1 = 100%</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An image was not selected</td>
</tr>
<tr>
<td>1</td>
<td>The image was drawn successfully</td>
</tr>
</tbody>
</table>
**DrawSpacedText**

**Text, Page layout**

**Description**

Draws text on the selected page, using the selected font at the predetermined font size. If no fonts have been added, then the 12 pt Helvetica will automatically be added and selected. Each character will be spaced at regular intervals. The individual characters will be aligned relative to the XPos variable depending on how the **SetTextAlign** function has been used.

**Syntax**

**Dylib**

```c
int DPLDrawSpacedText(int InstanceID, double XPos, double YPos,
                       double Spacing, wchar_t * Text);
```

**Objective-C class**

```objc
-(int)DrawSpacedText:(double)XPos :(double)YPos :(double)Spacing
:(NSString *)Text;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XPos</strong></td>
<td>The horizontal position of where to draw the text</td>
</tr>
<tr>
<td><strong>YPos</strong></td>
<td>The vertical position of where to draw the text. The reference point is the text baseline.</td>
</tr>
<tr>
<td><strong>Spacing</strong></td>
<td>The spacing between the same point on each character</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text to draw on the page</td>
</tr>
</tbody>
</table>
**DrawTableRows**

**Description**

Draws multiple rows from the specified table onto the selected page and returns the total height of the drawn rows. Only the number of rows that fit into the specified height will be drawn. Use the `GetTableLastDrawnRow` function to determine the row number of the last row.

In Quick PDF Library version 7.18 and earlier the result of this function was always reported in points. From version 7.19 and later the value returned by this function is correctly scaled according to the current co-ordinate system settings as set by the `SetMeasurementUnits` function.

**Syntax**

**Dylib**

```c
double DPLDrawTableRows(int InstanceID, int TableID, double Left,
                          double Top, double Height, int FirstRow, int LastRow);
```

**Objective-C class**

```c
- (double)DrawTableRows:(int)TableID :(double)Left :(double)Top
                          :(double)Height :(int)FirstRow :(int)LastRow;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <code>CreateTable</code> function</td>
</tr>
<tr>
<td>Left</td>
<td>The horizontal distance from the origin to the left edge of the table</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical distance from the origin to the top of the table</td>
</tr>
<tr>
<td>Height</td>
<td>The maximum height available to draw the table in</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The number of the first row to draw. Top row is row number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>0 = All remaining rows</td>
</tr>
<tr>
<td></td>
<td>Non-zero = The number of the final row to set</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No rows were drawn</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The total height of all the rows that were drawn onto the page.</td>
</tr>
</tbody>
</table>
**DrawText**

**Text, Page layout**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Draws text on the selected page, using the selected font at the predetermined font size. If no fonts have been added, then 12 pt Helvetica will automatically be added and selected. The alignment of the text can be changed with the **SetTextAlign** function.

**Syntax**

**Dylib**

```c
int DPLDrawText(int InstanceID, double XPos, double YPos, wchar_t * Text);
```

**Objective-C class**

```c
- (int)DrawText:(double)XPos :(double)YPos :(NSString *)Text;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XPos</strong></td>
<td>The horizontal position of where to draw the text. The reference point is usually to the left of the first character, unless the <strong>SetTextAlign</strong> function has been used to change the alignment.</td>
</tr>
<tr>
<td><strong>YPos</strong></td>
<td>The vertical position of where to draw the text. The reference point is the text baseline.</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text to draw on the page</td>
</tr>
</tbody>
</table>
DrawTextArc

Text, Page layout

Description

Draws text fitted to an imaginary arc with the specified center point and radius. The text will be drawn with its left edge at the requested angle, where 0 degrees is the "12 o'clock" position, and positive angles are clockwise. The `SetTextAlign` function can be used to change the alignment of the text relative to the specified angle.

Syntax

Dylib

```c
int DPLDrawTextArc(int InstanceID, double XPos, double YPos, double Radius, double Angle, wchar_t * Text, int DrawOptions);
```

Objective-C class

```c
- (int)DrawTextArc:(double)XPos :(double)YPos :(double)Radius :
  :(double)Angle :(NSString *)Text :(int)DrawOptions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>The horizontal co-ordinate of the center of the arc</td>
</tr>
<tr>
<td>YPos</td>
<td>The vertical co-ordinate of the center of the arc</td>
</tr>
<tr>
<td>Radius</td>
<td>The radius of the arc</td>
</tr>
<tr>
<td>Angle</td>
<td>The angle at which the text should be placed</td>
</tr>
<tr>
<td>Text</td>
<td>The actual text to draw</td>
</tr>
</tbody>
</table>
| DrawOptions | 0 = Draw the text outside the arc in a clockwise direction  
|            | 1 = Draw the text inside the arc in an anti-clockwise direction |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The text was blank or the DrawOptions parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>The text was drawn successfully</td>
</tr>
</tbody>
</table>
**DrawTextBox**

**Text, Page layout**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

This function is similar to the **DrawText** function, but the text is placed within the bounding box specified. The vertical alignment can be set using the Options parameter, and the horizontal alignment can be set with the **SetTextAlign** function. The text will be word-wrapped to fit inside the bounding box.

**Syntax**

**Dylib**

```c
int DPLDrawTextBox(int InstanceID, double Left, double Top, double Width, double Height, wchar_t * Text, int Options);
```

**Objective-C class**

```objc
- (int)DrawTextBox:(double)Left :(double)Top :(double)Width :(double)Height :(NSString *)Text :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the bounding box</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the bounding box</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the bounding box</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the bounding box</td>
</tr>
<tr>
<td>Text</td>
<td>The text to draw on the page</td>
</tr>
</tbody>
</table>
| Options   | 0 = Center vertical alignment  
            | 1 = Top vertical alignment  
            | 2 = Bottom vertical alignment  
            | 3 = Center vertical alignment, no wrapping  
            | 4 = Top vertical alignment, no wrapping  
            | 5 = Bottom vertical alignment, no wrapping |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Options parameter was out of range, or the Width parameter was too small to contain any text</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of lines of text actually drawn</td>
</tr>
</tbody>
</table>
**DrawTextBoxMatrix**

**Text, Page layout**

**Description**

This function is similar to the `DrawTextBox` function but the position/scaling/rotation is specified using a transformation matrix.

The vertical alignment can be set using the `Options` parameter, and the horizontal alignment can be set with the `SetTextAlign` function. The text will be word-wrapped to fit inside the bounding box.

**Syntax**

**Dylib**

```c
int DPLDrawTextBoxMatrix(int InstanceID, double Width, double Height, wchar_t * Text, int Options, double M11, double M12, double M21, double M22, double MDX, double MDY);
```

**Objective-C class**

```objective-c
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>The width of the bounding box</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the bounding box</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text to draw on the page</td>
</tr>
</tbody>
</table>
| **Options** | 0 = Center vertical alignment  
1 = Top vertical alignment  
2 = Bottom vertical alignment  
3 = Center vertical alignment, no wrapping  
4 = Top vertical alignment, no wrapping  
5 = Bottom vertical alignment, no wrapping |
| **M11** | Matrix component |
| **M12** | Matrix component |
| **M21** | Matrix component |
| **M22** | Matrix component |
| **MDX** | Matrix component |
| **MDY** | Matrix component |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The <code>Options</code> parameter was out of range, or the <code>Width</code> parameter was too small to contain any text</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of lines of text actually drawn</td>
</tr>
</tbody>
</table>
**DrawWrappedText**

**Text, Page layout**

**Description**

Draw text which is wrapped to a certain width. The `SetTextAlign` function can be used to change the alignment of the text.

**Syntax**

**Dylib**

```c
int DPLDrawWrappedText(int InstanceID, double XPos, double YPos, double Width, wchar_t * Text);
```

**Objective-C class**

```Objective-C
-(int)DrawWrappedText:(double)XPos :(double)YPos :(double)Width :(NSString *)Text;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos</td>
<td>The left edge of the text block</td>
</tr>
<tr>
<td>YPos</td>
<td>The baseline of the first line of text</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the text block</td>
</tr>
<tr>
<td>Text</td>
<td>The text to draw</td>
</tr>
</tbody>
</table>
EditableContentStream

Content Streams and Optional Content Groups

Description

Use this function to determine if the content stream part that was selected with the SelectContentStream function can be drawn on.

Syntax

**Dylib**

```cpp
int DPLEditableContentStream(int InstanceID);
```

**Objective-C class**

```objc
- (int)EditableContentStream
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected content stream part cannot be drawn on</td>
</tr>
<tr>
<td>1</td>
<td>The selected content stream part is editable</td>
</tr>
</tbody>
</table>
EmbedFile

Document properties

Description

Embeds a file into the PDF document and creates a file attachment link to the embedded file. The file can then be accessed in Acrobat under the File Attachments function. This is equivalent to calling AddEmbeddedFile followed by AddFileAttachment.

Syntax

Dylib

```c
int DPLEmbedFile(int InstanceID, wchar_t * Title, wchar_t * FileName, wchar_t * MIMEType);
```

Objective-C class

```objc
-(int)EmbedFile:(NSString *)Title :(NSString *)FileName :(NSString *)MIMEType;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>A unique title for this file. No two files can have the same title. If a file with this title already exists in the document the new file will not be embedded.</td>
</tr>
<tr>
<td>FileName</td>
<td>The full path and name of the file to embed.</td>
</tr>
<tr>
<td>MIMEType</td>
<td>The optional MIME type of the file, for example &quot;image/jpg&quot; for a JPEG image. See <a href="http://www.iana.org/assignments/media-types/">http://www.iana.org/assignments/media-types/</a> for a full list of MIME types. If the MIME type is not known it can be set to an empty string.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file could not be embedded</td>
</tr>
<tr>
<td>1</td>
<td>The file was embedded successfully</td>
</tr>
</tbody>
</table>
EmbeddedFileCount

Document properties

Description

Returns the number of embedded files in the document. This total only includes embedded files that are listed as file attachments and does not include embedded files that are only referenced by a link annotation.

Syntax

**Dylib**

```c
int DPLEmbeddedFileCount(int InstanceID);
```

**Objective-C class**

```objc
-(int)EmbeddedFileCount
```
EncapsulateContentStream

Content Streams and Optional Content Groups

Description

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function combines the content stream parts and surrounds the content stream with "save graphics state" and "restore graphics state" operators. This has the effect of clearing the current clipping path.

Some pages may contain unbalanced "save graphics state" and "restore graphics state" operators. The BalanceContentStream function can be used to repair such pages.

Syntax

Dylib

int DPLEncapsulateContentStream(int InstanceID);

Objective-C class

- (int)EncapsulateContentStream
## EncodePermissions

### Security and Signatures

#### Description

Create a value for the Permissions parameter of the Encrypt function.

#### Syntax

**Dylib**

```c
int DPLEncodePermissions(int InstanceID, int CanPrint, int CanCopy,
int CanChange, int CanAddNotes, int CanFillFields,
int CanCopyAccess, int CanAssemble, int CanPrintFull);
```

**Objective-C class**

```c
- (int)EncodePermissions:(int)CanPrint :(int)CanCopy :(int)CanChange
 :(int)CanAddNotes :(int)CanFillFields :(int)CanCopyAccess
 :(int)CanAssemble :(int)CanPrintFull;
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanPrint</td>
<td>Set this to 1 to allow the user to print the document</td>
</tr>
<tr>
<td>CanCopy</td>
<td>Set this to 1 to allow the user to copy text and graphics from the document</td>
</tr>
<tr>
<td>CanChange</td>
<td>Set this to 1 to allow the user to edit the document</td>
</tr>
<tr>
<td>CanAddNotes</td>
<td>Set this to 1 to allow the user to add annotations</td>
</tr>
<tr>
<td>CanFillFields</td>
<td>Set this to 1 to allow the user to fill in form fields. Only works with 128-bit encryption.</td>
</tr>
<tr>
<td>CanCopyAccess</td>
<td>Set this to 1 to enable copying for use with accessibility features. Only works with 128-bit encryption.</td>
</tr>
<tr>
<td>CanAssemble</td>
<td>Set this to 1 to allow the user to assemble the document. Only works with 128-bit encryption.</td>
</tr>
<tr>
<td>CanPrintFull</td>
<td>Set this to 0 to force low-resolution printing of the document only. This prevents the document from being distilled into a new PDF document. Only works with 128-bit encryption.</td>
</tr>
</tbody>
</table>

#### Return values

Result is a 32-bit encoded number which should be passed to the Encrypt function.
Encrypt

Security and Signatures

Description

This function adds the specified security settings to the selected document.
From Quick PDF Library 8.11, the actual encryption of the document is delayed until the document is saved so this function can be called at any time, even before further content is added to the document.

Syntax

Dylib

```c
int DPLEncrypt(int InstanceID, wchar_t * Owner, wchar_t * User,
               int Strength, int Permissions);
```

Objective-C class

```objective-c
- (int)Encrypt:(NSString *)Owner :(NSString *)User :(int)Strength
   :(int)Permissions;
```

Parameters

<table>
<thead>
<tr>
<th>Owner</th>
<th>The owner or master password for the document</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The user password for the document</td>
</tr>
<tr>
<td>Strength</td>
<td>The strength of encryption to use:</td>
</tr>
<tr>
<td></td>
<td>0 = 40-bit encryption</td>
</tr>
<tr>
<td></td>
<td>1 = 128-bit RC4 encryption</td>
</tr>
<tr>
<td></td>
<td>2 = 128-bit AES encryption (requires Acrobat 7 or later)</td>
</tr>
<tr>
<td></td>
<td>3 = 256-bit AES encryption (requires Acrobat 9 or later)</td>
</tr>
<tr>
<td></td>
<td>4 = 256-bit AES encryption (requires Acrobat X or later)</td>
</tr>
<tr>
<td>Permissions</td>
<td>A value created with the <code>EncodePermissions</code> function</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document could not be encrypted. Use the <code>LastErrorCode</code> function to determine the reason for failure.</td>
</tr>
<tr>
<td>1</td>
<td>The document was encrypted successfully</td>
</tr>
</tbody>
</table>
EncryptFile

Security and Signatures

Description

Encrypts a file on disk and saves the results to a new file. The entire document does not have to be loaded into memory so this function can be used to encrypt huge documents.

Syntax

Dylib

```c
int DPLEncryptFile(int InstanceID, wchar_t * InputFileName,
                   wchar_t * OutputFileName, wchar_t * Owner, wchar_t * User,
                   int Strength, int Permissions);
```

Objective-C class

``` Objective-C
- (int)EncryptFile:(NSString *)InputFileName :(NSString *)OutputFileName
  :(NSString *)Owner :(NSString *)User :(int)Strength
  :(int)Permissions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFileName</td>
<td>The name of the file to encrypt.</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>The name of the destination file to create.</td>
</tr>
<tr>
<td>Owner</td>
<td>The owner password to use for the encrypted file. This is sometimes called</td>
</tr>
<tr>
<td></td>
<td>the &quot;master&quot; password or the &quot;permissions&quot; password. This password will</td>
</tr>
<tr>
<td></td>
<td>be needed to change the document.</td>
</tr>
<tr>
<td>User</td>
<td>The user password to use for the encrypted file. This is sometimes called</td>
</tr>
<tr>
<td></td>
<td>the &quot;open&quot; password, it will allow the user to open the document but not to</td>
</tr>
<tr>
<td></td>
<td>use the document in ways not permitted.</td>
</tr>
<tr>
<td>Strength</td>
<td>The strength of encryption to use:</td>
</tr>
<tr>
<td></td>
<td>0 = 40-bit RC4 encryption</td>
</tr>
<tr>
<td></td>
<td>1 = 128-bit RC4 encryption</td>
</tr>
<tr>
<td></td>
<td>2 = 128-bit AES encryption (requires Acrobat 7 or later)</td>
</tr>
<tr>
<td></td>
<td>3 = 256-bit AES encryption (requires Acrobat 9 or later)</td>
</tr>
<tr>
<td></td>
<td>4 = 256-bit AES encryption (requires Acrobat X or later)</td>
</tr>
<tr>
<td>Permissions</td>
<td>A value created with the EncodePermissions function</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file could not be encrypted. Check the result of the</td>
</tr>
<tr>
<td></td>
<td>LastErrorCode function to determine the cause of the</td>
</tr>
<tr>
<td></td>
<td>failure.</td>
</tr>
<tr>
<td>1</td>
<td>The document was encrypted successfully</td>
</tr>
</tbody>
</table>
EncryptWithFingerprint

Security and Signatures

Description
Encrypts the selected document using the encryption "fingerprint" obtained from another document using the \texttt{GetEncryptionFingerprint} function. The selected document will be encrypted with the same owner and user passwords as the document the fingerprint was taken from.

Syntax

\textbf{Dylib}

\begin{verbatim}
int DPLEncryptWithFingerprint(int InstanceID, wchar_t * Fingerprint);
\end{verbatim}

\textbf{Objective-C class}

\begin{verbatim}
- (int)EncryptWithFingerprint:(NSString *)Fingerprint;
\end{verbatim}

Parameters

\begin{itemize}
\item \textbf{Fingerprint} A fingerprint returned by the \texttt{GetEncryptionFingerprint} function
\end{itemize}

Return values

\begin{itemize}
\item \texttt{0} The fingerprint was invalid or the document was already encrypted
\item \texttt{1} The document was successfully encrypted using the supplied fingerprint
\end{itemize}
EncryptionAlgorithm

Document properties, Security and Signatures

Description

Returns the encryption algorithm used to encrypt the selected document. The EncryptionStrength function can be used to determine the encryption key length.

Syntax

Dylib

```c
int DPLEncryptionAlgorithm(int InstanceID);
```

Objective-C class

```
-(int)EncryptionAlgorithm
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document is not encrypted</td>
</tr>
<tr>
<td>1</td>
<td>The document is encrypted using RC4 encryption</td>
</tr>
<tr>
<td>2</td>
<td>The document is encrypted using AES encryption</td>
</tr>
</tbody>
</table>
EncryptionStatus

Document properties, Security and Signatures

Description

Determines the encryption status of the selected document.

Syntax

Dylib

```c
int DPLEncryptionStatus(int InstanceID);
```

Objective-C class

```c
- (int)EncryptionStatus
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected document is not encrypted</td>
</tr>
<tr>
<td>1</td>
<td>The document is encrypted with Adobe &quot;Standard&quot; encryption</td>
</tr>
<tr>
<td>2</td>
<td>The document is encrypted with an unknown encryption</td>
</tr>
</tbody>
</table>
**EncryptionStrength**

Document properties, Security and Signatures

**Description**

If the selected document has been encrypted this function returns the encryption strength. This is the length of the key used to encrypt the contents of the document.

**Syntax**

**Dylib**

```c
int DPLEncryptionStrength(int InstanceID);
```

**Objective-C class**

```c
-(int)EncryptionStrength
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected document is not encrypted</td>
</tr>
<tr>
<td>40</td>
<td>The document has been encrypted with 40-bit encryption (Adobe Acrobat 3.x and 4.x)</td>
</tr>
<tr>
<td>128</td>
<td>The document has been encrypted with 128-bit encryption (Adobe Acrobat 5.x)</td>
</tr>
<tr>
<td>256</td>
<td>The document has been encrypted with 256-bit encryption (Acrobat 9 or Acrobat 10). Use the SecurityInfo function to determine which version of encryption was used.</td>
</tr>
</tbody>
</table>
For detailed page layouts the `BeginPageUpdate` function can be called before a group of drawing commands. The page layout commands will then be buffered until a matching call to this function.

**Syntax**

**Dylib**

```c
int DPLEndPageUpdate(int InstanceID);
```

**Objective-C class**

```c
-(int)EndPageUpdate
```
EndSignProcessToFile

Security and Signatures

Description

Completes a digital signature process and writes the signed document to a file. The result returned by EndSignProcessToFile will always be zero. To check the result of the digital signature signing process call the GetSignProcessResult function.

Syntax

**Dylib**

```c
int DPLEndSignProcessToFile(int InstanceID, int SignProcessID, wchar_t * OutputFile);
```

**Objective-C class**

```objc
- (int)EndSignProcessToFile:(int)SignProcessID :(NSString *)OutputFile;
```

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputFile</td>
<td>The path and name of the file to save the signed PDF to.</td>
</tr>
</tbody>
</table>
EndSignProcessToString

Security and Signatures

Description

Completes a digital signature process and returns the signed document as a string of 8-bit bytes. The result returned by EndSignProcessToString will always be zero. To check the result of the digital signature signing process call the GetSignProcessResult function.

Syntax

**Dylib**

```c
char * DPLEndSignProcessToString(int InstanceID, int SignProcessID);
```

**Objective-C class**

```objc
- (NSData *)EndSignProcessToString:(int)SignProcessID;
```

Parameters

ExtractFilePageContentToString

**Extraction, Page manipulation**

**Description**

Retrieves the page description operators that define the layout of any page in a PDF document. This function does not load the entire file into memory so it can be used with arbitrarily large documents.

**Syntax**

**Dylib**

```c
char * DPLExtractFilePageContentToString(int InstanceID,
            wchar_t * InputFileName, wchar_t * Password, int Page);
```

**Objective-C class**

```objective-c
- (NSData *)ExtractFilePageContentToString:(NSString *)InputFileName
                                      :(NSString *)Password :(int)Page;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFileName</td>
<td>The path and file name of the file to extract page content from.</td>
</tr>
<tr>
<td>Password</td>
<td>The password to use when opening the file</td>
</tr>
<tr>
<td>Page</td>
<td>The number of the page to extract. The first page in the document is page 1.</td>
</tr>
</tbody>
</table>
ExtractFilePageText

Description

Extracts the text of any page in a PDF file.

This function internally uses the direct access functionality. The entire file is not loaded into memory, so this function can be used on arbitrarily large documents.

Two different methods are provided for extracting text from the selected page in a variety of output formats.

The DASetTextExtractionWordGap, DASetTextExtractionOptions and DASetTextExtractionArea functions can be used to adjust the text extraction process.

Syntax

Dylib

```c
wchar_t * DPLExtractFilePageText(int InstanceID, wchar_t * InputFileName, wchar_t * Password, int Page, int Options);
```

Objective-C class

```c
- (NSString *)ExtractFilePageText:(NSString *)InputFileName :(NSString *)Password :(int)Page :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFileName</td>
<td>The path and file name of the file to extract text from.</td>
</tr>
<tr>
<td>Password</td>
<td>The password to use, if any, when opening the file</td>
</tr>
<tr>
<td>Page</td>
<td>The number of the page that must be extracts. The first page in the document is page 1.</td>
</tr>
</tbody>
</table>
| Options      | Using the standard text extraction algorithm: 0 = Extract text in human readable format 1 = Deprecated 2 = Return a CSV string including font, color, size and position of each piece of text on the page  
Using the more accurate but slower text extraction algorithm: 3 = Return a CSV string for each piece of text on the page with the following format: Font Name, Text Color, Text Size, X1, Y1, X2, Y2, X3, Y3, X4, Y4, Text The co-ordinates are the four points bounding the text, measured using the units set with the SetMeasurementUnits function and the origin set with the SetOrigin function. Co-ordinate order is anti-clockwise with the bottom left corner first. 4 = Similar to option 3, but individual words are returned, making searching for words easier 5 = Similar to option 3 but character widths are output after each block of text 6 = Similar to option 4 but character widths are output after each line of text 7 = Extract text in human readable format with improved accuracy compared to option 0 8 = Similar output format as option 0 but using the more accurate algorithm. Returns unformatted lines. |
**ExtractFilePageTextBlocks**

**Text, Extraction, Page properties**

**Description**

Similar to the `ExtractFilePageText` function but the results are stored in a text block list rather than returned as a CSV string.

This function internally uses the direct access functionality.

Once the results are in the text block list, functions such as `DAGetTextBlockCount`, `DAGetTextBlockText` and `DAGetTextBlockColor` can be used to retrieve the properties of each block of text.

**Syntax**

**Dylib**

```c
int DPLExtractFilePageTextBlocks(int InstanceID, wchar_t * InputFileName, wchar_t * Password, int Page, int Options);
```

**Objective-C class**

- `(int)ExtractFilePageTextBlocks:(NSString *)InputFileName (NSString *)Password :(int)Page :(int)Options;`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFileName</td>
<td>The path and file name of the file to extract text from.</td>
</tr>
<tr>
<td>Password</td>
<td>The password to use, if any, when opening the file</td>
</tr>
<tr>
<td>Page</td>
<td>The number of the page that must be extracts. The first page in the document is page 1.</td>
</tr>
<tr>
<td>Options</td>
<td>3 = Normal extraction</td>
</tr>
<tr>
<td></td>
<td>4 = Split words</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The text could not be extracted</td>
</tr>
<tr>
<td>1</td>
<td>A TextBlockListID value</td>
</tr>
</tbody>
</table>
ExtractFilePages

Document manipulation, Extraction, Page manipulation

Description

Extracts ranges of pages from a PDF document on disk and places the extracted pages into a new PDF document.

The ExtractFilePagesEx function (introduced in version 9.14) is able to produce smaller output files using a cross reference stream instead of a cross reference table.

Syntax

**Dylib**

```c
int DPLExtractFilePages(int InstanceID, wchar_t * InputFileName, wchar_t * Password, wchar_t * OutputFileName, wchar_t * RangeList);
```

**Objective-C class**

```objc
- (int)ExtractFilePages:(NSString *)InputFileName :(NSString *)Password :(NSString *)OutputFileName :(NSString *)RangeList;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputFileName</strong></td>
<td>The path and name of the document that contains the pages to extract.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>The password to use when opening the document</td>
</tr>
<tr>
<td><strong>OutputFileName</strong></td>
<td>The path and name of the document to create containing the extracted pages.</td>
</tr>
<tr>
<td><strong>RangeList</strong></td>
<td>The pages to extract, for example &quot;10,15,18-20,25-35&quot;. Invalid characters will be ignored. Reversed page ranges such as &quot;5-1&quot; will be accepted. Duplicate page numbers will be accepted but if a change is made to such a page the same changes will appear on the duplicate pages. The list of pages will not be sorted so the resulting document will have the pages in the specified order.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td>The pages could not be extracted. Use the <strong>LastErrorCode</strong> function to determine the cause of the failure.</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td>The pages were extracted successfully</td>
</tr>
</tbody>
</table>
**ExtractFilePagesEx**

**Document manipulation, Extraction, Page manipulation**

**Description**

Similar to the `ExtractFilePages` function but is able to generate smaller output files using cross reference streams rather than a cross reference table.

**Syntax**

**Dylib**

```c
int DPLEXtractFilePagesEx(int InstanceID, wchar_t * InputFileName,
                          wchar_t * Password, wchar_t * OutputFileName,
                          wchar_t * RangeList, int Options);
```

**Objective-C class**

```objc
- (int)ExtractFilePagesEx:(NSString *)InputFileName :(NSString *)Password
                     :(NSString *)OutputFileName :(NSString *)RangeList
                     :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>InputFileName</code></td>
<td>The path and name of the document that contains the pages to extract.</td>
</tr>
<tr>
<td><code>Password</code></td>
<td>The password to use when opening the document.</td>
</tr>
<tr>
<td><code>OutputFileName</code></td>
<td>The path and name of the document to create containing the extracted pages.</td>
</tr>
<tr>
<td><code>RangeList</code></td>
<td>The pages to extract, for example &quot;10,15,18-20,25-35&quot;. Invalid characters will be ignored. Reversed page ranges such as &quot;5-1&quot; will be accepted. Duplicate page numbers will be accepted but if a change is made to such a page the same changes will appear on the duplicate pages. The list of pages will not be sorted so the resulting document will have the pages in the specified order.</td>
</tr>
</tbody>
</table>
| `Options`       | 0 = Use a cross reference table  
                  1 = Use a cross reference stream (smaller output file size) |

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The pages could not be extracted. Use the <code>LastErrorCode</code> function to determine the cause of the failure.</td>
</tr>
<tr>
<td>1</td>
<td>The pages were extracted successfully</td>
</tr>
</tbody>
</table>
**ExtractPageRanges**

**Document manipulation, Extraction, Page manipulation**

**Description**

Use this function to extract one or more non-consecutive pages from a document to a new document.

**Syntax**

**Dylib**

```c
int DPLExtractPageRanges(int InstanceID, wchar_t * RangeList);
```

**Objective-C class**

```objc
- (int)ExtractPageRanges:(NSString *)Rangelist;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RangeList</strong></td>
<td>The pages to extract, for example &quot;10,15,18-20,25-35&quot;. Invalid characters and duplicate page numbers in the string will be ignored. Reversed page ranges such as &quot;5-1&quot; will be accepted. The list of pages will be sorted resulting in the pages being extracted in numerical order.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page extraction did not succeed. The original document remains as the selected document.</td>
</tr>
<tr>
<td>1</td>
<td>The page extraction was successful. The new document containing the selected pages is now the selected document.</td>
</tr>
</tbody>
</table>
ExtractPageTextBlocks

Text, Extraction

Description

Similar to the GetPageText function but the results are stored in a text block list rather than returned as a CSV string.

Once the results are in the text block list, functions such as GetTextBlockCount, GetTextBlockText and GetTextBlockColor can be used to retrieve the properties of each block of text.

Syntax

Dylib

```c
int DPLExtractPageTextBlocks(int InstanceID, int ExtractOptions);
```

Objective-C class

```
-(int)ExtractPageTextBlocks:(int)ExtractOptions;
```

Parameters

<table>
<thead>
<tr>
<th>ExtractOptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Normal extraction</td>
</tr>
<tr>
<td>4</td>
<td>Split words</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The text could not be extracted</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A TextBlockListID value</td>
</tr>
</tbody>
</table>
ExtractPages
Extraction, Page manipulation

Description
Copies the selected document to a new document, but retains only the specified pages.
If successful, the new document will be selected and the original document will be removed from memory.

Syntax
Dylib
int DPLEXtractPages(int InstanceID, int StartPage, int PageCount);

Objective-C class
- (int)ExtractPages:(int)StartPage :(int)PageCount;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPage</td>
<td>The page number of the first page to extract</td>
</tr>
<tr>
<td>PageCount</td>
<td>The total number of pages to extract</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Failed, use LastErrorCode for further details</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
FileListCount

Miscellaneous functions

Description

Returns the number of items in the specified file list.

Syntax

Dylib

```c
int DPLFileListCount(int InstanceID, wchar_t * ListName);
```

Objective-C class

```objective-c
- (int)FileListCount:(NSString *)ListName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListName</td>
<td>The name of the file list</td>
</tr>
</tbody>
</table>
FileListItem

Miscellaneous functions

Description

Returns the file name stored at the specified index in the named list.

Syntax

**Dylib**

```c
wchar_t * DPLFileListItem(int InstanceID, wchar_t * ListName, int Index);
```

**Objective-C class**

```objective-c
- (NSString *)FileListItem:(NSString *)ListName :(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListName</td>
<td>The name of the list to work with</td>
</tr>
<tr>
<td>Index</td>
<td>The index of the file name to retrieve. The first item has an index of 1.</td>
</tr>
</tbody>
</table>
FindFonts
Fonts, Document properties

Description

Analyses the selected document and finds all available fonts. The number of found fonts is returned. Calling this function a second time will return zero as all relevant fonts were found the first time the function was called. These fonts are then available in conjunction to the fonts added with the Add*Font functions and will also be counted in subsequent calls to the FontCount function.

Syntax

Dylib

```c
int DPLFindFonts(int InstanceID);
```

Objective-C class

```objc
-(int)FindFonts
```

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No fonts were found in the document</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of fonts that were found</td>
</tr>
</tbody>
</table>
FindFormFieldByTitle

Form fields

Description

Finds the index of the form field with the specified title.

Syntax

Dylib

int DPLFindFormFieldByTitle(int InstanceID, wchar_t * Title);

Objective-C class

- (int)FindFormFieldByTitle:(NSString *)Title;

Parameters

| Title | The title of the form field to find. |

Return values

| 0 | The form field could not be found |
| Non-zero | The Index of the form field with the specified title |
FindImages

Image handling, Document properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Searches the selected document for embedded images. Returns the number of images found.

Syntax

Dylib

```c
int DPLFindImages(int InstanceID);
```

Objective-C class

```c
- (int)FindImages
```
**FitImage**

*Image handling, Page layout*

**Description**

This function allows an image to be placed into an area on the page. The aspect ratio of the image is preserved, and the alignment and rotation of the image can be specified.

**Syntax**

**Dylib**

```c
int DPLFitImage(int InstanceID, double Left, double Top, double Width, double Height, int HAlign, int VAlign, int Rotate);
```

**Objective-C class**

```objective-c
- (int)FitImage:(double)Left :(double)Top :(double)Width :(double)Height :(int)HAlign :(int)VAlign :(int)Rotate;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The horizontal co-ordinate of the left-edge of the bounding box</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The vertical co-ordinate of the top-edge of the bounding box</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the bounding box</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the bounding box</td>
</tr>
<tr>
<td><strong>HAlign</strong></td>
<td>Horizontal alignment of the image within the bounding box: 0 = Left, 1 = Center, 2 = Right</td>
</tr>
<tr>
<td><strong>VAlign</strong></td>
<td>Vertical alignment of the image within the bounding box: 0 = Top, 1 = Center, 2 = Bottom</td>
</tr>
<tr>
<td><strong>Rotate</strong></td>
<td>The rotation of the image: 0 = Normal, 1 = 90 degrees anti-clockwise, 2 = 90 degrees clockwise, 3 = 180 degrees</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image could not be drawn. Either a valid image has not been selected or the HAlign, VAlign or Rotate parameters are out of range.</td>
</tr>
<tr>
<td>1</td>
<td>The image was drawn successfully</td>
</tr>
</tbody>
</table>
**FitRotatedTextBox**

**Text, Page layout**

**Description**

Similar to the **FitTextBox** function, but the angle of the box can be rotated by any angle. The text size is adjusted to ensure that all the text fits into the available space. The top-left corner of the box before it is rotated is used as the rotation point.

**Syntax**

**Dylib**

```c
int DPLFitRotatedTextBox(int InstanceID, double Left, double Top,
                        double Width, double Height, double Angle, wchar_t * Text,
                        int Options);
```

**Objective-C class**

```c
- (int)FitRotatedTextBox:(double)Left :(double)Top :(double)Width
  :(double)Height :(double)Angle :(NSString *)Text :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The horizontal co-ordinate of the top-left corner of the box before it is rotated</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The vertical co-ordinate of the top-left corner of the box before it is rotated</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the box before it is rotated</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the box before it is rotated</td>
</tr>
<tr>
<td><strong>Angle</strong></td>
<td>The angle in degrees that the box should be rotated by. A positive angle rotates the box in an anti-clockwise direction, a negative angle rotated the box in a clockwise direction.</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text that will be fitted into the box</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Vertical alignment: 0 = Centered 1 = Top 2 = Bottom</td>
</tr>
</tbody>
</table>

If 100 is added to these values long words will not be split up, the font size will be reduced until the longest word fits into the available width.

If 1000 is added to these values the font size will be allowed to increase until the text fills the available area.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Options parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>The rotated text box was drawn successfully</td>
</tr>
</tbody>
</table>
FitTextBox

Text, Page layout

Description

Similar to the DrawText function, but the text size is adjusted to ensure that all the text fits into the available space.

Syntax

Dylib

```c
int DPLFitTextBox(int InstanceID, double Left, double Top, double Width, double Height, wchar_t * Text, int Options);
```

Objective-C class

```objective-c
-(int)FitTextBox:(double)Left :(double)Top :(double)Width :(double)Height :(NSString *)Text :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the bounding box</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the bounding box</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the bounding box</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the bounding box</td>
</tr>
<tr>
<td>Text</td>
<td>The text to display in the box</td>
</tr>
<tr>
<td>Options</td>
<td>Vertical alignment:</td>
</tr>
<tr>
<td></td>
<td>0 = Centered</td>
</tr>
<tr>
<td></td>
<td>1 = Top</td>
</tr>
<tr>
<td></td>
<td>2 = Bottom</td>
</tr>
</tbody>
</table>

If 100 is added to these values long words will not be split up, the font size will be reduced until the longest word fits into the available width.

If 1000 is added to these values the font size will be allowed to increase until the text fills the available area.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Options specified were out of range</td>
</tr>
<tr>
<td>1</td>
<td>The text was drawn successfully</td>
</tr>
</tbody>
</table>
FlattenAnnot
Annotations and hotspot links, Page layout

Description
Flattens the specified annotation by merging the appearance stream with the selected page.

Syntax

**Dylib**

```c
int DPLFlattenAnnot(int InstanceID, int Index, int Options);
```

**Objective-C class**

```c
-(int)FlattenAnnot:(int)Index :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>This parameter is reserved for future use and should always be set to zero.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified annotation could not be flattened</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
FlattenFormField
Form fields, Page layout

Description

Use this function to draw the visual appearance onto the page it is associated with. The form field will then be removed from the document and only it's appearance will remain - it will no longer be an interactive field.

If the field is flattened successfully the field index of subsequent form fields will be decreased by 1. From version 9.11 this function no longer updates the form field's appearance stream before flattening. To update the appearance stream before flattening, use the UpdateAndFlattenFormField function or call UpdateAppearanceStream followed by a call to this function.

Syntax

Dylib

int DPLFlattenFormField(int InstanceID, int Index);

Objective-C class

- (int)FlattenFormField:(int)Index;

Parameters

| Index | The index of the form field to work with. The first form field has an index of 1. |

Return values

| 0     | The form field could not be found or it was not possible to flatten the form field |
| 1     | The form field was flattened successfully |
**FontCount**

**Fonts**

**Description**

Returns the total number of fonts added to the PDF file. This function does not take into account the fonts that may have already been in an existing PDF document which was loaded with the `LoadFromFile` function unless the `FindFonts` function has been called.

**Syntax**

**Dylib**

```c
int DPLFontCount(int InstanceID);
```

**Objective-C class**

```c
- (int)FontCount
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No fonts have been added to the document or <code>FindFonts</code> has not found any fonts in an existing document.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of fonts added to the PDF plus the number of fonts found with <code>FindFonts</code>.</td>
</tr>
</tbody>
</table>
**Fonts**

**Description**

Returns the font family of the selected font, if available.

**Syntax**

**Dylib**

```c
wchar_t * DPLFontFamily(int InstanceID);
```

**Objective-C class**

```objective-c
- (NSString *)FontFamily
```
FontHasKerning

Text, Fonts

Description

Indicated whether the selected font has kerning information.

Syntax

Dylib

int DPLFontHasKerning(int InstanceID);

Objective-C class

- (int)FontHasKerning

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected font does not have any kerning information</td>
</tr>
<tr>
<td>1</td>
<td>The selected font has at least one kerning pair</td>
</tr>
</tbody>
</table>
Returns the name of the selected font. A font is automatically selected when it is added to the document. The GetFontID and SelectFont functions can be used to select a different font.

**Syntax**

**Dylib**

```c
wchar_t * DPLFontName(int InstanceID);
```

**Objective-C class**

```objc
- (NSString *)FontName
```
FontReference

Fonts

Description

Returns the internal reference of the selected font.

Syntax

Dylib

wchar_t * DPLFontReference(int InstanceID);

Objective-C class

- (NSString *)FontReference
Description

Returns the size in bytes of the selected font. A value will only be returned for embedded TrueType or Type1 fonts. A value will not be returned for subsetted fonts or standard fonts.

Syntax

Dylib

int DPLFontSize(int InstanceID);

Objective-C class

- (int)FontSize
Fonts

FontType

Description

Used to determine the type of the selected font.

Syntax

Dylib

`int DPLFontType(int InstanceID);`

Objective-C class

- `(int)FontType`

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No font has been selected</td>
</tr>
<tr>
<td>1</td>
<td>Unknown</td>
</tr>
<tr>
<td>2</td>
<td>Standard</td>
</tr>
<tr>
<td>3</td>
<td>TrueType</td>
</tr>
<tr>
<td>4</td>
<td>Embedded TrueType</td>
</tr>
<tr>
<td>5</td>
<td>Packaged</td>
</tr>
<tr>
<td>6</td>
<td>Type1</td>
</tr>
<tr>
<td>7</td>
<td>Subsetted</td>
</tr>
<tr>
<td>8</td>
<td>Type3</td>
</tr>
<tr>
<td>9</td>
<td>Type1 CID</td>
</tr>
<tr>
<td>10</td>
<td>TrueType CID</td>
</tr>
<tr>
<td>11</td>
<td>CJK</td>
</tr>
</tbody>
</table>
**FormFieldCount**

**Form fields**

**Description**

Returns the total number of form fields in the selected document. The Index parameter of the various form field functions must be a number from 1 to the value returned by this function.

If a form field is deleted or flattened successfully it will be removed from the document, the total field count will be reduced and the field Index of the subsequent fields will be reduced by 1.

**Syntax**

**Dylib**

```c
int DPLFormFieldCount(int InstanceID);
```

**Objective-C class**

```objective-c
- (int)FormFieldCount
```

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>There are no form fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-zero</strong></td>
<td>The number of form fields in the document</td>
</tr>
</tbody>
</table>
FormFieldHasParent

Description

This function returns 1 if the specified form field is the child of another field.

Syntax

Dylib

```c
int DPLFormFieldHasParent(int InstanceID, int Index);
```

Objective-C class

```c
-(int)FormFieldHasParent:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>The index of the form field. The first field has an index of 1.</td>
</tr>
</tbody>
</table>
FormFieldJavaScriptAction
Form fields, JavaScript

Description

Adds JavaScript to a form field for any of the possible action types.

Syntax

**Dylib**

```c
int DPLFormFieldJavaScriptAction(int InstanceID, int Index,
wchar_t * ActionType, wchar_t * JavaScript);
```

**Objective-C class**

```objective-c
- (int)FormFieldJavaScriptAction:(int)Index :(NSString *)ActionType
  :(NSString *)JavaScript;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Index of the form field</td>
</tr>
<tr>
<td>ActionType</td>
<td>The action type:</td>
</tr>
<tr>
<td></td>
<td>E = An action to be performed when the cursor enters the annotation's active area</td>
</tr>
<tr>
<td></td>
<td>X = An action to be performed when the cursor exits the annotation's active area</td>
</tr>
<tr>
<td></td>
<td>D = An action to be performed when the mouse button is pressed inside the annotation's active area</td>
</tr>
<tr>
<td></td>
<td>U = An action to be performed when the mouse button is released inside the annotation's active area</td>
</tr>
<tr>
<td></td>
<td>Fo = An action to be performed when the annotation receives the input focus</td>
</tr>
<tr>
<td></td>
<td>Bi = An action to be performed when the annotation loses the input focus (blurred)</td>
</tr>
<tr>
<td></td>
<td>K = An action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This allows the keystroke to be checked for validity and rejected or modified.</td>
</tr>
<tr>
<td></td>
<td>F = An action to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.</td>
</tr>
<tr>
<td></td>
<td>V = An action to be performed when the field's value is changed. This allows the new value to be checked for validity.</td>
</tr>
<tr>
<td></td>
<td>C = An action to be performed in order to recalculate the value of this field when that of another field changes</td>
</tr>
<tr>
<td>JavaScript</td>
<td>The JavaScript to execute.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cannot find the form field</td>
</tr>
<tr>
<td>1</td>
<td>The JavaScript action was added to the form field successfully</td>
</tr>
</tbody>
</table>
FormFieldWebLinkAction

Form fields

Description

Adds an action to the specified form field that links to an internet address.

Syntax

Dylib

```c
int DPLFormFieldWebLinkAction(int InstanceID, int Index,
    wchar_t * ActionType, wchar_t * Link);
```

Objective-C class

```objective-c
- (int)FormFieldWebLinkAction:(int)Index :(NSString *)ActionType
    :(NSString *)Link;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to set the action of</td>
</tr>
<tr>
<td>ActionType</td>
<td>The action type:&lt;br&gt; E = An action to be performed when the cursor enters the annotation's active area&lt;br&gt; X = An action to be performed when the cursor exits the annotation's active area&lt;br&gt; D = An action to be performed when the mouse button is pressed inside the annotation's active area&lt;br&gt; U = An action to be performed when the mouse button is released inside the annotation's active area&lt;br&gt; Fo = An action to be performed when the annotation receives the input focus&lt;br&gt; Bl = An action to be performed when the annotation loses the input focus (blurred)&lt;br&gt; K = An action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This allows the keystroke to be checked for validity and rejected or modified.&lt;br&gt; F = An action to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.&lt;br&gt; V = An action to be performed when the field's value is changed. This allows the new value to be checked for validity.&lt;br&gt; C = An action to be performed in order to recalculate the value of this field when that of another field changes</td>
</tr>
</tbody>
</table>
| Link       | The URL to link to. Some examples:<br> "http://www.example.com"
            <br> "mailto:info@example.com" |

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found, or the ActionType was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The web link action was added to the form field successfully</td>
</tr>
</tbody>
</table>
GetActionDest

Annotations and hotspot links

Description

This function will return a DestID if the specified action has a destination entry. The DestID can be used with the GetDestPage, GetDestType and GetDestValue functions.

Syntax

**Dylib**

```c
int DPLGetActionDest(int InstanceID, int ActionID);
```

**Objective-C class**

```objc
- (int)GetActionDest:(int)ActionID;
```

Parameters

<table>
<thead>
<tr>
<th>ActionID</th>
<th>An ActionID as returned by the GetAnnotActionID, GetOutlineActionID or GetFormFieldActionID functions</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The specified action does not have a destination entry</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-zero</strong></td>
<td>A DestID that can be used with the destination functions.</td>
</tr>
</tbody>
</table>
**GetActionType**

**Annotations and hotspot links**

**Description**

Returns the action type of the specified action, for example "GoTo" or "GoToR".

**Syntax**

**Dylib**

```c
wchar_t * DPLGetActionType(int InstanceID, int ActionID);
```

**Objective-C class**

```objc
- (NSString *)GetActionType:(int)ActionID;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>ActionID</strong></th>
<th>An ActionID as returned by the <a href="#">GetAnnotActionID</a>, <a href="#">GetOutlineActionID</a> or <a href="#">GetFormFieldActionID</a> functions</th>
</tr>
</thead>
</table>
GetActionURL
Annotations and hotspot links

Description
Returns the target URL of the specified action.

Syntax
Dylib

wchar_t * DPLGetActionURL(int InstanceID, int ActionID);

Objective-C class

- (NSString *)GetActionURL:(int)ActionID;

Parameters
| ActionID | An ActionID as returned by the GetAnnotActionID, GetOutlineActionID or GetFormFieldActionID functions |
GetAnalysisInfo

Document properties

Description

Returns individual items from the results of the analysis done by the AnalyseFile function.

Syntax

Dylib

```c
wchar_t * DPLGetAnalysisInfo(int InstanceID, int AnalysisID, int AnalysisItem);
```

Objective-C class

```objective-c
-(NSString *)GetAnalysisInfo:(int)AnalysisID :(int)AnalysisItem;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalysisID</td>
<td>The ID of the set of analysis results to query, as returned by the AnalyseFile function</td>
</tr>
<tr>
<td>AnalysisItem</td>
<td>The specific analysis result to retrieve: 0 = File name (eg. &quot;c:\hello.pdf&quot;) 1 = File size (eg. &quot;2048&quot; for a file exactly 2K in size) 2 = Author 3 = Title 4 = Subject 5 = Keywords 6 = Creator 7 = Producer 8 = PDF version (eg. &quot;1.4&quot;) 9 = Page count (eg. &quot;120&quot;) 10 = Creation date 11 = Modification date 12 = Document ID 13 = The supplied password: &quot;None&quot; for no security &quot;User&quot; for the user password &quot;Owner&quot; for the owner password 14 = Document contains usage rights (eg. Reader Extensions) &quot;No&quot; if there is no usage rights dictionary &quot;Yes&quot; if there is a usage rights dictionary 15 = Name of signature in the usage rights dictionary 20..30 = Equivalent to SecurityInfo(0)..SecurityInfo(10) 31 = Number of form fields in the document 41..43 = Equivalent to SecurityInfo(11)..SecurityInfo(13)</td>
</tr>
</tbody>
</table>
GetAnnotActionID
Annotations and hotspot links

Description

This function will return an ActionID if the specified annotation has an action dictionary. The ActionID can be used with the `GetActionType` and `GetActionDest` functions and can also be compared to the values returned by `GetOutlineActionID` to determine if an annotation action is shared with an outline action.

Syntax

**Dylib**

```c
int DPLGetAnnotActionID(int InstanceID, int Index);
```

**Objective-C class**

```c
- (int)GetAnnotActionID:(int)Index;
```

Parameters

| **Index** | The index of the annotation. The first annotation on the page has an index of 1. |
GetAnnotDblProperty
Annotations and hotspot links

Description

Returns a property of the specified annotation.

Syntax

Dylib

double DPLGetAnnotDblProperty(int InstanceID, int Index, int Tag);

Objective-C class

- (double)GetAnnotDblProperty:(int)Index :(int)Tag;

Parameters

<table>
<thead>
<tr>
<th><strong>Index</strong></th>
<th>The index of the annotation. The first annotation on the page has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tag</strong></td>
<td>105 = Left</td>
</tr>
<tr>
<td></td>
<td>106 = Top</td>
</tr>
<tr>
<td></td>
<td>107 = Width</td>
</tr>
<tr>
<td></td>
<td>108 = Height</td>
</tr>
<tr>
<td></td>
<td>119 = Gray color component</td>
</tr>
<tr>
<td></td>
<td>120 = Red color component</td>
</tr>
<tr>
<td></td>
<td>121 = Green color component</td>
</tr>
<tr>
<td></td>
<td>122 = Blue color component</td>
</tr>
<tr>
<td></td>
<td>123 = Cyan color component</td>
</tr>
<tr>
<td></td>
<td>124 = Magenta color component</td>
</tr>
<tr>
<td></td>
<td>125 = Yellow color component</td>
</tr>
<tr>
<td></td>
<td>126 = Black color component</td>
</tr>
<tr>
<td></td>
<td>132 = Border width</td>
</tr>
</tbody>
</table>


**GetAnnotDest**

**Annotations and hotspot links**

**Description**

This function will return a DestID if the specified annotation has a destination entry. The DestID can be used with the `GetDestPage`, `GetDestType` and `GetDestValue` functions.

If the annotation does not have a destination entry, this function will return zero.

The `GetAnnotActionID` function might return a value that can be used with the `GetActionDest` function.

**Syntax**

**Dylib**

```c
int DPLGetAnnotDest(int InstanceID, int Index);
```

**Objective-C class**

```c
-(int)GetAnnotDest:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the annotation. The first annotation on the page has an index of 1.</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The specified annotation does not have a destination entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>A DestID that can be used with the destination functions.</td>
</tr>
</tbody>
</table>
GetAnnotIntProperty

Annotations and hotspot links

Description

Returns a property of the specified annotation.

Syntax

**Dylib**

```c
int DPLGetAnnotIntProperty(int InstanceID, int Index, int Tag);
```

**Objective-C class**

```
- (int)GetAnnotIntProperty:(int)Index :(int)Tag;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the annotation. The first annotation on the page has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>109 = Flags&lt;br&gt;116 = Page number of &quot;GoToR&quot; action (1 is first page)&lt;br&gt;128 = Index of the annotation that this annotation is in reply to&lt;br&gt;131 = Page number of &quot;GoTo&quot; action&lt;br&gt;133 = Returns 1 if a &quot;Launch&quot; or &quot;GoToR&quot; action's NewWindow property is set</td>
</tr>
</tbody>
</table>
GetAnnotQuadCount

Annotations and hotspot links

Description

Returns the number of quads (rectangular areas) within the specified annotation.

Syntax

Dylib

```c
int DPLGetAnnotQuadCount(int InstanceID, int Index);
```

Objective-C class

```objective-c
- (int)GetAnnotQuadCount:(int)Index;
```

Parameters

| Index | The index of the annotation. The first annotation on the page has an index of 1. |
GetAnnotQuadPoints

Description

Returns a component of the specified quad (rectangular area) contained within the specified annotation.

From version 7.25 the order of the co-ordinates has changed for consistency between GetPageText and SetAnnotQuadPoints.

Syntax

Dylib

```c
double DPLGetAnnotQuadPoints(int InstanceID, int Index, int QuadNumber,
                               int PointNumber);
```

Objective-C class

```objective-c
-(double)GetAnnotQuadPoints:(int)Index :(int)QuadNumber :(int)PointNumber;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the annotation. The first annotation on the page has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuadNumber</td>
<td>The number of the quad to access. The first quad has a QuadNumber of 1.</td>
</tr>
</tbody>
</table>
| PointNumber | 1 = The horizontal co-ordinate of the bottom-left corner  
|           | 2 = The vertical co-ordinate of the bottom-left corner  
|           | 3 = The horizontal co-ordinate of the bottom-right corner  
|           | 4 = The vertical co-ordinate of the bottom-right corner  
|           | 5 = The horizontal co-ordinate of the top-right corner  
|           | 6 = The vertical co-ordinate of the top-right corner  
|           | 7 = The horizontal co-ordinate of the top-left corner  
|           | 8 = The vertical co-ordinate of the top-left corner  |
GetAnnotSoundToFile
Annotations and hotspot links

Description
Copies the sound data stored in the specified annotation into a file.

Syntax

**Dylib**

```c
int DPLGetAnnotSoundToFile(int InstanceID, int Index, int Options,
wchar_t * SoundFileName);
```

**Objective-C class**

```objc
- (int)GetAnnotSoundToFile:(int)Index :(int)Options
:(NSString *)SoundFileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the annotation. The first annotation on the page has an index</td>
</tr>
<tr>
<td>of 1.</td>
<td></td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>0 = Sound data as stored in the PDF</td>
</tr>
<tr>
<td></td>
<td>1 = Encode data as a WAV file</td>
</tr>
<tr>
<td><strong>SoundFileName</strong></td>
<td>The path and name of the file to create containing the sound data.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The sound could not be written</td>
</tr>
<tr>
<td>1</td>
<td>The sound was written successfully</td>
</tr>
</tbody>
</table>
GetAnnotSoundToString

Annotations and hotspot links

Description

Returns the sound data stored in the specified annotation.

Syntax

Dylib

```c
char * DPLGetAnnotSoundToString(int InstanceID, int Index, int Options);
```

Objective-C class

```objective-c
- (NSData *)GetAnnotSoundToString:(int)Index :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
</tbody>
</table>
| **Options** | 0 = Sound data as stored in the PDF  
1 = Encode data as a WAV file |
GetAnnotStrProperty
Annotations and hotspot links

Description
Returns a property of the specified annotation.

Syntax

Dylib

wchar_t * DPLGetAnnotStrProperty(int InstanceID, int Index, int Tag);

Objective-C class

- (NSString *)GetAnnotStrProperty:(int)Index :(int)Tag;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Annotation type</td>
</tr>
<tr>
<td>102</td>
<td>Contents</td>
</tr>
<tr>
<td>103</td>
<td>Name</td>
</tr>
<tr>
<td>104</td>
<td>Modified date</td>
</tr>
<tr>
<td>110</td>
<td>Author</td>
</tr>
<tr>
<td>111</td>
<td>URL of a link annotation</td>
</tr>
<tr>
<td>112</td>
<td>Action type of link annotation, e.g. &quot;URI&quot;, &quot;Launch&quot;, &quot;GoToR&quot;</td>
</tr>
<tr>
<td>113</td>
<td>The &quot;Win&quot; file name of a &quot;Launch&quot; action</td>
</tr>
<tr>
<td>114</td>
<td>The &quot;F&quot; file name of a &quot;Launch&quot; action</td>
</tr>
<tr>
<td>115</td>
<td>The &quot;F&quot; file name of a &quot;GoToR&quot; action</td>
</tr>
<tr>
<td>117</td>
<td>The name of the annotation icon</td>
</tr>
<tr>
<td>118</td>
<td>Color space, e.g. &quot;Gray&quot;, &quot;RGB&quot;, &quot;CMYK&quot;</td>
</tr>
<tr>
<td>127</td>
<td>Subject of the annotation</td>
</tr>
<tr>
<td>129</td>
<td>The &quot;UF&quot; file name of a &quot;Launch&quot; action</td>
</tr>
<tr>
<td>130</td>
<td>The &quot;UF&quot; file name of a &quot;GoToR&quot; action</td>
</tr>
</tbody>
</table>
GetBarcodeWidth

Vector graphics, Page layout

Description

Returns the total width of a barcode based on the width of the smallest bars in the barcode.

Syntax

**Dylib**

```c
double DPLGetBarcodeWidth(int InstanceID, double NominalWidth,
wchar_t * Text, int Barcode);
```

**Objective-C class**

```objective-c
- (double)GetBarcodeWidth:(double)NominalWidth :(NSString *)Text
 :(int)Barcode;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NominalWidth</strong></td>
<td>The desired width of the narrowest bars in the barcode</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The barcode data</td>
</tr>
</tbody>
</table>
| **Barcode**  | 1 = Code39 (or Code 3 of 9)  
2 = EAN-13  
3 = Code128  
4 = PostNet  
5 = Interleaved 2 of 5 |
GetBaseURL
Document properties, Annotations and hotspot links

Description

Returns the Base URL for all URL links in the document. For example, if the Base URL was set to "http://www.example.com/" and a URL link destination was set to "index.html" then the link will point to "http://www.example.com/index.html".

Use the AddLinkToWeb function to add a URL link to the current page.

Syntax

Dylib

wchar_t * DPLGetBaseURL(int InstanceID);

Objective-C class

- (NSString *)GetBaseURL
GetCSDictEPSG

Measurement and coordinate units

Description

Returns the EPSG reference code for a coordinate system dictionary (see www.epsg.org).

Syntax

Dylib

```c
int DPLGetCSDictEPSG(int InstanceID, int CSDictID);
```

Objective-C class

```c
-(int)GetCSDictEPSG:(int)CSDictID;
```

Parameters

<table>
<thead>
<tr>
<th>CSDictID</th>
<th>A value returned from the GetMeasureDictGCSDict or GetMeasureDictDCSDict functions</th>
</tr>
</thead>
</table>
GetCSDictType
Measurement and coordinate units

Description
Returns the coordinate system type for a coordinate system dictionary.

Syntax
Dylib

```c
int DPLGetCSDictType(int InstanceID, int CSDictID);
```

Objective-C class

```objc
- (int)GetCSDictType:(int)CSDictID;
```

Parameters

<table>
<thead>
<tr>
<th>CSDictID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSDictID</td>
<td>A value returned from the <code>GetMeasureDictGCSDict</code> or <code>GetMeasureDictDCSDict</code> functions</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The CSDictID parameter was incorrect</td>
</tr>
<tr>
<td>1</td>
<td>A geographic coordinate system (GEOGCS)</td>
</tr>
<tr>
<td>2</td>
<td>A projected coordinate system (PROJCS)</td>
</tr>
</tbody>
</table>
GetCSDictWKT

Measurement and coordinate units

Description

Returns the Well Known Text (WKT) description of a coordinate system dictionary.

Syntax

Dylib

wchar_t * DPLGetCSDictWKT(int InstanceID, int CSDictID);

Objective-C class

- (NSString *)GetCSDictWKT:(int)CSDictID;

Parameters

<table>
<thead>
<tr>
<th>CSDictID</th>
<th>A value returned from the GetMeasureDictGCSDict or GetMeasureDictDCSDict functions</th>
</tr>
</thead>
</table>
GetCanvasDC

Vector graphics, Document management

Description

Creates a canvas of the specified size and returns a Windows device context DC that can be drawn on using Win32 drawing commands. When drawing operations are complete, call the `LoadFromCanvasDC` function to create a new document from the supplied drawing commands.

The return value is defined as either an unsigned integer or a signed integer on different platforms and editions of the library.

Syntax

**Dylib**

```c
int DPLGetCanvasDC(int InstanceID, int CanvasWidth, int CanvasHeight);
```

**Objective-C class**

```objc
- (int)GetCanvasDC:(int)CanvasWidth :(int)CanvasHeight;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanvasWidth</td>
<td>The width of the canvas</td>
</tr>
<tr>
<td>CanvasHeight</td>
<td>The height of the canvas</td>
</tr>
</tbody>
</table>
GetCatalogInformation

Document properties

Description

This function allows you to retrieve custom information from the "Catalog" section of the document.

Syntax

Dylib

wchar_t * DPLGetCatalogInformation(int InstanceID, wchar_t * Key);

Objective-C class

- (NSString *)GetCatalogInformation:(NSString *)Key;

Parameters

| Key    | The name of the key to retrieve. This key must have a special prefix assigned to you by Adobe to avoid conflicts with other software. |
GetContentStreamToString
Page properties, Content Streams and Optional Content Groups, Page manipulation

Description

Returns the PDF page description commands in the content stream part that was selected with the SelectContentStream function.

Syntax

**Dylib**

```c
char * DPLGetContentStreamToString(int InstanceID);
```

**Objective-C class**

```objc
- (NSData *)GetContentStreamToString
```
**GetCustomInformation**

**Document properties**

**Description**

Returns a custom value from the document. This function and the `SetCustomInformation` function can be used to store and retrieve custom document metadata.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetCustomInformation(int InstanceID, wchar_t * Key);
```

**Objective-C class**

```objc
- (NSString *)GetCustomInformation:(NSString *)Key;
```

**Parameters**

| Key | Specifies which key to retrieve the value of |

**Return values**

The value of the specified key, or an empty string if the key could not be found. An empty string will also be returned if the key is "Author", "Keywords", "Subject", "Title", "Creator" or "Producer". For these keys, use the `GetInformation function`. 
GetCustomKeys

Document properties

Description

Returns all the custom keys in either the Document Information Dictionary or Document Catalog as a CSV string. See the StoreCustomData and RetrieveCustomData functions for details of how to manipulate the data stored under these keys.

Syntax

Dylib

```c
wchar_t * DPLGetCustomKeys(int InstanceID, int Location);
```

Objective-C class

```c
- (NSString *)GetCustomKeys:(int)Location;
```

Parameters

<table>
<thead>
<tr>
<th>Location</th>
<th>The location to extract custom key names from:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Document Information Dictionary</td>
</tr>
<tr>
<td>2</td>
<td>Document Catalog</td>
</tr>
</tbody>
</table>
GetDefaultPrinterName

Rendering and printing

Description

Returns the name of the default printer. This name can be used with the PrintDocument or NewCustomPrinter functions.

Syntax

**Dylib**

```c
wchar_t * DPLGetDefaultPrinterName(int InstanceID);
```

**Objective-C class**

```objc
-(NSString *)GetDefaultPrinterName
```
GetDestName

Annotations and hotspot links

Description

Returns the name of the specified destination.

Syntax

Dylib

```c
wchar_t * DPLGetDestName(int InstanceID, int DestID);
```

Objective-C class

```objc
- (NSString *)GetDestName:(int)DestID;
```

Parameters

| DestID | The ID of the destination to analyse. A valid destination ID is returned by the `GetOutlineDest` function. |
GetDestPage
Annotations and hotspot links

Description
Returns the page number of the specified destination, or zero if the destination is invalid or does not contain a page number.

Syntax
Dylib

```c
int DPLGetDestPage(int InstanceID, int DestID);
```

Objective-C class

```objc
- (int)GetDestPage:(int)DestID;
```

Parameters

<table>
<thead>
<tr>
<th>DestID</th>
<th>The ID of the destination to analyse. A valid destination ID is returned by the <code>GetOutlineDest</code> function.</th>
</tr>
</thead>
</table>
GetDestType

Annotations and hotspot links

Description

Returns the type of the specified destination.

Syntax

**Dylib**

```c
int DPLGetDestType(int InstanceID, int DestID);
```

**Objective-C class**

```objc
- (int)GetDestType:(int)DestID;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestID</td>
<td>The ID of the destination to analyse. A valid destination ID is returned by the <strong>GetOutlineDest</strong> function.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;XYZ&quot; - the target page is positioned at the Left and Top properties of the destination, and the Zoom property specifies the zoom percentage</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Fit&quot; - the entire page is zoomed to fit the window</td>
</tr>
<tr>
<td>3</td>
<td>&quot;FitH&quot; - the page is zoomed so that the entire width of the page is visible. The height of the page may be greater or less than the height of the window. The page is positioned vertically at the Top property of the destination.</td>
</tr>
<tr>
<td>4</td>
<td>&quot;FitV&quot; - the page is zoomed so that the entire height of the page can be seen. The width of the page may be greater or less than the width of the window. The page is positioned horizontally at the Left property of the destination.</td>
</tr>
<tr>
<td>5</td>
<td>&quot;FitR&quot; - the page is zoomed so that a certain rectangle on the page is visible. The Left, Top, Right and Bottom properties of the destination define the rectangle on the page.</td>
</tr>
<tr>
<td>6</td>
<td>&quot;FitB&quot; - the page is zoomed so that it's bounding box is visible</td>
</tr>
<tr>
<td>7</td>
<td>&quot;FitBH&quot; - the page is positioned vertically at the value of the Top property of the destination, and the page is zoomed so that the entire width of the page's bounding box is visible</td>
</tr>
<tr>
<td>8</td>
<td>&quot;FitBV&quot; - the page is positioned at the value of the Left property of the destination is visible, and the page is zoomed just enough to fit the entire height of the bounding box into the window</td>
</tr>
</tbody>
</table>
Descrition

Returns the value of a property of the specified destination.

Syntax

Dylib

```c
double DPLGetDestValue(int InstanceID, int DestID, int ValueKey);
```

Objective-C class

```objc
- (double)GetDestValue:(int)DestID :(int)ValueKey;
```

Parameters

<table>
<thead>
<tr>
<th>DestID</th>
<th>The ID of the destination to analyse. A valid destination ID is returned by the <code>GetOutlineDest</code> function.</th>
</tr>
</thead>
</table>
| ValueKey | 1 = Left  
|          | 2 = Top   
|          | 3 = Right 
|          | 4 = Bottom|
|          | 5 = Zoom  |
GetDocJavaScript

Document properties, JavaScript

Description

Retrieves the JavaScript linked to a specified document action.

Syntax

Dylib

wchar_t * DPLGetDocJavaScript(int InstanceID, wchar_t * ActionType);

Objective-C class

- (NSString *)GetDocJavaScript:(NSString *)ActionType;

Parameters

<table>
<thead>
<tr>
<th>ActionType</th>
<th>Retrieve the JavaScript linked to this action:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;DC&quot;</td>
<td>Document close</td>
</tr>
<tr>
<td>&quot;WS&quot;</td>
<td>Will save</td>
</tr>
<tr>
<td>&quot;DS&quot;</td>
<td>Did save</td>
</tr>
<tr>
<td>&quot;WP&quot;</td>
<td>Will print</td>
</tr>
<tr>
<td>&quot;DP&quot;</td>
<td>Did print</td>
</tr>
</tbody>
</table>
GetDocumentFileName

Document management

Description

Returns the file name of the selected document if it was opened using `LoadFromFile`.

Syntax

**Dylib**

```c
wchar_t * DPLGetDocumentFileName(int InstanceID);
```

**Objective-C class**

```c
- (NSString *)GetDocumentFileName
```
GetDocumentFileSize

Document properties

**Description**

Returns the file size of the selected document.

The size cannot be determined dynamically - it will only be set directly after a call to `LoadFromFile`, `LoadFromStream`, `LoadFromString`, `LoadFromVariant`, `SaveToFile`, `SaveToStream`, `SaveToString` or `SaveToVariant`.

**Syntax**

**Dylib**

```c
int DPLGetDocumentFileSize(int InstanceID);
```

**Objective-C class**

```objective-c
-(int)GetDocumentFileSize
```
GetDocumentID

Document management

Description

Returns the ID of the document with the specified index.

Syntax

Dylib

    int DPLGetDocumentID(int InstanceID, int Index);

Objective-C class

    - (int)GetDocumentID:(int)Index;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the document to query. Must be 1 or greater.</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The specified index was out of range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>The ID of the specified document</td>
</tr>
</tbody>
</table>
GetDocumentIdentifier

Document properties

Description

Returns the document identifier. This identifier consists of two parts, each string. The first string does not change when the document is resaved with an "incremental update" in Acrobat. This can be seen as the permanent identifier for the document. The second part will change each time the document is resaved, even if the resave is an incremental update.

Syntax

Dylib

```c
wchar_t * DPLGetDocumentIdentifier(int InstanceID, int Part, int Options);
```

Objective-C class

```c
- (NSString *)GetDocumentIdentifier:(int)Part :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Part</th>
<th>0 = Permanent identifier</th>
<th>1 = Changeable identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td>0 = Return the identifier as a string of characters</td>
<td>1 = Return the identifier as a hexadecimal string</td>
</tr>
</tbody>
</table>
GetDocumentMetadata

Document properties

Description

Returns the document's metadata, if any.

Syntax

Dylib

wchar_t * DPLGetDocumentMetadata(int InstanceID);

Objective-C class

- (NSString *)GetDocumentMetadata
GetDocumentRepaired

Document properties, Document management

Description

Indicates whether the document was repaired when it was loaded.

Syntax

Dylib

```c
int DPLGetDocumentRepaired(int InstanceID);
```

Objective-C class

```
-(int)GetDocumentRepaired
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document was not repaired</td>
</tr>
<tr>
<td>1</td>
<td>The document was repaired</td>
</tr>
</tbody>
</table>
GetDocumentResourceList

Document properties

Description

Returns a list of the PDF resource names used in the document. For advanced use only.

Syntax

**Dylib**

```c
wchar_t * DPLGetDocumentResourceList(int InstanceID);
```

**Objective-C class**

```objective-c
- (NSString *)GetDocumentResourceList
```
GetEmbeddedFileContentToFile

Description

Extracts the specified embedded file and writes the content to the specified file.

Syntax

Dylib

```c
int DPLGetEmbeddedFileContentToFile(int InstanceID, int Index,
                                   wchar_t * FileName);
```

Objective-C class

```objective-c
- (int)GetEmbeddedFileContentToFile:(int)Index :(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the embedded file. Must be a value between 1 and the value returned by <code>EmbeddedFileCount</code>.</td>
</tr>
<tr>
<td>FileName</td>
<td>The path and file name of the file to write the contents to.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not write to the specified file or Index parameter was invalid.</td>
</tr>
<tr>
<td>1</td>
<td>Embedded file contents written to the specified file successfully.</td>
</tr>
</tbody>
</table>
GetEmbeddedFileContentToString

**Document properties**

**Description**

Extracts the specified embedded file and returns the content as a string.

**Syntax**

**Dylib**

```c
char * DPLGetEmbeddedFileContentToString(int InstanceID, int Index);
```

**Objective-C class**

```objc
-(NSData *)GetEmbeddedFileContentToString:(int)Index;
```

**Parameters**

| Index | The index of the embedded file. Must be a value between 1 and the value returned by `EmbeddedFileCount`. |
GetEmbeddedFileID

Description

Returns the ID of the specified embedded file. This ID can be used with the AddLinkToEmbeddedFile function.

Syntax

Dylib

int DPLGetEmbeddedFileID(int InstanceID, int Index);

Objective-C class

- (int)GetEmbeddedFileID:(int)Index;

Parameters

| Index | The index of the embedded file. Must be a value between 1 and the value returned by EmbeddedFileCount. |

Return values

| 0     | The specified index was invalid |
| Non-zero | An EmbeddedFileID value |
GetEmbeddedFileIntProperty

Document properties

Description

Retrieves an integer property of the specified embedded file.

Syntax

Dylib

int DPLGetEmbeddedFileIntProperty(int InstanceID, int Index, int Tag);

Objective-C class

- (int)GetEmbeddedFileIntProperty:(int)Index :(int)Tag;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The index of the embedded file. Must be a value between 1 and the value returned by EmbeddedFileCount.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Deprecated (previously same as 6)</td>
</tr>
<tr>
<td>6</td>
<td>File size in bytes</td>
</tr>
</tbody>
</table>
GetEmbeddedFileStrProperty

**Document properties**

**Description**

Retrieves a string property of the specified embedded file.
Use the `SetEmbeddedFileStrProperty` function to change the values.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetEmbeddedFileStrProperty(int InstanceID, int Index, int Tag);
```

**Objective-C class**

``` objective-c
- (NSString *)GetEmbeddedFileStrProperty:(int)Index :(int)Tag;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File name</td>
</tr>
<tr>
<td>2</td>
<td>MIME type</td>
</tr>
<tr>
<td>3</td>
<td>Creation date</td>
</tr>
<tr>
<td>4</td>
<td>Modification date</td>
</tr>
<tr>
<td>5</td>
<td>Title</td>
</tr>
<tr>
<td>7</td>
<td>Description</td>
</tr>
</tbody>
</table>

**Tag**

- 1 = File name
- 2 = MIME type
- 3 = Creation date
- 4 = Modification date
- 5 = Title
- 7 = Description
GetEncryptionFingerprint

Document properties, Security and Signatures

Description

Returns all the encryption information for the selected document. This encryption "fingerprint" can be used to encrypt a different document using the EncryptWithFingerprint function. This allows a new document to be encrypted with the same passwords as an existing document without actually knowing these passwords.

Syntax

**Dylib**

```c
wchar_t * DPLGetEncryptionFingerprint(int InstanceID);
```

**Objective-C class**

```objective-c
- (NSString *)GetEncryptionFingerprint
```
GetFileMetadata

Document properties

Description

Returns the metadata in a file, if any.

Syntax

Dylib

wchar_t * DPLGetFileMetadata(int InstanceID, wchar_t * InputFileName, wchar_t * Password);

Objective-C class

- (NSString *)GetFileMetadata:(NSString *)InputFileName:(NSString *)Password;

Parameters

<table>
<thead>
<tr>
<th>InputFileName</th>
<th>The path and name of the document to extract metadata from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>The password to use when opening the document</td>
</tr>
</tbody>
</table>
GetFirstChildOutline

Outlines

Description

Returns the ID of the outline that is the first child of the specified outline.

Syntax

Dylib

```c
int DPLGetFirstChildOutline(int InstanceID, int OutlineID);
```

Objective-C class

```objc
-(int)GetFirstChildOutline:(int)OutlineID;
```

Parameters

| OutlineID | The ID of the outline item to work with. This ID is returned by the `NewOutline` or `NewStaticOutline` functions, or retrieved with the `GetOutlineID` function or `Get*Outline` functions. |
GetFirstOutline

Outlines

Description

Returns the ID of the first outline in the hierarchy.

Syntax

Dylib

int DPLGetFirstOutline(int InstanceID);

Objective-C class

- (int)GetFirstOutline
**GetFontEncoding**

**Fonts**

**Description**

Returns the font encoding of the selected font.

**Syntax**

**Dylib**

```c
int DPLGetFontEncoding(int InstanceID);
```

**Objective-C class**

```c
-(int)GetFontEncoding
```

**Return values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unknown</td>
</tr>
<tr>
<td>1</td>
<td>MacRomanEncoding</td>
</tr>
<tr>
<td>2</td>
<td>WinAnsiEncoding</td>
</tr>
<tr>
<td>3</td>
<td>MacExpertEncoding</td>
</tr>
<tr>
<td>5</td>
<td>No encoding</td>
</tr>
</tbody>
</table>
GetFontFlags

Fonts

Description

Returns the value of the specified bit in the flags property of the selected font.

Syntax

**Dylib**

```c
int DPLGetFontFlags(int InstanceID, int FontFlagItemID);
```

**Objective-C class**

```c
- (int)GetFontFlags:(int)FontFlagItemID;
```

Parameters

<table>
<thead>
<tr>
<th>FontFlagItemID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fixed</td>
</tr>
<tr>
<td>2</td>
<td>Serif</td>
</tr>
<tr>
<td>3</td>
<td>Symbolic</td>
</tr>
<tr>
<td>4</td>
<td>Script</td>
</tr>
<tr>
<td>5</td>
<td>Italic</td>
</tr>
<tr>
<td>6</td>
<td>AllCap</td>
</tr>
<tr>
<td>7</td>
<td>SmallCap</td>
</tr>
<tr>
<td>8</td>
<td>ForceBold</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Flag is not set</td>
</tr>
<tr>
<td>1</td>
<td>Flag is set</td>
</tr>
</tbody>
</table>
GetFontID

Text, Fonts

Description

Returns the ID of the specified font. Before this function is used a call to FindFonts or FontCount must be made in order to generate a list of fonts available for use in the selected document.

Syntax

Dylib

int DPLGetFontID(int InstanceID, int Index);

Objective-C class

- (int)GetFontID:(int)Index;

Parameters

Index               The index of the font. The first font has an index of 1.

Return values

0                   The index is out of bounds
Non-zero            The ID of the specified font
GetFontObjectNumber

Fonts

Description

This specialized function returns the internal object number of the selected font. This object number can sometimes be used by other systems.

Syntax

Dylib

```c
int DPLGetFontObjectNumber(int InstanceID);
```

Objective-C class

```objective-c
-(int)GetFontObjectNumber
```
GetFormFieldActionID

Form fields, Annotations and hotspot links

Description

Returns an ActionID for the specified form field which can be used with the various action manipulation functions such as `GetActionType`, `GetActionDest`, `GetActionURL` and `SetActionURL`.

There are different trigger events and each one has its own action.

Syntax

**Dylib**

```c
int DPLGetFormFieldActionID(int InstanceID, int Index, wchar_t * TriggerEvent);
```

**Objective-C class**

```objc
- (int)GetFormFieldActionID:(int)Index :(NSString *)TriggerEvent;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
</table>
| TriggerEvent | Retrieve the action for the specified trigger event:  
Empty string = simple activation action  
E = annotation enter action  
X = annotation exit action  
D = annotation button down action  
U = annotation button up action  
Fo = annotation input focus action  
Bl = annotation input blur (opposite of focus) action  
PO = annotation page open action  
PC = annotation page close action  
PV = annotation page visible action  
PI = annotation page invisible action  
K = form field change action  
F = form field format action  
V = form field validate action  
C = form field calculate action |

Return values

| 0 | The field index was invalid or the field does not have an action associated with the specified trigger event |
| Non-zero | The form field's ActionID for the specified trigger event |
**GetFormFieldAlignment**

**Form fields**

**Description**

Retrieves the text alignment of the specified form field.

**Syntax**

**Dylib**

```
int DPLGetFormFieldAlignment(int InstanceID, int Index);
```

**Objective-C class**

```
- (int)GetFormFieldAlignment:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Left alignment (this value is also returned if the form field could not be found)</td>
</tr>
<tr>
<td>1</td>
<td>Centered</td>
</tr>
<tr>
<td>2</td>
<td>Right aligned</td>
</tr>
</tbody>
</table>
GetFormFieldAnnotFlags

Form fields

Description

Get the "annotation" flags for the specified form field. This is for advanced use. See the PDF specification for full details.

Syntax

**Dylib**

```
int DPLGetFormFieldAnnotFlags(int InstanceID, int Index);
```

**Objective-C class**

```
- (int)GetFormFieldAnnotFlags:(int)Index;
```

Parameters

| Index | The index of the form field to check |
**GetFormFieldBackgroundColor**

**Form fields, Color**

**Description**

Returns the background color of the specified field.

**Syntax**

**Dylib**

```c
double DPLGetFormFieldBackgroundColor(int InstanceID, int Index, int ColorComponent);
```

**Objective-C class**

```c
-(double)GetFormFieldBackgroundColor:(int)Index :(int)ColorComponent;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field to examine</td>
</tr>
</tbody>
</table>
| **ColorComponent** | 1 = Red  
2 = Green  
3 = Blue |
GetFormFieldBackgroundColorType

Form fields, Color

Description

Returns the number of color components of the specified field's background.

Syntax

**Dylib**

```c
int DPLGetFormFieldBackgroundColorType(int InstanceID, int Index);
```

**Objective-C class**

```objc
- (int)GetFormFieldBackgroundColorType:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the field to examine</th>
</tr>
</thead>
</table>

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gray</td>
</tr>
<tr>
<td>3</td>
<td>RGB</td>
</tr>
<tr>
<td>4</td>
<td>CMYK</td>
</tr>
</tbody>
</table>
GetFormFieldBorderColor

Form fields, Color

Description

Returns the color of the specified field's border.

Syntax

Dylib

```c
double DPLGetFormFieldBorderColor(int InstanceID, int Index,
                                 int ColorComponent);
```

Objective-C class

```objc
-(double)GetFormFieldBorderColor:(int)Index :(int)ColorComponent;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to examine</th>
</tr>
</thead>
</table>
| ColorComponent | 1 = Red  
|                     | 2 = Green  
|                     | 3 = Blue  |
GetFormFieldBorderColorType

Form fields, Color

Description

Returns the number of color components of the specified field’s border.

Syntax

**Dylib**

```c
int DPLGetFormFieldBorderColorType(int InstanceID, int Index);
```

**Objective-C class**

```c
- (int)GetFormFieldBorderColorType:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to examine</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>1</th>
<th>Gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>RGB</td>
</tr>
<tr>
<td>4</td>
<td>CMYK</td>
</tr>
</tbody>
</table>
GetFormFieldBorderProperty

Form fields

Description

Returns various properties of the specified field's border.

Syntax

Dylib

```c
double DPLGetFormFieldBorderProperty(int InstanceID, int Index,
int PropKey);
```

Objective-C class

```objc
- (double)GetFormFieldBorderProperty:(int)Index :(int)PropKey;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to examine</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropKey</td>
<td>1 = Border width</td>
</tr>
<tr>
<td></td>
<td>2 = Dash on</td>
</tr>
<tr>
<td></td>
<td>3 = Dash off</td>
</tr>
</tbody>
</table>
GetFormFieldBorderStyle

Form fields

Description

Returns the border style of the specified field.

Syntax

Dylib

```c
int DPLGetFormFieldBorderStyle(int InstanceID, int Index);
```

Objective-C class

```objc
-(int)GetFormFieldBorderStyle:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to examine</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>Solid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dashed</td>
</tr>
<tr>
<td>2</td>
<td>Beveled</td>
</tr>
<tr>
<td>3</td>
<td>Inset</td>
</tr>
<tr>
<td>4</td>
<td>Underline</td>
</tr>
</tbody>
</table>
GetFormFieldBound

Form fields

Description

Returns the bounding box of the specified form field.

Syntax

Dylib

double DPLGetFormFieldBound(int InstanceID, int Index, int Edge);

Objective-C class

- (double)GetFormFieldBound:(int)Index :(int)Edge;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to measure. The first form field has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td>The required edge:</td>
</tr>
<tr>
<td></td>
<td>0 = Left</td>
</tr>
<tr>
<td></td>
<td>1 = Top</td>
</tr>
<tr>
<td></td>
<td>2 = Width</td>
</tr>
<tr>
<td></td>
<td>3 = Height</td>
</tr>
</tbody>
</table>

Return values

| 0     | Could not find the specified form field                                  |
| Non-zero | The requested measurement                                  |
GetFormFieldCaption

Form fields

Description

Returns the caption of a form field.

Syntax

Dylib

wchar_t * DPLGetFormFieldCaption(int InstanceID, int Index);

Objective-C class

- (NSString *)GetFormFieldCaption:(int)Index;

Parameters

| Index | The index of the form field |
GetFormFieldCheckStyle

Form fields

Description

Returns the checkbox style of the specified form field.

Syntax

Dylib

int DPLGetFormFieldCheckStyle(int InstanceID, int Index);

Objective-C class

-(int)GetFormFieldCheckStyle:(int)Index;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>

Return values

| 0 | Cross |
| 1 | Check (Tick) |
| 2 | Dot (Radio) |
| 3 | XP Check |
| 4 | XP Radio |
| 5 | Diamond |
| 6 | Square |
| 7 | Start |
Form fields can be arranged in a hierarchical structure, and the title of the form field will be the full path to the field, for example "names.first" or "address.zipcode". This function will return only the last part of the title, "first" or "zipcode" in this example.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetFormFieldChildTitle(int InstanceID, int Index);
```

**Objective-C class**

```c
- (NSString *)GetFormFieldChildTitle:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to retrieve the title of</td>
</tr>
</tbody>
</table>
GetFormFieldChoiceType

Form fields

Description
Determine whether a choice form field is a combo box or list box field.

Syntax

Dylib

```c
int DPLGetFormFieldChoiceType(int InstanceID, int Index);
```

Objective-C class

```objective-c
-(int)GetFormFieldChoiceType:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field is not a choice form field</td>
</tr>
<tr>
<td>1</td>
<td>The form field is a scrollable list box</td>
</tr>
<tr>
<td>2</td>
<td>The form field is a drop-down combo box</td>
</tr>
<tr>
<td>3</td>
<td>The form field is a multiselect scrollable list box</td>
</tr>
<tr>
<td>4</td>
<td>The form field is a drop-down combo box with an edit box</td>
</tr>
</tbody>
</table>
GetFormFieldColor

Form fields, Color

Description

Retrieves the color of the text in the form field. This function must be called three times to retrieve all components of the color (red, green and blue).

Syntax

Dylib

double DPLGetFormFieldColor(int InstanceID, int Index, int ColorComponent);

Objective-C class

- (double)GetFormFieldColor:(int)Index :(int)ColorComponent;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
</table>
| ColorComponent | 1 = Red  
2 = Green  
3 = Blue |
GetFormFieldComb

Form fields

Description

Returns 1 if the specified form field is marked as a comb field, where each character in the value occupies the same space in the field.

Syntax

Dylib

```c
int DPLGetFormFieldComb(int InstanceID, int Index);
```

Objective-C class

```c
- (int)GetFormFieldComb:(int)Index;
```

Parameters

| Index | The index of the form field |
GetFormFieldDefaultValue

Form fields

Description

Returns the default value of a form field. This is the value that the field will have when the form is reset.

Syntax

**Dylib**

```c
wchar_t * DPLGetFormFieldDefaultValue(int InstanceID, int Index);
```

**Objective-C class**

```objective-c
- (NSString *)GetFormFieldDefaultValue:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
GetFormFieldDescription

Form fields

Description

Retrieves the description of the specified form field if it has one.

Syntax

Dylib

wchar_t * DPLGetFormFieldDescription(int InstanceID, int Index);

Objective-C class

- (NSString *)GetFormFieldDescription:(int)Index;

Parameters

| Index   | The index of the form field to work with |
GetFormFieldFlags

Form fields

Description

Retrieves a form field's flags. This setting is for advanced purposes and most users will not need to use it.

Syntax

**Dylib**

```c
int DPLGetFormFieldFlags(int InstanceID, int Index);
```

**Objective-C class**

```c
-(int)GetFormFieldFlags:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cannot find the form field</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The flags for the specified form field</td>
</tr>
</tbody>
</table>
GetFormFieldFontName

Form fields

Description

Retrieves the name of the font that the specified form field is using.

Syntax

**Dylib**

```c
wchar_t * DPLGetFormFieldFontName(int InstanceID, int Index);
```

**Objective-C class**

```objc
- (NSString *)GetFormFieldFontName:(int)Index;
```

Parameters

| Index     | The index of the form field to work with. The first form field has an index of 1. |
GetFormFieldJavaScript

Form fields

Description

Retrieves the JavaScript associated with the specified action for the specified form field.

Syntax

Dylib

```c
wchar_t * DPLGetFormFieldJavaScript(int InstanceID, int Index,
wchar_t * ActionType);
```

Objective-C class

```c
- (NSString *)GetFormFieldJavaScript:(int)Index :(NSString *)ActionType;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionType</td>
<td>The action type:</td>
</tr>
<tr>
<td>E</td>
<td>An action to be performed when the cursor enters the annotation's active area</td>
</tr>
<tr>
<td>X</td>
<td>An action to be performed when the cursor exits the annotation's active area</td>
</tr>
<tr>
<td>D</td>
<td>An action to be performed when the mouse button is pressed inside the annotation's active area</td>
</tr>
<tr>
<td>U</td>
<td>An action to be performed when the mouse button is released inside the annotation's active area</td>
</tr>
<tr>
<td>Fo</td>
<td>An action to be performed when the mouse button is released inside the annotation's active area</td>
</tr>
<tr>
<td>Bl</td>
<td>An action to be performed when the annotation loses the input focus (blurred)</td>
</tr>
<tr>
<td>K</td>
<td>An action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This allows the keystroke to be checked for validity and rejected or modified.</td>
</tr>
<tr>
<td>F</td>
<td>An action to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.</td>
</tr>
<tr>
<td>V</td>
<td>An action to be performed when the field's value is changed. This allows the new value to be checked for validity.</td>
</tr>
<tr>
<td>C</td>
<td>An action to be performed in order to recalculate the value of this field when that of another field changes</td>
</tr>
</tbody>
</table>
GetFormFieldKidCount

Form fields

Description

Returns the number of children fields that the specified field has.

Syntax

Dylib

```c
int DPLGetFormFieldKidCount(int InstanceID, int Index);
```

Objective-C class

```objectivec
- (int)GetFormFieldKidCount:(int)Index;
```

Parameters

| Index | The index of the form field. The first field has an index of 1. |
GetFormFieldKidTempIndex

Form fields

Description

Returns a temporary index for the item fields (kids) of a radio button or checkbox form field group. An index of 1 will select the first radio or checkbox in the group, 2 the second and so on. The number of kids can be determined by calling GetFormFieldKidCount. This temporary index can be used with the regular form field functions such as GetFormFieldTabOrder and GetFormFieldValue.

If you need to update the subname for a choice field then you should use SetFormFieldSubChoice instead.

Syntax

Dylib

```c
int DPLGetFormFieldKidTempIndex(int InstanceID, int Index, int SubIndex);
```

Objective-C class

```objective-c
- (int)GetFormFieldKidTempIndex:(int)Index :(int)SubIndex;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the radio-button form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubIndex</td>
<td>The index of the sub-field. The first sub-field has an index of 1.</td>
</tr>
</tbody>
</table>
## GetFormFieldMaxLen

### Form fields

#### Description

Retrieves the maximum allowed length for a text form field.

#### Syntax

**Dylib**

```c
int DPLGetFormFieldMaxLen(int InstanceID, int Index);
```

**Objective-C class**

```objective-c
- (int)GetFormFieldMaxLen:(int)Index;
```

#### Parameters

| Index | The index of the form field to work with. The first form field has an index of 1. |

#### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field does not have a maximum length specified</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The maximum length of the form field</td>
</tr>
</tbody>
</table>
GetFormFieldNoExport

Form fields

Description

Returns the state of a field's NoExport flag.
The field will not be exported by a submit-form action if the NoExport flag is set.

Syntax

**Dylib**

```c
int DPLGetFormFieldNoExport(int InstanceID, int Index);
```

**Objective-C class**

```objectivec
- (int)GetFormFieldNoExport:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The field's NoExport flag is not set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The field's NoExport flag is set</td>
</tr>
</tbody>
</table>
GetFormFieldPage

Form fields

Description

Returns the page number that the specified form field is on.

Syntax

Dylib

```c
int DPLGetFormFieldPage(int InstanceID, int Index);
```

Objective-C class

```c
- (int)GetFormFieldPage:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to locate</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found, or the form field does not have valid page information</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The page number of the page that the form field is displayed on</td>
</tr>
</tbody>
</table>
GetFormFieldPrintable

Form fields

Description

Returns 1 if the specified field will be printed.

Syntax

Dylib

int DPLGetFormFieldPrintable(int InstanceID, int Index);

Objective-C class

- (int)GetFormFieldPrintable:(int)Index;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to check</th>
</tr>
</thead>
</table>

GetFormFieldReadOnly

Form fields

Description

Returns the state of a field's ReadOnly flag.
The user cannot change the value of a form field if the ReadOnly flag is set.

Syntax

**Dylib**

```c
int DPLGetFormFieldReadOnly(int InstanceID, int Index);
```

**Objective-C class**

```objc
- (int)GetFormFieldReadOnly:(int)Index;
```

Parameters

| **Index** | The index of the form field |

Return values

| **0** | The field's ReadOnly flag is not set |
| **1** | The field's ReadOnly flag is set |
GetFormFieldRequired

Form fields

Description

Returns the state of a field's is Required flag.
If this flag is set the field must have a value when the form is exported by a submit-form action.

Syntax

Dylib

```c
int DPLGetFormFieldRequired(int InstanceID, int Index);
```

Objective-C class

```c
- (int)GetFormFieldRequired:(int)Index;
```

Parameters

| Index | The index of the form field |

Return values

| 0     | The field's Required flag is not set |
| 1     | The field's Required flag is set |
GetFormFieldRichTextString

Form fields

Description

Retrieves the rich text (RV) or default style (DS) string of the specified form field using the given key. The format of the return value is defined in the PDF Specification under the section titled "Field Dictionaries".

Syntax

**Dylib**

```c
wchar_t * DPLGetFormFieldRichTextString(int InstanceID, int Index, wchar_t * Key);
```

**Objective-C class**

```objective-c
-(NSString *)GetFormFieldRichTextString:(int)Index :(NSString *)Key;
```

Parameters

| **Index** | The index of the required form field. The first form field has an index of 1. |
| **Key**   | The Key value to return. |
|           | "RV" = returns the rich text string |
|           | "DS" = returns the default style string |
GetFormFieldRotation

Form fields

Description

Returns the angle in degrees that the form field is rotated by. This is always a multiple of 90 degrees.

Syntax

**Dylib**

```c
int DPLGetFormFieldRotation(int InstanceID, int Index);
```

**Objective-C class**

```objc
- (int)GetFormFieldRotation:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to query</th>
</tr>
</thead>
</table>
GetFormFieldSubCount

Form fields

Description

For radio button, checkbox items and choice fields (scrollable list box or combo box drop-down list), this function returns the number of possible values the form field can be set to.

Syntax

Dylib

```c
int DPLGetFormFieldSubCount(int InstanceID, int Index);
```

Objective-C class

```objc
- (int)GetFormFieldSubCount:(int)Index;
```

Parameters

| Index | The index of the form field to examine |

Return values

| 0     | The form field could not be found or it does not have sub-values |
| Non-zero | The number of possible values the form field can be set to |
GetFormFieldSubDisplayName

Form fields

Description

Similar to GetFormFieldSubName but returns the display name of the specified choice field item.

Syntax

Dylib

wchar_t * DPLGetFormFieldSubDisplayName(int InstanceID, int Index, int SubIndex);

Objective-C class

- (NSString *)GetFormFieldSubDisplayName:(int)Index :(int)SubIndex;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to examine</td>
</tr>
<tr>
<td>SubIndex</td>
<td>The index of the sub-value to retrieve</td>
</tr>
</tbody>
</table>
**GetFormFieldSubName**

**Form fields**

**Description**

For radio button, checkbox and choice (scrollable list box or combo box drop-down list) form fields, this function returns the specified possible value.

For choice fields the `GetFormFieldSubDisplay_Name` function can be used to retrieve the display name of the choice item.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetFormFieldSubName(int InstanceID, int Index, int SubIndex);
```

**Objective-C class**

```c
- (NSString *)GetFormFieldSubName:(int)Index :(int)SubIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field to examine</td>
</tr>
<tr>
<td><strong>SubIndex</strong></td>
<td>The index of the sub-value to retrieve</td>
</tr>
</tbody>
</table>
GetFormFieldTabOrder

Form fields

Description

Returns the tab order of the specified form field. The first form field on the page has a tab order of 1.

Syntax

Dylib

```c
int DPLGetFormFieldTabOrder(int InstanceID, int Index);
```

Objective-C class

```objective-c
-(int)GetFormFieldTabOrder:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>

GetFormFieldTabOrderEx

Form fields

Description

Returns the tab order of the specified form field. Similar to the GetFormFieldTabOrder function but the order is adjusted to match certain popular PDF viewers. The first form field on the page has a tab order of 1.

Syntax

Dylib

```c
int DPLGetFormFieldTabOrderEx(int InstanceID, int Index, int Options);
```

Objective-C class

```c
-(int)GetFormFieldTabOrderEx:(int)Index :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
| Options   | 0 = Acrobat style  
           | 1 = Nuance style |

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Index parameters was invalid or the Options parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
GetFormFieldTextFlags

Form fields

Description

Returns certain properties of a text field.

Syntax

**Dylib**

```c
int DPLGetFormFieldTextFlags(int InstanceID, int Index, int ValueKey);
```

**Objective-C class**

```objective-c
- (int)GetFormFieldTextFlags:(int)Index :(int)ValueKey;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
| **ValueKey** | Indicates which property to analyse:  
1 = Multiline  
2 = Password  
3 = FileSelect  
4 = DoNotSpellCheck  
5 = DoNotScroll |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The flag for the specific property is not turned on. For example, if ValueKey is 5 and the function returns 0 this indicates that the form field is allowed to scroll.</td>
</tr>
<tr>
<td>1</td>
<td>The flag is turned on. For example, if ValueKey is 2 and the function returns 1 this indicates that the form field is a password field.</td>
</tr>
</tbody>
</table>
GetFormFieldTextSize

Form fields

Description

Retrieves the size of the text in the specified form field. A value of 0 indicates that the form field autosizes the text to fit into the available space.

Syntax

**Dylib**

```c
double DPLGetFormFieldTextSize(int InstanceID, int Index);
```

**Objective-C class**

```objective-c
- (double)GetFormFieldTextSize:(int)Index;
```

Parameters

| Index | The index of the form field to work with. The first form field has an index of 1. |
GetFormFieldTitle

Form fields

Description

Returns the title of the specified form field.

Syntax

Dylib

wchar_t * DPLGetFormFieldTitle(int InstanceID, int Index);

Objective-C class

- (NSString *)GetFormFieldTitle:(int)Index;

Parameters

| Index | The index of the required form field. The first form field has an index of 1. |
GetFormFieldType

Form fields

Description

Returns the type of the specified form field.

Syntax

Dylib

int DPLGetFormFieldType(int InstanceID, int Index);

Objective-C class

- (int)GetFormFieldType:(int)Index;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unknown</td>
</tr>
<tr>
<td>1</td>
<td>Text</td>
</tr>
<tr>
<td>2</td>
<td>Pushbutton</td>
</tr>
<tr>
<td>3</td>
<td>Checkbox</td>
</tr>
<tr>
<td>4</td>
<td>Radiobutton</td>
</tr>
<tr>
<td>5</td>
<td>Choice</td>
</tr>
<tr>
<td>6</td>
<td>Signature</td>
</tr>
<tr>
<td>7</td>
<td>Parent</td>
</tr>
</tbody>
</table>

Return values

0 Unknown
1 Text
2 Pushbutton
3 Checkbox
4 Radiobutton
5 Choice
6 Signature
7 Parent
GetFormFieldValue

Form fields

Description
Retrieves the value of the specified form field.

Syntax

**Dylib**

```c
wchar_t * DPLGetFormFieldValue(int InstanceID, int Index);
```

**Objective-C class**

```c
- (NSString *)GetFormFieldValue:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th><strong>Index</strong></th>
<th>The index of the form field to retrieve the value of</th>
</tr>
</thead>
</table>
GetFormFieldValueByTitle

Form fields

Description

Returns the value of the form field with the specified title.

Syntax

Dylib

```c
wchar_t * DPLGetFormFieldValueByTitle(int InstanceID, wchar_t * Title);
```

Objective-C class

```objc
-(NSString *)GetFormFieldValueByTitle:(NSString *)Title;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The title of the field.</td>
</tr>
</tbody>
</table>
GetFormFieldVisible

Form fields

Description

Returns 1 if the specified field will be visible when the document is viewed.

Syntax

Dylib

int DPLGetFormFieldVisible(int InstanceID, int Index);

Objective-C class

- (int)GetFormFieldVisible:(int)Index;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to check</th>
</tr>
</thead>
</table>

**Description**

Returns the internet address that the specified form field's action points to, if any.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetFormFieldWebLink(int InstanceID, int Index, wchar_t * ActionType);
```

**Objective-C class**

```c
- (NSString *)GetFormFieldWebLink:(int)Index :(NSString *)ActionType;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to change</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionType</td>
<td>The action type:</td>
</tr>
<tr>
<td>E</td>
<td>An action to be performed when the cursor enters the annotation's active area</td>
</tr>
<tr>
<td>X</td>
<td>An action to be performed when the cursor exits the annotation's active area</td>
</tr>
<tr>
<td>D</td>
<td>An action to be performed when the mouse button is pressed inside the annotation's active area</td>
</tr>
<tr>
<td>U</td>
<td>An action to be performed when the mouse button is released inside the annotation's active area</td>
</tr>
<tr>
<td>Fo</td>
<td>An action to be performed when the annotation receives the input focus (blurred)</td>
</tr>
<tr>
<td>Bl</td>
<td>An action to be performed when the annotation loses the input focus</td>
</tr>
<tr>
<td>K</td>
<td>An action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This allows the keystroke to be checked for validity and rejected or modified.</td>
</tr>
<tr>
<td>F</td>
<td>An action to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.</td>
</tr>
<tr>
<td>V</td>
<td>An action to be performed when the field's value is changed. This allows the new value to be checked for validity.</td>
</tr>
<tr>
<td>C</td>
<td>An action to be performed in order to recalculate the value of this field when that of another field changes</td>
</tr>
</tbody>
</table>
GetFormFontCount

Fonts, Form fields

Description

Returns the number of fonts available to fields in the form.

Syntax

Dylib

int DPLGetFormFontCount(int InstanceID);

Objective-C class

- (int)GetFormFontCount
GetFormFontName

Fonts, Form fields

Description

Returns the name of the font with the specified index.

Syntax

Dylib

wchar_t * DPLGetFormFontName(int InstanceID, int FontIndex);

Objective-C class

- (NSString *)GetFormFontName:(int)FontIndex;

Parameters

<table>
<thead>
<tr>
<th>FontIndex</th>
<th>The index of the font to work with. The first font in the form has an index of 1. Use GetFormFontCount to determine the number of fonts available in the form.</th>
</tr>
</thead>
</table>
GetGlobalJavaScript

Document properties, JavaScript

**Description**

Retrieves the global JavaScript for the specified package.

**Syntax**

**Dylib**

`wchar_t * DPLGetGlobalJavaScript(int InstanceID, wchar_t * PackageName);`

**Objective-C class**

`- (NSString *)GetGlobalJavaScript:(NSString *)PackageName;`

**Parameters**

| **PackageName** | The JavaScript stored under this package name will be retrieved. |
GetHTMLTextHeight

Text, HTML text

Description

Returns the height that a certain block of HTML text will occupy if drawn onto the page. See Appendix A for details of the supported HTML tags.

Syntax

Dylib

```c
double DPLGetHTMLTextHeight(int InstanceID, double Width, wchar_t * HTMLText);
```

Objective-C class

```c
-(double)GetHTMLTextHeight:(double)Width :(NSString *)HTMLText;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>The width of the area the text would be drawn into</td>
</tr>
<tr>
<td>HTMLText</td>
<td>The HTML to determine the height of. See Appendix A for details of the supported HTML tags.</td>
</tr>
</tbody>
</table>
GetHTMLTextLineCount

Text, HTML text

Description

Returns the number of lines a block of HTML text will take up if it is drawn using the DrawHTMLText function. See Appendix A for details of the supported HTML tags.

Syntax

Dylib

```c
int DPLGetHTMLTextLineCount(int InstanceID, double Width,
    wchar_t * HTMLText);
```

Objective-C class

```objc
-(int)GetHTMLTextLineCount:(double)Width :(NSString *)HTMLText;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>The width of the area the text would be drawn into</td>
</tr>
<tr>
<td><strong>HTMLText</strong></td>
<td>The HTML text to determine the number of lines of</td>
</tr>
</tbody>
</table>
GetHTMLTextWidth

**Text, HTML text**

**Description**

Returns the actual horizontal size of a block of HTML text when wrapped to a maximum width by the **DrawHTMLText** function. See **Appendix A** for details of the supported HTML tags.

**Syntax**

**Dylib**

```c
double DPLGetHTMLTextWidth(int InstanceID, double MaxWidth, wchar_t * HTMLText);
```

**Objective-C class**

```objc
- (double)GetHTMLTextWidth:(double)MaxWidth :(NSString *)HTMLText;
```

**Parameters**

| **MaxWidth** | The width of the area the text would be drawn into |
| **HTMLText** | The HTML text to determine the width of |
GetImageID

Image handling

Description

Returns the ID of the specified image.

Syntax

Dylib

\[ \text{int DPLGetImageID(int InstanceID, int Index);} \]

Objective-C class

- (int)GetImageID:(int)Index;

Parameters

| Index   | The index of the image. The first image has an index of 1. |

Return values

| 0       | The index is out of bounds |
| Non-zero | The ID of the specified image |
**GetImageListCount**

**Image handling**

**Description**
Returns the number of images in an image list.

**Syntax**

**Dylib**

```c
int DPLGetImageListCount(int InstanceID, int ImageListID);
```

**Objective-C class**

```objc
- (int)GetImageListCount:(int)ImageListID;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>ImageListID</strong></th>
<th>A value returned by the <code>GetPageImageList</code> function</th>
</tr>
</thead>
</table>
GetImageListItemDataToString  

**Image handling**  

**Description**  

Returns the image data of the specified image list item as a string of 8-bit bytes.

**Syntax**

**Dylib**

```c
char * DPLGetImageListItemDataToString(int InstanceID, int ImageListID,
int ImageIndex, int Options);
```

**Objective-C class**

```objc
- (NSData *)GetImageListItemDataToString:(int)ImageListID :(int)ImageIndex
 ::(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageListID</td>
<td>A value returned by the <code>GetPageImageList</code> function</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image in the list. The first image has an index of 1.</td>
</tr>
<tr>
<td>Options</td>
<td>Reserved for future use. Should be set to 0.</td>
</tr>
</tbody>
</table>
**GetImageListItemDblProperty**

**Image handling**

**Description**

Returns a Double type property of the specified image list item.

**Syntax**

**Dylib**

```c
double DPLGetImageListItemDblProperty(int InstanceID, int ImageListID, int ImageIndex, int PropertyID);
```

**Objective-C class**

```objective-c
- (double)GetImageListItemDblProperty:(int)ImageListID :(int)ImageIndex :(int)PropertyID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ImageListID</strong></td>
<td>A value returned by the <strong>GetPageImageList</strong> function</td>
</tr>
<tr>
<td><strong>ImageIndex</strong></td>
<td>The index of the image in the list. The first image has an index of 1.</td>
</tr>
</tbody>
</table>
| **PropertyID** | 501 = Horizontal co-ordinate of top-left corner  
502 = Vertical co-ordinate of top-left corner  
503 = Horizontal co-ordinate of top-right corner  
504 = Vertical co-ordinate of top-right corner  
505 = Horizontal co-ordinate of bottom-right corner  
506 = Vertical co-ordinate of bottom-right corner  
507 = Horizontal co-ordinate of bottom-left corner  
508 = Vertical co-ordinate of bottom-left corner |
GetImageListItemIntProperty

Image handling

Description

Returns an Integer type property of the specified image list item.

Syntax

Dylib

```c
int DPLGetImageListItemIntProperty(int InstanceID, int ImageListID, int ImageIndex, int PropertyID);
```

Objective-C class

```objective-c
- (int)GetImageListItemIntProperty:(int)ImageListID :(int)ImageIndex :(int)PropertyID;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageListID</td>
<td>A value returned by the <code>GetPageImageList</code> function</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image in the list. The first image has an index of 1.</td>
</tr>
<tr>
<td>PropertyID</td>
<td>400 = Image type&lt;br&gt;401 = Width in pixels&lt;br&gt;402 = Height in pixels&lt;br&gt;403 = Bits per pixel&lt;br&gt;404 = Color space type</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JPEG (for image type)&lt;br&gt;DeviceGray (for color space type)</td>
</tr>
<tr>
<td>2</td>
<td>BMP (for image type)&lt;br&gt;DeviceRGB (for color space type)</td>
</tr>
<tr>
<td>3</td>
<td>TIFF (for image type)&lt;br&gt;DeviceCMYK (for color space type)</td>
</tr>
<tr>
<td>4</td>
<td>PNG (for image type)</td>
</tr>
<tr>
<td>-1</td>
<td>Unknown (for color space type)</td>
</tr>
</tbody>
</table>
GetImageMeasureDict

Measurement and coordinate units

Description

Returns the measurement dictionary for the selected image as a MeasureDictID value.

Syntax

**Dylib**

```
int DPLGetImageMeasureDict(int InstanceID);
```

**Objective-C class**

```
-(int)GetImageMeasureDict
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The measure dictionary of the selected image could not be found</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A MeasureDictID value</td>
</tr>
</tbody>
</table>
GetImagePageCount

Image handling, Miscellaneous functions

Description

Returns the number of pages in the specified image file. Most images consist of 1 page, but TIFF images may contain multiple pages.

Syntax

Dylib

```c
int DPLGetImagePageCount(int InstanceID, wchar_t * FileName);
```

Objective-C class

```objc
- (int)GetImagePageCount:(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The name of the image file to analyse.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image file is invalid or does not exist</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of pages in the specified image</td>
</tr>
</tbody>
</table>
GetImagePageCountFromString

Image handling, Miscellaneous functions

Description

Returns the number of pages in the provided image data. Most images consist of 1 page, but TIFF images may contain multiple pages.

Syntax

Dylib

```c
int DPLGetImagePageCountFromString(int InstanceID, char * Source);
```

Objective-C class

```c
- (int)GetImagePageCountFromString:(NSData *)Source;
```

Parameters

| Source     | A string containing the image data. In the ActiveX version of the library this string must contain 16-bit characters, only the lower 8-bits of each character will be used. |

Return values

| 0           | The image data is invalid |
| Non-zero    | The number of pages in the image |
**GetImagePtDataDict**

**Measurement and coordinate units**

**Description**

Returns the PtData dictionary for the selected image as a PtDataDictID value.

**Syntax**

**Dylib**

```
int DPLGetImagePtDataDict(int InstanceID);
```

**Objective-C class**

```
-(int)GetImagePtDataDict
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The PtData dictionary for the selected image could not be found</td>
</tr>
<tr>
<td><strong>Non-zero</strong></td>
<td>A PtDataDictID value</td>
</tr>
</tbody>
</table>
GetInformation

Document properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Get the properties of the selected document.

Syntax

**Dylib**

```c
wchar_t * DPLGetInformation(int InstanceID, int Key);
```

**Objective-C class**

```c
- (NSString *)GetInformation:(int)Key;
```

Parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>The property to get:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PDF Version</td>
</tr>
<tr>
<td>1</td>
<td>Author</td>
</tr>
<tr>
<td>2</td>
<td>Title</td>
</tr>
<tr>
<td>3</td>
<td>Subject</td>
</tr>
<tr>
<td>4</td>
<td>Keywords</td>
</tr>
<tr>
<td>5</td>
<td>Creator</td>
</tr>
<tr>
<td>6</td>
<td>Producer</td>
</tr>
<tr>
<td>7</td>
<td>Creation date</td>
</tr>
<tr>
<td>8</td>
<td>Modification date</td>
</tr>
</tbody>
</table>
GetInstalledFontsByCharset

Fonts

Description

Returns a list of the names of fonts that are installed. These font names can be used with the
AddTrueTypeFont and AddSubsettedFont functions.
The list is filtered by the specified character set. To show all fonts, set CharsetIndex to 2
(corresponding to DEFAULT_CHARSET).

Syntax

Dylib

wchar_t * DPLGetInstalledFontsByCharset(int InstanceID, int CharsetIndex,
int Options);

Objective-C class

- (NSString *)GetInstalledFontsByCharset:(int)CharsetIndex :(int)Options;

Parameters

<table>
<thead>
<tr>
<th>CharsetIndex</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ANSI</td>
</tr>
<tr>
<td>2</td>
<td>Default</td>
</tr>
<tr>
<td>3</td>
<td>Symbol</td>
</tr>
<tr>
<td>4</td>
<td>Shift JIS</td>
</tr>
<tr>
<td>5</td>
<td>Hangeul</td>
</tr>
<tr>
<td>6</td>
<td>GB2312</td>
</tr>
<tr>
<td>7</td>
<td>Chinese Big 5</td>
</tr>
<tr>
<td>8</td>
<td>OEM</td>
</tr>
<tr>
<td>9</td>
<td>Johab</td>
</tr>
<tr>
<td>10</td>
<td>Hebrew</td>
</tr>
<tr>
<td>11</td>
<td>Arabic</td>
</tr>
<tr>
<td>12</td>
<td>Greek</td>
</tr>
<tr>
<td>13</td>
<td>Turkish</td>
</tr>
<tr>
<td>14</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>15</td>
<td>Thai</td>
</tr>
<tr>
<td>16</td>
<td>East Europe</td>
</tr>
<tr>
<td>17</td>
<td>Russian</td>
</tr>
<tr>
<td>18</td>
<td>Mac</td>
</tr>
<tr>
<td>19</td>
<td>Baltic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Font names enclosed in double quotes with comma delimiters</td>
</tr>
<tr>
<td>1</td>
<td>Font names in plain text with CRLF delimiters</td>
</tr>
</tbody>
</table>
GetInstalledFontsByCodePage

Fonts

Description

Returns a list of the names of fonts that are installed. These font names can be used with the AddTrueTypeFont and AddSubsettedFont functions. The list is filtered by the specified code page. To show all fonts, set CodePage to 0 (corresponding to DEFAULT_CHARSET).

Syntax

Dylib

wchar_t * DPLGetInstalledFontsByCodePage(int InstanceID, int CodePage, int Options);

Objective-C class

- (NSString *)GetInstalledFontsByCodePage:(int)CodePage :(int)Options;

Parameters

<table>
<thead>
<tr>
<th>CodePage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DEFAULT_CHARSET</td>
</tr>
<tr>
<td>437</td>
<td>OEM_CHARSET</td>
</tr>
<tr>
<td>850</td>
<td>OEM_CHARSET</td>
</tr>
<tr>
<td>852</td>
<td>OEM_CHARSET</td>
</tr>
<tr>
<td>874</td>
<td>THAI_CHARSET</td>
</tr>
<tr>
<td>932</td>
<td>SHIFTJIS_CHARSET</td>
</tr>
<tr>
<td>936</td>
<td>GB2312_CHARSET</td>
</tr>
<tr>
<td>949</td>
<td>HANGEUL_CHARSET</td>
</tr>
<tr>
<td>950</td>
<td>CHINESEBIG5_CHARSET</td>
</tr>
<tr>
<td>1250</td>
<td>EASTEUROPE_CHARSET</td>
</tr>
<tr>
<td>1251</td>
<td>RUSSIAN_CHARSET</td>
</tr>
<tr>
<td>1252</td>
<td>ANSI_CHARSET</td>
</tr>
<tr>
<td>1253</td>
<td>GREEK_CHARSET</td>
</tr>
<tr>
<td>1254</td>
<td>TURKISH_CHARSET</td>
</tr>
<tr>
<td>1255</td>
<td>HEBREW_CHARSET</td>
</tr>
<tr>
<td>1256</td>
<td>ARABIC_CHARSET</td>
</tr>
<tr>
<td>1257</td>
<td>BALTIC_CHARSET</td>
</tr>
<tr>
<td>1258</td>
<td>VIETNAMESE_CHARSET</td>
</tr>
<tr>
<td>1361</td>
<td>JOHAB_CHARSET</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Font names enclosed in double quotes with comma delimiters</td>
</tr>
<tr>
<td>1</td>
<td>Font names in plain text with CRLF delimiters</td>
</tr>
</tbody>
</table>
GetKerning
Text, Fonts

Description

Returns the amount of kerning for the specified character pair.

Syntax

Dylib

int DPLGetKerning(int InstanceID, wchar_t * CharPair);

Objective-C class

- (int)GetKerning:(NSString *)CharPair;

Parameters

| CharPair        | A two-character string containing the characters making the kerning pair, for example "AW" |

Return values

The amount the space between the kerning pair will be reduced by. This is the same value as shown in graphics programs such as Adobe Illustrator. A value of 1000 is the same as the height of the text.
GetLatestPrinterNames

Description

Similar to the GetPrinterNames function but returns the latest list of printers rather than the cached list that was enumerated when the app started. This function may take some time to execute depending on the number of network printers installed.

Syntax

Dylib

```c
wchar_t * DPLGetLatestPrinterNames(int InstanceID);
```

Objective-C class

```objective-c
- (NSString *)GetLatestPrinterNames
```
GetMaxObjectNumber

Document properties, Miscellaneous functions

Description

Returns the highest object number in the selected document. This is for advanced use.

Syntax

Dylib

```c
int DPLGetMaxObjectNumber(int InstanceID);
```

Objective-C class

```objective-c
- (int)GetMaxObjectNumber
```
GetMeasureDictBoundsCount

Measurement and coordinate units

Description

Returns the number of items in a measurement dictionary Bounds array.

Syntax

Dylib

```c
int DPLGetMeasureDictBoundsCount(int InstanceID, int MeasureDictID);
```

Objective-C class

```c
- (int)GetMeasureDictBoundsCount:(int)MeasureDictID;
```

Parameters

| MeasureDictID | A value returned from the GetImageMeasureDict function |
GetMeasureDictBoundsItem

Measurement and coordinate units

Description

Returns an item from a measurement dictionary Bounds array.

Syntax

Dylib

```c
double DPLGetMeasureDictBoundsItem(int InstanceID, int MeasureDictID,
int ItemIndex);
```

Objective-C class

```objc
- (double)GetMeasureDictBoundsItem:(int)MeasureDictID :(int)ItemIndex;
```

Parameters

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>A value returned from the <code>GetImageMeasureDict</code> function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemIndex</td>
<td>The index of the item to return. The first item has an index of 1.</td>
</tr>
</tbody>
</table>
GetMeasureDictCoordinateSystem

Measurement and coordinate units

Description

Returns the coordinate system type of a measurement dictionary.

Syntax

Dylib

int DPLGetMeasureDictCoordinateSystem(int InstanceID, int MeasureDictID);

Objective-C class

- (int)GetMeasureDictCoordinateSystem:(int)MeasureDictID;

Parameters

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A value returned from the GetImageMeasureDict function</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect</td>
</tr>
<tr>
<td>1</td>
<td>The measurement dictionary is a rectilinear coordinate system (RL)</td>
</tr>
<tr>
<td>2</td>
<td>The measurement dictionary is a geospatial coordinate system (GEO)</td>
</tr>
</tbody>
</table>
GetMeasureDictDCSDict

Measurement and coordinate units

Description

Returns the DCS coordinate system dictionary of a measurement dictionary (used for display purposes) as a CSDictID value.

Syntax

**Dylib**

```c
int DPLGetMeasureDictDCSDict(int InstanceID, int MeasureDictID);
```

**Objective-C class**

```c
- (int)GetMeasureDictDCSDict:(int)MeasureDictID;
```

Parameters

| MeasureDictID | A value returned from the `GetImageMeasureDict` function |

Return values

| 0 | The MeasureDictID parameter was incorrect |
| Non-zero | A CSDictID value |
GetMeasureDictGCSDict

Measurement and coordinate units

Description

Returns the GCS coordinate system dictionary of a measurement dictionary as a CSDictID value.

Syntax

Dylib

```c
int DPLGetMeasureDictGCSDict(int InstanceID, int MeasureDictID);
```

Objective-C class

```objective-c
-(int)GetMeasureDictGCSDict:(int)MeasureDictID;
```

Parameters

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A value returned from the GetImageMeasureDict function</td>
<td></td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A CSDict value</td>
</tr>
</tbody>
</table>
GetMeasureDictGPTSCount

Measurement and coordinate units

Description

Returns the number of items in the GPTS array of a measurement dictionary.

Syntax

Dylib

    int DPLGetMeasureDictGPTSCount(int InstanceID, int MeasureDictID);

Objective-C class

    - (int)GetMeasureDictGPTSCount:(int)MeasureDictID;

Parameters

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>A value returned from the GetImageMeasureDict function</th>
</tr>
</thead>
</table>

**GetMeasureDictGPTSItem**

**Measurement and coordinate units**

**Description**

Returns a value from the GPTS array of a measurement dictionary.

**Syntax**

**Dylib**

```c
double DPLGetMeasureDictGPTSItem(int InstanceID, int MeasureDictID, int ItemIndex);
```

**Objective-C class**

```objc
-(double)GetMeasureDictGPTSItem:(int)MeasureDictID :(int)ItemIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeasureDictID</td>
<td>A value returned from the <a href="#">GetImageMeasureDict</a> function</td>
</tr>
<tr>
<td>ItemIndex</td>
<td>The index of the item. The first item has an index of 1.</td>
</tr>
</tbody>
</table>
GetMeasureDictLPTSCCount
Measurement and coordinate units

Description

Returns the number of items in the LPTS array of a measurement dictionary.

Syntax

Dylib

```c
int DPLGetMeasureDictLPTSCCount(int InstanceID, int MeasureDictID);
```

Objective-C class

```objective-c
- (int)GetMeasureDictLPTSCount:(int)MeasureDictID;
```

Parameters

| MeasureDictID | A value returned from the GetImageMeasureDict function |
GetMeasureDictLPTSItem

Measurement and coordinate units

Description

Returns a value from the CPTS array of a measurement dictionary.

Syntax

**Dylib**

```c
double DPLGetMeasureDictLPTSItem(int InstanceID, int MeasureDictID, int ItemIndex);
```

**Objective-C class**

```c
- (double)GetMeasureDictLPTSItem:(int)MeasureDictID :(int)ItemIndex;
```

Parameters

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>A value returned from the GetImageMeasureDict function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemIndex</td>
<td>The index of the item. The first item has an index of 1.</td>
</tr>
</tbody>
</table>
GetMeasureDictPDU

Measurement and coordinate units

Syntax

**Dylib**

```c
int DPLGetMeasureDictPDU(int InstanceID, int MeasureDictID, int UnitIndex);
```

**Objective-C class**

```objective-c
-(int)GetMeasureDictPDU:(int)MeasureDictID :(int)UnitIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A value returned from the <em>GetImageMeasureDict</em> function</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UnitIndex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Linear display units</td>
</tr>
<tr>
<td>2</td>
<td>Area display units</td>
</tr>
<tr>
<td>3</td>
<td>Angular display units</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect.</td>
</tr>
</tbody>
</table>
| 1     | Linear units: M (a meter)  
|       | Area units: SQM (a square meter)  
|       | Angular units: DEG (a degree)  |
| 2     | Linear units: KM (a kilometer)  
|       | Area units: HA (a hectare)  
|       | Angular units: GRD (a grad = 0.9 degrees)  |
| 3     | Linear units: FT (an international foot)  
|       | Area units: SQKM (a square kilometer)  |
| 4     | Linear units: USFT (a U.S. Survey foot)  
|       | Area units: SQFT (a square foot)  |
| 5     | Linear units: MI (an international mile)  
|       | Area units: A (a acre)  |
| 6     | Linear units: MI (an international nautical mile)  
|       | Area units: SQMI (a square mile)  |
GetNamedDestination

Document properties, Annotations and hotspot links

Description

Locates the named destination with the specified name and returns a DestID that can be used with the GetDestPage, GetDestType and GetDestValue functions.

Syntax

Dylib

```c
int DPLGetNamedDestination(int InstanceID, wchar_t * DestName);
```

Objective-C class

```objective-c
- (int)GetNamedDestination:(NSString *)DestName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestName</td>
<td>The name of the named destination to search for</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified named destination could not be found</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A DestID that can be used with the destination functions</td>
</tr>
</tbody>
</table>
GetNextOutline

Outlines

Description

Returns the ID of the outline that is below the specified outline at the same level.

Syntax

Dylib

int DPLGetNextOutline(int InstanceID, int OutlineID);

Objective-C class

- (int)GetNextOutline:(int)OutlineID;

Parameters

| OutlineID | The ID of the outline item to work with. This ID is returned by the NewOutline or NewStaticOutline functions, or retrieved with the GetOutlineID function or Get*Outline functions. |
GetObjectCount

Miscellaneous functions

Description

Returns the number of raw PDF objects in the document.

Syntax

**Dylib**

```c
int DPLGetObjectCount(int InstanceID);
```

**Objective-C class**

```c
- (int)GetObjectCount
```
**GetObjectDecodeError**

**Miscellaneous functions**

**Description**

This function can be used to determine if an error was encountered during decoding of the raw PDF object from the file.

**Syntax**

**Dylib**

```c
int DPLGetObjectDecodeError(int InstanceID, int ObjectNumber);
```

**Objective-C class**

```objective-c
- (int)GetObjectDecodeError:(int)ObjectNumber;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>ObjectName</strong></th>
<th>The number of the object to retrieve. The first object is numbered 1 and the last object has an object number equal to the result of the GetObjectCount function.</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The object was decoded successfully</td>
</tr>
<tr>
<td>1</td>
<td>The object could not be decoded</td>
</tr>
</tbody>
</table>
GetObjectToString

Miscellaneous functions

Description

Returns the raw PDF object data for the specified object number. This is for advanced use only.

Syntax

Dylib

char * DPLGetObjectToString(int InstanceID, int ObjectNumber);

Objective-C class

- (NSData *)GetObjectToString:(int)ObjectNumber;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectNumber</td>
<td>The number of the object to retrieve. The first object is numbered 1 and the last object has an object number equal to the result of the GetObjectCount function.</td>
</tr>
</tbody>
</table>
GetOpenActionDestination

Document properties

Description

Retrieves the ID of the open action destination, if any. This ID can be used with the GetDestPage, GetDestType and GetDestValue functions to obtain information about the open action destination.

Syntax

Dylib

```c
int DPLGetOpenActionDestination(int InstanceID);
```

Objective-C class

```objective-c
- (int)GetOpenActionDestination
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document does not have an open action destination</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A DestID that can be used with the GetDestPage, GetDestType and GetDestValue functions</td>
</tr>
</tbody>
</table>
GetOpenActionJavaScript

**Document properties, JavaScript**

**Description**

Retrieves the JavaScript linked to the document's open action.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetOpenActionJavaScript(int InstanceID);
```

**Objective-C class**

```objective-c
- (NSString *)GetOpenActionJavaScript
```
GetOptionalContentConfigCount

**Content Streams and Optional Content Groups**

**Description**

Returns the number of optional content configuration dictionaries in the selected document. The first optional content configuration dictionary is used to specify the initial state of the optional content groups when the document is first opened by a PDF viewer. Other configuration dictionaries are used in other circumstances.

The `GetOptionalContentConfigState` function can be used to determine the state of the optional content groups as defined by a particular optional content configuration dictionary.

**Syntax**

**Dylib**

```c
int DPLGetOptionalContentConfigCount(int InstanceID);
```

**Objective-C class**

```c
- (int)GetOptionalContentConfigCount
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document does not have any optional content configuration dictionaries.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The number of optional content configuration dictionaries in the document.</td>
</tr>
</tbody>
</table>
GetOptionalContentConfigLocked
Content Streams and Optional Content Groups

Description
This function is used to determine if an optional content group is locked as defined by the specified optional content configuration dictionary.

Syntax

Dylib

```c
int DPLGetOptionalContentConfigLocked(int InstanceID,
int OptionalContentConfigID, int OptionalContentGroupID);
```

Objective-C class

```c
- (int)GetOptionalContentConfigLocked:(int)OptionalContentConfigID
  :(int)OptionalContentGroupID;
```

Parameters

<table>
<thead>
<tr>
<th>OptionalContentConfigID</th>
<th>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptionalContentGroupID</td>
<td>An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The optional content group is unlocked</td>
</tr>
<tr>
<td>1</td>
<td>The optional content group is locked</td>
</tr>
</tbody>
</table>
GetOptionalContentConfigOrderCount

Content Streams and Optional Content Groups

**Description**

Returns the number of items in the order array of the specified optional content configuration dictionary.

The order array defines a tree structure with labels and optional content group items that can be used in the user interface of the PDF viewer application.

**Syntax**

**Dylib**

```c
int DPLGetOptionalContentConfigOrderCount(int InstanceID,
 int OptionalContentConfigID);
```

**Objective-C class**

```c
- (int)GetOptionalContentConfigOrderCount:(int)OptionalContentConfigID;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>OptionalContentConfigID</strong></th>
<th>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</th>
</tr>
</thead>
</table>
GetOptionalContentConfigOrderItemID
Content Streams and Optional Content Groups

Description
Returns the OptionalContentGroupID for an item in the order array of the specified optional content configuration dictionary.

Syntax
Dylib

```c
int DPLGetOptionalContentConfigOrderItemID(int InstanceID,
int OptionalContentConfigID, int ItemIndex);
```

Objective-C class

```objc
- (int)GetOptionalContentConfigOrderItemID:(int)OptionalContentConfigID
:(int)ItemIndex;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OptionalContentConfigID</strong></td>
<td>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</td>
</tr>
<tr>
<td><strong>ItemIndex</strong></td>
<td>The index number of the item in the order array. The first item has an index number of 1 and the last item has an index equal to the value returned by the <strong>GetOptionalContentConfigOrderCount</strong> function.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified item could not be found or it is a label item and does not have an associated optional content group.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The OptionalContentGroupID of the item</td>
</tr>
</tbody>
</table>
GetOptionalContentConfigOrderItemLabel

Content Streams and Optional Content Groups

Description

Returns the label text for an item in the order array of the specified optional content configuration dictionary.

Syntax

Dylib

wchar_t * DPLGetOptionalContentConfigOrderItemLabel(int InstanceID, int OptionalContentConfigID, int ItemIndex);

Objective-C class

- (NSString *)GetOptionalContentConfigOrderItemLabel :(int)OptionalContentConfigID :(int)ItemIndex;

Parameters

<table>
<thead>
<tr>
<th>OptionalContentConfigID</th>
<th>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemIndex</td>
<td>The index number of the item in the order array. The first item has an index number of 1 and the last item has an index equal to the value returned by the GetOptionalContentConfigOrderCount function.</td>
</tr>
</tbody>
</table>
GetOptionalContentConfigOrderItemLevel

Content Streams and Optional Content Groups

Description

Returns the hierarchical level for an item in the order array of the specified optional content configuration dictionary.

The first item has a level of 1.

Syntax

**Dylib**

```c
int DPLGetOptionalContentConfigOrderItemLevel(int InstanceID,
                                             int OptionalContentConfigID, int ItemIndex);
```

**Objective-C class**

```c
- (int)GetOptionalContentConfigOrderItemLevel:(int)OptionalContentConfigID
                                         :(int)ItemIndex;
```

Parameters

<table>
<thead>
<tr>
<th>OptionalContentConfigID</th>
<th>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemIndex</td>
<td>The index number of the item in the order array. The first item has an index number of 1 and the last item has an index equal to the value returned by the <code>GetOptionalContentConfigOrderCount</code> function.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The specified item could not be found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>The level of the specified item</td>
</tr>
</tbody>
</table>
GetOptionalContentConfigOrderItemType

Content Streams and Optional Content Groups

Description

Returns the item type for an item in the order array of the specified optional content configuration
dictionary.
Items are either optional content groups or text labels.

Syntax

Dylib

```c
int DPLGetOptionalContentConfigOrderItemType(int InstanceID,
                                           int OptionalContentConfigID, int ItemIndex);
```

Objective-C class

```objc
-(int)GetOptionalContentConfigOrderItemType:(int)OptionalContentConfigID
    :(int)ItemIndex;
```

Parameters

<table>
<thead>
<tr>
<th>OptionalContentConfigID</th>
<th>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemIndex</td>
<td>The index number of the item in the order array. The first item has an index number of 1 and the last item has an index equal to the value returned by the <code>GetOptionalContentConfigOrderCount</code> function.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The specified item could not be found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The specified item is an optional content group. The <code>GetOptionalContentConfigOrderItemID</code> function can be used to determine the OptionalContentGroupID.</td>
</tr>
<tr>
<td>2</td>
<td>The specified item is a text label.</td>
</tr>
</tbody>
</table>
**GetOptionalContentConfigState**

**Content Streams and Optional Content Groups**

**Description**

This function is used to determine the state of an optional content group as defined by the specified optional content configuration dictionary.

**Syntax**

**Dylib**

```c
int DPLGetOptionalContentConfigState(int InstanceID,
                                      int OptionalContentConfigID, int OptionalContentGroupID);
```

**Objective-C class**

```c
- (int)GetOptionalContentConfigState:(int)OptionalContentConfigID
  :(int)OptionalContentGroupID;
```

**Parameters**

<table>
<thead>
<tr>
<th>OptionalContentConfigID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OptionalContentGroupID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>An ID returned by the <strong>NewOptionalContentGroup</strong>, <strong>GetOptionalContentGroupID</strong> or <strong>GetOptionalContentConfigOrderItemID</strong> functions</td>
<td></td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The OptionalContentConfigID parameter or the OptionalContentGroupID parameter is not valid.</td>
</tr>
<tr>
<td>1</td>
<td>The state of the optional content group is set to ON when this optional content configuration dictionary is applied.</td>
</tr>
<tr>
<td>2</td>
<td>The state of the optional content group is set to OFF when this optional content configuration dictionary is applied.</td>
</tr>
<tr>
<td>3</td>
<td>The state of the optional content group is not changed when this optional content configuration dictionary is applied.</td>
</tr>
</tbody>
</table>
GetOptionalContentGroupID
Content Streams and Optional Content Groups

Description
Returns the ID of the optional content group with the specified index.

Syntax

Dylib

    int DPLGetOptionalContentGroupID(int InstanceID, int Index);

Objective-C class

    - (int)GetOptionalContentGroupID:(int)Index;

Parameters

| Index | The index of the optional content group. The first group has an index of 1. Use the OptionalContentGroupCount function to determine the number of optional content groups in the document. |

Return values

| 0     | The Index parameter was out of range |
GetOptionalContentGroupName

Content Streams and Optional Content Groups

Description

Returns the name of the specified optional content group.

Syntax

Dylib

wchar_t * DPLGetOptionalContentGroupName(int InstanceID,
int OptionalContentGroupID);

Objective-C class

- (NSString *)GetOptionalContentGroupName:(int)OptionalContentGroupID;

Parameters

<table>
<thead>
<tr>
<th>OptionalContentGroupID</th>
<th>An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions</th>
</tr>
</thead>
</table>
GetOptionalContentGroupPrintable
Content Streams and Optional Content Groups

Description

Returns the printable state of the specified optional content group.

Syntax

**Dylib**

```c
int DPLGetOptionalContentGroupPrintable(int InstanceID,
int OptionalContentGroupID);
```

**Objective-C class**

```c
- (int)GetOptionalContentGroupPrintable:(int)OptionalContentGroupID;
```

Parameters

| OptionalContentGroupID | An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified optional content group is not printable</td>
</tr>
<tr>
<td>1</td>
<td>The specified optional content group is printable</td>
</tr>
</tbody>
</table>
GetOptionalContentGroupVisible

Content Streams and Optional Content Groups

Description

Returns the visible state of the specified optional content group.

Syntax

Dylib

```c
int DPLGetOptionalContentGroupVisible(int InstanceID,
int OptionalContentGroupID);
```

Objective-C class

```objective-c
- (int)GetOptionalContentGroupVisible:(int)OptionalContentGroupID;
```

Parameters

| OptionalContentGroupID | An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions |

Return values

| 0 | The specified optional content group is not visible |
| 1 | The specified optional content group is visible |
GetOrigin

Description

Returns the co-ordinate system origin as set with the SetOrigin function.

Syntax

**Dylib**

```c
int DPLGetOrigin(int InstanceID);
```

**Objective-C class**

```c
- (int)GetOrigin
```
GetOutlineActionID

Annotations and hotspot links, Outlines

Description

This function will return an ActionID if the specified outline has an action dictionary. The ActionID can be used with the GetActionType function and can also be compared to the values returned by GetAnnotActionID to determine if an outline action is shared with an annotation action.

Syntax

**Dylib**

```c
int DPLGetOutlineActionID(int InstanceID, int OutlineID);
```

**Objective-C class**

```c
-(int)GetOutlineActionID:(int)OutlineID;
```

Parameters

<table>
<thead>
<tr>
<th><strong>OutlineID</strong></th>
<th>The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID.</th>
</tr>
</thead>
</table>
GetOutlineColor

Color, Outlines

Description

Returns the color component of the outline as a value between 0 and 1.

Syntax

**Dylib**

```c
double DPLGetOutlineColor(int InstanceID, int OutlineID,
                          int ColorComponent);
```

**Objective-C class**

```objc
- (double)GetOutlineColor:(int)OutlineID :(int)ColorComponent;
```

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>The ID of the outline as returned by the <strong>NewOutline</strong> function. Alternatively, use the <strong>GetOutlineID</strong> function to get a valid outline ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorComponent</td>
<td>The component of the color:</td>
</tr>
<tr>
<td></td>
<td>0 = Red</td>
</tr>
<tr>
<td></td>
<td>1 = Green</td>
</tr>
<tr>
<td></td>
<td>2 = Blue</td>
</tr>
</tbody>
</table>
GetOutlineDest

Outlines

Description

Retrieves information about the destination the specified outline links to.

Syntax

**Dylib**

```c
int DPLGetOutlineDest(int InstanceID, int OutlineID);
```

**Objective-C class**

```objective-c
-(int)GetOutlineDest:(int)OutlineID;
```

Parameters

**OutlineID**  The ID of the outline as returned by the `NewOutline` function. Alternatively, use the `GetOutlineID` function to get a valid outline ID.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The outline does not have a valid destination or the outline could not be found</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A destination ID (or DestID) that can be used with the <code>GetDestPage</code>, <code>GetDestType</code> and <code>GetDestValue</code> functions</td>
</tr>
</tbody>
</table>
GetOutlineID

Outlines

Description

Returns the Outline ID of the outline item (bookmark) with the specified index. The first outline item has an index of 1.

Syntax

Dylib

```c
int DPLGetOutlineID(int InstanceID, int Index);
```

Objective-C class

```objective-c
-(int)GetOutlineID:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the outline item to retrieve the ID of. The first outline item has an index of 1.</th>
</tr>
</thead>
</table>

**GetOutlineJavaScript**

**JavaScript, Outlines**

**Description**

Returns the JavaScript associated with the outline, if any.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetOutlineJavaScript(int InstanceID, int OutlineID);
```

**Objective-C class**

```c
- (NSString *)GetOutlineJavaScript:(int)OutlineID;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>OutlineID</strong></th>
<th>The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID.</th>
</tr>
</thead>
</table>
**GetOutlineObjectNumber**

**Outlines**

**Description**

Returns the PDF object number of the specified outline item.
This function is for advanced use only.

**Syntax**

**Dylib**

```c
int DPLGetOutlineObjectNumber(int InstanceID, int OutlineID);
```

**Objective-C class**

```objc
-(int)GetOutlineObjectNumber:(int)OutlineID;
```

**Parameters**

| OutlineID | The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID. |
**GetOutlineOpenFile**

**Outlines**

**Description**

Returns the file name that the outline links to, if any.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetOutlineOpenFile(int InstanceID, int OutlineID);
```

**Objective-C class**

```objective-c
- (NSString *)GetOutlineOpenFile:(int)OutlineID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OutlineID</strong></td>
<td>The ID of the outline as returned by the <strong>NewOutline</strong> function. Alternatively, use the <strong>GetOutlineID</strong> function to get a valid outline ID.</td>
</tr>
</tbody>
</table>
GetOutlinePage

Outlines

Description

Returns the page number that the outline links to.

Syntax

**Dylib**

```c
int DPLGetOutlinePage(int InstanceID, int OutlineID);
```

**Objective-C class**

```objc
- (int)GetOutlinePage:(int)OutlineID;
```

Parameters

| **OutlineID** | The ID of the outline as returned by the **NewOutline** function. Alternatively, use the **GetOutlineID** function to get a valid outline ID. |
GetOutlineStyle

Outlines

Description

Returns the style of the outline.

Syntax

Dylib

    int DPLGetOutlineStyle(int InstanceID, int OutlineID);

Objective-C class

    - (int)GetOutlineStyle:(int)OutlineID;

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The ID of the outline as returned by the <strong>NewOutline</strong> function. Alternatively, use the <strong>GetOutlineID</strong> function to get a valid outline ID.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th></th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>1</td>
<td>Italic</td>
</tr>
<tr>
<td>2</td>
<td>Bold</td>
</tr>
<tr>
<td>3</td>
<td>Bold Italic</td>
</tr>
</tbody>
</table>
GetOutlineWebLink

Outlines

Description

Returns the web link (internet URL) that the outline links to, if any.

Syntax

Dylib

wchar_t * DPLGetOutlineWebLink(int InstanceID, int OutlineID);

Objective-C class

- (NSString *)GetOutlineWebLink:(int)OutlineID;

Parameters

| OutlineID | The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID. |
**GetPageBox**

**Page properties**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Returns the dimensions of the selected page's boundary rectangles.

The MediaBox represents the physical medium of the page.

The CropBox represents the visible region of the page, the contents will be clipped to this region.

The BleedBox is similar to the CropBox, but is the rectangle used in a production environment.

The TrimBox indicates the intended dimensions of the finished page after trimming, and the ArtBox defines the extent of the page's meaningful content as intended by the page's creator.

If the document does not have a CropBox but it does have a MediaBox then the CropBox will be the same as the MediaBox. If the document does not have any of the other boxes this function will return the values from the CropBox.

**Syntax**

**Dylib**

```c
double DPLGetPageBox(int InstanceID, int BoxType, int Dimension);
```

**Objective-C class**

```c
- (double)GetPageBox:(int)BoxType :(int)Dimension;
```

**Parameters**

<table>
<thead>
<tr>
<th>BoxType</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MediaBox</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CropBox</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>BleedBox</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TrimBox</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ArtBox</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Left</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Top</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Width</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Height</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Right</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Bottom</td>
<td></td>
</tr>
</tbody>
</table>
GetPageColorSpaces

Color, Page properties

Description

Returns a CSV string containing the list of color spaces defined in the resource tree of the selected page.

Syntax

**Dylib**

```
wchar_t * DPLGetPageColorSpaces(int InstanceID, int Options);
```

**Objective-C class**

```
- (NSString *)GetPageColorSpaces:(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options</strong></td>
<td>This parameter should be set to 0.</td>
</tr>
</tbody>
</table>
GetPageContentToString

Page properties, Page manipulation

Description

This function returns the PDF page description commands which make up the content of the selected page. This is for advanced use only, and will probably be meaningless unless you have an understanding of the Adobe PDF specification.

Previous versions of Quick PDF Library only returned the content of the selected content stream part.

From version 8.11 this function returns the content of the entire page and the GetContentStreamToString function can be used to retrieve the PDF page description commands of the content stream part selected with the SelectContentStream function.

Syntax

Dylib

    char * DPLGetPageContentToString(int InstanceID);

Objective-C class

    -(NSData *)getPageContentToString

GetPageImageList  
Image handling, Page properties

Description
This function finds all the images on the selected page and returns an ImageListID that can be used with the GetImageListCount, GetImageListItemIntProperty, GetImageListItemDblProperty, GetImageListItemDataToString, GetImageListItemDataToVariant and SaveImageListItemDataToFile functions.

Syntax

**Dylib**

```c
int DPLGetPageImageList(int InstanceID, int Options);
```

**Objective-C class**

```objective-c
- (int)GetPageImageList:(int)Options;
```

Parameters

| Options | Reserved for future use, should be set to 0. |

Return values

| 0 | The images on the page could not be enumerated. |
| Non-zero | An ImageListID value |
GetPageJavaScript

Color, JavaScript, Page properties

Description

Retrieves the JavaScript linked to the specified page event.

Syntax

Dylib

wchar_t * DPLGetPageJavaScript(int InstanceID, wchar_t * ActionType);

Objective-C class

- (NSString *)GetPageJavaScript:(NSString *)ActionType;

Parameters

<table>
<thead>
<tr>
<th>ActionType</th>
<th>Retrieves the JavaScript linked to this action:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;O&quot; = (capital letter O) This event occurs when the page is opened</td>
<td></td>
</tr>
<tr>
<td>&quot;C&quot; = This event occurs when the page is closed</td>
<td></td>
</tr>
</tbody>
</table>
GetPageLGIDictContent

Page properties, Measurement and coordinate units

Description

Returns the content of the specified LGIDict dictionary on the selected page.

Syntax

Dylib

wchar_t * DPLGetPageLGIDictContent(int InstanceID, int DictIndex);

Objective-C class

- (NSString *)getPageLGIDictContent:(int)DictIndex;

Parameters

| DictIndex | The index of the dictionary. The first dictionary has an index of 1. Use the LGIDictCount function to determine the total number of LGIDict dictionaries attached to the selected page. |
getPageLGIDictCount

Page properties, Measurement and coordinate units

Description

Returns the number of LGIDict dictionaries attached to the selected page.

Syntax

Dylib

int DPLGetPageLGIDictCount(int InstanceID);

Objective-C class

- (int)getPageLGIDictCount
**GetPageLabel**

**Page properties**

**Description**

Returns the page label for the specified page.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetPageLabel(int InstanceID, int Page);
```

**Objective-C class**

```objc
- (NSString *)GetPageLabel:(int)Page;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>The number of the page to retrieve the page number of</td>
</tr>
</tbody>
</table>
GetPageLayout

Document properties

Description

    Returns the initial page layout of the selected document.

Syntax

    Dylib

    int DPLGetPageLayout(int InstanceID);

    Objective-C class

    - (int)GetPageLayout

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Single page</td>
</tr>
<tr>
<td>1</td>
<td>One column</td>
</tr>
<tr>
<td>2</td>
<td>Two columns, odd-numbered pages on the left</td>
</tr>
<tr>
<td>3</td>
<td>Two columns, odd-numbered pages on the right</td>
</tr>
<tr>
<td>4</td>
<td>Two pages, odd-numbered pages on the left</td>
</tr>
<tr>
<td>5</td>
<td>Two pages, odd-numbered pages on the right</td>
</tr>
<tr>
<td>6</td>
<td>No preference set in document</td>
</tr>
</tbody>
</table>
GetPageMetricsToString

Description

Returns the dimensions (MediaBox and CropBox) and rotation of the specified page range in the document.

Syntax

Dylib

char * DPLGetPageMetricsToString(int InstanceID, int StartPage, int EndPage, int Options);

Objective-C class

- (NSData *)getPageMetricsToString:(int)StartPage :(int)EndPage :(int)Options;

Parameters

<table>
<thead>
<tr>
<th>StartPage</th>
<th>The first page in the range</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndPage</td>
<td>The last page in the range</td>
</tr>
<tr>
<td>Options</td>
<td>1 = Binary output</td>
</tr>
<tr>
<td></td>
<td>Nine double values per page are streamed in a continuous array. For each page there are the elements of the MediaBox, the elements of the CropBox and the value of the Rotation entry.</td>
</tr>
<tr>
<td></td>
<td>2 = Text output</td>
</tr>
<tr>
<td></td>
<td>The values are displayed in text format, separated by tab (char 9) characters and with CRLF after each page.</td>
</tr>
</tbody>
</table>
GetPageMode
Document properties

Description
Returns the initial page mode of the document.

Syntax
Dylib

int DPLGetPageMode(int InstanceID);

Objective-C class
- (int)GetPageMode

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal view</td>
</tr>
<tr>
<td>1</td>
<td>Show the outlines pane</td>
</tr>
<tr>
<td>2</td>
<td>Show the thumbnails pane</td>
</tr>
<tr>
<td>3</td>
<td>Show the document in full screen mode</td>
</tr>
<tr>
<td>4</td>
<td>Optional content group panel visible</td>
</tr>
<tr>
<td>5</td>
<td>Attachments panel visible</td>
</tr>
</tbody>
</table>
GetPageText

Description

This function provides two different methods for extracting text from the selected page, and presents the results in a variety of formats.

The SetTextExtractionWordGap, SetTextExtractionOptions and SetTextExtractionArea functions can be used to adjust the text extraction process.

Syntax

Dylib

wchar_t * DPLGetPageText(int InstanceID, int ExtractOptions);

Objective-C class

- (NSString *)GetPageText:(int)ExtractOptions;

Parameters

ExtractOptions Using the standard text extraction algorithm:
0 = Extract text in human readable format
1 = Deprecated
2 = Return a CSV string including font, color, size and position of each piece of text on the page

Using the more accurate but slower text extraction algorithm:
3 = Return a CSV string for each piece of text on the page with the following format:
Font Name, Text Color, Text Size, X1, Y1, X2, Y2, X3, Y3, X4, Y4, Text
The co-ordinates are the four points bounding the text, measured using the units set with the SetMeasurementUnits function and the origin set with the SetOrigin function. Co-ordinate order is anti-clockwise with the bottom left corner first.
4 = Similar to option 3, but individual words are returned, making searching for words easier
5 = Similar to option 3 but character widths are output after each block of text
6 = Similar to option 4 but character widths are output after each line of text
7 = Extract text in human readable format with improved accuracy compared to option 0
8 = Similar output format as option 0 but using the more accurate algorithm. Returns unformatted lines.

Return values

The text of the selected page, or an empty string if a problem occurred. Lines are separated with CR-LF characters.
GetPageViewportCount
Page properties, Measurement and coordinate units

Description

Returns the number of viewports defined for the selected page.
The GetPageViewportID function can be used to obtain a ViewPortID that can be used with the GetViewportName and GetViewportMeasureDict functions.

Syntax

Dylib

```c
int DPLGetPageViewportCount(int InstanceID);
```

Objective-C class

```objc
-(int)GetPageViewportCount
```
**GetPageViewPortID**

**Page properties, Measurement and coordinate units**

**Description**

Returns a ViewPortID value for the specified viewport of the selected page. This value can be used with the `GetViewportName` and `GetViewportMeasureDict` functions. Use the `GetPageViewportCount` function to determine the number of viewports on the page.

**Syntax**

**Dylib**

```c
int DPLGetPageViewPortID(int InstanceID, int Index);
```

**Objective-C class**

```c
- (int)GetPageViewPortID:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the viewport. The first viewport on the page has an index value of 1.</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The view port at the specified index could not be found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>A ViewPortID value</td>
</tr>
</tbody>
</table>
GetParentOutline

Outlines

Description

Returns the ID of the outline that is the parent of the specified outline.

Syntax

Dylib

```c
int DPLGetParentOutline(int InstanceID, int OutlineID);
```

Objective-C class

```objective-c
- (int)GetParentOutline:(int)OutlineID;
```

Parameters

| OutlineID | The ID of the outline item to work with. This ID is returned by the `NewOutline` or `NewStaticOutline` functions, or retrieved with the `GetOutlineID` function or `Get*Outline` functions. |
GetPrevOutline

Outlines

Description

Returns the ID of the outline that is above the specified outline at the same level.

Syntax

Dylib

```c
int DPLGetPrevOutline(int InstanceID, int OutlineID);
```

Objective-C class

```c
- (int)GetPrevOutline:(int)OutlineID;
```

Parameters

| OutlineID | The ID of the outline item to work with. This ID is returned by the NewOutline or NewStaticOutline functions, or retrieved with the GetOutlineID function or Get*Outline functions. |
GetPrintPreviewBitmapToString

Rendering and printing

Description

Returns a binary string containing a BMP image representing a preview of how printing will look.

Syntax

Dylib

```c
char * DPLGetPrintPreviewBitmapToString(int InstanceID,
wchar_t * PrinterName, int PreviewPage, int PrintOptions,
int MaxDimension, int PreviewOptions);
```

Objective-C class

```objective-c
- (NSData *)GetPrintPreviewBitmapToString:(NSString *)PrinterName
    :(int)PreviewPage :(int)PrintOptions :(int)MaxDimension
    :(int)PreviewOptions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrinterName</td>
<td>The name of the printer to use for printing. This is the name that appears in the Windows Print Manager. Use the <code>GetPrinterNames</code> function to return a list of valid printers on the system. A value returned by the <code>NewCustomPrinter</code> function can also be used here.</td>
</tr>
<tr>
<td>PreviewPage</td>
<td>The page number to preview</td>
</tr>
<tr>
<td>PrintOptions</td>
<td>Use the <code>PrintOptions</code> function to obtain a value for this parameter</td>
</tr>
<tr>
<td>MaxDimension</td>
<td>The maximum width or height of the preview bitmap</td>
</tr>
<tr>
<td>PreviewOptions</td>
<td>Reserved for future use, should be set to zero.</td>
</tr>
</tbody>
</table>
GetPrinterBins

Rendering and printing

Description

This function returns a string containing the bin numbers and names for all the bins (paper trays) available for the specified printer. The string returned contains a line of text for each bin, the lines of text are separated with CR/LF characters. Each line contains a numeric bin number, a comma, and the name of the bin, in double quotes. The bin numbers can be used with the SetupCustomPrinter function.

Syntax

Dylib

wchar_t * DPLGetPrinterBins(int InstanceID, wchar_t * PrinterName);

Objective-C class

- (NSString *)GetPrinterBins:(NSString *)PrinterName;

Parameters

| PrinterName | The name of the printer to query. This is the name that appears in the Windows Print Manager. Use the GetPrinterNames function to return a list of valid printers on the system. A value returned by the NewCustomPrinter function can also be used here. |
GetPrinterDevModeToString

Rendering and printing

Description

Returns a binary string containing the DEVMODE structure for the specified printer. Use the SetPrinterDevModeFromString function to apply this DEVMODE structure during the printing process.

Syntax

Dylib

char * DPLGetPrinterDevModeToString(int InstanceID, wchar_t * PrinterName);

Objective-C class

-(NSData *)GetPrinterDevModeToString:(NSString *)PrinterName;

Parameters

| PrinterName | The name of the printer to use for printing. This is the name that appears in the Windows Print Manager. Use the GetPrinterNames function to return a list of valid printers on the system. A value returned by the NewCustomPrinter function can also be used here. |
GetPrinterMediaTypes

Rendering and printing

Description

This function returns a string containing the media type numbers and names for all the media types available for the specified printer. The string returned contains a line of text for each media type, the lines of text are separated with CR/LF characters. Each line contains a numeric media type number, a comma, and the name of the media type, in double quotes.

Syntax

Dylib

```c
wchar_t * DPLGetPrinterMediaTypes(int InstanceID, wchar_t * PrinterName);
```

Objective-C class

```objective-c
- (NSString *)GetPrinterMediaTypes:(NSString *)PrinterName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrinterName</td>
<td>The name of the printer to query. This is the name that appears in the Windows Print Manager. Use the <code>GetPrinterNames</code> function to return a list of valid printers on the system. A value returned by the <code>NewCustomPrinter</code> function can also be used here.</td>
</tr>
</tbody>
</table>
GetPrinterNames

Rendering and printing

Description

Returns a CSV string containing the names of all the available printers on the system. The result is the cached list that was enumerated when the app was started. The new GetLatestPrinterNames function returns the latest list of printers.

Syntax

**Dylib**

```c
wchar_t * DPLGetPrinterNames(int InstanceID);
```

**Objective-C class**

```objc
- (NSString *)GetPrinterNames
```
GetRenderScale

Rendering and printing

Description

Returns the render scale as set by the SetRenderScale function.

Syntax

Dylib

```c
double DPLGetRenderScale(int InstanceID);
```

Objective-C class

```c
- (double)GetRenderScale
```
GetSignProcessByteRange

Description

Returns an element of the byte range array of a pass-thru digital signature. The values should be handled as 32-bit unsigned integers with two values combined to form a 64-bit file position.

Syntax

Dylib

```c
int DPLGetSignProcessByteRange(int InstanceID, int SignProcessID, int ArrayPosition);
```

Objective-C class

```objective-c
- (int)GetSignProcessByteRange:(int)SignProcessID :(int)ArrayPosition;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SignProcessID</strong></td>
<td>A value returned by the <code>NewSignProcessFromFile</code>, <code>NewSignProcessFromStream</code> or <code>NewSignProcessFromString</code> functions.</td>
</tr>
</tbody>
</table>
| **ArrayPosition** | 1 = ByteArray[0] low 32-bits  
2 = ByteArray[1] low 32-bits  
3 = ByteArray[2] low 32-bits  
5 = ByteArray[0] high 32-bits  
6 = ByteArray[1] high 32-bits  
7 = ByteArray[2] high 32-bits  
8 = ByteArray[3] high 32-bits |
GetSignProcessResult

Security and Signatures

Description

Returns the signing result of a digital signature process.

Syntax

Dylib

```c
int DPLGetSignProcessResult(int InstanceID, int SignProcessID);
```

Objective-C class

```c
- (int)GetSignProcessResult:(int)SignProcessID;
```

Parameters

|---------------|---------------------------------------------------------------------------------------------------------------|

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The file was signed successfully</td>
</tr>
<tr>
<td>2</td>
<td>Input PDF not found</td>
</tr>
<tr>
<td>3</td>
<td>Input PDF cannot be read</td>
</tr>
<tr>
<td>4</td>
<td>Input PDF password incorrect</td>
</tr>
<tr>
<td>5</td>
<td>Certificate file not found</td>
</tr>
<tr>
<td>6</td>
<td>Certificate file is invalid</td>
</tr>
<tr>
<td>7</td>
<td>Incorrect certificate password</td>
</tr>
<tr>
<td>8</td>
<td>Unknown certificate format</td>
</tr>
<tr>
<td>9</td>
<td>No private key found in certificate file</td>
</tr>
<tr>
<td>10</td>
<td>Could not write output file</td>
</tr>
<tr>
<td>11</td>
<td>Could not apply signature</td>
</tr>
<tr>
<td>12</td>
<td>The signature field name was blank</td>
</tr>
</tbody>
</table>
GetStringListCount

Miscellaneous functions

Description

Returns the number of strings in the specified string list.

Syntax

Dylib

int DPLGetStringListCount(int InstanceID, int StringListID);

Objective-C class

- (int)GetStringListCount:(int)StringListID;

Parameters

| StringListID | The ID of the string list as returned by the CheckFileCompliance function. |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>There are no strings in the specified string list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>The number of strings in the list.</td>
</tr>
</tbody>
</table>
GetStringListItem

Miscellaneous functions

Description

Returns an item from the specified string list.

Syntax

**Dylib**

```c
wchar_t * DPLGetStringListItem(int InstanceID, int StringListID,
                                 int ItemIndex);
```

**Objective-C class**

```Objective-C
- (NSString *)GetStringListItem:(int)StringListID :(int)ItemIndex;
```

Parameters

- **StringListID**  The ID of the string list as returned by the `CheckFileCompliance` function.
- **ItemIndex**  The index of the item to return. The first item in the list has an index value of 1. The last item in the list has an index value equal to the return value of the `GetStringListCount` function.
GetTabOrderMode
Form fields, Annotations and hotspot links

Description
This function returns the current tabbing order for all the annotations and formfields on the currently selected page.
If you use SetFormFieldTabOrder then you should set the tabbing order to 'S'tructure mode for the required pages.

Syntax
Dylib

wchar_t * DPLGetTabOrderMode(int InstanceID);

Objective-C class

- (NSString *)GetTabOrderMode

Return values

<table>
<thead>
<tr>
<th>'S'</th>
<th>Structure mode (The order the Annots are defined)</th>
</tr>
</thead>
<tbody>
<tr>
<td>'R'</td>
<td>Row mode (Left to right, top to bottom order)</td>
</tr>
<tr>
<td>'C'</td>
<td>Column mode (Top to bottom, left to right order)</td>
</tr>
<tr>
<td>''</td>
<td>No tabbing order has been defined</td>
</tr>
</tbody>
</table>
GetTableCellDblProperty

Page layout

Description

Returns a numeric property of the specified table cell.

Syntax

Dylib

```c
double DPLGetTableCellDblProperty(int InstanceID, int TableID,
    int RowNumber, int ColumnNumber, int Tag);
```

Objective-C class

```objective-c
- (double)GetTableCellDblProperty:(int)TableID :(int)RowNumber
    :(int)ColumnNumber :(int)Tag;
```

Parameters

<table>
<thead>
<tr>
<th>TableID</th>
<th>A TableID returned by the CreateTable function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowNumber</td>
<td>The the row number of the cell. Top row is row number 1.</td>
</tr>
<tr>
<td>ColumnNumber</td>
<td>The the column number of the cell. Left most column is column number 1.</td>
</tr>
<tr>
<td>Tag</td>
<td>101 to 104 = Left, top, width and height of cell</td>
</tr>
<tr>
<td></td>
<td>105 = Text size</td>
</tr>
<tr>
<td></td>
<td>106 = Red or cyan component of the background color</td>
</tr>
<tr>
<td></td>
<td>107 = Green or magenta component of the background color</td>
</tr>
<tr>
<td></td>
<td>108 = Blue or yellow component of the background color</td>
</tr>
<tr>
<td></td>
<td>109 = Black component of the background color</td>
</tr>
<tr>
<td></td>
<td>110 = Red or cyan component of the text color</td>
</tr>
<tr>
<td></td>
<td>111 = Green or magenta component of the text color</td>
</tr>
<tr>
<td></td>
<td>112 = Blue or yellow component of the text color</td>
</tr>
<tr>
<td></td>
<td>113 = Black component of the text color</td>
</tr>
<tr>
<td></td>
<td>114 to 117 = Red or cyan component of the left, top, right and bottom border</td>
</tr>
<tr>
<td></td>
<td>118 to 121 = Green or magenta component of the left, top, right and bottom border</td>
</tr>
<tr>
<td></td>
<td>122 to 125 = Blue or yellow component of the left, top, right and bottom border</td>
</tr>
<tr>
<td></td>
<td>126 to 129 = Black component of the left, top, right and bottom border</td>
</tr>
<tr>
<td></td>
<td>130 to 133 = Padding of the edge next to the left, top, right and bottom border</td>
</tr>
<tr>
<td></td>
<td>134 to 137 = Width of the left, top, right and bottom border</td>
</tr>
</tbody>
</table>
GetTableCellIntProperty

Page layout

Description

Returns an integer property of the specified table cell.

Syntax

Dylib

```c
int DPLGetTableCellIntProperty(int InstanceID, int TableID, int RowNumber,
int ColumnNumber, int Tag);
```

Objective-C class

```objective-c
- (int)GetTableCellIntProperty:(int)TableID :(int)RowNumber
 :(int)ColumnNumber :(int)Tag;
```

Parameters

<table>
<thead>
<tr>
<th>TableID</th>
<th>A TableID returned by the CreateTable function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowNumber</td>
<td>The the row number of the cell. Top row is row number 1.</td>
</tr>
<tr>
<td>ColumnNumber</td>
<td>The the column number of the cell. Left most column is column number 1.</td>
</tr>
</tbody>
</table>
| Tag | 201 = Cell alignment (see the SetTableCellAlignment function)  
202 = Merged cell row span  
203 = Merged cell column span  
204 = Number of color components in the background color (3 for RGB, 4 for CMYK)  
205 = Number of color components in the text color (3 for RGB, 4 for CMYK)  
206 to 209 = Number of color components in the left, top, right and bottom border color (3 for RGB, 4 for CMYK) |
GetTableCellStrProperty

Description

Returns a string property of the specified table cell.

Syntax

Dylib

```c
wchar_t * DPLGetTableCellStrProperty(int InstanceID, int TableID,
        int RowNumber, int ColumnNumber, int Tag);
```

Objective-C class

```objc
- (NSString *)GetTableCellStrProperty:(int)TableID :(int)RowNumber
        :(int)ColumnNumber :(int)Tag;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the CreateTable function</td>
</tr>
<tr>
<td>RowNumber</td>
<td>The the row number of the cell. Top row is row number 1.</td>
</tr>
<tr>
<td>ColumnNumber</td>
<td>The the column number of the cell. Left most column is column number 1.</td>
</tr>
<tr>
<td>Tag</td>
<td>301 = Cell contents</td>
</tr>
</tbody>
</table>
GetTableColumnCount

Page layout

Description

Returns the number of columns in the specified table.

Syntax

Dylib

```c
int DPLGetTableColumnCount(int InstanceID, int TableID);
```

Objective-C class

```objective-c
- (int)GetTableColumnCount:(int)TableID;
```

Parameters

| TableID | A TableID returned by the CreateTable function |
GetTableLastDrawnRow

Description

Returns the row number of the last row that was drawn onto the page by the DrawTableRows function.

Syntax

Dylib

    int DPLGetTableLastDrawnRow(int InstanceID, int TableID);

Objective-C class

    -(int)GetTableLastDrawnRow:(int)TableID;

Parameters

TableID  A TableID returned by the CreateTable function

Return values

0  No rows from the specified table have been drawn

Non-zero  The row number of the last drawn row. The top row is row number 1.
GetTableRowCount

Description

Returns the number of rows in the specified table.

Syntax

**Dylib**

```c
int DPLGetTableRowCount(int InstanceID, int TableID);
```

**Objective-C class**

```objc
-(int)GetTableRowCount:(int)TableID;
```

Parameters

| TableID | A TableID returned by the **CreateTable** function |
GetTempPath

**Miscellaneous functions**

**Description**

Retrieves the current setting for the folder that will be used to store temporary files generated by functions such as **MergeFileList**.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetTempPath(int InstanceID);
```

**Objective-C class**

```objective-c
- (NSString *)GetTempPath
```
**GetTextAscent**

**Text, Fonts, Page layout**

**Description**

Returns the size of the selected font, measured from the baseline to the top of capital letters (without any accents).

**Syntax**

**Dylib**

```c
double DPLGetTextAscent(int InstanceID);
```

**Objective-C class**

```c
- (double)GetTextAscent
```

**Return values**

The ascent of the selected font.
**GetTextBlockBound**

**Text, Fonts, Extraction**

**Description**

Returns one of the bounds of the specified text block.

**Syntax**

**Dylib**

```c
double DPLGetTextBlockBound(int InstanceID, int TextBlockListID, int Index, int BoundIndex);
```

**Objective-C class**

```objc
- (double)GetTextBlockBound:(int)TextBlockListID :(int)Index :(int)BoundIndex;
```

**Parameters**

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the ExtractPageTextBlocks function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
| BoundIndex      | 1 = Bottom left horizontal coordinate  
                  2 = Bottom left vertical coordinate  
                  3 = Top left horizontal coordinate  
                  4 = Top left vertical coordinate  
                  5 = Top right horizontal coordinate  
                  6 = Top right vertical coordinate  
                  7 = Bottom right horizontal coordinate  
                  8 = Bottom right vertical coordinate |
GetTextBlockCharWidth

Text, Fonts, Extraction

Description

Returns the width of a particular character within the specified text block.

Syntax

Dylib

```c
double DPLGetTextBlockCharWidth(int InstanceID, int TextBlockListID,
                                int Index, int CharIndex);
```

Objective-C class

```objective-c
-(double)GetTextBlockCharWidth:(int)TextBlockListID :(int)Index
               :(int)CharIndex;
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TextBlockListID</strong></td>
<td>A value returned by the <a href="#">ExtractPageTextBlocks</a> function</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
<tr>
<td><strong>CharIndex</strong></td>
<td>The index of the character to retrieve the width of. The first character has an index of 1.</td>
</tr>
</tbody>
</table>
GetTextBlockColor

Text, Extraction, Color

Description

Returns one component of the color of the text in the specified text block. The color component value is returned as a value between 0 and 1.

Syntax

Dylib

double DPLGetTextBlockColor(int InstanceID, int TextBlockListID, int Index, int ColorComponent);

Objective-C class

-(double)GetTextBlockColor:(int)TextBlockListID :(int)Index :(int)ColorComponent;

Parameters

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the ExtractPageTextBlocks function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
<tr>
<td>ColorComponent</td>
<td>For RGB: 1 = Red, 2 = Green, 3 = Blue For CMYK: 1 = Cyan, 2 = Magenta, 3 = Yellow, 4 = Black</td>
</tr>
</tbody>
</table>
**GetTextBlockColorType**

**Text, Extraction, Color**

**Description**

Returns the type of color of the text in the specified text block.

**Syntax**

**Dylib**

```c
int DPLGetTextBlockColorType(int InstanceID, int TextBlockListID, int Index);
```

**Objective-C class**

```objc
- (int)GetTextBlockColorType:(int)TextBlockListID :(int)Index;
```

**Parameters**

- **TextBlockListID** A value returned by the `ExtractPageTextBlocks` function
- **Index** The index of the text block. The first text block in the list has an index of 1.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>RGB</td>
</tr>
<tr>
<td>4</td>
<td>CMYK</td>
</tr>
</tbody>
</table>
**GetTextBlockCount**

**Text, Extraction**

**Description**

Returns the number of text blocks in the specified text block list.

**Syntax**

**Dylib**

```c
int DPLGetTextBlockCount(int InstanceID, int TextBlockListID);
```

**Objective-C class**

```objective-c
- (int)GetTextBlockCount:(int)TextBlockListID;
```

**Parameters**

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the ExtractPageTextBlocks function</th>
</tr>
</thead>
</table>

GetTextBlockFontName
Text, Fonts, Extraction

Description

Returns the font name of the text in the specified text block.

Syntax

Dylib

```c
wchar_t * DPLGetTextBlockFontName(int InstanceID, int TextBlockListID,
    int Index);
```

Objective-C class

```c
- (NSString *)GetTextBlockFontName:(int)TextBlockListID :(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the <code>ExtractPageTextBlocks</code> function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
GetTextBlockFontSize
Text, Fonts, Extraction

Description

Returns the font size of the text in the specified text block.

Syntax

Dylib

```c
double DPLGetTextBlockFontSize(int InstanceID, int TextBlockListID, int Index);
```

Objective-C class

```objc
- (double)GetTextBlockFontSize:(int)TextBlockListID :(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the <code>ExtractPageTextBlocks</code> function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
GetTextBlockText

Text, Extraction

Description

Returns the text in the specified text block.

Syntax

**Dylib**

```c
wchar_t * DPLGetTextBlockText(int InstanceID, int TextBlockListID, int Index);
```

**Objective-C class**

```c
-(NSString *)GetTextBlockText:(int)TextBlockListID :(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the <strong>ExtractPageTextBlocks</strong> function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the text block. The first text block in the list has an index of 1.</td>
</tr>
</tbody>
</table>
Description

Returns the bounding box of the font. This is the largest rectangle which can enclose every character of the font. The top and bottom are measured from the baseline of the font.

Syntax

**Dylib**

```c
double DPLGetTextBound(int InstanceID, int Edge);
```

**Objective-C class**

```c
- (double)GetTextBound:(int)Edge;
```

**Parameters**

<table>
<thead>
<tr>
<th>Edge</th>
<th>The edge measurement to retrieve:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Left</td>
</tr>
<tr>
<td>2</td>
<td>Top</td>
</tr>
<tr>
<td>3</td>
<td>Right</td>
</tr>
<tr>
<td>4</td>
<td>Bottom</td>
</tr>
</tbody>
</table>

**Return values**

| 0                | The edge specified was not valid |
| Non-zero         | The specified edge measurement  |
**Description**

Returns the size of the selected font, measured from the baseline to the bottom of the tails of lowercase letters such as g and y.

**Syntax**

**Dylib**

```c
double DPLGetTextDescent(int InstanceID);
```

**Objective-C class**

```objective-c
- (double)getTextDescent
```

**Return values**

The descent of the selected font
GetTextHeight

Description

Returns the height of the selected font. This is the sum of `GetTextBound(2)` and `-GetTextBound(4)`.

Syntax

**Dylib**

```c
double DPLGetTextHeight(int InstanceID);
```

**Objective-C class**

```c
-(double)GetTextHeight
```

Return values

The height of the selected font
GetTextSize

Text, Fonts, Page layout

Description

Retrieves the current text size.

Syntax

Dylib

```c
double DPLGetTextSize(int InstanceID);
```

Objective-C class

```c
- (double)GetTextSize
```
**Description**

Calculate the width of the specified text, based on the selected font and font size.

**Syntax**

**Dylib**

```c
double DPLGetTextWidth(int InstanceID, wchar_t * Text);
```

**Objective-C class**

```objc
-(double)GetWidth:(NSString *)Text;
```

**Parameters**

| Text | The text to determine the width for |

**Return values**

| The width of the specified text |
GetUnicodeCharactersFromEncoding
Text, Fonts, Miscellaneous functions

Description

Returns a string containing all the Unicode characters from the specified encoding.

Syntax

Dylib

```c
wchar_t * DPLGetUnicodeCharactersFromEncoding(int InstanceID,
   int Encoding);
```

Objective-C class

```objc
- (NSString *)GetUnicodeCharactersFromEncoding:(int)Encoding;
```

Parameters

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>WinAnsiEncoding</td>
</tr>
</tbody>
</table>
**GetViewPortBBox**

**Page properties, Measurement and coordinate units**

**Description**

Returns details of the BBox entry of a viewport dictionary.

**Syntax**

**Dylib**

```c
double DPLGetViewPortBBox(int InstanceID, int ViewPortID, int Dimension);
```

**Objective-C class**

```objective-c
- (double)GetViewPortBBox:(int)ViewPortID :(int)Dimension;
```

**Parameters**

<table>
<thead>
<tr>
<th>ViewPortID</th>
<th>A value returned by the GetPageViewPortID function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>0 = Left</td>
</tr>
<tr>
<td></td>
<td>1 = Top</td>
</tr>
<tr>
<td></td>
<td>2 = Width</td>
</tr>
<tr>
<td></td>
<td>3 = Height</td>
</tr>
<tr>
<td></td>
<td>4 = Right</td>
</tr>
<tr>
<td></td>
<td>5 = Bottom</td>
</tr>
</tbody>
</table>


GetViewPortMeasureDict
Page properties, Measurement and coordinate units

Description

Returns the measurement dictionary of the specified viewport as a MeasureDictID value.

Syntax

Dylib

```c
int DPLGetViewPortMeasureDict(int InstanceID, int ViewPortID);
```

Objective-C class

```c
- (int)GetViewPortMeasureDict:(int)ViewPortID;
```

Parameters

| ViewPortID | A value returned by the GetPageViewPortID function |

Return values

| 0 | The specified ViewPortID was invalid or the viewport does not have a measurement dictionary |
| Non-zero | A MeasureDictID value |
GetViewPortName

**Page properties, Measurement and coordinate units**

**Description**

Returns the name of the specified viewport.

**Syntax**

**Dylib**

```c
wchar_t * DPLGetViewPortName(int InstanceID, int ViewPortID);
```

**Objective-C class**

```swift
-(NSString *)GetViewPortName:(int)ViewPortID;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>ViewPortID</strong></th>
<th>A value returned by the GetPageViewPortID function</th>
</tr>
</thead>
</table>
**GetViewPortPtDataDict**

*Page properties, Measurement and coordinate units*

**Description**

Returns the PtData dictionary of the specified viewport.

**Syntax**

**Dylib**

```c
int DPLGetViewPortPtDataDict(int InstanceID, int ViewPortID);
```

**Objective-C class**

```c
-(int)GetViewPortPtDataDict:(int)ViewPortID;
```

**Parameters**

<table>
<thead>
<tr>
<th>ViewPortID</th>
<th>A value returned by the <em>[GetPageViewPortID]</em> function</th>
</tr>
</thead>
</table>

**Return values**

| 0 | The ViewPortID parameter was incorrect or the viewport does not have a PtData |
| Non-zero | A PtDataID value |
GetViewerPreferences

Document properties

Description

Returns the viewer preferences for the document.

Syntax

Dylib

```c
int DPLGetViewerPreferences(int InstanceID, int Option);
```

Objective-C class

```c
- (int)GetViewerPreferences:(int)Option;
```

Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hide toolbar</td>
</tr>
<tr>
<td>2</td>
<td>Hide menubar</td>
</tr>
<tr>
<td>3</td>
<td>Hide window user interface</td>
</tr>
<tr>
<td>4</td>
<td>Resize window to first page size</td>
</tr>
<tr>
<td>5</td>
<td>Center window</td>
</tr>
<tr>
<td>6</td>
<td>Display document title</td>
</tr>
<tr>
<td>7</td>
<td>Page mode after full screen</td>
</tr>
<tr>
<td>8</td>
<td>Predominant text reading order</td>
</tr>
<tr>
<td>9</td>
<td>Display boundary for viewing</td>
</tr>
<tr>
<td>10</td>
<td>Clipping boundary for viewing</td>
</tr>
<tr>
<td>11</td>
<td>Display boundary for printing</td>
</tr>
<tr>
<td>12</td>
<td>Clipping boundary for printing</td>
</tr>
<tr>
<td>13</td>
<td>Default print dialog: scaling</td>
</tr>
<tr>
<td>14</td>
<td>Default print dialog: duplex</td>
</tr>
<tr>
<td>15</td>
<td>Default print dialog: auto paper tray</td>
</tr>
<tr>
<td>16</td>
<td>Default print dialog: number of copies</td>
</tr>
</tbody>
</table>

Return values

See the `SetViewerPreferences` function to determine possible return values for each Option value.
GetWrappedText

Text, Page layout

Description

Get the positions where text will wrap, based on the current font and text size.

Syntax

Dylib

```c
wchar_t * DPLGetWrappedText(int InstanceID, double Width,
                           wchar_t * Delimiter, wchar_t * Text);
```

Objective-C class

```objc
- (NSString *)GetWrappedText:(double)Width :(NSString *)Delimiter
                           :(NSString *)Text;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>The width of the block to wrap the text to</td>
</tr>
<tr>
<td>Delimiter</td>
<td>The string to place between each line</td>
</tr>
<tr>
<td>Text</td>
<td>The text to wrap</td>
</tr>
</tbody>
</table>

Return values

Returns the lines of the text block, separated by the Delimiter string
GetWrappedTextBreakString

Text

Description

Similar to the GetWrappedText function, but preserves the break strings originally in the text. This is useful for splitting text into different areas on the page.

Syntax

Dylib

wchar_t * DPLGetWrappedTextBreakString(int InstanceID, double Width, wchar_t * Delimiter, wchar_t * Text);

Objective-C class

- (NSString *)GetWrappedTextBreakString:(double)Width :(NSString *)Delimiter :(NSString *)Text;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>The width that the text should be wrapped to</td>
</tr>
<tr>
<td>Delimiter</td>
<td>The delimiter to use between wrapped lines</td>
</tr>
<tr>
<td>Text</td>
<td>The text to wrap</td>
</tr>
</tbody>
</table>
GetWrappedTextHeight

Text, Page layout

Description
Get the height of a block of text wrapped to a certain width, based on the current font and text size.

Syntax

**Dylib**

```c
double DPLGetWrappedTextHeight(int InstanceID, double Width, wchar_t * Text);
```

**Objective-C class**

```objective-c
-(double)GetWrappedTextHeight:(double)Width :(NSString *)Text;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>The width of the block to wrap the text to</td>
</tr>
<tr>
<td>Text</td>
<td>The text to wrap</td>
</tr>
</tbody>
</table>

Return values

Returns the height of the text block
GetWrappedTextLineCount

**Description**

Determine the number of lines a block of text wrapped to a certain width will take up, based on the current font and text size.

**Syntax**

**Dylib**

```c
int DPLGetWrappedTextLineCount(int InstanceID, double Width, wchar_t * Text);
```

**Objective-C class**

```objective-c
- (int)GetWrappedTextLineCount:(double)Width :(NSString *)Text;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>The width of the block to wrap the text to</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>The text to wrap</td>
</tr>
</tbody>
</table>

**Return values**

The number of lines
GetXFAFormFieldCount

Form fields

Description

Returns the number of XFA form fields in the selected document.

Syntax

Dylib

```c
int DPLGetXFAFormFieldCount(int InstanceID);
```

Objective-C class

```objc
- (int)GetXFAFormFieldCount
```
GetXFAFormFieldName

Form fields

Description

Returns the name of the specified form field.

Syntax

Dylib

wchar_t * DPLGetXFAFormFieldName(int InstanceID, int Index);

Objective-C class

- (NSString *)GetXFAFormFieldName:(int)Index;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the XFA form field. The first XFA form field has an index of 1 and the last XFA form field has a value as returned by the GetXFAFormFieldCount function.</th>
</tr>
</thead>
</table>

GetXFAFormFieldNames

Form fields

Description

Returns a list of the names of the XFA form fields separated by the specified delimiter.

Syntax

Dylib

wchar_t * DPLGetXFAFormFieldNames(int InstanceID, wchar_t * Delimiter);

Objective-C class

- (NSString *)GetXFAFormFieldNames:(NSString *)Delimiter;

Parameters

| Delimiter | The delimiter to use to separate the XFA form field names. |
GetXFAFormField

Description

Returns the value of the specified XFA form field.

Syntax

Dylib

wchar_t * DPLGetXFAFormFieldValue(int InstanceID, wchar_t * XFAFieldName);

Objective-C class

- (NSString *)GetXFAFormFieldValue:(NSString *)XFAFieldName;

Parameters

| XFAFieldName | The name of the XFA field to work with. |
GetXFAToString

Form fields

Description

Returns the complete XFA form contents as an XML string.

Syntax

Dylib

```c
char * DPLGetXFAToString(int InstanceID, int Options);
```

Objective-C class

```objc
- (NSData *)GetXFAToString:(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Options</th>
<th>Reserved for future use. Should be set to zero.</th>
</tr>
</thead>
</table>
GlobalJavaScriptCount

Document properties, JavaScript

Description

Returns the number of global JavaScript packages in the selected document.

Syntax

Dylib

```c
int DPLGlobalJavaScriptCount(int InstanceID);
```

Objective-C class

```c
-(int)GlobalJavaScriptCount
```
GlobalJavaScriptPackageName

**Document properties, JavaScript**

**Description**

Returns the name of the JavaScript package with the specified index. This package name can be used with the `GetGlobalJavaScript` function.

**Syntax**

**Dylib**

```c
wchar_t * DPLGlobalJavaScriptPackageName(int InstanceID, int Index);
```

**Objective-C class**

```objc
- (NSString *)GlobalJavaScriptPackageName:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the global JavaScript package. The first package has an index of 1. The last package has an index equal to the value returned by the <code>GlobalJavaScriptCount</code> function.</th>
</tr>
</thead>
</table>
HasFontResources

Fonts, Document properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Determines if the selected document has font resources. If the document does not have font resources it can be assumed to be an image only PDF.

Syntax

Dylib

```c
int DPLHasFontResources(int InstanceID);
```

Objective-C class

```c
- (int)HasFontResources
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected document does not have font resources</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The selected document has font resources</td>
</tr>
</tbody>
</table>
HasPageBox
Page properties

Description
Indicates whether the selected page has the specified boundary rectangle.

Syntax
Dylib

    int DPLHasPageBox(int InstanceID, int BoxType);

Objective-C class

    -(int)HasPageBox:(int)BoxType;

Parameters

<table>
<thead>
<tr>
<th>BoxType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MediaBox</td>
</tr>
<tr>
<td>2</td>
<td>CropBox</td>
</tr>
<tr>
<td>3</td>
<td>BleedBox</td>
</tr>
<tr>
<td>4</td>
<td>TrimBox</td>
</tr>
<tr>
<td>5</td>
<td>ArtBox</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page does not have the specified boundary rectangle</td>
</tr>
<tr>
<td>1</td>
<td>The page has the specified boundary rectangle</td>
</tr>
<tr>
<td>2</td>
<td>The page does not have the specified boundary rectangle, but there is a value in a parent page tree node that is being inherited by the page</td>
</tr>
</tbody>
</table>
HidePage
Page properties, Page manipulation

Description

Hides the selected page. This is similar to deleting the page, but the page contents are not removed from the PDF document. This is sometimes useful when used in conjunction with the ClonePages function.

Syntax

Dylib

int DPLHidePage(int InstanceID);

Objective-C class

- (int)HidePage
**ImageCount**

**Image handling, Document properties**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Returns the total number of images added to the PDF file. This function does not take into account the images that may have already been in an existing PDF document which was loaded with the `LoadFromFile` function.

**Syntax**

**Dylib**

```c
int DPLImageCount(int InstanceID);
```

**Objective-C class**

```objective-c
- (int)ImageCount
```
Description

Returns the color of the center pixel in the selected image. This could be used to identify a placeholder image for later replacement.

Syntax

**Dylib**

```
int DPLImageFillColor(int InstanceID);
```

**Objective-C class**

```
-(int)ImageFillColor
```
ImageHeight

Image handling

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

The height of the selected image.

Syntax

**Dylib**

```c
int DPLImageHeight(int InstanceID);
```

**Objective-C class**

```objective-c
- (int)ImageHeight
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image has been selected</td>
</tr>
<tr>
<td><strong>Non-zero</strong></td>
<td>The height in pixels of the selected image</td>
</tr>
</tbody>
</table>
**ImageHorizontalResolution**

**Image handling**

**Description**

Returns the horizontal resolution of the selected image, if it is available. Presently only the resolution of JFIF/JPEG, Exif/JPEG, TIFF and BMP images can be retrieved. Use the **ImageResolutionUnits** function to determine if this measurement is in dots-per-inch (DPI) or dots-per-centimetre (DPCM).

**Syntax**

**Dylib**

```c
int DPLImageHorizontalResolution(int InstanceID);
```

**Objective-C class**

```c
- (int)ImageHorizontalResolution
```
ImageResolutionUnits

Image handling

Description

Use this function to determine the units of the ImageHorizontalResolution and ImageVerticalResolution results.

Syntax

Dylib

    int DPLImageResolutionUnits(int InstanceID);

Objective-C class

- (int)ImageResolutionUnits

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unknown</td>
</tr>
<tr>
<td>1</td>
<td>No units, values specify the aspect ratio</td>
</tr>
<tr>
<td>2</td>
<td>Dots per inch (DPI)</td>
</tr>
<tr>
<td>3</td>
<td>Dots per centimetre (DPCM)</td>
</tr>
</tbody>
</table>
**ImageType**

**Image handling**

**Description**

The type of the selected image.

**Syntax**

**Dylib**

```c
int DPLImageType(int InstanceID);
```

**Objective-C class**

```c
-(int)ImageType
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image is selected</td>
</tr>
<tr>
<td>1</td>
<td>The selected image is a JPEG image</td>
</tr>
<tr>
<td>2</td>
<td>The selected image is a BMP image</td>
</tr>
<tr>
<td>3</td>
<td>The selected image is a TIFF image</td>
</tr>
</tbody>
</table>
ImageVerticalResolution

Image handling

Description

Returns the vertical resolution of the selected image, if it is available. Presently only the resolution of JFIF/JPEG, Exif/JPEG, TIFF and BMP images can be retrieved. Use the ImageResolutionUnits function to determine if this measurement is in dots-per-inch (DPI) or dots-per-centimetre (DPCM).

Syntax

**Dylib**

```c
int DPLImageVerticalResolution(int InstanceID);
```

**Objective-C class**

```c
- (int)ImageVerticalResolution
```
ImageWidth

Image handling

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

The width of the selected image.

Syntax

Dylib

int DPLImageWidth(int InstanceID);

Objective-C class

- (int)ImageWidth

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image has been selected</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The width in pixels of the selected image</td>
</tr>
</tbody>
</table>
ImportEMFFromFile

Vector graphics, Image handling

Description

Adds a WMF or EMF image from a file to the selected document. Once an image has been added to the document it can be drawn on any page multiple times without further increasing the size of the PDF file.

Syntax

Dylib

```c
int DPLImportEMFFromFile(int InstanceID, wchar_t * FileName,
                          int FontOptions, int GeneralOptions);
```

Objective-C class

```objc
- (int)ImportEMFFromFile:(NSString *)FileName :(int)FontOptions
            :(int)GeneralOptions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The file name of the image to add.</td>
</tr>
<tr>
<td>FontOptions</td>
<td>If GeneralOptions is 1 this parameter is ignored, otherwise the following</td>
</tr>
<tr>
<td></td>
<td>values take effect:</td>
</tr>
<tr>
<td></td>
<td>0 = Use the first font added to the PDF</td>
</tr>
<tr>
<td></td>
<td>1 = Automatically add fonts as non-embedded TrueType fonts</td>
</tr>
</tbody>
</table>
| GeneralOptions | 0 = Import as a vector image  
|             | 1 = Import as a bitmap image                                                 |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image could not be added</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The image was added successfully. The ImageID is returned which can be</td>
</tr>
<tr>
<td></td>
<td>passed to functions like SelectImage and DrawImage.</td>
</tr>
</tbody>
</table>
InsertPages

Document management, Page manipulation

Description

Inserts one or more blank pages into the document.

Syntax

Dylib

```c
int DPLInsertPages(int InstanceID, int StartPage, int PageCount);
```

Objective-C class

```objc
-(int)InsertPages:(int)StartPage :(int)PageCount;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPage</td>
<td>The page number of the first page to insert</td>
</tr>
<tr>
<td>PageCount</td>
<td>The total number of pages to insert</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Failed</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The new total number of pages in the document</td>
</tr>
</tbody>
</table>
InsertTableColumns

Page layout

Description

Adds columns to the specified table at any position

Syntax

Dylib

```c
int DPLInsertTableColumns(int InstanceID, int TableID, int Position,
                           int NewColumnCount);
```

Objective-C class

```objective-c
- (int)InsertTableColumns:(int)TableID :(int)Position :(int)NewColumnCount;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the CreateTable function</td>
</tr>
<tr>
<td>Position</td>
<td>The position to insert the new columns. Minimum value is 1. Maximum value is one greater than the value returned by the GetTableColumnCount function.</td>
</tr>
<tr>
<td>NewColumnCount</td>
<td>The number of columns to add to the table</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Columns could not be added. Check the TableID parameter and make sure NewColumnCount is greater than or equal to 1. The Position parameter must also be within range.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The total number of columns in the table after adding the new columns.</td>
</tr>
</tbody>
</table>
InsertTableRows

Page layout

Description

Adds rows to the specified table at any position

Syntax

Dylib

```c
int DPLInsertTableRows(int InstanceID, int TableID, int Position,
int NewRowCount);
```

Objective-C class

```objective-c
-(int)InsertTableRows:(int)TableID :(int)Position :(int)NewRowCount;
```

Parameters

<table>
<thead>
<tr>
<th>TableID</th>
<th>A TableID returned by the CreateTable function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>The position to insert the new rows. Minimum value is 1. Maximum value is one greater than the value returned by the GetTableRowCount function.</td>
</tr>
<tr>
<td>NewRowCount</td>
<td>The number of rows to add to the table</td>
</tr>
</tbody>
</table>

Return values

| 0          | Rows could not be added. Check the TableID parameter and make sure NewRowCount is greater than or equal to 1. The Position parameter must also be within range. |
| Non-zero   | The total number of rows in the table after adding the new rows. |
IsAnnotFormField
Form fields, Annotations and hotspot links

Description
For an annotation to be a form field it must be attached to the document form.
This function checks if the specified annotation is allowed to be attached to the document form and whether it is currently attached.
For an annotation to be attached to the document form it must be a Widget annotation and it cannot be a child of another annotation.

Syntax
Dylib

int DPLIsAnnotFormField(int InstanceID, int Index);

Objective-C class

-(int)IsAnnotFormField:(int)Index;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified annotation is not a Widget annotation or it is the child of another annotation.</td>
</tr>
<tr>
<td>1</td>
<td>The specified annotation is a form field and is currently attached to the document form.</td>
</tr>
<tr>
<td>2</td>
<td>The specified annotation is in the correct format to be a form field but it is not currently attached to the document form. Use the AttachAnnotToForm function to attach it.</td>
</tr>
</tbody>
</table>
**IsTaggedPDF**

**Description**

Determines if the selected document has the MarkInfo/Marked property set.

**Syntax**

**Dylib**

```c
int DPLIsTaggedPDF(int InstanceID);
```

**Objective-C class**

```objective-c
-(int)IsTaggedPDF
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document is not tagged</td>
</tr>
<tr>
<td>1</td>
<td>The document is tagged</td>
</tr>
</tbody>
</table>
**LastErrorCode**

**Miscellaneous functions**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Use this function to determine the reason certain functions failed.

**Syntax**

Dylib

```c
int DPLLastErrorCode(int InstanceID);
```

**Objective-C class**

```c
- (int)LastErrorCode
```

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>The Strength parameter passed to the Encrypt function was invalid</td>
</tr>
<tr>
<td>102</td>
<td>The Permissions parameter passed to the Encrypt function was invalid. Use the EncodePermissions function to construct a value for this parameter</td>
</tr>
<tr>
<td>103</td>
<td>The Encrypt function was used on a document that was already encrypted</td>
</tr>
<tr>
<td>104</td>
<td>The Encrypt function failed for an unknown reason</td>
</tr>
<tr>
<td>201</td>
<td>The SetInformation function failed because the document is encrypted</td>
</tr>
<tr>
<td>202</td>
<td>The Key parameter passed to the SetInformation function was out of range</td>
</tr>
<tr>
<td>301</td>
<td>An invalid combination of barcode and option was sent to the DrawBarcode function</td>
</tr>
<tr>
<td>302</td>
<td>Non-numeric characters were sent to DrawBarcode using EAN-13</td>
</tr>
<tr>
<td>303</td>
<td>The EAN-13 barcode has an invalid checksum character</td>
</tr>
<tr>
<td>401</td>
<td>Could not open input file</td>
</tr>
<tr>
<td>402</td>
<td>Output file already exists and could not be deleted</td>
</tr>
<tr>
<td>403</td>
<td>Could not open output file</td>
</tr>
<tr>
<td>404</td>
<td>Invalid password</td>
</tr>
<tr>
<td>405</td>
<td>Document is not encrypted</td>
</tr>
<tr>
<td>406</td>
<td>Document is already encrypted</td>
</tr>
<tr>
<td>407</td>
<td>Invalid encryption strength</td>
</tr>
<tr>
<td>408</td>
<td>Invalid permissions</td>
</tr>
<tr>
<td>409</td>
<td>Invalid file structure, file is damaged</td>
</tr>
<tr>
<td>410</td>
<td>One of the input files is encrypted</td>
</tr>
<tr>
<td>411</td>
<td>File not found</td>
</tr>
<tr>
<td>412</td>
<td>Invalid page range list</td>
</tr>
<tr>
<td>501</td>
<td>The specified FileHandle was invalid</td>
</tr>
<tr>
<td>999</td>
<td>The function could not be used because the library is not unlocked</td>
</tr>
</tbody>
</table>
LastRenderError

Miscellaneous functions, Rendering and printing

Description

Returns the exception information in cases where the renderer encountered an error.

Syntax

Dylib

wchar_t * DPLastRenderError(int InstanceID);

Objective-C class

- (NSString *)LastRenderError
LibraryVersion

Miscellaneous functions

Description

Returns the version of the library, for example "7.12".

Syntax

**Dylib**

```c
wchar_t * DPLLibraryVersion(int InstanceID);
```

**Objective-C class**

```objective-c
- (NSString *)LibraryVersion
```
# LicenseInfo

## Miscellaneous functions

### Description

Returns information about the unlock license used.

### Syntax

**Dylib**

```c
wchar_t * DPLLicenseInfo(int InstanceID);
```

**Objective-C class**

```objc
- (NSString *)LicenseInfo
```
Linearized

Document properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Reports whether the selected document was loaded from a linearized file. This is for informational purposes only. If the file is resaved it will no longer be linearized.

Syntax

**Dylib**

```
int DPLLinearized(int InstanceID);
```

**Objective-C class**

```
- (int)linearized
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The original file was not linearized</td>
</tr>
<tr>
<td>1</td>
<td>The original file was linearized</td>
</tr>
</tbody>
</table>
**LoadFromCanvasDC**

**Vector graphics, Document management**

**Description**

Creates a new document from the drawing operations applied to the DC returned by the `GetCanvasDC` function.

When the Options parameter is set to 3, use the `NoEmbedFontListAdd` function to add fonts to the no embed font list.

**Syntax**

**Dylib**

```c
int DPLLoadFromCanvasDC(int InstanceID, double DPI, int Options);
```

**Objective-C class**

```objective-c
- (int)LoadFromCanvasDC:(double)DPI :(int)Options;
```

**Parameters**

- **DPI**
  
  The DPI to use for the new document. For example, if the canvas was created with a width and height of 96 and the DPI is specified as 192, the resulting document will be 0.5 inches in width and height.

- **Options**
  
  - 1 = Convert the drawing commands to a single image using GDI+
  - 0 = Process the drawing commands as vector graphics, fonts are not embedded
  - 1 = Process the drawing commands as vector graphics, fonts are embedded but not compressed
  - 2 = Process the drawing commands as vector graphics, fonts are embedded and compressed
  - 3 = Process the drawing commands as vector graphics, fonts not in the no embed font list are embedded and compressed

**Return values**

- **0**
  
  A canvas has not been created

- **1**
  
  The canvas DC was processed correctly and a new document has been created
**LoadFromFile**

**Document management**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Loads a PDF document from a file on disk. If the function succeeds, the loaded document will be selected and its DocumentID can be retrieved using the `SelectedDocument` function.

**Syntax**

**Dylib**

```c
int DPLLoadFromFile(int InstanceID, wchar_t * FileName, wchar_t * Password);
```

**Objective-C class**

```objective-c
- (int)LoadFromFile:(NSString *)FileName :(NSString *)Password;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The path and file name of the file to load.</td>
</tr>
<tr>
<td>Password</td>
<td>The password to open the file</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file could not be read or processed. Use the <code>LastErrorCode</code> function to determine the cause of the failure.</td>
</tr>
<tr>
<td>1</td>
<td>The file was loaded successfully</td>
</tr>
</tbody>
</table>
LoadFromString

Document management

Description

Similar to the LoadFromFile function, except the data for the PDF document is passed in as a string. If the function succeeds, the loaded document will be selected and its DocumentID can be retrieved using the SelectedDocument function.

Syntax

Dylib

```c
int DPLLoadFromString(int InstanceID, char * Source, wchar_t * Password);
```

Objective-C class

```objective-c
- (int)LoadFromString:(NSData *)Source :(NSString *)Password;
```

Parameters

<table>
<thead>
<tr>
<th>Source</th>
<th>The source data to load the PDF document from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>The password to load the file</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The PDF could not be loaded</td>
</tr>
<tr>
<td>1</td>
<td>The PDF was loaded from the string successfully</td>
</tr>
</tbody>
</table>
LoadState

Vector graphics, Page layout

Description

Loads the graphics state previously stored with SaveState.

Syntax

**Dylib**

```c
int DPLLoadState(int InstanceID);
```

**Objective-C class**

```c
-(int)LoadState
```
**MergeDocument**

**Document manipulation**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Use this function to join another document to the selected document. After merging, the second document is deleted.

Form fields and annotations from the second document are preserved but outlines (bookmarks) are not.

**Syntax**

**Dylib**

```c
int DPLMergeDocument(int InstanceID, int DocumentID);
```

**Objective-C class**

```c
-(int)MergeDocument:(int)DocumentID;
```

**Parameters**

| DocumentID | The ID of the document to join to the selected document |

**Return values**

| 0 | The documents could not be merged together |
| 1 | The merging was successful |
MergeFileList

Document manipulation

Description

Merges all the files in a named file list and saves the resulting merged document to the specified file. Use the ClearFileList, FileListCount and AddToFileList functions to construct the named file list. There must be two or more files in the file list in order for the merging to succeed.

Outlines (bookmarks), form fields and annotations from all the documents will be present in the merged document.

Syntax

Dylib

```c
int DPLMergeFileList(int InstanceID, wchar_t * ListName, wchar_t * OutputFileName);
```

Objective-C class

```objective-c
-(int)MergeFileList:(NSString *)ListName :(NSString *)OutputFileName;
```

Parameters

<table>
<thead>
<tr>
<th>ListName</th>
<th>The name of the list of files to merge together</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputFileName</td>
<td>The path and file name of the file to create which will contain the merged files.</td>
</tr>
</tbody>
</table>

Return values

The number of documents which were successfully merged together. If this is less than the intended number use the FileListItem function to find the file which caused the merge process to end prematurely.
**MergeFileListFast**

**Document manipulation**

**Description**

Similar to the `MergeFileList` function, but uses an advanced algorithm to improve speed.

A new file list will be created during merging that will contain the result of the merge process for each of the items in the specified file list. The new file list will have the same name as the original file list with the word Result appended. For example, if the original file list was called "MyFiles", then the new file list will be called "MyFilesResult". This new file list will not contain file names, but will contain a text description of the status of the matching file during the merge process.

There must be two or more files in the file list in order for the merging to succeed.

Form fields and annotations from all the documents will be present in the merged document but only outlines (bookmarks) from the first document will be in the merged document.

**Syntax**

**Dylib**

```c
int DPLMergeFileListFast(int InstanceID, wchar_t * ListName, wchar_t * OutputFileName);
```

**Objective-C class**

```objc
- (int)MergeFileListFast:(NSString *)ListName :(NSString *)OutputFileName;
```

**Parameters**

<table>
<thead>
<tr>
<th>ListName</th>
<th>The name of the file list to use. All the files in this list will be merged together.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputFileName</td>
<td>The path and file name of the output file to create. This file will contain all the files from the file list.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The merge process could not be completed. Use the GetLastError function to determine the cause of the error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>The number of files that were successfully merged</td>
</tr>
</tbody>
</table>
**MergeFiles**

**Document manipulation**

**Description**

Merges two files on disk and saves the merged document to a new file. The files are accessed directly on disk, the entire file does not have to be loaded into memory so this function can be used with huge documents. The files must not be encrypted. Monitor the size of the output file while this function runs to work out the progress.

Outlines (bookmarks), form fields and annotations from the both documents will be present in the merged document.

**Syntax**

**Dylib**

```c
int DPLMergeFiles(int InstanceID, wchar_t * FirstFileName, wchar_t * SecondFileName, wchar_t * OutputFileName);
```

**Objective-C class**

```objc
- (int)MergeFiles:(NSString *)FirstFileName : (NSString *)SecondFileName : (NSString *)OutputFileName;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstFileName</td>
<td>The name of the first file to merge.</td>
</tr>
<tr>
<td>SecondFileName</td>
<td>The name of the second file to merge.</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>The name of the file to create which will contain the merged document.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The files could not be merged. Use the <code>LastErrorCode</code> function to determine the cause of the failure.</td>
</tr>
<tr>
<td>1</td>
<td>The files were merged successfully and the new merged document was created</td>
</tr>
</tbody>
</table>
MergeTableCells

Description

Merges multiple cells from the specified table into one cell.

Syntax

**Dylib**

```c
int DPLMergeTableCells(int InstanceID, int TableID, int FirstRow,
    int FirstColumn, int LastRow, int LastColumn);
```

**Objective-C class**

```c
- (int)MergeTableCells:(int)TableID :(int)FirstRow :(int)FirstColumn
    :(int)LastRow :(int)LastColumn;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <a href="#">CreateTable</a> function</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The the number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
</tbody>
</table>
MoveContentStream
Content Streams and Optional Content Groups

Description
A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.
This function can be used to change the order in which the content stream parts are drawn onto the page to bring certain information to the front or push it to the back.
Content stream parts that you want placed at the back should be drawn first (index of 1).

Syntax

Dylib

int DPLMoveContentStream(int InstanceID, int FromPosition, int ToPosition);

Objective-C class

- (int)MoveContentStream:(int)FromPosition :(int)ToPosition;

Parameters

FromPosition  The current content stream part index. The first content stream part has an index of 1. The last content stream part has an index equal to the value returned by the ContentStreamCount function.

ToPosition  The new content stream part index.

Return values

0  The content stream part could not be moved

1  Success
MoveOutlineAfter

Outlines

Description

Moves an outline item to appear directly after another outline item. The outline will be moved along with all children nodes.

Syntax

Dylib

int DPLMoveOutlineAfter(int InstanceID, int OutlineID, int SiblingID);

Objective-C class

- (int)MoveOutlineAfter:(int)OutlineID :(int)SiblingID;

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiblingID</td>
<td>The outline will be moved to a position after the outline with this ID</td>
</tr>
</tbody>
</table>

Return values

| 0         | The outline was not moved, the OutlineID or SiblingID parameters were invalid or were the same value                     |
| 1         | The outline was moved successfully                                                                                  |
MoveOutlineBefore

Outlines

Description

Moves an outline item to appear directly before another outline item. The outline will be moved along with all children nodes.

Syntax

Dylib

```c
int DPLMoveOutlineBefore(int InstanceID, int OutlineID, int SiblingID);
```

Objective-C class

```objectivec
-(int)MoveOutlineBefore:(int)OutlineID :(int)SiblingID;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutlineID</td>
<td>The ID of the outline as returned by the <code>NewOutline</code> function. Alternatively, use the <code>GetOutlineID</code> function to get a valid outline ID.</td>
</tr>
<tr>
<td>SiblingID</td>
<td>The outline will be moved to a position before the outline with this ID</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The outline was not moved, the OutlineID or SiblingID parameters were invalid or were the same value</td>
</tr>
<tr>
<td>1</td>
<td>The outline was moved successfully</td>
</tr>
</tbody>
</table>
MovePage

Document management, Page manipulation

Description

Moves the selected page to a new position in the document.

Syntax

Dylib

```c
int DPLMovePage(int InstanceID, int NewPosition);
```

Objective-C class

```objc
- (int)MovePage:(int)NewPosition;
```

Parameters

- **NewPosition**  
The new position of the page

Return values

- **0**  
The page could not be moved. Check the value of the NewPosition parameter.
- **1**  
The page was moved successfully
MovePath

Vector graphics, Path definition and drawing

Description

Starts a new sub-path within the current path. This allows complex shapes to be created (for example, with pieces cut out).

Syntax

Dylib

int DPLMovePath(int InstanceID, double NewX, double NewY);

Objective-C class

- (int)MovePath:(double)NewX :(double)NewY;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewX</td>
<td>The new horizontal co-ordinate of the starting point of the new sub-path</td>
</tr>
<tr>
<td>NewY</td>
<td>The new vertical co-ordinate of the starting point of the new sub-path</td>
</tr>
</tbody>
</table>
MultiplyScale

Measurement and coordinate units

Description

Multiplies the drawing scale by a specified factor. For example, multiplying the scale by 0.5 will draw graphics at half their size with the same drawing commands.

Syntax

Dylib

```c
int DPLMultiplyScale(int InstanceID, double MultScaleBy);
```

Objective-C class

```objective-c
-(int)MultiplyScale:(double)MultScaleBy;
```

Parameters

<table>
<thead>
<tr>
<th><strong>MultScaleBy</strong></th>
<th>The factor to multiply the current drawing scale by</th>
</tr>
</thead>
</table>
NewChildFormField

Form fields

Description

Adds a new form field to the selected page as a child of another field.

Syntax

Dylib

```c
int DPLNewChildFormField(int InstanceID, int Index, wchar_t * Title,
int FieldType);
```

Objective-C class

```objective-c
- (int)NewChildFormField:(int)Index :(NSString *)Title :(int)FieldType;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the parent field.</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the new form field. The title cannot contain the period &quot;.&quot; character.</td>
</tr>
<tr>
<td>FieldType</td>
<td>The type of the field to create:</td>
</tr>
<tr>
<td></td>
<td>1 = Text</td>
</tr>
<tr>
<td></td>
<td>2 = Pushbutton</td>
</tr>
<tr>
<td></td>
<td>3 = Checkbox</td>
</tr>
<tr>
<td></td>
<td>4 = Radiobutton</td>
</tr>
<tr>
<td></td>
<td>5 = Choice</td>
</tr>
<tr>
<td></td>
<td>6 = Signature</td>
</tr>
<tr>
<td></td>
<td>7 = Parent</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The new form field could not be created</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The form field was created successfully, and this is the index of the new field</td>
</tr>
</tbody>
</table>
NewContentStream

Content Streams and Optional Content Groups

Description

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function creates a new content stream part on the selected page. If required, the new content stream part can then be moved behind the existing information on the page using the MoveContentStream function.

Syntax

Dylib

int DPLNewContentStream(int InstanceID);

Objective-C class

- (int)NewContentStream

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The new content stream part could not be created</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The index of the new content stream part. The first part has an index of 1.</td>
</tr>
</tbody>
</table>
NewCustomPrinter
Rendering and printing

Description

Creates a custom printer and returns the name of the custom printer. The returned printer name can be used as the PrinterName parameter of the PrintDocument function. Before printing, the properties of the printer can be set using the SetupCustomPrinter function.

Syntax

**Dylib**

```c
wchar_t * DPLNewCustomPrinter(int InstanceID,
wchar_t * OriginalPrinterName);
```

**Objective-C class**

```objective-c
- (NSString *)NewCustomPrinter:(NSString *)OriginalPrinterName;
```

Parameters

| OriginalPrinterName | The name of the printer to use for printing. This is the name that appears in the Windows Print Manager. Use the GetPrinterNames function to return a list of valid printers on the system. |
NewDestination

Annotations and hotspot links, Document management

Description

Creates a new destination object that can be used with the AddLinkToDestination, GetDestPage, GetDestType or GetDestValue functions.

Syntax

**Dylib**

```c
int DPLNewDestination(int InstanceID, int DestPage, int Zoom,
                      int DestType, double Left, double Top, double Right,
                      double Bottom);
```

**Objective-C class**

```objc
-(int)NewDestination:(int)DestPage :(int)Zoom :(int)DestType
:(double)Left :(double)Top :(double)Right :(double)Bottom;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestPage</td>
<td>The page number that this destination object links to</td>
</tr>
<tr>
<td>Zoom</td>
<td>The zoom percentage to use for the destination object, valid values from 0 to 6400. Only used for DestType = 1, should be set to 0 for other DestTypes.</td>
</tr>
<tr>
<td>DestType</td>
<td>1 = &quot;XYZ&quot; - the target page is positioned at the point specified by the Left and Top parameters. The Zoom parameter specifies the zoom percentage. 2 = &quot;Fit&quot; - the entire page is zoomed to fit the window. None of the other parameters are used and should be set to zero. 3 = &quot;FitH&quot; - the page is zoomed so that the entire width of the page is visible. The height of the page may be greater or less than the height of the window. The page is positioned at the vertical position specified by the Top parameter. 4 = &quot;FitV&quot; - the page is zoomed so that the entire height of the page can be seen. The width of the page may be greater or less than the width of the window. The page is positioned at the horizontal position specified by the Left parameter. 5 = &quot;FitR&quot; - the page is zoomed so that a certain rectangle on the page is visible. The Left, Top, Right and Bottom parameters define the rectangular area on the page. 6 = &quot;FitB&quot; - the page is zoomed so that it's bounding box is visible. 7 = &quot;FitBH&quot; - the page is positioned vertically at the position specified by the Top parameter. The page is zoomed so that the entire width of the page's bounding box is visible. 8 = &quot;FitBV&quot; - the page is positioned at the horizontal position specified by the Left parameter. The page is zoomed just enough to fit the entire height of the bounding box into the window.</td>
</tr>
<tr>
<td>Left</td>
<td>The horizontal position used by DestType = 1, 4, 5 and 8</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical position used by DestType = 1, 3, 5 and 7</td>
</tr>
<tr>
<td>Right</td>
<td>The horizontal position of the righthand edge of the rectangle. Used by DestType = 5</td>
</tr>
<tr>
<td>Bottom</td>
<td>The horizontal position of the bottom of the rectangle. Used by DestType = 5</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The DestPage parameter was invalid</td>
</tr>
<tr>
<td>Non-zero</td>
<td>A DestID value</td>
</tr>
</tbody>
</table>
NewDocument

Document management

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Creates a new blank document.

Syntax

Dylib

```c
int DPLNewDocument(int InstanceID);
```

Objective-C class

```objc
-(int)NewDocument
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>There was an error while trying to create the new document. This should never occur.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the new document</td>
</tr>
</tbody>
</table>
NewFormField

Form fields

Description

Adds a new form field to the selected page.

Syntax

Dylib

```c
int DPLNewFormField(int InstanceID, wchar_t * Title, int FieldType);
```

Objective-C class

```objective-c
- (int)NewFormField:(NSString *)Title :(int)FieldType;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The title of the new form field. The title cannot contain the period &quot;.&quot; character.</td>
</tr>
<tr>
<td>FieldType</td>
<td>The type of the field to create:</td>
</tr>
<tr>
<td></td>
<td>1 = Text</td>
</tr>
<tr>
<td></td>
<td>2 = Pushbutton</td>
</tr>
<tr>
<td></td>
<td>3 = Checkbox</td>
</tr>
<tr>
<td></td>
<td>4 = Radiobutton</td>
</tr>
<tr>
<td></td>
<td>5 = Choice</td>
</tr>
<tr>
<td></td>
<td>6 = Signature</td>
</tr>
<tr>
<td></td>
<td>7 = Parent</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The new form field could not be created</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The form field was created successfully, and this is the index of the new field</td>
</tr>
</tbody>
</table>
NewInternalPrinterObject

Rendering and printing

Description

Creates a new internal printer object for use by subsequent printing operations.

Syntax

**Dylib**

```c
int DPLNewInternalPrinterObject(int InstanceID, int Options);
```

**Objective-C class**

```c
- (int)NewInternalPrinterObject:(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options</strong></td>
<td>Must be set to 0</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The options parameter was not zero or the new internal printer object could not be created</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
NewNamedDestination

Annotations and hotspot links

Description

Creates a named destination.

Syntax

**Dylib**

```c
int DPLNewNamedDestination(int InstanceID, wchar_t * DestName, int DestID);
```

**Objective-C class**

```c
-(int)NewNamedDestination:(NSString *)DestName :(int)DestID;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DestName</strong></td>
<td>The name of the destination</td>
</tr>
<tr>
<td><strong>DestID</strong></td>
<td>The destination to assign a name to</td>
</tr>
</tbody>
</table>
**NewOptionalContentGroup**

**Content Streams and Optional Content Groups**

**Description**

Creates a new optional content group. The group name will appear in the Layers tab in Acrobat 6 or later.

**Syntax**

**Dylib**

```c
int DPLNewOptionalContentGroup(int InstanceID, wchar_t * GroupName);
```

**Objective-C class**

```objective-c
-(int)NewOptionalContentGroup:(NSString *)GroupName;
```

**Parameters**

<table>
<thead>
<tr>
<th>GroupName</th>
<th>The name of the optional content group. This name is displayed in the PDF viewer user interface.</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The new optional content group could not be created</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>An ID that can be used as the OptionalContentGroupId parameter with the other optional content group functions</td>
</tr>
</tbody>
</table>
**NewOutline**

**Outlines**

**Description**

Adds a new outline item to the document. Outline items can be added in a hierarchical structure. In Acrobat Reader, outlines are referred to as bookmarks.

**Syntax**

**Dylib**

```c
int DPLNewOutline(int InstanceID, int Parent, wchar_t * Title,
int DestPage, double DestPosition);
```

**Objective-C class**

```c
- (int)NewOutline:(int)Parent :(NSString *)Title :(int)DestPage :
(double)DestPosition;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>0 for a root item, or the ID of the parent item if this is a child item (returned by the <code>NewOutline</code> function). Alternatively, use the <code>GetOutlineID</code> function to get a valid outline ID.</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the outline item.</td>
</tr>
<tr>
<td>DestPage</td>
<td>The destination page number that this outline item links to</td>
</tr>
<tr>
<td>DestPosition</td>
<td>The vertical position on the destination page to link to</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The item could not be added</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the item which was added successfully</td>
</tr>
</tbody>
</table>
NewPage

Page manipulation

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Create a new page. The new page is added to the end of the document, and will have the same width and height as the selected page.

Syntax

Dylib

```c
int DPLNewPage(int InstanceID);
```

Objective-C class

```c
- (int)NewPage
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page could not be added. This should never occur.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The page number of the page that was added</td>
</tr>
</tbody>
</table>
**NewPageFromCanvasDC**

**Vector graphics, Page manipulation**

**Description**

Adds a new page to the selected document from the drawing operations applied to the DC returned by the GetCanvasDC function.

When the Options parameter is set to 3 or 4, use the NoEmbedFontListAdd function to add fonts to the no embed font list.

**Syntax**

**Dylib**

```c
int DPLNewPageFromCanvasDC(int InstanceID, double DPI, int Options);
```

**Objective-C class**

```objective-c
- (int)NewPageFromCanvasDC:(double)DPI :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>DPI</th>
<th>The DPI to use for the new document. For example, if the canvas was created with a width and height of 96 and the DPI is specified as 192, the resulting document will be 0.5 inches in width and height.</th>
</tr>
</thead>
</table>
| Options  | -1 = Convert the drawing commands to a single image using GDI+  
0 = Process the drawing commands as vector graphics, fonts are not embedded  
1 = Process the drawing commands as vector graphics, fonts are embedded but not compressed  
2 = Process the drawing commands as vector graphics, fonts are embedded and compressed  
3 = Process the drawing commands as vector graphics, fonts not in the no embed font list are embedded and compressed  
4 = Same as 3 but fonts already added during previous calls to this function or the LoadFromCanvasDC function are reused |

**Return values**

| 0 | A canvas has not been created |
| 1 | The canvas DC was processed correctly and a new document has been created |
NewPages

Page manipulation

Description

This function is similar to the NewPage function, but allows you to add more than one new page to the selected document.

Syntax

Dylib

int DPLNewPages(int InstanceID, int PageCount);

Objective-C class

- (int)NewPages:(int)PageCount;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageCount</td>
<td>The number of pages to add to the document</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The pages could not be added. This should never occur.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The total number of pages in the document after the new pages were added</td>
</tr>
</tbody>
</table>
**NewPostScriptXObject**

**Description**

Adds a PostScript XObject to the document. If the PostScript XObject is drawn onto the page with the `DrawPostScriptXObject` function the contents of the PostScript XObject will be placed into the generated PostScript for the page when printed to a PostScript printer.

**Syntax**

**Dylib**

```c
int DPLNewPostScriptXObject(int InstanceID, wchar_t * PS);
```

**Objective-C class**

```c
-(int)NewPostScriptXObject:(NSString *)PS;
```

**Parameters**

| PS  | The PostScript that will be inserted |

**Return values**

| 0   | The PostScript XObject could not be added |
| Non-zero | A reference to the PostScript XObject which can be used with the `DrawPostScriptXObject` function |
NewRGBAxialShader

Vector graphics, Color

Description

This function adds an axial shader to the current document. The color is varied linearly from one color to another between two points and is used for linear gradient fills.

The shader can be used with the SetTextShader, SetLineShader and SetFillShader functions to set the color of subsequently drawn vector graphics and text.

Syntax

Dylib

```c
int DPLNewRGBAxialShader(int InstanceID, wchar_t * ShaderName,
                        double StartX, double StartY, double StartRed,
                        double StartGreen, double StartBlue, double EndX, double EndY,
                        double EndRed, double EndGreen, double EndBlue, int Extend);
```

Objective-C class

```objective-c
- (int)NewRGBAxialShader:(NSString *)ShaderName :(double)StartX
   :(double)StartY :(double)StartRed :(double)StartGreen
   :(double)StartBlue :(double)EndX :(double)EndY :(double)EndRed
   :(double)EndGreen :(double)EndBlue :(int)Extend;
```

Parameters

<table>
<thead>
<tr>
<th>ShaderName</th>
<th>The name of the shader. Should be a simple string consisting of alphanumeric characters and no whitespace. This name is used with the SetTextShader, SetLineShader and SetFillShader functions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartX</td>
<td>The horizontal co-ordinate of the start point</td>
</tr>
<tr>
<td>StartY</td>
<td>The vertical co-ordinate of the start point</td>
</tr>
<tr>
<td>StartRed</td>
<td>The red component of the start color</td>
</tr>
<tr>
<td>StartGreen</td>
<td>The green component of the start color</td>
</tr>
<tr>
<td>StartBlue</td>
<td>The blue component of the start color</td>
</tr>
<tr>
<td>EndX</td>
<td>The horizontal co-ordinate of the end point</td>
</tr>
<tr>
<td>EndY</td>
<td>The vertical co-ordinate of the end point</td>
</tr>
<tr>
<td>EndRed</td>
<td>The red component of the end color</td>
</tr>
<tr>
<td>EndGreen</td>
<td>The green component of the end color</td>
</tr>
<tr>
<td>EndBlue</td>
<td>The blue component of the end color</td>
</tr>
</tbody>
</table>
| Extend     | 0 = do not extend the beyond the start and end points  
   1 = extend the shader using solid color                                                                                      |

Return values

| 0          | The shader could not be added, possibly a shader with this name has already been added                                                                                           |
| 1          | The shader was added successfully                                                                                                                                                    |
NewSignProcessFromFile

Security and Signatures

Description

Creates a new digital signature process using a file as the source document.

Syntax

**Dylib**

```c
int DPLNewSignProcessFromFile(int InstanceID, wchar_t * InputFile, wchar_t * Password);
```

**Objective-C class**

```c
- (int)NewSignProcessFromFile:(NSString *)InputFile :(NSString *)Password;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputFile</strong></td>
<td>The path and name of the file to sign</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>The password to open the PDF, if any</td>
</tr>
</tbody>
</table>
Description

Creates a new digital signature process using a string of 8-bit bytes as the source.

Syntax

**Dylib**

```c
int DPLNewSignProcessFromString(int InstanceID, char * Source, wchar_t * Password);
```

**Objective-C class**

```objc
- (int)NewSignProcessFromString:(NSData *)Source :(NSString *)Password;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>A string containing the document to be signed</td>
</tr>
<tr>
<td>Password</td>
<td>The password to open the PDF, if any</td>
</tr>
</tbody>
</table>
NewStaticOutline

Outlines

Description

This function creates a new outline without an action. The action can later be set using the `SetOutlineDestination`, `SetOutlineWebLink` or `SetOutlineJavaScript` functions.

Syntax

**Dylib**

```c
int DPLNewStaticOutline(int InstanceID, int Parent, wchar_t * Title);
```

**Objective-C class**

```objective-c
- (int)NewStaticOutline:(int)Parent :(NSString *)Title;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>0 for a root item, or the ID of the parent item if this is a child item</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the outline item.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The outline item could not be added</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the outline item that was added</td>
</tr>
</tbody>
</table>
NewTilingPatternFromCapturedPage

Vector graphics, Color

Description

This function converts a captured page into a tiling pattern and adds the pattern to the current document.

The pattern can be used with the SetFillTilingPattern function to set the color of subsequently drawn vector graphics.

Syntax

**Dylib**

```
int DPLNewTilingPatternFromCapturedPage(int InstanceID,
wchar_t * PatternName, int CaptureID);
```

**Objective-C class**

```
- (int)NewTilingPatternFromCapturedPage:(NSString *)PatternName
:(int)CaptureID;
```

Parameters

<table>
<thead>
<tr>
<th>PatternName</th>
<th>The name of the tiling pattern. Should be a simple string consisting of alphanumeric characters and no whitespace. This name is used with the SetFillTilingPattern function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaptureID</td>
<td>The ID returned by the CapturePage or CapturePageEx functions.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The captured page could not be converted into a tiling pattern. The CaptureID parameter might be invalid or the PatternName has already been used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
NoEmbedFontListAdd
Vector graphics, Fonts, Miscellaneous functions

Description
 Adds a font name to the no embed font list used by the LoadFromCanvasDC and NewPageFromCanvasDC functions.

Syntax

Dylib

```c
int DPLNoEmbedFontListAdd(int InstanceID, wchar_t * FontName);
```

Objective-C class

```c
-(int)NoEmbedFontListAdd:(NSString *)FontName;
```

Parameters

| FontName | The font name to add to the list |

Return values

| 0 | The font name is already in the list |
| 1 | The font name was added to the list successfully |
NoEmbedFontListCount

Vector graphics, Fonts, Miscellaneous functions

Description

Returns the number of font names in the no embed font list used by the LoadFromCanvasDC and NewPageFromCanvasDC functions.

Syntax

Dylib

```c
int DPLNoEmbedFontListCount(int InstanceID);
```

Objective-C class

```objective-c
-(int)NoEmbedFontListCount
```
NoEmbedFontListGet

Vector graphics, Fonts, Miscellaneous functions

Description

Returns the font name at the specified index in the no embed font list used by the LoadFromCanvasDC and NewPageFromCanvasDC functions.

Syntax

**Dylib**

```c
wchar_t* DPLNoEmbedFontListGet(int InstanceID, int Index);
```

**Objective-C class**

```objc
- (NSString *)NoEmbedFontListGet:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the font name in the list. The first name has an Index value of 1.</th>
</tr>
</thead>
</table>
NoEmbedFontListRemoveAll

Vector graphics, Fonts, Miscellaneous functions

Description

Removes all the font names from the no embed font list used by the LoadFromCanvasDC and NewPageFromCanvasDC functions.

Syntax

**Dylib**

```c
int DPLNoEmbedFontListRemoveAll(int InstanceID);
```

**Objective-C class**

```objc
- (int)NoEmbedFontListRemoveAll
```
NoEmbedFontListRemoveIndex

Vector graphics, Fonts, Miscellaneous functions

Description

Removes the font name at the specified index from the no embed font list used by the LoadFromCanvasDC and NewPageFromCanvasDC functions.

Syntax

Dylib

int DPLNoEmbedFontListRemoveIndex(int InstanceID, int Index);

Objective-C class

- (int)NoEmbedFontListRemoveIndex:(int)Index;

Parameters

| Index | The index of the font name in the list. The first name has an Index value of 1. |

Return values

| 0     | The specified index was out of range |
| 1     | The font name was successfully removed from the list |
NoEmbedFontListRemoveName

Description

Removes the specified font name from the no embed font list used by the `LoadFromCanvasDC` and `NewPageFromCanvasDC` functions.

Syntax

**Dylib**

```c
int DPLNoEmbedFontListRemoveName(int InstanceID, wchar_t * FontName);
```

**Objective-C class**

```objective-c
- (int)NoEmbedFontListRemoveName:(NSString *)FontName;
```

Parameters

| FontName | The font name to remove from the list |

Return values

| 0 | The specified font name was not found in the list |
| 1 | The font name was successfully removed from the list |
NormalizePage

Page manipulation

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Moves and/or rotates the contents of the page so that subsequent drawing operations are at the expected position on the page. All the page boundary boxes are adjusted to the physical size of the page and the page's rotation attribute is reset to zero.

**Syntax**

**Dylib**

```c
int DPLNormalizePage(int InstanceID, int NormalizeOptions);
```

**Objective-C class**

```c
-(int)NormalizePage:(int)NormalizeOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>NormalizeOptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Standard normalization</td>
</tr>
<tr>
<td>1</td>
<td>Normalize and also balance the graphics state stack</td>
</tr>
<tr>
<td>2</td>
<td>Maintain existing page structure</td>
</tr>
<tr>
<td>3</td>
<td>Maintain existing page structure and balance the stack</td>
</tr>
</tbody>
</table>
OpenOutline

Outlines

Description

Expands an outline item (bookmark).

Syntax

Dylib

```
int DPLOpenOutline(int InstanceID, int OutlineID);
```

Objective-C class

```
- (int)OpenOutline:(int)OutlineID;
```

Parameters

| OutlineID | The ID of the outline item to work with. This ID is returned by the NewOutline or NewStaticOutline functions, or retrieved with the GetOutlineID function or Get*Outline functions. |

Return values

| 0          | The Outline ID provided was invalid |
| 1          | The outline item was expanded       |
OptionalContentGroupCount

Content Streams and Optional Content Groups

Description

Returns the number of optional content groups in the selected document.

Syntax

**Dylib**

```c
int DPLOptionalContentGroupCount(int InstanceID);
```

**Objective-C class**

```objc
-(int)OptionalContentGroupCount
```
OutlineCount

Outlines

Description

Returns the number of outline items (bookmarks) in the selected document.

Syntax

Dylib

```c
int DPLOutlineCount(int InstanceID);
```

Objective-C class

```c
- (int)OutlineCount
```
OutlineTitle

Outlines

Description

Returns the title of an outline item (bookmark).

Syntax

Dylib

wchar_t * DPLOutlineTitle(int InstanceID, int OutlineID);

Objective-C class

- (NSString *)OutlineTitle:(int)OutlineID;

Parameters

| OutlineID | The ID of the outline item to work with. This ID is returned by the NewOutline or NewStaticOutline functions, or retrieved with the GetOutlineID function or Get*Outline functions. |
PageCount

Document properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Reports the total number of pages in the selected document.

Syntax

Dylib

```c
int DPLPageCount(int InstanceID);
```

Objective-C class

- (int)PageCount
**PageHasFontResources**

**Page properties**

**Description**

Analyses the specified page to identify font resources.

**Syntax**

**Dylib**

```c
int DPLPageHasFontResources(int InstanceID, int PageNumber);
```

**Objective-C class**

```objc
- (int)PageHasFontResources:(int)PageNumber;
```

**Parameters**

<table>
<thead>
<tr>
<th>PageNumber</th>
<th>The number of the page to analyse</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified page does not have font resources</td>
</tr>
<tr>
<td>1</td>
<td>The specified page has at least one font resource</td>
</tr>
</tbody>
</table>
**PageHeight**

**Page properties**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Returns the height of the selected page.

**Syntax**

**Dylib**

```c
double DPLPageHeight(int InstanceID);
```

**Objective-C class**

```objc
- (double)PageHeight
```

**Return values**

The height of the selected page (in points, millimetres or inches)
Page JavaScript Action

**JavaScript, Page properties**

**Description**

This function is used to add JavaScript to a page open or page close event.

**Syntax**

**Dylib**

```c
int DPLPageJavaScriptAction(int InstanceID, wchar_t * ActionType, wchar_t * JavaScript);
```

**Objective-C class**

```objective-c
- (int)PageJavaScriptAction:(NSString *)ActionType :(NSString *)JavaScript;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>ActionType</strong></th>
<th>The event to add the JavaScript to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;O&quot; = (capital letter O) This event occurs when the page is opened</td>
<td></td>
</tr>
<tr>
<td>&quot;C&quot; = This event occurs when the page is closed</td>
<td></td>
</tr>
</tbody>
</table>

| **JavaScript** | This is the JavaScript to execute when the event occurs. |

**Return values**

<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified ActionType was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The JavaScript was added successfully</td>
</tr>
</tbody>
</table>
PageRotation

Page properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Returns the rotation of the selected page. This value should always be a multiple of 90 degrees.

Syntax

Dylib

    int DPLPageRotation(int InstanceID);

Objective-C class

    - (int)PageRotation
**Description**

Returns the width of the selected page.

**Syntax**

**Dylib**

```c
double DPLPageWidth(int InstanceID);
```

**Objective-C class**

```c
- (double)PageWidth
```

**Return values**

The width of the selected page (in points, millimetres or inches)
PrintDocument

Rendering and printing

Description

Renders certain pages from the selected document to the specified printer.

Syntax

Dylib

```c
int DPLPrintDocument(int InstanceID, wchar_t * PrinterName, int StartPage,
int EndPage, int Options);
```

Objective-C class

```objective-c
- (int)PrintDocument:(NSString *)PrinterName :(int)StartPage :(int)EndPage
:(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrinterName</td>
<td>The name of the printer to use for printing. This is the name that appears in the Windows Print Manager. Use the <code>GetPrinterNames</code> function to return a list of valid printers on the system. A value returned by the <code>NewCustomPrinter</code> function can also be used here.</td>
</tr>
<tr>
<td>StartPage</td>
<td>The first page to print</td>
</tr>
<tr>
<td>EndPage</td>
<td>The last page to print</td>
</tr>
<tr>
<td>Options</td>
<td>Use the <code>PrintOptions</code> function to obtain a value for this parameter</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The pages could not be printed, usually caused by the StartPage and EndPage parameters being out of range</td>
</tr>
<tr>
<td>1</td>
<td>The pages were printed successfully</td>
</tr>
</tbody>
</table>
PrintDocumentToFile

Rendering and printing

Description

Renders certain pages from the selected document to the specified printer. The print output is directed to the specified spool file.

Not all printer drivers support the DocInfo.lpszOutput field so results may vary.

Syntax

Dylib

```c
int DPLPrintDocumentToFile(int InstanceID, wchar_t * PrinterName,
                         int StartPage, int EndPage, int Options, wchar_t * FileName);
```

Objective-C class

```objc
- (int)PrintDocumentToFile:(NSString *)PrinterName :(int)StartPage
               :(int)EndPage :(int)Options :(NSString *)FileName;
```

Parameters

- **PrinterName** The name of the printer to use for printing. This is the name that appears in the Windows Print Manager. Use the `GetPrinterNames` function to return a list of valid printers on the system. A value returned by the `NewCustomPrinter` function can also be used here.
- **StartPage** The first page to print
- **EndPage** The last page to print
- **Options** Use the `PrintOptions` function to obtain a value for this parameter
- **FileName** The file name where print output should be spooled to.
PrintOptions
Rendering and printing

Description
This function is used to construct a value that can be used as the Options parameter to the PrintDocument function.

Syntax

Dylib

```c
    int DPLPrintOptions(int InstanceID, int PageScaling, int AutoRotateCenter, wchar_t * Title);
```

Objective-C class

```objective-c
- (int)PrintOptions:(int)PageScaling :(int)AutoRotateCenter :(NSString *)Title;
```

Parameters

| PageScaling     | 0 = None  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Fit to paper</td>
</tr>
<tr>
<td></td>
<td>2 = Shrink large pages</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AutoRotateCenter</th>
<th>0 = Do not rotate pages automatically</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Rotate pages to fit on the output medium, and center on the page</td>
</tr>
</tbody>
</table>

| Title            | The title of the document. This title is used by Windows in the Print Manager and for network title pages |
**PrintPages**

**Rendering and printing**

**Description**

Renders a page range list from the selected document to the specified printer.

**Syntax**

**Dylib**

```c
int DPLPrintPages(int InstanceID, wchar_t * PrinterName,
                  wchar_t * PageRanges, int Options);
```

**Objective-C class**

```objc
- (int)PrintPages:(NSString *)PrinterName :(NSString *)PageRanges
               :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PrinterName</strong></td>
<td>The name of the printer to use for printing. This is the name that appears in the Windows Print Manager. Use the <code>GetPrinterNames</code> function to return a list of valid printers on the system. A value returned by the <code>NewCustomPrinter</code> function can also be used here.</td>
</tr>
<tr>
<td><strong>PageRanges</strong></td>
<td>A list of pages to print, for example &quot;1-10,12,14&quot;</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Use the <code>PrintOptions</code> function to obtain a value for this parameter</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An error occurred</td>
</tr>
<tr>
<td>1</td>
<td>The pages were printed successfully</td>
</tr>
</tbody>
</table>
PrintPagesToFile

Rendering and printing

Description

Renders a list of page ranges from the selected document to the specified printer. The print output is directed to the specified spool file.

Not all printer drivers support the DocInfo.lpszOutput field so results may vary.

Syntax

Dylib

```c
int DPLPrintPagesToFile(int InstanceID, wchar_t * PrinterName,
                         wchar_t * PageRanges, int Options, wchar_t * FileName);
```

Objective-C class

```objective-c
- (int)PrintPagesToFile:(NSString *)PrinterName :(NSString *)PageRanges
                :(int)Options :(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrinterName</td>
<td>The name of the printer to use for printing. This is the name that appears in the Windows Print Manager. Use the <code>GetPrinterNames</code> function to return a list of valid printers on the system. A value returned by the <code>NewCustomPrinter</code> function can also be used here.</td>
</tr>
<tr>
<td>PageRanges</td>
<td>A list of pages to print, for example &quot;1-10,12,14&quot;</td>
</tr>
<tr>
<td>Options</td>
<td>Use the <code>PrintOptions</code> function to obtain a value for this parameter</td>
</tr>
<tr>
<td>FileName</td>
<td>Use the <code>PrintOptions</code> function to obtain a value for this parameter</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An error occurred</td>
</tr>
<tr>
<td>1</td>
<td>The pages were printed successfully</td>
</tr>
</tbody>
</table>
ReleaseBuffer

Miscellaneous functions

Description

Releases a buffer created with the CreateBuffer function.

Syntax

Dylib

```c
int DPLReleaseBuffer(int InstanceID, char * Buffer);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>A value returned from the CreateBuffer function</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The InstanceID was invalid, or the Buffer has already been released or is invalid</td>
</tr>
<tr>
<td>1</td>
<td>The buffer was released successfully</td>
</tr>
</tbody>
</table>
ReleaseImage

Image handling

Description

Releases the temporary memory used by an image that was added to the PDF after the document was opened (using functions such as AddImageFromFile) or an image that was found using the FindImages function.

Releasing the image does not affect the PDF itself, images that have already been drawn onto the page will not be removed.

After the image has been released the ImageID is no longer valid and cannot be used with functions such as SelectImage.

Syntax

Dylib

```c
int DPLReleaseImage(int InstanceID, int ImageID);
```

Objective-C class

```objc
- (int)ReleaseImage:(int)ImageID;
```

Parameters

| ImageID | The ID of the image to release |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image could not be released. The ImageID parameter could be invalid or the ImageID doesn't reference an image contained in the selected document.</td>
</tr>
<tr>
<td>1</td>
<td>The image was released successfully.</td>
</tr>
</tbody>
</table>
ReleaseImageList

Image handling, Page properties

Description

Releases the specified image list including all the image data extracted from the images in the list. Releasing the image list does not affect the original images.

Syntax

Dylib

```c
int DPLReleaseImageList(int InstanceID, int ImageListID);
```

Objective-C class

```objc
- (int)ReleaseImageList:(int)ImageListID;
```

Parameters

| ImageListID | A value returned by the GetPageImageList function |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image list could not be released. The ImageListID parameter could be invalid or the ImageListID doesn't reference an image list from the selected document.</td>
</tr>
<tr>
<td>1</td>
<td>The image list was released successfully.</td>
</tr>
</tbody>
</table>
ReleaseLibrary

Miscellaneous functions

Description

Frees the object created with the CreateLibrary function.

Syntax

Dylib

```c
int DPLReleaseLibrary(int InstanceID);
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The library could not be released. The InstanceID value may be incorrect.</td>
</tr>
<tr>
<td>1</td>
<td>The library was released successfully</td>
</tr>
</tbody>
</table>
ReleaseSignProcess

Security and Signatures

Description

Releases a signature process from memory.

Syntax

Dylib

int DPLReleaseSignProcess(int InstanceID, int SignProcessID);

Objective-C class

- (int)ReleaseSignProcess:(int)SignProcessID;

Parameters

<table>
<thead>
<tr>
<th>SignProcessID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A value returned by the NewSignProcessFromFile, NewSignProcessFromStream or NewSignProcessFromString functions.</td>
<td></td>
</tr>
</tbody>
</table>

Return values

| 0 | Invalid SignProcessID |
| 1 | Successfully deleted the signing process |
ReleaseStringList

**Description**

Releases the specified string list.

**Syntax**

**Dylib**

```c
int DPLReleaseStringList(int InstanceID, int StringListID);
```

**Objective-C class**

```objc
-(int)ReleaseStringList:(int)StringListID;
```

**Parameters**

| StringListID | The ID of the string list as returned by the CheckFileCompliance function. |

**Return values**

| 0 | The string list could not be released, the StringListID parameter is invalid. |
| 1 | Success |
ReleaseTextBlocks

Text, Extraction

Description

Releases the memory used by a text block list.

Syntax

Dylib

```c
int DPLReleaseTextBlocks(int InstanceID, int TextBlockListID);
```

Objective-C class

```objective-c
- (int)ReleaseTextBlocks:(int)TextBlockListID;
```

Parameters

<table>
<thead>
<tr>
<th>TextBlockListID</th>
<th>A value returned by the ExtractPageTextBlocks function</th>
</tr>
</thead>
</table>

RemoveAppearanceStream

**Form fields**

**Description**

Removes the appearance stream of the specified form field.

**Syntax**

**Dylib**

```c
int DPLRemoveAppearanceStream(int InstanceID, int Index);
```

**Objective-C class**

```c
- (int)RemoveAppearanceStream:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The index of the form field to work with. The first form field has an index of 1.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The appearance stream of the form field was removed successfully</td>
</tr>
</tbody>
</table>
RemoveCustomInformation

**Document properties**

**Description**

Removes a custom metadata item from the document.

**Syntax**

**Dylib**

```c
int DPLRemoveCustomInformation(int InstanceID, wchar_t * Key);
```

**Objective-C class**

```objective-c
-(int)RemoveCustomInformation:(NSString *)Key;
```

**Parameters**

<table>
<thead>
<tr>
<th>Key</th>
<th>Specifies which key to remove</th>
</tr>
</thead>
</table>
RemoveDocument

Document management

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Removes the specified document, freeing up memory.
Quick PDF Library will always ensure that there is at least one document loaded at all times.
In version 7.18 and earlier, it was only possible to remove a document if there were at least two documents loaded.
From version 7.19 this function will always succeed. If the specified document was the only loaded document it will be removed and replaced with a new blank document.

Syntax

Dylib

int DPLRemoveDocument(int InstanceID, int DocumentID);

Objective-C class

- (int)RemoveDocument:(int)DocumentID;

Parameters

| DocumentID | The ID of the document to remove |

Return values

| 0           | The specified document does not exist or could not be removed. |
| 1           | The specified document was removed successfully |
**RemoveEmbeddedFile**

**Description**

Removes the specified embedded file from the document.

**Syntax**

**Dylib**

```c
int DPLRemoveEmbeddedFile(int InstanceID, int Index);
```

**Objective-C class**

```objc
- (int)RemoveEmbeddedFile:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the embedded file. Must be a value between 1 and the value returned by <code>EmbeddedFileCount</code>.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The embedded file could not be removed.</td>
</tr>
<tr>
<td>1</td>
<td>The embedded file was successfully removed from the document.</td>
</tr>
</tbody>
</table>
RemoveFormFieldBackgroundColor

**Form fields**

**Description**

Removes the form field's background color entry

**Syntax**

**Dylib**

```c
int DPLRemoveFormFieldBackgroundColor(int InstanceID, int Index);
```

**Objective-C class**

```c
- (int)RemoveFormFieldBackgroundColor:(int)Index;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The Index parameter was incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
RemoveFormFieldBorderColor

Form fields

Description

Removes the form field's border color entry

Syntax

**Dylib**

```c
int DPLRemoveFormFieldBorderColor(int InstanceID, int Index);
```

**Objective-C class**

```c
- (int)RemoveFormFieldBorderColor:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The Index parameter was incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
**RemoveFormFieldChoiceSub**

**Form fields**

**Description**

Removes a subname entry from a choice based form field.

**Syntax**

**Dylib**

```c
int DPLRemoveFormFieldChoiceSub(int InstanceID, int Index,
                                wchar_t * Subname);
```

**Objective-C class**

```objective-c
- (int)RemoveFormFieldChoiceSub:(int)Index :(NSString *)Subname;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field</td>
</tr>
<tr>
<td><strong>Subname</strong></td>
<td>The string value of the subname to delete</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The subname was not deleted. The specified form field may not have been a choice form field.</td>
</tr>
<tr>
<td>1</td>
<td>The subname was successfully deleted</td>
</tr>
</tbody>
</table>
RemoveGlobalJavaScript

Document properties, JavaScript

Description

Removes a block of JavaScript from the global JavaScript store.

Syntax

**Dylib**

```
int DPLRemoveGlobalJavaScript(int InstanceID, wchar_t * PackageName);
```

**Objective-C class**

```
-(int)RemoveGlobalJavaScript:(NSString *)PackageName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PackageName</strong></td>
<td>The name that that JavaScript was stored under.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified package name could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The JavaScript was removed successfully</td>
</tr>
</tbody>
</table>
**RemoveOpenAction**

**Document properties**

**Description**

Removes any open action from the document.

**Syntax**

**Dylib**

```c
int DPLRemoveOpenAction(int InstanceID);
```

**Objective-C class**

```objective-c
-(int)RemoveOpenAction
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An unexpected error occurred</td>
</tr>
<tr>
<td>1</td>
<td>The open action, if any, was removed from the document successfully</td>
</tr>
</tbody>
</table>
RemoveOutline

Outlines

Description

Removes an outline from the document.

Syntax

Dylib

```c
int DPLRemoveOutline(int InstanceID, int OutlineID);
```

Objective-C class

```objective-c
- (int)RemoveOutline:(int)OutlineID;
```

Parameters

| OutlineID | The ID of the outline item to work with. This ID is returned by the NewOutline or NewStaticOutline functions, or retrieved with the GetOutlineID function or Get*Outline functions. |

Return values

| 0 | The Outline ID provided was invalid |
| 1 | The outline was removed successfully |
RemoveSharedContentStreams

Content Streams and Optional Content Groups

Description

This function ensures that none of the pages in the selected document have shared content streams. This is necessary before imposing a document with the CapturePage or CapturePageEx functions.

Syntax

Dylib

int DPLRemoveSharedContentStreams(int InstanceID);

Objective-C class

- (int)RemoveSharedContentStreams
**RemoveStyle**

**Text**

**Description**

Removes a style that was previously saved using the **SaveStyle** function. The style name is case sensitive, it must exactly match the style name used with the **SaveStyle** function.

**Syntax**

**Dylib**

```c
int DPLRemoveStyle(int InstanceID, wchar_t * StyleName);
```

**Objective-C class**

```objc
- (int)RemoveStyle:(NSString *)StyleName;
```

**Parameters**

| **StyleName** | The name to associate with the style. This name is case sensitive. |

**Return values**

| **0** | The specified StyleName could not be found |
| **1** | The style was removed successfully |
RemoveUsageRights

Document manipulation, Document properties

Description

Removes any usage rights from the document.

Syntax

Dylib

```c
int DPLRemoveUsageRights(int InstanceID);
```

Objective-C class

```c
- (int)RemoveUsageRights
```

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Usage rights were not found in the document.</td>
</tr>
<tr>
<td>1</td>
<td>Usage rights were successfully removed from the document.</td>
</tr>
</tbody>
</table>
**RemoveXFAEntries**

**Document properties, Form fields**

**Description**

Removes the XFA form field entry from the document's form.

**Syntax**

**Dylib**

```c
int DPLRemoveXFAEntries(int InstanceID, int Options);
```

**Objective-C class**

```objective-c
- (int)RemoveXFAEntries:(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Options</th>
<th>Reserved for future use, should be set to 0.</th>
</tr>
</thead>
</table>

RenderAsMultipageTIFFToFile

Image handling, Rendering and printing

Description

Renders the specified pages from the selected document to a multi-page TIFF file. ImageOptions 1, TIFF (G4) output, is only available on Windows Vista and Windows Server 2008 and later.

Syntax

Dylib

```c
int DPLRenderAsMultipageTIFFToFile(int InstanceID, double DPI,wchar_t * PageRanges, int ImageOptions, int OutputOptions,wchar_t * FileName);
```

Objective-C class

```objective-c
-(int)RenderAsMultipageTIFFToFile:(double)DPI :(NSString *)PageRanges:(int)ImageOptions :(int)OutputOptions :(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPI</td>
<td>The DPI to render the pages at</td>
</tr>
<tr>
<td>PageRanges</td>
<td>A list of pages to render, for example &quot;5-10,3,12&quot;.</td>
</tr>
<tr>
<td>ImageOptions</td>
<td>0=24-bit RGB TIFF</td>
</tr>
<tr>
<td></td>
<td>1=1-bit G4 TIFF</td>
</tr>
<tr>
<td>OutputOptions</td>
<td>Reserved for future use, should be set to 0.</td>
</tr>
<tr>
<td>FileName</td>
<td>The file name and path of the TIFF file to create</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalid parameters or cannot create file</td>
</tr>
<tr>
<td>1</td>
<td>The multipage TIFF was created successfully</td>
</tr>
</tbody>
</table>
RenderDocumentToFile
Rendering and printing

Description
Renders certain pages from the selected document to an image file on disk.

Syntax
Dylib

```c
int DPLRenderDocumentToFile(int InstanceID, double DPI, int StartPage,
int EndPage, int Options, wchar_t * FileName);
```

Objective-C class

```c
- (int)RenderDocumentToFile:(double)DPI :(int)StartPage :(int)EndPage
:(int)Options :(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPI</td>
<td>The DPI to use for the rendering. A value of 72 will give the same result as Acrobat when the zoom level is 100%.</td>
</tr>
<tr>
<td>StartPage</td>
<td>The first page to print</td>
</tr>
<tr>
<td>EndPage</td>
<td>The last page to print</td>
</tr>
</tbody>
</table>
| Options   | 0 = BMP output  
1 = JPEG output  
2 = WMF output  
3 = EMF output  
4 = EPS output  
5 = PNG output  
6 = GIF output  
7 = TIFF output  
8 = EMF+ output  
9 = HTML5 output  
10 = G4 TIFF output |
| FileName  | The path and filename to use for the file. Each page will be stored in a separate file. If this parameter contains "%p" this will be replaced by the page number, otherwise the page number will be appended to the end of the filename before the extension. For example, if FileName is "output.jpg" and page 10 is rendered the image will be stored in a file called "output10.jpg". If FileName is "page%poutput.bmp" and page 5 is rendered the image will be stored in a file called "page5output.bmp". |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The pages were not rendered successfully. This is usually caused by the StartPage or EndPage parameters being out of range.</td>
</tr>
<tr>
<td>1</td>
<td>The pages were rendered successfully</td>
</tr>
</tbody>
</table>
RenderPageToDC

Rendering and printing

Description

This function renders a page from the selected document directly onto a graphics surface.

Syntax

Dylib

int DPLRenderPageToDC(int InstanceID, double DPI, int Page, int DC);

Objective-C class

- (int)RenderPageToDC:(double)DPI :(int)Page :(int)DC;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPI</td>
<td>The DPI to use when rendering the page</td>
</tr>
<tr>
<td>Page</td>
<td>The page number to render</td>
</tr>
<tr>
<td>DC</td>
<td>The device context handle</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Page could not be rendered</td>
</tr>
<tr>
<td>1</td>
<td>Page was rendered successfully</td>
</tr>
</tbody>
</table>
**RenderPageToFile**

**Rendering and printing**

**Description**

This function renders a page from the selected document to a file on disk. The data written to disk depends on the Options parameter.

**Syntax**

**Dylib**

```c
int DPLRenderPageToFile(int InstanceID, double DPI, int Page, int Options, wchar_t * FileName);
```

**Objective-C class**

```objective-c
- (int)RenderPageToFile:(double)DPI :(int)Page :(int)Options :(NSString *)FileName;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPI</td>
<td>The DPI to use when rendering the page. Values over 300 will cause excessive memory usage.</td>
</tr>
<tr>
<td>Page</td>
<td>The page number to render</td>
</tr>
</tbody>
</table>
| Options   | 0 = BMP output  
            1 = JPEG output  
            2 = WMF output  
            3 = EMF output  
            4 = EPS output  
            5 = PNG output  
            6 = GIF output  
            7 = TIFF (LZW) output  
            8 = EMF+ output  
            9 = HTML5 output  
            10 = TIFF (G4) output |
| FileName  | The path and file name of the file to create to store the rendered page image data in. |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page could not be rendered</td>
</tr>
<tr>
<td>1</td>
<td>The page was rendered correctly and the image file was saved to disk</td>
</tr>
<tr>
<td>2</td>
<td>The file could not be written to disk</td>
</tr>
</tbody>
</table>
RenderPageToString

Rendering and printing

Description

This function renders a page from the selected document to a string. The data in the returned string depends on the Options parameter.

Syntax

**Dylib**

```c
char * DPLRenderPageToString(int InstanceID, double DPI, int Page, int Options);
```

**Objective-C class**

```c
- (NSData *)RenderPageToString:(double)DPI :(int)Page :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPI</td>
<td>The DPI to use when rendering the page. Values over 300 will cause excessive memory usage.</td>
</tr>
<tr>
<td>Page</td>
<td>The page number to render</td>
</tr>
</tbody>
</table>
| Options   | 0 = BMP output  
1 = JPEG output  
2 = WMF output  
3 = EMF output  
4 = EPS output  
5 = PNG output  
6 = GIF output  
7 = TIFF output  
8 = EMF+ output  
9 = HTML5 output  
10 = TIFF (G4) output |
**ReplaceFonts**

*Fonts, Document manipulation*

**Description**

Replaces embedded fonts with equivalent standard fonts, reducing the file size. In version 9.11 and earlier, only Courier and Courier-Bold were replaced. As of version 9.12, the following fonts are replaced:

- Courier
- Courier-Bold
- Courier-BoldOblique
- Courier-Oblique
- Helvetica
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Oblique
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Symbol
- ZapfDingbats

**Syntax**

**Dylib**

```c
int DPLReplaceFonts(int InstanceID);
```

**Objective-C class**

```c
- (int)ReplaceFonts
```
ReplaceImage

Image handling, Page layout

Description

Replaces an image on the selected page with another image. The original image is not removed from the document and can be reused. If the original image is no longer needed it can be cleared using the ClearImage function.

Syntax

Dylib

```c
int DPLReplaceImage(int InstanceID, int OriginalImageID, int NewImageID);
```

Objective-C class

```c
- (int)ReplaceImage:(int)OriginalImageID :(int)NewImageID;
```

Parameters

<table>
<thead>
<tr>
<th>OriginalImageID</th>
<th>The ImageID of the image to be replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewImageID</td>
<td>The ImageID of the image to replace the existing image</td>
</tr>
</tbody>
</table>
ReplaceTag
Page manipulation

Description
This function searches through the contents of the current page, and replaces all occurrences of Tag with NewValue.

Syntax

Dylib

```c
int DPLReplaceTag(int InstanceID, wchar_t * Tag, wchar_t * NewValue);
```

Objective-C class

```objc
- (int)ReplaceTag:(NSString *)Tag :(NSString *)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Tag</th>
<th>The text to search for</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewValue</td>
<td>The replacement text</td>
</tr>
</tbody>
</table>

Return values

Returns the number of times the text was replaced
Description

Use this function to activate an alternative printing system that allows the printer status to be returned. Many of
the status codes returned are supplied by the printer driver there is no guarantee that values will contain
meaningful information for all printers.

The first step is to call this function with StatusCommand=101 to enable printer status monitoring. Optionally, the
print job can be started in the paused state by calling this function again with StatusCommand=103. This might be
necessary for small print jobs that would otherwise finish before the status can be read.

The print job can then be started as usual with one of the printing functions: PrintDocument, PrintDocumentToFile or PrintDocumentToPrinterObject.

Once the print job has started, this function can be called again repeatedly to obtain the printer status for the print
job over time. If the print job was started in the paused state, actual printing will only begin once this function is
called with StatusCommand=402.

Syntax

Dylib

```c
int DPLRequestPrinterStatus(int InstanceID, int StatusCommand);
```

Objective-C class

```c
- (int)RequestPrinterStatus:(int)StatusCommand;
```

Parameters

<table>
<thead>
<tr>
<th>StatusCommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Turn off printer status monitoring.</td>
</tr>
<tr>
<td>101</td>
<td>Turn on printer status monitoring.</td>
</tr>
<tr>
<td>102</td>
<td>Returns 1 if printer status monitoring is active.</td>
</tr>
<tr>
<td>103</td>
<td>Start print job in paused state.</td>
</tr>
<tr>
<td>104</td>
<td>Start printing immediately.</td>
</tr>
<tr>
<td>105</td>
<td>Returns 1 if print job will be started in paused state.</td>
</tr>
<tr>
<td>200</td>
<td>Returns 1 if print job data exists.</td>
</tr>
<tr>
<td>201</td>
<td>Returns the Windows Spooler JobID.</td>
</tr>
<tr>
<td>202</td>
<td>Returns the job priority, from 1 to 99.</td>
</tr>
<tr>
<td>203</td>
<td>Returns the job's position in the print queue.</td>
</tr>
<tr>
<td>204</td>
<td>Returns the total page count.</td>
</tr>
<tr>
<td>205</td>
<td>Returns the number of pages that have been printed. This is usually zero if the data type is &quot;RAW&quot;.</td>
</tr>
<tr>
<td>206</td>
<td>Returns the number of milliseconds since the print job was started.</td>
</tr>
<tr>
<td>207</td>
<td>Returns the print job data type Returns 1 if the data type contains &quot;RAW&quot;</td>
</tr>
<tr>
<td></td>
<td>Returns 2 if the data type contains &quot;EMF&quot;</td>
</tr>
<tr>
<td></td>
<td>Returns 3 if the data type contains &quot;TEXT&quot;</td>
</tr>
<tr>
<td></td>
<td>Returns 4 if the data type contains &quot;XPS&quot;</td>
</tr>
<tr>
<td>300</td>
<td>Returns the encoded job status.</td>
</tr>
<tr>
<td>301</td>
<td>Returns 1 if the job is paused.</td>
</tr>
<tr>
<td>302</td>
<td>Returns 1 if there is an error.</td>
</tr>
<tr>
<td>303</td>
<td>Returns 1 if the job is being deleted.</td>
</tr>
<tr>
<td>304</td>
<td>Returns 1 if the job is spooling.</td>
</tr>
<tr>
<td>305</td>
<td>Returns 1 if the job is printing.</td>
</tr>
<tr>
<td>306</td>
<td>Returns 1 if the printer is offline.</td>
</tr>
<tr>
<td>307</td>
<td>Returns 1 if the printer is out of paper.</td>
</tr>
<tr>
<td>308</td>
<td>Returns 1 if the job has printed.</td>
</tr>
<tr>
<td>309</td>
<td>Returns 1 if the job has been deleted.</td>
</tr>
<tr>
<td>310</td>
<td>Returns 1 if the driver cannot print the print job.</td>
</tr>
<tr>
<td>311</td>
<td>Returns 1 if the printer has an error that requires the user to do something.</td>
</tr>
<tr>
<td>312</td>
<td>Returns 1 if the job has been restarted.</td>
</tr>
<tr>
<td>313</td>
<td>For Windows XP and later, returns 1 if the job has been sent to the printer (job may not be printed yet).</td>
</tr>
<tr>
<td>314</td>
<td>For Windows Vista and later, returns 1 if the job has been retained in the print queue and cannot be deleted.</td>
</tr>
<tr>
<td>401</td>
<td>Pause the print job.</td>
</tr>
<tr>
<td>402</td>
<td>Resume a paused print job.</td>
</tr>
<tr>
<td>403</td>
<td>Delete the print job.</td>
</tr>
</tbody>
</table>
RetrieveCustomDataToFile

Document properties

Description
Retrieves custom data from the PDF that was previously stored with StoreCustomDataFromString or StoreCustomDataFromFile. The retrieved data is written to the specified file.

Syntax

Dylib

```c
int DPLRetrieveCustomDataToFile(int InstanceID, wchar_t * Key, wchar_t * FileName, int Location);
```

Objective-C class

```objective-c
- (int)RetrieveCustomDataToFile:(NSString *)Key :(NSString *)FileName :(int)Location;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>The key that the data was stored under. If the location is the Document Catalog then the key must have a special prefix assigned to you by Adobe to avoid conflicts with other software. If the location is the Document Information Dictionary any key can be used but should be chosen with care so they make sense to the user.</td>
</tr>
<tr>
<td>FileName</td>
<td>The path and file name of the file to save the retrieved data to.</td>
</tr>
</tbody>
</table>
| Location  | 1 = Retrieve the data from the Document Information Dictionary  
            2 = Retrieve the data from the Document Catalog |

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>There was no data stored in the specified key, or the file to save the data to already exists and could not be overwritten</td>
</tr>
<tr>
<td>1</td>
<td>The data was retrieved and written to the specified file successfully</td>
</tr>
</tbody>
</table>
RetrieveCustomDataToString

Document properties

Description

Retrieves custom data from the PDF that was previously stored with the StoreCustomData function.

Syntax

Dylib

```c
char * DPLRetrieveCustomDataToString(int InstanceID, char * Key,
    int Location);
```

Objective-C class

```objectivec
- (NSData *)RetrieveCustomDataToString:(NSData *)Key :(int)Location;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>The key that the data was stored under. If the location is the Document Catalog then the key must have a special prefix assigned to you by Adobe to avoid conflicts with other software. If the location is the Document Information Dictionary any key can be used but should be chosen with care so they make sense to the user.</td>
</tr>
</tbody>
</table>
| Location  | 1 = Retrieve the data from the Document Information Dictionary  
2 = Retrieve the data from the Document Catalog |
ReverseImage

Image handling

Description

This function reverses the interpretation of the color components in the selected image. For example, a green pixel (0, 255, 0) will become a purple pixel (255, 0, 255) and a black pixel will become a white pixel.

Syntax

**Dylib**

```c
int DPLReverseImage(int InstanceID, int Reset);
```

**Objective-C class**

```objective-c
-(int)ReverseImage:(int)Reset;
```

Parameters

<table>
<thead>
<tr>
<th><strong>Reset</strong></th>
<th>Indicates whether the /Decode parameter in the image dictionary should be removed. This is necessary when the image is used as a stencil mask in Acrobat 4.0, but may give different results for different source image types (BMP, TIFF and PNG). Experimentation will be necessary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Keep the /Decode array and reverse the image</td>
</tr>
<tr>
<td>1</td>
<td>Remove the /Decode array</td>
</tr>
</tbody>
</table>
RotatePage

Page properties, Page manipulation

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Used to rotate the page by a multiple of 90 degrees. This will also rotate the co-ordinate system on the page so that it remains the same with respect to the orientation of the page. The rotation is absolute, for example calling the function twice with a parameter of 90 will result in a page rotated by 90 degrees, not 180 degrees.

Syntax

**Dylib**

```c
int DPLRotatePage(int InstanceID, int PageRotation);
```

**Objective-C class**

```c
-(int)RotatePage:(int)PageRotation;
```

Parameters

<table>
<thead>
<tr>
<th>PageRotation</th>
<th>The number of degrees to rotate the page by. Must be a multiple of 90 degrees (90, 180 or 270).</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The page could not be rotated, probably because the rotation specified was not a multiple of 90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The page was rotated successfully</td>
</tr>
</tbody>
</table>
SaveFontToFile

Fonts

Description

This function is useful for extracting fonts from a PDF that have been found with the FindFonts function. Only embedded TrueType fonts can be saved.

Syntax

**Dylib**

```c
int DPLSaveFontToFile(int InstanceID, wchar_t * FileName);
```

**Objective-C class**

```objective-c
-(int)SaveFontToFile:(NSString *)FileName;
```

Parameters

| FileName | The path and file name of the file that should be created to store the font data in. |

Return values

| 0      | The font is not embedded so there is no font data to save to the file |
| 1      | The embedded font data was written to the file successfully |
SaveImageListItemDataToFile

Image handling

Description
Saves the image data of an image list item to a file on disk.

Syntax

Dylib

```c
int DPLSaveImageListItemDataToFile(int InstanceID, int ImageListID, int ImageIndex, int Options, wchar_t * ImageFileName);
```

Objective-C class

```c
-(int)SaveImageListItemDataToFile:(int)ImageListID :(int)ImageIndex :(int)Options :(NSString *)ImageFileName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageListID</td>
<td>A value returned by the <code>GetPageImageList</code> function</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image in the list. The first image has an index of 1.</td>
</tr>
<tr>
<td>Options</td>
<td>Reserved for future use. Should be set to 0.</td>
</tr>
<tr>
<td>ImageFileName</td>
<td>The path and filename of the file to create</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Image data could not be saved</td>
</tr>
<tr>
<td>1</td>
<td>Image data was saved successfully</td>
</tr>
</tbody>
</table>
SaveImageToFile

Image handling

Description

Saves the selected image to a file on disk. Only certain images can be saved. If the ImageType function returns 0 then the image type is in an unsupported format and cannot be saved.

Syntax

Dylib

```c
int DPLSaveImageToFile(int InstanceID, wchar_t * FileName);
```

Objective-C class

```objective-c
- (int)SaveImageToFile:(NSString *)FileName;
```

Parameters

| FileName | The name of the image file to create. |

Return values

| 0 | The image could not be saved. Either an image is not selected or the file could not be created. |
| 1 | The image was saved successfully |
SaveImageToString

Image handling

Description

Use this function to save the selected image to a string. Only certain image types can be saved, see the SaveImageToFile function for further information.

Syntax

Dylib

char * DPLSaveImageToString(int InstanceID);

Objective-C class

- (NSData *)SaveImageToString
SaveState
Vector graphics, Page layout

Description

Saves the current graphics state, which can be loaded later with the LoadState function.

Syntax

Dylib

int DPLSaveState(int InstanceID);

Objective-C class

- (int)SaveState
**Text**

**Description**

Saves the current text properties under a named style. This style can then be applied quickly with a single call to the `ApplyStyle` function. The properties that are saved include the font name, font size, text color, alignment, underline and highlight style, spacing and scaling.

**Syntax**

**Dylib**

```c
int DPLSaveStyle(int InstanceID, wchar_t * StyleName);
```

**Objective-C class**

```objc
-(int)SaveStyle:(NSString *)StyleName;
```

**Parameters**

| **StyleName** | The name to associate with the style. This name is case sensitive. |
SaveToFile

Document management

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Saves the selected document to a file on disk.

Syntax

Dylib

int DPLSaveToFile(int InstanceID, wchar_t * FileName);

Objective-C class

- (int)SaveToFile:(NSString *)FileName;

Parameters

| FileName | The name of the file to create. |

Return values

| 0         | The file could not be created |
| 1         | The file was created successfully |
SaveToString
Document management

Description

Similar to the SaveToFile function, but instead of creating a file the data for the PDF file is returned as a string.

Syntax

Dylib

    char * DPLSaveToString(int InstanceID);

Objective-C class

    -(NSData * )SaveToString
SecurityInfo

Document properties, Security and Signatures

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Returns information about the security settings of the selected document.

Syntax

Dylib

int DPLSecurityInfo(int InstanceID, int SecurityItem);

Objective-C class

- (int)SecurityInfo:(int)SecurityItem;

Parameters

<table>
<thead>
<tr>
<th>SecurityItem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Security Method</td>
</tr>
<tr>
<td>1</td>
<td>User Password</td>
</tr>
<tr>
<td>2</td>
<td>Owner Password</td>
</tr>
<tr>
<td>3</td>
<td>Printing</td>
</tr>
<tr>
<td>4</td>
<td>Changing the Document</td>
</tr>
<tr>
<td>5</td>
<td>Content Copying or Extraction</td>
</tr>
<tr>
<td>6</td>
<td>Authoring Comments and Form Fields</td>
</tr>
<tr>
<td>7</td>
<td>Form Field Fill-in or Signing</td>
</tr>
<tr>
<td>8</td>
<td>Content Accessibility Enabled</td>
</tr>
<tr>
<td>9</td>
<td>Document Assembly</td>
</tr>
<tr>
<td>10</td>
<td>Encryption Level</td>
</tr>
<tr>
<td>11</td>
<td>Opened with User password</td>
</tr>
<tr>
<td>12</td>
<td>Opened with Owner password</td>
</tr>
<tr>
<td>13</td>
<td>Variable Encryption Strength</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Adobe Standard Security</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Fully Allowed</td>
</tr>
<tr>
<td>5</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>6</td>
<td>Allowed</td>
</tr>
<tr>
<td>7</td>
<td>40-bit RC4 (Acrobat 3.x, 4.x)</td>
</tr>
<tr>
<td>8</td>
<td>128-bit RC4 (Acrobat 5.x)</td>
</tr>
<tr>
<td>9</td>
<td>Unknown</td>
</tr>
<tr>
<td>10</td>
<td>Low resolution</td>
</tr>
<tr>
<td>11</td>
<td>Blank</td>
</tr>
<tr>
<td>12</td>
<td>128-bit AES (Acrobat 7)</td>
</tr>
<tr>
<td>13</td>
<td>256-bit AES (Acrobat 9)</td>
</tr>
<tr>
<td>14</td>
<td>Variable length RC4 (use SecurityItem=13 to determine the length)</td>
</tr>
<tr>
<td>15</td>
<td>256-bit AES (Acrobat X)</td>
</tr>
</tbody>
</table>
**SelectContentStream**

**Content Streams and Optional Content Groups**

**Description**

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function selects one of the selected page's content stream parts.

All drawing operations are only carried out on the selected content stream part.

**Syntax**

**Dylib**

```c
int DPLSelectContentStream(int InstanceID, int NewIndex);
```

**Objective-C class**

```c
- (int)SelectContentStream:(int)NewIndex;
```

**Parameters**

**NewIndex**  
The index of the content stream part to select. The first content stream part has an index of 1.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified layer could not be selected</td>
</tr>
<tr>
<td>1</td>
<td>The specified layer was selected successfully</td>
</tr>
</tbody>
</table>
SelectDocument

Document management

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Selects a document.

Syntax

Dylib

```
int DPLSelectDocument(int InstanceID, int DocumentID);
```

Objective-C class

```
- (int)SelectDocument:(int)DocumentID;
```

Parameters

<table>
<thead>
<tr>
<th>DocumentID</th>
<th>The ID of the document to select</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The document could not be selected, the ID could not be found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The specified document was selected successfully</td>
</tr>
</tbody>
</table>
SelectFont

Text, Fonts

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Select one of the fonts which have been added to the selected document. The FontID must be a valid ID as returned by one of the Add*Font functions.

Syntax

Dylib

```c
int DPLSelectFont(int InstanceID, int FontID);
```

Objective-C class

```objective-c
-(int)SelectFont:(int)FontID;
```

Parameters

| FontID     | The ID of the font to select |

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified ID could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The font was selected successfully</td>
</tr>
</tbody>
</table>
SelectImage

Image handling, Page layout

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Select one of the images that have been added to the selected document.

**Syntax**

**Dylib**

```c
int DPLSelectImage(int InstanceID, int ImageID);
```

**Objective-C class**

```c
- (int)SelectImage:(int)ImageID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ImageID</strong></td>
<td>The ID of the image to select</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified ID could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The image was selected successfully</td>
</tr>
</tbody>
</table>
SelectPage

Page layout, Page manipulation

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Selects a page of the selected document.

**Syntax**

**Dylib**

```c
int DPLSelectPage(int InstanceID, int PageNumber);
```

**Objective-C class**

```objective-c
- (int) SelectPage:(int) PageNumber;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNumber</td>
<td>The page to select</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified page could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The page was selected successfully</td>
</tr>
</tbody>
</table>
SelectRenderer
Rendering and printing

Description
Select the renderer to use during rendering.
If Cairo is used, the SetCairoFileName function should be used to set the path to the Cairo DLL.
Rendering using Cairo is currently experimental and is only enabled for the RenderPageToDC and DARenderPageToDC functions.

Syntax
Dylib

```
int DPLSelectRenderer(int InstanceID, int RendererID);
```

Objective-C class

```
-(int)SelectRenderer:(int)RendererID;
```

Parameters

<table>
<thead>
<tr>
<th>RendererID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GDI+</td>
</tr>
<tr>
<td>2</td>
<td>Cairo</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified renderer could not be selected</td>
</tr>
<tr>
<td>1</td>
<td>The GDI+ renderer was selected</td>
</tr>
<tr>
<td>2</td>
<td>The Cairo renderer was selected</td>
</tr>
</tbody>
</table>
SelectedDocument

Document management

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Returns the ID of the selected document.

Syntax

Dylib

```c
int DPLSelectedDocument(int InstanceID);
```

Objective-C class

```objective-c
-(int)SelectedDocument
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A document has not been selected. This should never occur.</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the selected document</td>
</tr>
</tbody>
</table>
SelectedFont

Text, Fonts

Description

Returns the ID of the selected font.

Syntax

Dylib

int DPLSelectedFont(int InstanceID);

Objective-C class

- (int)SelectedFont

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No font has been selected</td>
</tr>
<tr>
<td>Non-zero</td>
<td>The ID of the selected font</td>
</tr>
</tbody>
</table>
**SelectedImage**

**Image handling, Page layout**

**Description**

Returns the ID of the selected image.

**Syntax**

**Dylib**

```c
int DPLSelectedImage(int InstanceID);
```

**Objective-C class**

```c
- (int)SelectedImage
```

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image has been selected</td>
</tr>
<tr>
<td><strong>Non-zero</strong></td>
<td>The ID of the selected image</td>
</tr>
</tbody>
</table>
**SelectedPage**

Page layout, Page manipulation

**Description**

Returns currently selected page.

**Syntax**

**Dylib**

```c
int DPLSelectedPage(int InstanceID);
```

**Objective-C class**

```c
- (int)SelectedPage
```
SetActionURL

Annotations and hotspot links

Description
Sets the target URL of the specified action.

Syntax

**Dylib**

```c
int DPLSetActionURL(int InstanceID, int ActionID, wchar_t * NewURL);
```

**Objective-C class**

```objc
- (int)SetActionURL:(int)ActionID :(NSString *)NewURL;
```

Parameters

<table>
<thead>
<tr>
<th>ActionID</th>
<th>An ActionID as returned by the GetAnnotActionID, GetOutlineActionID or GetFormFieldActionID functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewURL</td>
<td>The new URL target</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified ActionID was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The action's target URL was set successfully</td>
</tr>
</tbody>
</table>
SetAnnotBorderColor
Color, Annotations and hotspot links

Description
Sets the border color for the specified annotation.

Syntax

Dylib

    int DPLSetAnnotBorderColor(int InstanceID, int Index, double Red, double Green, double Blue);

Objective-C class

    - (int)SetAnnotBorderColor:(int)Index :(double)Red :(double)Green :(double)Blue;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
<tr>
<td>Red</td>
<td>The red component of the color</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>
SetAnnotBorderStyle
Annotations and hotspot links

Description

Sets the border style of the specified annotation.

Syntax

Dylib

```c
int DPLSetAnnotBorderStyle(int InstanceID, int Index, double Width,
int Style, double DashOn, double DashOff);
```

Objective-C class

```objc
- (int)SetAnnotBorderStyle:(int)Index :(double)Width :(int)Style
 :(double)DashOn :(double)DashOff;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the border</td>
</tr>
<tr>
<td><strong>Style</strong></td>
<td>The style of the border: 0 = Solid 1 = Dashed 2 = Beveled 3 = Inset Anything else = Solid</td>
</tr>
<tr>
<td><strong>DashOn</strong></td>
<td>The length of the dash. Only valid if the border style is &quot;dashed&quot;.</td>
</tr>
<tr>
<td><strong>DashOff</strong></td>
<td>The length of the spaces between the dashes. Only valid if the border style is &quot;dashed&quot;.</td>
</tr>
</tbody>
</table>
SetAnnotContents

Annotations and hotspot links

Description

Changes the contents of an annotation.

Syntax

Dylib

```c
int DPLSetAnnotContents(int InstanceID, int Index, wchar_t * NewContents);
```

Objective-C class

```objective-c
- (int)SetAnnotContents:(int)Index :(NSString *)NewContents;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
<tr>
<td>NewContents</td>
<td>The new contents of the annotation</td>
</tr>
</tbody>
</table>
SetAnnotDblProperty
Annotations and hotspot links

Description
Sets an double property of the specified annotation.

Syntax
Dylib

```c
int DPLSetAnnotDblProperty(int InstanceID, int Index, int Tag,
                            double NewValue);
```

Objective-C class

```
- (int)SetAnnotDblProperty:(int)Index :(int)Tag :(double)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the annotation. The first annotation on the page has an index of 1.</th>
</tr>
</thead>
</table>
| Tag   | 105 = Left  
|       | 106 = Top  
|       | 107 = Width  
|       | 108 = Height  |
| NewValue | The new value of the specified annotation and property. |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The annotation specified by the Index parameter was out of range or the Tag parameter was not valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The annotation property was set successfully</td>
</tr>
</tbody>
</table>
SetAnnotIntProperty

Annotations and hotspot links

Description

Sets an integer property of the specified annotation.

Syntax

Dylib

```c
int DPLSetAnnotIntProperty(int InstanceID, int Index, int Tag,
int NewValue);
```

Objective-C class

```c
- (int)SetAnnotIntProperty:(int)Index :(int)Tag :(int)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
</tbody>
</table>
| Tag       | 116 = Page number of "GoToR" action (1 is first page)  
131 = Page number of "GoTo" action |
| NewValue  | The new value of the specified annotation and property. |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The annotation specified by the Index parameter was out of range or the Tag parameter was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The annotation property was set successfully</td>
</tr>
</tbody>
</table>
SetAnnotQuadPoints

Annotations and hotspot links

Description

Sets the co-ordinates of the specified quad (rectangular area) contained within the specified annotation. If the QuadNumber is higher than the number of quads that the annotation already has then a new quad will be added to the annotation.

From version 7.25 the order of the co-ordinates has changed for consistency between GetPageText and GetAnnotQuadPoints.

Syntax

Dylib

```c
int DPLSetAnnotQuadPoints(int InstanceID, int Index, int QuadNumber,
   double X1, double Y1, double X2, double Y2, double X3,
   double Y3, double X4, double Y4);
```

Objective-C class

```objective-c
- (int)SetAnnotQuadPoints:(int)Index :(int)QuadNumber :(double)X1
   :(double)Y1 :(double)X2 :(double)Y2 :(double)X3 :(double)Y3
   :(double)X4 :(double)Y4;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
<tr>
<td>QuadNumber</td>
<td>The index of the annotation's quad to set. The first quad has a QuadNumber of 1. If QuadNumber is greater than the number of existing quads then a new quad will be added to the annotation.</td>
</tr>
<tr>
<td>X1</td>
<td>The horizontal co-ordinate of the bottom-left corner.</td>
</tr>
<tr>
<td>Y1</td>
<td>The vertical co-ordinate of the bottom-left corner.</td>
</tr>
<tr>
<td>X2</td>
<td>The horizontal co-ordinate of the bottom-right corner.</td>
</tr>
<tr>
<td>Y2</td>
<td>The vertical co-ordinate of the bottom-right corner.</td>
</tr>
<tr>
<td>X3</td>
<td>The horizontal co-ordinate of the top-right corner.</td>
</tr>
<tr>
<td>Y3</td>
<td>The vertical co-ordinate of the top-right corner.</td>
</tr>
<tr>
<td>X4</td>
<td>The horizontal co-ordinate of the top-left corner.</td>
</tr>
<tr>
<td>Y4</td>
<td>The vertical co-ordinate of the top-left corner.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The QuadNumber parameter was less than 1.</td>
</tr>
<tr>
<td>1</td>
<td>The quad was changed or a new quad was added.</td>
</tr>
</tbody>
</table>
SetAnnotRect
Annotations and hotspot links

Description
Sets the size and position of the specified annotation.

Syntax

**Dylib**
```c
int DPLSetAnnotRect(int InstanceID, int Index, double Left, double Top, double Width, double Height);
```

**Objective-C class**
```objc
-(int)SetAnnotRect:(int)Index :(double)Left :(double)Top :(double)Width :(double)Height;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
<tr>
<td>Left</td>
<td>The new horizontal co-ordinate of the left edge of the annotation</td>
</tr>
<tr>
<td>Top</td>
<td>The new vertical co-ordinate of the top edge of the annotation</td>
</tr>
<tr>
<td>Width</td>
<td>The new width of the annotation</td>
</tr>
<tr>
<td>Height</td>
<td>The new height of the annotation</td>
</tr>
</tbody>
</table>
SetAnnotStrProperty

Annotations and hotspot links

Description

Sets a string property of the specified annotation.

Syntax

Dylib

```c
int DPLSetAnnotStrProperty(int InstanceID, int Index, int Tag,
                          wchar_t * NewValue);
```

Objective-C class

```objective-c
- (int)SetAnnotStrProperty:(int)Index :(int)Tag :(NSString *)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
</tbody>
</table>
| Tag       | 102 = Contents  
            | 103 = Name  
            | 110 = Subject  
            | 111 = URL of a link annotation  
            | 113 = The "Win" file name of a "Launch" action  
            | 114 = The "F" file name of a "Launch" action  
            | 115 = The "F" file name of a "GoToR" action  
            | 127 = Subject  
            | 129 = The "UF" file name of a "Launch" action  
            | 130 = The "UF" file name of a "GoToR" action |
| NewValue   | The new value of the specified annotation and property. |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The annotation specified by the Index parameter was out of range or the Tag parameter was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The annotation property was set successfully</td>
</tr>
</tbody>
</table>
SetAnsiMode

**Description**

This function sets the mode used by the DLL to convert strings to and from Unicode.

**Syntax**

**Dylib**

```c
int DPLSetAnsiMode(int InstanceID, int NewAnsiMode);
```

**Objective-C class**

```objc
- (int)SetAnsiMode:(int)NewAnsiMode;
```

**Parameters**

<table>
<thead>
<tr>
<th>NewAnsiMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Conversion using the current code page</td>
</tr>
<tr>
<td>1</td>
<td>Conversion using UTF-8 encoding</td>
</tr>
</tbody>
</table>
SetBaseURL

Document properties, Annotations and hotspot links

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Sets the Base URL for all URL links in the document.

For example, if the Base URL was set to "http://www.example.com/" and a URL link destination was set to "index.html" then the link will point to "http://www.example.com/index.html".

Use the AddLinkToWeb function to add a URL link to the current page.

Syntax

Dylib

```c
int DPLSetBaseURL(int InstanceID, wchar_t * NewBaseURL);
```

Objective-C class

```c
-(int)setBaseURL:(NSString *)NewBaseURL;
```

Parameters

`NewBaseURL` The base URL to use for all URL link annotations in the document.
SetBlendMode

Vector graphics, Image handling, Text

Description

Sets the blend mode for subsequently drawn graphics.

Syntax

Dylib

```c
int DPLSetBlendMode(int InstanceID, int BlendMode);
```

Objective-C class

```objective-c
-(int)SetBlendMode:(int)BlendMode;
```

Parameters

<table>
<thead>
<tr>
<th>BlendMode</th>
<th>The blend mode to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>1</td>
<td>Multiply</td>
</tr>
<tr>
<td>2</td>
<td>Screen</td>
</tr>
<tr>
<td>3</td>
<td>Overlay</td>
</tr>
<tr>
<td>4</td>
<td>Darken</td>
</tr>
<tr>
<td>5</td>
<td>Lighten</td>
</tr>
<tr>
<td>6</td>
<td>Color Dodge</td>
</tr>
<tr>
<td>7</td>
<td>Color Burn</td>
</tr>
<tr>
<td>9</td>
<td>Hard Light</td>
</tr>
<tr>
<td>10</td>
<td>Soft Light</td>
</tr>
<tr>
<td>11</td>
<td>Difference</td>
</tr>
<tr>
<td>12</td>
<td>Exclusion</td>
</tr>
<tr>
<td>13</td>
<td>Hue</td>
</tr>
<tr>
<td>14</td>
<td>Saturation</td>
</tr>
<tr>
<td>15</td>
<td>Luminosity</td>
</tr>
</tbody>
</table>
SetBreakString

**Text**

**Description**

Sets the string to use to mark line breaks. This string allows text to be split when using the *WrappedText functions.

**Syntax**

**Dylib**

```c
int DPLSetBreakString(int InstanceID, wchar_t * NewBreakString);
```

**Objective-C class**

```objective-c
-(int)SetBreakString:(NSString *)NewBreakString;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>NewBreakString</strong></th>
<th>The string of characters to use as a break character, for example Chr(13) + Chr(10)</th>
</tr>
</thead>
</table>
**SetCSDictEPSG**

**Measurement and coordinate units**

**Description**

Sets the EPSG reference code for a coordinate system dictionary (see www.epsg.org).

**Syntax**

**Dylib**

```c
int DPLSetCSDictEPSG(int InstanceID, int CSDictID, int NewEPSG);
```

**Objective-C class**

```objective-c
-(int)SetCSDictEPSG:(int)CSDictID :(int)NewEPSG;
```

**Parameters**

<table>
<thead>
<tr>
<th>CSDictID</th>
<th>A value returned from the <code>GetMeasureDictGCSDict</code> or <code>GetMeasureDictDCSDict</code> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewEPSG</td>
<td>The new value for the EPSG reference code</td>
</tr>
</tbody>
</table>

**Return values**

| 0              | The CSDictID parameter was incorrect                                                     |
| 1              | Success                                                                                    |
SetCSDictType

Measurement and coordinate units

Description

Sets the coordinate system type of a coordinate system dictionary.

Syntax

**Dylib**

```
int DPLSetCSDictType(int InstanceID, int CSDictID, int NewDictType);
```

**Objective-C class**

```
-(int)SetCSDictType:(int)CSDictID :(int)NewDictType;
```

Parameters

<table>
<thead>
<tr>
<th>CSDictID</th>
<th>A value returned from the <code>GetMeasureDictGCSDict</code> or <code>GetMeasureDictDCSDict</code> functions</th>
</tr>
</thead>
</table>
| NewDictType | 1 = Geographic coordinate system (GEOGCS)  
2 = Projected coordinate system (PROJCS) |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The CSDictID parameter was incorrect or the NewDictType parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetCSDictWKT

Measurement and coordinate units

Description

Sets the Well Known Text (WKT) describing a coordinate system dictionary.

Syntax

Dylib

```c
int DPLSetCSDictWKT(int InstanceID, int CSDictID, wchar_t * NewWKT);
```

Objective-C class

```objective-c
- (int)SetCSDictWKT:(int)CSDictID :(NSString *)NewWKT;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSDictID</td>
<td>A value returned from the <code>GetMeasureDictGCSDict</code> or <code>GetMeasureDictDCSDict</code> functions</td>
</tr>
<tr>
<td>NewWKT</td>
<td>The new Well Known Text description</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The CSDictID parameter was incorrect</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetCairoFileName

Description

Sets the path and file name of the Cairo DLL. The `SelectRenderer` function can be used to select the Cairo renderer rather than the default GDI+ renderer.

The Cairo DLL is usually dependent on other DLLs. If these are not all stored in the same directory as the application, or a system directory, the Windows API function `SetDllDirectory` should be used to add the correct path before calling any rendering functions.

Rendering using Cairo is currently experimental.

Syntax

**Dylib**

```c
int DPLSetCairoFileName(int InstanceID, wchar_t * FileName);
```

**Objective-C class**

```objective-c
- (int)SetCairoFileName:(NSString *)FileName;
```

Parameters

| FileName | The path and file name of the Cairo DLL. |

Return values

<table>
<thead>
<tr>
<th></th>
<th>The specified DLL was not a valid Cairo DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The specified Cairo DLL was valid</td>
</tr>
</tbody>
</table>
SetCapturedPageOptional

Content Streams and Optional Content Groups, Page layout

Description

Links the captured page to an optional content group. This allows the captured page to be selectively shown in Acrobat 6 or later.

Syntax

**Dylib**

```c
int DPLSetCapturedPageOptional(int InstanceID, int CaptureID, int OptionalContentGroupID);
```

**Objective-C class**

```c
- (int)SetCapturedPageOptional:(int)CaptureID :(int)OptionalContentGroupID;
```

Parameters

<table>
<thead>
<tr>
<th>CaptureID</th>
<th>The ID returned by the CapturePage function when a page was previously captured</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptionalContentGroupID</td>
<td>An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions</td>
</tr>
</tbody>
</table>

Return values

| 0 | The CaptureID or OptionalContentGroupID parameters were not valid |
| 1 | The captured page was linked to the optional content group successfully |
SetCapturedPageTransparencyGroup
Content Streams and Optional Content Groups, Page layout

Syntax

Dylib

```c
int DPLSetCapturedPageTransparencyGroup(int InstanceID, int CaptureID,
int CS, int Isolate, int Knockout);
```

Objective-C class

```objc
- (int)SetCapturedPageTransparencyGroup:(int)CaptureID :(int)CS
:(int)Isolate :(int)Knockout;
```

Parameters

<table>
<thead>
<tr>
<th>CaptureID</th>
<th>The ID returned by the CapturePage function when a page was previously captured</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>The color space to use: 1 = RGB 2 = CMYK</td>
</tr>
<tr>
<td>Isolate</td>
<td>This parameter has no effect and is reserved for future use. It should always be set to 0.</td>
</tr>
<tr>
<td>Knockout</td>
<td>Indicates whether items added to the page are drawn over each other or &quot;knocked out&quot; of the page. In knockout mode a &quot;hole&quot; is made through existing objects on the page in the shape of the new object. The new object is then drawn against the background. 0 = Do not knockout 1 = Knockout</td>
</tr>
</tbody>
</table>

Return values

| 0 | An error occurred |
| 1 | Success |
**SetCatalogInformation**

**Document properties**

**Description**

This function allows you to store custom information in the PDF document. This is similar to the **SetCustomInformation** function, but the information is stored in the Document Catalog instead of the Document Information Dictionary. Metadata should be stored in the Document Information Dictionary using **SetCustomInformation**, private content or structural information should be stored in the Document Catalog using this function.

**Syntax**

**Dylib**

```c
int DPLSetCatalogInformation(int InstanceID, wchar_t * Key,
                              wchar_t * NewValue);
```

**Objective-C class**

```objc
- (int)SetCatalogInformation:(NSString *)Key :(NSString *)NewValue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Key</th>
<th>The name of the key to set. This key must have a special prefix assigned to you by Adobe to avoid conflicts with other software.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewValue</td>
<td>The new value of the specified key.</td>
</tr>
</tbody>
</table>

**Return values**

| 0          | The key specified could not be set, it may have been a system key                                                   |
| 1          | The value of the specified key was set successfully                                                                  |
SetCharWidth
Text, Form fields

Description
Sets the width of a specific character in the selected font.
The width uses is a ratio to the text size. For example, if a value of 750 is used the width of the character when output as 12pt text would be $(750 / 1000) \times 12$.

Syntax

Dylib

```c
int DPLSetCharWidth(int InstanceID, int CharCode, int NewWidth);
```

Objective-C class

```c
-(int)SetCharWidth:(int)CharCode :(int)NewWidth;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharCode</td>
<td>The glyph character code that should be set. For example, 65 for “A”.</td>
</tr>
<tr>
<td>NewWidth</td>
<td>The new width</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A font has not been selected</td>
</tr>
<tr>
<td>1</td>
<td>The width was set successfully</td>
</tr>
</tbody>
</table>
SetClippingPath

Vector graphics, Path definition and drawing

Description

Uses the current path as a clipping path for subsequent drawing operations. The current path is combined with the existing clipping path, this means that the clipping area can only be made smaller.

To restore the clipping path, call `SaveState` before calling this function and then `LoadState` to restore the clipping path to its previous state.

Syntax

**Dylib**

```c
int DPLSetClippingPath(int InstanceID);
```

**Objective-C class**

```objc
- (int)SetClippingPath
```
SetClippingPathEvenOdd

Vector graphics, Path definition and drawing

Description

Similar to the SetClippingPath function, but uses the "even odd" method for dealing with situations where parts of the path overlap.

Syntax

**Dylib**

```c
int DPLSetClippingPathEvenOdd(int InstanceID);
```

**Objective-C class**

```objective_c
-(int)SetClippingPathEvenOdd
```
SetCompatibility

Miscellaneous functions

Description

Sets Quick PDF Library to operate in the same way as previous versions of the library to maintain backwards compatibility.

Syntax

Dylib

```c
int DPLSetCompatibility(int InstanceID, int CompatibilityItem,
                         int CompatibilityMode);
```

Objective-C class

```objective-c
- (int)SetCompatibility:(int)CompatibilityItem :(int)CompatibilityMode;
```

Parameters

<table>
<thead>
<tr>
<th>CompatibilityItem</th>
<th>100 = DrawTableRows return value scaling (version 7.18)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompatibilityMode</td>
<td>0 = Turn off compatibility</td>
</tr>
<tr>
<td></td>
<td>1 = Turn on compatibility</td>
</tr>
</tbody>
</table>

Return values

| 0                  | Either CompatibilityItem or CompatibilityMode was out of range |
| 1                  | The compatibility mode was set successfully               |
SetContentStreamFromString

Page properties, Content Streams and Optional Content Groups, Page manipulation

Description

Sets the PDF page description commands in the content stream part that was selected with the SelectContentStream function.

Syntax

**Dylib**

```c
int DPLSetContentStreamFromString(int InstanceID, char * Source);
```

**Objective-C class**

```objc
- (int)SetContentStreamFromString:(NSData *)Source;
```

Parameters

<table>
<thead>
<tr>
<th><strong>Source</strong></th>
<th>The new PDF page description commands for the content stream part</th>
</tr>
</thead>
</table>
SetContentStreamOptional

Content Streams and Optional Content Groups

Description

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function links the content stream that was selected using the SelectContentStream function to an optional content group. This allows the content stream part to be selectively shown in Acrobat 6 or later.

Syntax

Dylib

```c
int DPLSetContentStreamOptional(int InstanceID,
                                int OptionalContentGroupID);
```

Objective-C class

```objc
- (int)SetContentStreamOptional:(int)OptionalContentGroupID;
```

Parameters

<table>
<thead>
<tr>
<th>OptionalContentGroupID</th>
<th>An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The OptionalContentGroupID parameter was not valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The content stream part was linked to the optional content group successfully</td>
</tr>
</tbody>
</table>
SetCropBox

Page properties

Description

Sets the visible area of the selected page. The non-visible area will be "cropped" and will not be displayed or printed.

Syntax

**Dylib**

```c
int DPLSetCropBox(int InstanceID, double Left, double Top, double Width, double Height);
```

**Objective-C class**

```objective-c
- (int)SetCropBox:(double)Left :(double)Top :(double)Width :(double)Height;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal co-ordinate of the left edge of the cropping rectangle</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical co-ordinate of the top edge of the cropping rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the cropping rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the cropping rectangle</td>
</tr>
</tbody>
</table>
**SetCustomInformation**

**Document properties**

**Description**

This function is used to store custom metadata in the document. These values can later be read from the document with the `GetCustomInformation` function. The data is stored in the Document Information Dictionary. Private content or structural information should rather be stored in the Document Catalog using the `SetCatalogInformation` function.

**Syntax**

**Dylib**

```c
int DPLSetCustomInformation(int InstanceID, wchar_t * Key, wchar_t * NewValue);
```

**Objective-C class**

```c
- (int)SetCustomInformation:(NSString *)Key :(NSString *)NewValue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Key</th>
<th>Specifies which key to set</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewValue</td>
<td>The value to set the key to.</td>
</tr>
</tbody>
</table>

**Return values**

| 0  | The value could not be set. The Key parameter cannot be "Producer", "Creator", "Subject", "Title", "Keywords" or "Author". For these keys use the `SetInformation` function. |
| 1  | The value of the key was set successfully |
SetCustomLineDash

Vector graphics

Description

Sets a custom line dash pattern.

Syntax

Dylib

```c
int DPLSetCustomLineDash(int InstanceID, wchar_t * DashPattern,
                        double DashPhase);
```

Objective-C class

```objc
- (int)SetCustomLineDash:(NSString *)DashPattern :(double)DashPhase;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DashPattern</td>
<td>A list of numeric values separated with commas. Alternate values are used for dashes and spaces. A period must be used for numbers with decimal fractions. For example, to make a dash-dot-dot pattern the following could be used: &quot;20.5,10,11,10,11,10&quot;</td>
</tr>
<tr>
<td>DashPhase</td>
<td>The distance within the pattern to start the dashed line. For example, if DashPattern is &quot;20,10,40,10&quot; and DashPhase is set to 5, the dashed line will start with a dash of size 15. The next dash will be 40, then 20, then 40, etc. with spaces of 10 between each dash.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The dash pattern was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The custom dash pattern was set successfully</td>
</tr>
</tbody>
</table>
SetDPLRFileName

**Description**

Sets the path and file name of the DPLR DLL. The `SelectRenderer` function can be used to select the DPLR renderer rather than the default GDI+ renderer.

Rendering using DPLR is currently experimental.

**Syntax**

**Dylib**

```c
int DPLSetDPLRFileName(int InstanceID, wchar_t * FileName);
```

**Objective-C class**

```objective-c
-(int)SetDPLRFileName:(NSString *)FileName;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>FileName</strong></th>
<th>The path and file name of the DPLR DLL.</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified DLL was not a valid DPLR DLL</td>
</tr>
<tr>
<td>1</td>
<td>The specified DPLR DLL was valid</td>
</tr>
</tbody>
</table>
SetDestProperties

Annotations and hotspot links

Description

Changes various properties of an existing destination.

Syntax

Dylib

```c
int DPLSetDestProperties(int InstanceID, int DestID, int Zoom,
int DestType, double Left, double Top, double Right,
double Bottom);
```

Objective-C class

```objc
- (int)SetDestProperties:(int)DestID :(int)Zoom :(int)DestType
:(double)Left :(double)Top :(double)Right :(double)Bottom;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestID</td>
<td>The ID of the destination to analyse. A valid destination ID is returned by the <code>GetOutlineDest</code> function.</td>
</tr>
<tr>
<td>Zoom</td>
<td>The zoom percentage to use when the outline destination is opened, valid values from 0 to 6400. Only used for DestType = 1, should be set to 0 for other DestTypes.</td>
</tr>
</tbody>
</table>
| DestType  | 1 = "XYZ" - the target page is positioned at the point specified by the Left and Top parameters. The Zoom parameter specifies the zoom percentage.  
2 = "Fit" - the entire page is zoomed to fit the window. None of the other parameters are used and should be set to zero.  
3 = "FitH" - the page is zoomed so that the entire width of the page is visible. The height of the page may be greater or less than the height of the window. The page is positioned at the vertical position specified by the Top parameter.  
4 = "FitV" - the page is zoomed so that the entire height of the page can be seen. The width of the page may be greater or less than the width of the window. The page is positioned at the horizontal position specified by the Left parameter.  
5 = "FitR" - the page is zoomed so that a certain rectangle on the page is visible. The Left, Top, Right and Bottom parameters define the rectangular area on the page.  
6 = "FitB" - the page is zoomed so that its bounding box is visible.  
7 = "FitBH" - the page is positioned vertically at the position specified by the Top parameter. The page is zoomed so that the entire width of the page's bounding box is visible.  
8 = "FitBV" - the page is positioned at the horizontal position specified by the Left parameter. The page is zoomed just enough to fit the entire height of the bounding box into the window. |
| Left      | The horizontal position used by DestType = 1, 4, 5 and 8 |
| Top       | The vertical position used by DestType = 1, 3, 5 and 7 |
| Right     | The horizontal position of the righthand edge of the rectangle. Used by DestType = 5 |
| Bottom    | The horizontal position of the bottom of the rectangle. Used by DestType = 5 |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The destination properties could not be set. The DestID parameter might be invalid or the Zoom and DestType parameters could be out of range.</td>
</tr>
<tr>
<td>1</td>
<td>The destination properties were set successfully</td>
</tr>
</tbody>
</table>
**SetDestValue**

**Annotations and hotspot links**

**Description**

Sets one of the properties of the specified destination.

**Syntax**

**Dylib**

```c
int DPLSetDestValue(int InstanceID, int DestID, int ValueKey, double NewValue);
```

**Objective-C class**

```objective-c
- (int)SetDestValue:(int)DestID :(int)ValueKey :(double)NewValue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestID</td>
<td>The ID of the destination to analyse. A valid destination ID is returned by the <code>GetOutlineDest</code> function.</td>
</tr>
</tbody>
</table>
| ValueKey  | 1=Left  
2=Top  
3=Bottom  
4=Right  
5=Zoom |
| NewValue  | The new value for the specified destination property |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The destination value could not be set. The DestID parameter might be invalid or the DestType parameter could be out of range.</td>
</tr>
<tr>
<td>1</td>
<td>The destination type was set successfully</td>
</tr>
</tbody>
</table>
SetDocumentMetadata

Document properties

Description

Set's the document metadata. The metadata must be a valid XMP string, see Adobe's website for XMP documentation.

Syntax

Dylib

```c
int DPLSetDocumentMetadata(int InstanceID, wchar_t * XMP);
```

Objective-C class

```objective-c
-(int)SetDocumentMetadata:(NSString *)XMP;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMP</td>
<td>The XMP metadata</td>
</tr>
</tbody>
</table>

Return values

This function always returns 1
SetEmbeddedFileStrProperty

Document properties

Description

Sets a property of the specified embedded file.

Syntax

Dylib

```c
int DPLSetEmbeddedFileStrProperty(int InstanceID, int Index, int Tag, wchar_t * NewValue);
```

Objective-C class

```objective-c
- (int)SetEmbeddedFileStrProperty:(int)Index :(int)Tag :
(NSString *)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the embedded file. Must be a value between 1 and the value returned by <code>EmbeddedFileCount</code>.</td>
</tr>
</tbody>
</table>
| Tag       | 1 = File name  
2 = MIME type  
3 = Creation date  
4 = Modification date  
5 = Title  
7 = Description |
| NewValue  | The new value of the specified property. |
SetFillColor

Description

Sets the fill color for any subsequently drawn graphics. The values for Red, Green and Blue range from 0 to 1, where 0 indicates 0% and 1 indicates 100% of the color.

Syntax

Dylib

```c
int DPLSetFillColor(int InstanceID, double Red, double Green, double Blue);
```

Objective-C class

```objective-c
-(int)SetFillColor:(double)Red :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>The red component of the color</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>
SetFillColorCMYK

Vector graphics, Color

Description

Sets the fill color of subsequently drawn graphics. Similar to the SetFillColor function, but allows a color in the CMYK color space to be used. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLSetFillColorCMYK(int InstanceID, double C, double M, double Y, double K);
```

Objective-C class

```objective-c
-(int)SetFillColorCMYK:(double)C :(double)M :(double)Y :(double)K;
```

Parameters

<table>
<thead>
<tr>
<th>C</th>
<th>The cyan component of the color</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>The magenta component of the color</td>
</tr>
<tr>
<td>Y</td>
<td>The yellow component of the color</td>
</tr>
<tr>
<td>K</td>
<td>The black component of the color</td>
</tr>
</tbody>
</table>
SetFillColorSep
Vector graphics, Color

Description
Sets the fill color of subsequently drawn graphics. Similar to the SetFillColor function, but a tint of a separation color added with the AddSeparationColor function is used.

Syntax

Dylib

```c
int DPLSetFillColorSep(int InstanceID, wchar_t * ColorName, double Tint);
```

Objective-C class

```c
-(int)SetFillColorSep:(NSString *)ColorName :(double)Tint;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorName</td>
<td>The name of the separation color that was used with the AddSeparationColor function</td>
</tr>
<tr>
<td>Tint</td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The separation color name could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The fill color was set successfully</td>
</tr>
</tbody>
</table>
SetFillShader

Vector graphics, Path definition and drawing, Color

Description

Sets the fill to the specified shader for subsequently drawn graphics.

Syntax

**Dylib**

```c
int DPLSetFillShader(int InstanceID, wchar_t * ShaderName);
```

**Objective-C class**

```objc
- (int)SetFillShader:(NSString *)ShaderName;
```

Parameters

| **ShaderName** | The shader name that was used when the shader was created. |

Return values

| **0** | The shader could not be found |
| **1** | The shader fill was setup correctly |
SetFillTilingPattern

Vector graphics, Color

Description

Sets the current fill to the specified tiling pattern.

Syntax

Dylib

```c
int DPLSetFillTilingPattern(int InstanceID, wchar_t * PatternName);
```

Objective-C class

```objc
- (int)SetFillTilingPattern:(NSString *)PatternName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PatternName</td>
<td>The pattern name that was used with the <code>NewTilingPatternFromCapturedPage</code> function</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The PatternName parameter was invalid</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetFindImagesMode
Image handling, Document management, Page properties

Description
Sets the search mode used by the FindImages function.
The default search mode runs a recursive search in the resources of all the pages and annotations in the document. This is the fastest method and requires the least amount of memory, however unused images will not be found.
The full search mode examines each object in the document. This takes more time and requires more memory, however all images will be located even if they are not used by any of the pages or annotations in the document.

Syntax

**Dylib**
```
int DPLSetFindImagesMode(int InstanceID, int NewFindImagesMode);
```

**Objective-C class**
```
- (int)SetFindImagesMode:(int)NewFindImagesMode;
```

Parameters

<table>
<thead>
<tr>
<th>NewFindImagesMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default search mode</td>
</tr>
<tr>
<td>2</td>
<td>Full search mode</td>
</tr>
<tr>
<td>3</td>
<td>Default search mode, full convert</td>
</tr>
<tr>
<td>4</td>
<td>Full search mode, full convert</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An invalid value for the NewFindImagesMode parameter was used</td>
</tr>
<tr>
<td>1</td>
<td>The search mode was changed successfully</td>
</tr>
</tbody>
</table>
SetFontEncoding

Fonts

Description

Sets the encoding for the selected font.

Syntax

Dylib

```c
int DPLSetFontEncoding(int InstanceID, int Encoding);
```

Objective-C class

```objectivec
- (int)SetFontEncoding:(int)Encoding;
```

Parameters

<table>
<thead>
<tr>
<th>Encoding</th>
<th>The encoding to use for the font:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>StandardEncoding</td>
</tr>
<tr>
<td>1</td>
<td>MacRomanEncoding</td>
</tr>
<tr>
<td>2</td>
<td>WinAnsiEncoding</td>
</tr>
<tr>
<td>3</td>
<td>Deprecated (was PDFDocEncoding)</td>
</tr>
<tr>
<td>4</td>
<td>MacExpertEncoding</td>
</tr>
<tr>
<td>5</td>
<td>Do not specify encoding</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>No font was selected, or the encoding could not be set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The encoding for the selected font was set successfully</td>
</tr>
</tbody>
</table>
Sets the flags for the selected font. Usually these flags are set automatically when the font is added, but in some circumstance (for example with symbolic Type1 fonts) the flags cannot be automatically set. This function allows you to ensure the fonts have the correct flags.

**Syntax**

**Dylib**

```c
int DPLSetFontFlags(int InstanceID, int Fixed, int Serif, int Symbolic, int Script, int Italic, int AllCap, int SmallCap, int ForceBold);
```

**Objective-C class**

```c
```

**Parameters**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Fixed** | 0 = Font is proportional or variable width  
 |        | 1 = Font is fixed width, all glyphs have the same width                  |
| **Serif** | 0 = Glyphs do not have serifs (short strokes drawn at an angle on the top and bottom of glyph stems)  
 |        | 1 = Glyphs have serifs                                                  |
| **Symbolic** | 0 = Font contains glyphs in the standard Latin character set  
 |        | 1 = Font contains symbols                                                |
| **Script** | 0 = Font contains regular glyphs                                           
 |        | 1 = Glyphs resemble cursive handwriting                                    |
| **Italic** | 0 = Regular font                                                          
 |        | 1 = Glyphs have dominant vertical strokes that are slanted               |
| **AllCap** | 0 = Font contains lowercase letters                                         
 |        | 1 = Font contains only uppercase letters                                   |
| **SmallCap** | 0 = Regular font                                                          
 |        | 1 = Lowercase glyphs look like the corresponding uppercase glyphs but are smaller in size |
| **ForceBold** | 0 = Regular font                                                          
 |        | 1 = Force font to be rendered with a bold effect even at small sizes     |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A font has not been selected</td>
</tr>
<tr>
<td>1</td>
<td>The font flags were set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldAlignment

Form fields

Description

Sets the alignment for the specified form field.

Syntax

**Dylib**

```c
int DPLSetFormFieldAlignment(int InstanceID, int Index, int Alignment);
```

**Objective-C class**

```objective-c
- (int)SetFormFieldAlignment:(int)Index :(int)Alignment;
```

Parameters

<table>
<thead>
<tr>
<th><strong>Index</strong></th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
</table>
| **Alignment** | The alignment to use for the form field:  
0 = Left alignment  
1 = Centered  
2 = Right alignment |

Return values

| **0** | The form field index was invalid |
| **1** | The alignment of the form field was set successfully |
SetFormFieldAnnotFlags

**Form fields**

**Description**

Set the "annotation" flags for the specified form field. This is for advanced use, see the PDF specification for details.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldAnnotFlags(int InstanceID, int Index, int NewFlags);
```

**Objective-C class**

```objective-c
- (int)SetFormFieldAnnotFlags:(int)Index :(int)NewFlags;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to change</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewFlags</td>
<td>The new flags value to apply</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The specified form field could not be found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The &quot;annotation&quot; flags for the specified form field were set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldBackgroundColor

Form fields, Color

**Description**

Sets the background color of the specified form field. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

**Syntax**

**Dylib**

```
int DPLSetFormFieldBackgroundColor(int InstanceID, int Index, double Red, double Green, double Blue);
```

**Objective-C class**

```
- (int)SetFormFieldBackgroundColor:(int)Index :(double)Red :(double)Green :(double)Blue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>The red component of the color</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>

**Return values**

| 0        | The form field could not be found or the parameters were invalid. |
| 1        | The background color of the form field was set successfully |
SetFormFieldBackgroundColorCMYK

Description

Sets the background color of the specified form field. Similar to the SetFormFieldBorderColor function, but the color components are specified in the CMYK color space (Cyan, Magenta, Yellow, Black). The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

int DPLSetFormFieldBackgroundColorCMYK(int InstanceID, int Index, double C, double M, double Y, double K);

Objective-C class

- (int)SetFormFieldBackgroundColorCMYK:(int)Index :(double)C :(double)M :(double)Y :(double)K;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>The cyan component of the color</td>
</tr>
<tr>
<td>M</td>
<td>The magenta component of the color</td>
</tr>
<tr>
<td>Y</td>
<td>The yellow component of the color</td>
</tr>
<tr>
<td>K</td>
<td>The black component of the color</td>
</tr>
</tbody>
</table>

Return values

| 0     | The form field could not be found |
| 1     | The background color of the specified form field was set successfully |
**SetFormFieldBackgroundColorGray**

**Form fields, Color**

**Description**

Sets the background color of the specified form field. Similar to the `SetFormFieldBackgroundColor` function, but a single color component is specified in the Gray color space. Possible values are in the range 0 to 1.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldBackgroundColorGray(int InstanceID, int Index, double Gray);
```

**Objective-C class**

```objectivec
- (int)SetFormFieldBackgroundColorGray:(int)Index :(double)Gray;
```

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>Gray</td>
<td>The gray component</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The background color of the specified form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldBackgroundColorSep
Form fields, Color

Description

Sets the background color of the specified form field. Similar to the SetFormFieldBorderColor function, but a tint of a separation color added with the AddSeparationColor function is used. The PDF specification does not support separation color spaces for form fields, so the results may not always work, especially if the form field is later edited in Acrobat. This feature has been added for situations where the form field will be flattened.

Syntax

Dylib

```c
int DPLSetFormFieldBackgroundColorSep(int InstanceID, int Index, wchar_t * ColorName, double Tint);
```

Objective-C class

```c
-(int)SetFormFieldBackgroundColorSep:(int)Index :(NSString *)ColorName
:(double)Tint;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorName</td>
<td>The name of the separation color that was used with the AddSeparationColor function</td>
</tr>
<tr>
<td>Tint</td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

Return values

| 0 | The form field could not be found, or the separation color name could not be found |
| 1 | The background color of the specified form field was set successfully |
SetFormFieldBorderColor

Form fields, Color

Description

Sets the border color of the specified form field. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLSetFormFieldBorderColor(int InstanceID, int Index, double Red, double Green, double Blue);
```

Objective-C class

```objc
- (int)SetFormFieldBorderColor:(int)Index :(double)Red :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>Red</td>
<td>The red component of the color</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found or the parameters were invalid</td>
</tr>
<tr>
<td>1</td>
<td>The border color of the form field was set successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldBorderColorCMYK**

**Form fields, Color**

**Description**

Sets the border color of the specified form field. Similar to the `SetFormFieldBorderColor` function, but the color components are specified in the CMYK color space (Cyan, Magenta, Yellow, Black). The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldBorderColorCMYK(int InstanceID, int Index, double C,
    double M, double Y, double K);
```

**Objective-C class**

```objc
- (int)SetFormFieldBorderColorCMYK:(int)Index :(double)C :(double)M
    :(double)Y :(double)K;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>C</td>
<td>The amount of cyan for the color. 0 indicates no cyan, 1 indicates maximum cyan.</td>
</tr>
<tr>
<td>M</td>
<td>The amount of magenta for the color. Equivalent to the separation color. 0 indicates no magenta, 1 indicates maximum magenta.</td>
</tr>
<tr>
<td>Y</td>
<td>The amount of yellow for the color. 0 indicates no yellow, 1 indicates maximum yellow.</td>
</tr>
<tr>
<td>K</td>
<td>The amount of black for the color. 0 indicates no black, 1 indicates maximum black.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The border color of the specified form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldBorderColorGray
Form fields, Color

Description

Sets the border color of the specified form field. Similar to the SetFormFieldBorderColor function, but a single color component is specified in the Gray color space. Possible values are in the range 0 to 1.

Syntax

Dylib

```
int DPLSetFormFieldBorderColorGray(int InstanceID, int Index, double Gray);
```

Objective-C class

```
-(int)SetFormFieldBorderColorGray:(int)Index :(double)Gray;
```

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>Gray</td>
<td>The gray component</td>
</tr>
</tbody>
</table>

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The background color of the specified form field was set successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldBorderColorSep**

**Form fields, Color**

**Description**

Sets the border color of the specified form field. Similar to the `SetFormFieldBorderColor` function, but a tint of a separation color added with the `AddSeparationColor` function is used. The PDF specification does not support separation color spaces for form fields, so the results may not always work, especially if the form field is later edited in Acrobat. This feature has been added for situations where the form field will be flattened.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldBorderColorSep(int InstanceID, int Index, wchar_t * ColorName, double Tint);
```

**Objective-C class**

```objective-c
-(int)SetFormFieldBorderColorSep:(int)Index :(NSString *)ColorName :(double)Tint;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>ColorName</td>
<td>The name of the separation color that was used with the <code>AddSeparationColor</code> function</td>
</tr>
<tr>
<td>Tint</td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found, or the separation color name could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The border color of the specified form field was set successfully</td>
</tr>
</tbody>
</table>

SetFormFieldBorderStyle

Description

Sets the width and line style of the specified form field's border.

Syntax

Dylib

```c
int DPLSetFormFieldBorderStyle(int InstanceID, int Index, double Width,
                                int Style, double DashOn, double DashOff);
```

Objective-C class

```objective-c
- (int)SetFormFieldBorderStyle:(int)Index :(double)Width :(int)Style
                           :(double)DashOn :(double)DashOff;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the border</td>
</tr>
<tr>
<td>Style</td>
<td>The style of the border:</td>
</tr>
<tr>
<td></td>
<td>0 = Solid</td>
</tr>
<tr>
<td></td>
<td>1 = Dashed</td>
</tr>
<tr>
<td></td>
<td>2 = Beveled</td>
</tr>
<tr>
<td></td>
<td>3 = Inset</td>
</tr>
<tr>
<td></td>
<td>Anything else = Solid</td>
</tr>
<tr>
<td>DashOn</td>
<td>The length of the dash. Only valid if the border style is &quot;dashed&quot;.</td>
</tr>
<tr>
<td>DashOff</td>
<td>The length of the space between dashes. Only valid if the border style is &quot;dashed&quot;.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found or the parameters were invalid</td>
</tr>
<tr>
<td>1</td>
<td>The border style of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldBounds

Form fields

Description

Changes the physical size and position of the specified form field.

Syntax

Dylib

```c
int DPLSetFormFieldBounds(int InstanceID, int Index, double Left,
    double Top, double Width, double Height);
```

Objective-C class

```objc
- (int)SetFormFieldBounds:(int)Index :(double)Left :(double)Top
    :(double)Width :(double)Height;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field to adjust</td>
</tr>
<tr>
<td><strong>Left</strong></td>
<td>The new co-ordinate of the left edge of the form field</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The new co-ordinate of the top of the form field</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The new width of the form field</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The new height of the form field</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td>The form field could not be found</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td>The form field was resized and moved successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldCalcOrder**

**Form fields**

**Description**

Sets or changes the calculation order for form fields.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldCalcOrder(int InstanceID, int Index, int Order);
```

**Objective-C class**

```objectivec
- (int)SetFormFieldCalcOrder:(int)Index :(int)Order;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to add to the list of calculated field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>The order this field should be calculated in. A value of 0 means this field is the first field to be calculated. A value of 1 means this field is the second field to be calculated. Use a value of -1 to specify this field should be calculated last out of the fields which have already been added to the calculation order list.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The specified form field could not be found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The specified form field was added to the calculation order list, or moved to the new position if it was already in the list</td>
</tr>
</tbody>
</table>
SetFormFieldCaption

Form fields

Description

Sets the caption of the form field. This applies to buttons, checkboxes and radiobutton form fields only.

Syntax

Dylib

```c
int DPLSetFormFieldCaption(int InstanceID, int Index, wchar_t * NewCaption);
```

Objective-C class

```objective-c
-(int)SetFormFieldCaption:(int)Index :(NSString *)NewCaption;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewCaption</td>
<td>The new caption for the form field</td>
</tr>
</tbody>
</table>

Return values

| 0   | The form field could not be found or the parameters were invalid |
| 1   | The caption of the form field was set successfully |
SetFormFieldCheckStyle

Form fields

Description

Sets the check style for checkbox fields or radio-button sub-fields.

Syntax

Dylib

```c
int DPLSetFormFieldCheckStyle(int InstanceID, int Index, int CheckStyle,
int Position);
```

Objective-C class

```c
- (int)SetFormFieldCheckStyle:(int)Index :(int)CheckStyle :(int)Position;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to work with. The first form field has an index of 1.</td>
</tr>
</tbody>
</table>
| CheckStyle | 0 = Cross  
             1 = Check (Tick)  
             2 = Dot (Radio)  
             3 = XP check  
             4 = XP Radio  
             5 = Diamond  
             6 = Square  
             7 = Star |
| Position   | 0 = Left align  
             1 = Center  
             2 = Right align |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>One of the parameters was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The check style was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldChildTitle

Form fields

Description

Sets the title of the specified form field. For form fields arranged in a hierarchy this function only sets the last part of the field name. For example, a field with the name "Address.ZipCode" can be changed to "Address.PostalCode".

Syntax

Dylib

```c
int DPLSetFormFieldChildTitle(int InstanceID, int Index, 
    wchar_t * NewTitle);
```

Objective-C class

```objective-c
- (int)SetFormFieldChildTitle:(int)Index :(NSString *)NewTitle;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field to set the title of</td>
</tr>
<tr>
<td><strong>NewTitle</strong></td>
<td>The new value for the last part of the title for the specified field.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The title of the specified form field was changed successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldChoiceSub**

**Form fields**

**Description**

Sets the export and display values of an existing sub-field that is part of a choice form field. If the display value is an empty string then it will be set to the same string as the export value. The `AddFormFieldChoiceSub` function can be used to change a sub-field entry in an existing choice form field.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldChoiceSub(int InstanceID, int Index, int SubIndex, wchar_t * SubName, wchar_t * DisplayName);
```

**Objective-C class**

```c
- (int)SetFormFieldChoiceSub:(int)Index :(int)SubIndex :(NSString *)SubName :(NSString *)DisplayName;
```

**Parameters**

- **Index**
  The index of the choice form field

- **SubIndex**
  The index of the sub-field. The first sub-field has an index of 1.

- **SubName**
  The export value of the new sub-field.

- **DisplayName**
  The display value of the new sub-field.

**Return values**

- **0**
  The sub-field was not added. The specified form field may not have been a choice form field.

- **1**
  The form field was updated successfully.
SetFormFieldChoiceType

Form fields

Description

Sets a choice form field to be a combo box or list box.

Syntax

Dylib

```
int DPLSetFormFieldChoiceType(int InstanceID, int Index, int ChoiceType);
```

Objective-C class

```
-(int)SetFormFieldChoiceType:(int)Index :(int)ChoiceType;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
| ChoiceType| 1 = Set the form field to be a scrollable list box  
          | 2 = Set the form field to be a drop-down combo box  
          | 3 = Set the form field to be a multiselect scrollable list box 
          | 4 = Set the form field to be a drop-down combo box with edit box |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The field was not changed</td>
</tr>
<tr>
<td>1</td>
<td>The field was changed successfully</td>
</tr>
</tbody>
</table>

**SetFormFieldColor**

**Form fields, Color**

**Description**

Sets the color of the text in the form field. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldColor(int InstanceID, int Index, double Red, double Green, double Blue);
```

**Objective-C class**

```objective-c
-(int)SetFormFieldColor:(int)Index :(double)Red :(double)Green :(double)Blue;
```

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field to work with. The first form field has an index of 1.</td>
</tr>
<tr>
<td><strong>Red</strong></td>
<td>The red component of the color</td>
</tr>
<tr>
<td><strong>Green</strong></td>
<td>The green component of the color</td>
</tr>
<tr>
<td><strong>Blue</strong></td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td>The form field could not be found</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td>The form field text color was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldColorCMYK

Form fields, Color

**Description**
Sets the color of the text in the specified form field. Similar to the `SetFormFieldBorderColor` function, but the color components are specified in the CMYK color space (Cyan, Magenta, Yellow, Black). The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldColorCMYK(int InstanceID, int Index, double C,
                             double M, double Y, double K);
```

**Objective-C class**

```objective-c
- (int)SetFormFieldColorCMYK:(int)Index :(double)C :(double)M :(double)Y
  :(double)K;
```

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>C</td>
<td>The cyan component of the color</td>
</tr>
<tr>
<td>M</td>
<td>The magenta component of the color</td>
</tr>
<tr>
<td>Y</td>
<td>The yellow component of the color</td>
</tr>
<tr>
<td>K</td>
<td>The black component of the color</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The text color of the specified form field was set successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldColorSep**

**Form fields, Color**

**Description**

Sets the color of the text in the specified form field. Similar to the `SetFormFieldBorderColor` function, but a tint of a separation color added with the `AddSeparationColor` function is used. The PDF specification does not support separation color spaces for form fields, so the results may not always work, especially if the form field is later edited in Acrobat. This feature has been added for situations where the form field will be flattened.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldColorSep(int InstanceID, int Index, wchar_t * ColorName, double Tint);
```

**Objective-C class**

```c
-(int)SetFormFieldColorSep:(int)Index :(NSString *)ColorName :(double)Tint;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorName</td>
<td>The name of the separation color that was used with the <code>[f:AddSeparationColor]</code> function</td>
</tr>
<tr>
<td>Tint</td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

**Return values**

| 0 | The form field could not be found, or the separation color name could not be found |
| 1 | The text color of the specified form field was set successfully |
SetFormFieldComb

Form fields

Description

Marks a form field as a comb field, where each character in the value occupies the same space in the field. The field must be a test field, and the SetFormFieldMaxLen function must be called to specify the number of characters in the field.

Syntax

Dylib

```c
int DPLSetFormFieldComb(int InstanceID, int Index, int Comb);
```

Objective-C class

```objective-c
-(int)SetFormFieldComb:(int)Index :(int)Comb;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
| **Comb**  | 0 = Regular field  
 1 = Comb field |
SetFormFieldDefaultValue

Form fields

Description

Sets the default value of the field. This is the value which is shown when the reset button is pressed, if one is on the form.

Syntax

Dylib

```c
int DPLSetFormFieldDefaultValue(int InstanceID, int Index, wchar_t * NewDefaultValue);
```

Objective-C class

```objc
-(int)SetFormFieldDefaultValue:(int)Index :(NSString *)NewDefaultValue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to change</td>
</tr>
<tr>
<td>NewDefaultValue</td>
<td>The new default value for the form field. For multi-line text fields you can use Chr(13) or Chr(13) + Chr(10) to force a line feed between lines.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The default value of the specified form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldDescription

Form fields

Description

Sets the description of the specified form field.

Syntax

Dylib

```c
int DPLSetFormFieldDescription(int InstanceID, int Index,
    wchar_t * NewDescription);
```

Objective-C class

```objective-c
- (int)SetFormFieldDescription:(int)Index :(NSString *)NewDescription;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to change</td>
</tr>
<tr>
<td>NewDescription</td>
<td>The new description.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The specified form field’s description was set successfultly</td>
</tr>
</tbody>
</table>
SetFormFieldFlags

Description

Sets the internal flags for the form field. This setting is for advanced purposes and most users will not need to use it.

Syntax

**Dylib**

```c
int DPLSetFormFieldFlags(int InstanceID, int Index, int NewFlags);
```

**Objective-C class**

```objective-c
- (int)SetFormFieldFlags:(int)Index :(int)NewFlags;
```

Parameters

<table>
<thead>
<tr>
<th><strong>Index</strong></th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NewFlags</strong></td>
<td>The new value of the flags. Consult the PDF specification for further details.</td>
</tr>
</tbody>
</table>

Return values

| **0** | Cannot find the form field |
| **1** | The flags were set successfully |
SetFormFieldFont

Form fields

Description

Sets the font that the specified form field must use.

Syntax

Dylib

```c
int DPLSetFormFieldFont(int InstanceID, int Index, int FontIndex);
```

Objective-C class

```objective-c
- (int)SetFormFieldFont:(int)Index :(int)FontIndex;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontIndex</td>
<td>The index of the font to use. The first font in the form has an index of 1. Use GetFormFontCount to determine the number of fonts available in the form.</td>
</tr>
</tbody>
</table>

Return values

| 0 | Bad font index or form field not found |
| 1 | Font was set successfully |
**SetFormFieldHighlightMode**

**Form fields**

**Description**

Sets the highlight mode for the specified form field.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldHighlightMode(int InstanceID, int Index, int NewMode);
```

**Objective-C class**

```objc
- (int)SetFormFieldHighlightMode:(int)Index :(int)NewMode;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>NewMode</td>
<td>The highlighting mode:</td>
</tr>
<tr>
<td></td>
<td>0 = None</td>
</tr>
<tr>
<td></td>
<td>1 = Invert</td>
</tr>
<tr>
<td></td>
<td>2 = Outline</td>
</tr>
<tr>
<td></td>
<td>3 = Push</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found or the parameters were invalid</td>
</tr>
<tr>
<td>1</td>
<td>The highlight mode of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldIcon

**Description**

Sets the icon of a button form field. To create an icon: add a new page to the document, set the size and draw images or text onto the page, and then capture the page using the `CapturePage` function. For a "down" or "rollover" icon to be displayed correctly the form field's highlight mode must be set to "push", see the `SetFormFieldHighlightMode` function.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldIcon(int InstanceID, int Index, int IconType, int CaptureID);
```

**Objective-C class**

```objective-c
- (int)SetFormFieldIcon:(int)Index :(int)IconType :(int)CaptureID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>IconType</td>
<td>The type of icon to assign:</td>
</tr>
<tr>
<td></td>
<td>0 = Normal icon</td>
</tr>
<tr>
<td></td>
<td>1 = Rollover icon</td>
</tr>
<tr>
<td></td>
<td>2 = Down icon</td>
</tr>
<tr>
<td>CaptureID</td>
<td>The ID returned by the <code>CapturePage</code> function</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found or the parameters were invalid</td>
</tr>
<tr>
<td>1</td>
<td>The specified icon of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldIconStyle

Form fields

Description

Sets the position, scaling and layout of a button form field's icon. These parameters apply to all the icons assigned to a button (up, down and rollover).

Syntax

Dylib

```c
int DPLSetFormFieldIconStyle(int InstanceID, int Index, int Placement,
   int Scale, int ScaleType, int HorizontalShift,
   int VerticalShift);
```

Objective-C class

```objc
- (int)SetFormFieldIconStyle:(int)Index :(int)Placement :(int)Scale
   :(int)ScaleType :(int)HorizontalShift :(int)VerticalShift;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Placement</td>
<td>The icon placement: 0 = No icon; caption only 1 = No caption; icon only 2 = Caption below the icon 3 = Caption above the icon 4 = Caption to the right of the icon 5 = Caption to the left of the icon 6 = Caption overlaid directly on the icon</td>
</tr>
<tr>
<td>Scale</td>
<td>The conditions under which to scale the icon: 0 = Always scale 1 = Only scale when the icon is bigger than the button 2 = Only scale when the icon is smaller than the button 3 = Never scale</td>
</tr>
<tr>
<td>ScaleType</td>
<td>The type of scaling to use: 0 = Ignore aspect ratio 1 = Maintain aspect ratio</td>
</tr>
<tr>
<td>HorizontalShift</td>
<td>The percentage of space placed to the left of the icon, for example: 0 = Align left 50 = Center horizontally 100 = Align right</td>
</tr>
<tr>
<td>VerticalShift</td>
<td>The percentage of space placed beneath the icon, for example: 0 = Align bottom 50 = Center vertically 100 = Align top</td>
</tr>
</tbody>
</table>

Return values

| 0 | The form field could not be found or the parameters were invalid |
| 1 | The icon style of the form field was set successfully |
SetFormFieldMaxLen

Form fields

Description

Sets the maximum number of characters that will be accepted for the specified text form field.

Syntax

Dylib

```c
int DPLSetFormFieldMaxLen(int InstanceID, int Index, int NewMaxLen);
```

Objective-C class

```objective-c
- (int)SetFormFieldMaxLen:(int)Index :(int)NewMaxLen;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewMaxLen</td>
<td>The new maximum length to use for the form field</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The form field index was invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The maximum length of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldNoExport

Form fields

Description

Sets the state of a field's NoExport flag.
The field will not be exported by a submit-form action if the NoExport flag is set.

Syntax

Dylib

```c
int DPLSetFormFieldNoExport(int InstanceID, int Index, int NoExport);
```

Objective-C class

```c
- (int)SetFormFieldNoExport:(int)Index :(int)NoExport;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
<tr>
<td>NoExport</td>
<td>0 = Clear the field's NoExport flag</td>
</tr>
<tr>
<td></td>
<td>1 = Set the field's NoExport flag</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not find the specified form field</td>
</tr>
<tr>
<td>1</td>
<td>The NoExport flag was set successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldOptional**

*Form fields, Content Streams and Optional Content Groups*

**Description**

Adds a form field to an optional content group.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldOptional(int InstanceID, int Index,
int OptionalContentGroupID);
```

**Objective-C class**

```objc
-(int)SetFormFieldOptional:(int)Index :(int)OptionalContentGroupID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
| OptionalContentGroupID | An ID returned by the [NewOptionalContentGroup](#),
                          | [GetOptionalContentGroupID](#) or
                          | [GetOptionalContentConfigOrderItemID](#) functions                       |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The OptionalContentGroupID or Index parameter was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The field was added to the optional content group successfully</td>
</tr>
</tbody>
</table>
SetFormFieldPage

Form fields

Description

Moves the specified form field onto another page.

Syntax

Dylib

```c
int DPLSetFormFieldPage(int InstanceID, int Index, int NewPage);
```

Objective-C class

```c
-(int)SetFormFieldPage:(int)Index :(int)NewPage;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to move</td>
</tr>
<tr>
<td>NewPage</td>
<td>The page number to move the form field to</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Can't find the form field or the new destination page is invalid</td>
</tr>
<tr>
<td>1</td>
<td>Form field moved successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldPrintable**

**Form fields**

**Description**

Set whether the specified form field should be printed or not.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldPrintable(int InstanceID, int Index, int Printable);
```

**Objective-C class**

```objective-c
- (int)SetFormFieldPrintable:(int)Index :(int)Printable;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to change</td>
</tr>
</tbody>
</table>
| Printable | 0 = Do not print  
|           | 1 = Print |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The printable flag of the specified form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldReadOnly

Form fields

Description

Sets the state of a field's ReadOnly flag.
The user cannot change the value of a form field if the ReadOnly flag is set.

Syntax

Dylib

```c
int DPLSetFormFieldReadOnly(int InstanceID, int Index, int ReadOnly);
```

Objective-C class

```objc
-(int)SetFormFieldReadOnly:(int)Index :(int)ReadOnly;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>
| ReadOnly   | 0 = Clear the field's ReadOnly flag  
|            | 1 = Set the field's ReadOnly flag   |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not find the specified form field</td>
</tr>
<tr>
<td>1</td>
<td>The ReadOnly flag was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldRequired

Form fields

Description

Sets the state of a field's is Required flag.
If this flag is set the field must have a value when the form is exported by a submit-form action.

Syntax

Dylib

```c
int DPLSetFormFieldRequired(int InstanceID, int Index, int Required);
```

Objective-C class

```c
-(int)SetFormFieldRequired:(int)Index :(int)Required;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>
| Required | 0 = Clear the field's Required flag  
|         | 1 = Set the field's Required flag |

Return values

| 0 | Could not find the specified form field |
| 1 | The Required flag was set successfully |
**SetFormFieldResetAction**

**Form fields**

**Description**

Adds a reset action to a button form field. When actioned all formfields will be reset to their default values.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldResetAction(int InstanceID, int Index, wchar_t * ActionType);
```

**Objective-C class**

```c
- (int)SetFormFieldResetAction:(int)Index :(NSString *)ActionType;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>
| ActionType | The action type:  
E = An action to be performed when the cursor enters the annotation's active area  
X = An action to be performed when the cursor exits the annotation's active area  
D = An action to be performed when the mouse button is pressed inside the annotation's active area  
U = An action to be performed when the mouse button is released inside the annotation's active area  
Fo = An action to be performed when the annotation receives the input focus  
Bl = An action to be performed when the annotation loses the input focus (blurred)  
K = An action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This allows the keystroke to be checked for validity and rejected or modified.  
F = An action to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.  
V = An action to be performed when the field's value is changed. This allows the new value to be checked for validity.  
C = An action to be performed in order to recalculate the value of this field when that of another field changes |

**Return values**

| 0 | Could not set the field action |
| 1 | Success |
**SetFormFieldRichTextString**

**Form fields**

**Description**

Sets the rich text (RV) or default style (DS) string of the specified form field using the given key.

**Syntax**

**Dylib**

```
int DPLSetFormFieldRichTextString(int InstanceID, int Index,
                                wchar_t * Key, wchar_t * NewValue);
```

**Objective-C class**

```
- (int)SetFormFieldRichTextString:(int)Index :(NSString *)Key
    :(NSString *)NewValue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the required form field. The first form field has an index of 1.</td>
</tr>
<tr>
<td><strong>Key</strong></td>
<td>The Key used to set the required string</td>
</tr>
<tr>
<td></td>
<td>&quot;RV&quot; = sets the rich text string</td>
</tr>
<tr>
<td></td>
<td>&quot;DS&quot; = sets the default style string</td>
</tr>
<tr>
<td><strong>NewValue</strong></td>
<td>The new value for the specified key. The required format of the input string is defined in the PDF Specification under the section titled &quot;Field Dictionaries&quot;.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not find the specified form field</td>
</tr>
<tr>
<td>1</td>
<td>The specified value of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldRotation

Form fields

Description

Sets the rotation of a form field anti-clockwise relative to the page.

Syntax

Dylib

```c
int DPLSetFormFieldRotation(int InstanceID, int Index, int Angle);
```

Objective-C class

```c
- (int)SetFormFieldRotation:(int)Index :(int)Angle;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angle</td>
<td>The angle to rotate the field by. Must be one of the following values: 0, 90, 180 or 270.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The form field could not be found or the specified angle was not valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The rotation of the specified form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldSignatureImage

Image handling, Form fields, Security and Signatures

Description

Sets the visual appearance of a signature form field to use the specified image.

Syntax

Dylib

```c
int DPLSetFormFieldSignatureImage(int InstanceID, int Index, int ImageID, int Options);
```

Objective-C class

```c
- (int)SetFormFieldSignatureImage:(int)Index :(int)ImageID :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the signature form field to work with. The first form field has an index of 1.</td>
</tr>
<tr>
<td>ImageID</td>
<td>A valid image ID as returned by the <code>SelectedImage</code> or <code>GetImageID</code> functions.</td>
</tr>
<tr>
<td>Options</td>
<td>0 = The image is stretched in both directions to fill the field size without any rotation</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field was not a signature field, the Index parameter was out of range or the ImageID parameter was invalid.</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetFormFieldStandardFont
Fonts, Form fields

Description
Sets a form field to use a standard font. A standard font must be used in Acrobat 4 and earlier if the form field contains a border or is rotated.

Syntax
Dylib
int DPLSetFormFieldStandardFont(int InstanceID, int Index, int StandardFontID);

Objective-C class
- (int)SetFormFieldStandardFont:(int)Index :(int)StandardFontID;

Parameters
<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field</th>
</tr>
</thead>
</table>
| StandardFontID | The ID of the font to add:  
| 0 = Courier  
| 1 = CourierBold  
| 2 = CourierBoldOblique  
| 3 = CourierOblique  
| 4 = Helvetica  
| 5 = HelveticaBold  
| 6 = HelveticaBoldOblique  
| 7 = HelveticaOblique  
| 8 = TimesRoman  
| 9 = TimesBold  
| 10 = TimesItalic  
| 11 = TimesBoldItalic  
| 12 = Symbol  
| 13 = ZapfDingbats |
SetFormFieldSubmitAction

Form fields

Description

Adding a submit action to a button form field.

Syntax

**Dylib**

```c
int DPLSetFormFieldSubmitAction(int InstanceID, int Index, wchar_t * ActionType, wchar_t * Link);
```

**Objective-C class**

```objc
- (int)SetFormFieldSubmitAction:(int)Index :(NSString *)ActionType :(NSString *)Link;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
| **ActionType** | The action type:  
  E = An action to be performed when the cursor enters the annotation’s active area  
  X = An action to be performed when the cursor exits the annotation’s active area  
  D = An action to be performed when the mouse button is pressed inside the annotation’s active area  
  U = An action to be performed when the mouse button is released inside the annotation’s active area  
  Fo = An action to be performed when the annotation receives the input focus  
  Bl = An action to be performed when the annotation loses the input focus (blurred)  
  K = An action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This allows the keystroke to be checked for validity and rejected or modified.  
  F = An action to be performed before the field is formatted to display its current value. This allows the field’s value to be modified before formatting.  
  V = An action to be performed when the field’s value is changed. This allows the new value to be checked for validity.  
  C = An action to be performed in order to recalculate the value of this field when that of another field changes |
| **Link** | The URL of the server script that will process the form submission |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not set the field action</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetFormFieldSubmitActionEx

Form fields

Description

Adds a submit action to a button form field with a flags parameter for setting various submit options. Please refer to section "Form Actions" of the official PDF Specifications.

Syntax

Dylib

```c
int DPLSetFormFieldSubmitActionEx(int InstanceID, int Index, wchar_t * ActionType, wchar_t * Link, int Flags);
```

Objective-C class

```objective-c
- (int)SetFormFieldSubmitActionEx:(int)Index :(NSString *)ActionType : (NSString *)Link :(int)Flags;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field</td>
</tr>
</tbody>
</table>
| ActionType| The action type:  
E = An action to be performed when the cursor enters the annotation's active area  
X = An action to be performed when the cursor exits the annotation's active area  
D = An action to be performed when the mouse button is pressed inside the annotation's active area  
U = An action to be performed when the mouse button is released inside the annotation's active area  
Fo = An action to be performed when the annotation receives the input focus  
Bl = An action to be performed when the annotation loses the input focus (blurred)  
K = An action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This allows the keystroke to be checked for validity and rejected or modified.  
F = An action to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.  
V = An action to be performed when the field's value is changed. This allows the new value to be checked for validity.  
C = An action to be performed in order to recalculate the value of this field when that of another field changes |
| Link      | The URL of the server script that will process the form submission. |
| Flags     | Adobe defined flags value for the formfield submit action. |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not set the field action</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetFormFieldTabOrder

Form fields

Description

Sets the tab order of the specified form field. A position of 1 indicates that the form field is the first field on the page.

If you use this function then you should call SetTabOrderMode with 'S' to set the tabbing mode to Structure mode.

Syntax

**Dylib**

```c
int DPLSetFormFieldTabOrder(int InstanceID, int Index, int Order);
```

**Objective-C class**

```objective-c
-(int)SetFormFieldTabOrder:(int)Index :(int)Order;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The index of the form field that should be moved to a new position in the tab order</td>
</tr>
<tr>
<td><strong>Order</strong></td>
<td>The new position this form field should be in the tab order. The first position in the tab order has a value of 1.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found or the new tab order was out of range</td>
</tr>
<tr>
<td>1</td>
<td>The tab order of the specified form field was updated successfully</td>
</tr>
</tbody>
</table>
SetFormFieldTextFlags

Form fields

Description

Sets various options for text form fields.

Syntax

Dylib

```
int DPLSetFormFieldTextFlags(int InstanceID, int Index, int Multiline,
   int Password, int FileSelect, int DoNotSpellCheck,
   int DoNotScroll);
```

Objective-C class

```
-(int)SetFormFieldTextFlags:(int)Index :(int)Multiline :(int)Password
   :(int)FileSelect :(int)DoNotSpellCheck :(int)DoNotScroll;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the form field to work with. The first form field has an index of 1.</td>
</tr>
</tbody>
</table>
| Multiline     | 0 = Field's text is restricted to one line
|               | 1 = Field may contain multiple lines of text                               |
| Password      | 0 = The field is not a password field                                       |
|               | 1 = The field is a password, characters will be displayed as asterisks      |
| FileSelect    | 0 = The field is not a file select field                                    |
|               | 1 = The contents of the file specified by the text entered in this field will be submitted as the value of the form field |
| DoNotSpellCheck | 0 = The field will be spell checked                                       |
|               | 1 = The field will not be spell checked                                    |
| DoNotScroll   | 0 = Field can scroll                                                       |
|               | 1 = Field is not allowed to scroll                                          |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The options for the text field were set successfully</td>
</tr>
</tbody>
</table>
**SetFormFieldTextSize**

**Text, Form fields**

**Description**

Sets the size of the text in the specified form field. A value of 0 indicates that the form field autosizes the text to fit into the available space.

**Syntax**

**Dylib**

```c
int DPLSetFormFieldTextSize(int InstanceID, int Index, double NewTextSize);
```

**Objective-C class**

```c
- (int)SetFormFieldTextSize:(int)Index :(double)NewTextSize;
```

**Parameters**

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewTextSize</td>
<td>The new size in points of the form field's font</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The form field font size was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldTitle

Form fields

Description

Renames the title of a parent form field. No validation is performed so you should make sure the title is unique.

Syntax

Dylib

```c
int DPLSetFormFieldTitle(int InstanceID, int Index, wchar_t * NewTitle);
```

Objective-C class

```objective-c
- (int)SetFormFieldTitle:(int)Index :(NSString *)NewTitle;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the formfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewTitle</td>
<td>The new title name for the formfield</td>
</tr>
</tbody>
</table>

Return values

| 0    | The form field title could not be changed |
| 1    | The field field title was changed successfully |
SetFormFieldValue

Form fields

Description
Sets the initial value of a form field. The appearance stream for the form field is generated if
SetNeedAppearances(1) has been called.

Syntax

Dylib
int DPLSetFormFieldValue(int InstanceID, int Index, wchar_t * NewValue);

Objective-C class
-(int)SetFormFieldValue:(int)Index :(NSString *)NewValue;

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the required form field. The first form field has an index of 1.</th>
</tr>
</thead>
</table>
| NewValue   | The new value of the form field. For multi-line text fields you can use Chr(13) or
            | Chr(13) + Chr(10) to force a line break.                                     |

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not find the specified form field</td>
</tr>
<tr>
<td>1</td>
<td>The default value of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldValueByTitle

Form fields

Description
Sets the value of all the form fields with the specified title. The appearance streams for the form fields are generated if `SetNeedAppearances(1)` has been called.

Syntax

Dylib

```c
int DPLSetFormFieldValueByTitle(int InstanceID, wchar_t * Title, wchar_t * NewValue);
```

Objective-C class

```objective-c
- (int)SetFormFieldValueByTitle:(NSString *)Title :(NSString *)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Title</th>
<th>The title of the form field to set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewValue</td>
<td>The new value of the form field. For multi-line text fields you can use Chr(13) or Chr(13) + Chr(10) to force a line feed between lines.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The form field could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The value of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetFormFieldVisible

Form fields

Description

Hides or shows a form field.

Syntax

Dylib

```c
int DPLSetFormFieldVisible(int InstanceID, int Index, int Visible);
```

Objective-C class

```Objective-C
- (int)SetFormFieldVisible:(int)Index :(int)Visible;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the required form field. The first form field has an index of 1.</td>
</tr>
</tbody>
</table>
| Visible   | 0 = Hide the form field  
|           | 1 = Show the form field |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Could not find the specified form field</td>
</tr>
<tr>
<td>1</td>
<td>The visibility of the form field was set successfully</td>
</tr>
</tbody>
</table>
SetGDIPlusFileName

Rendering and printing

Description
Sets the path and filename of the GDI+ DLL (gdiplus.dll) used by the various rendering functions. This can usually be left at the default, which means the DLL will most probably be stored in the Windows/System folder, but on webservers, etc. it may be necessary to store the file in a different location.

Syntax

Dylib

```c
int DPLSetGDIPlusFileName(int InstanceID, wchar_t * DLLFileName);
```

Objective-C class

```objective-c
- (int)SetGDIPlusFileName:(NSString *)DLLFileName;
```

Parameters

<table>
<thead>
<tr>
<th>DLLFileName</th>
<th>The path and file name of the GDI+ DLL, for example &quot;c:\dlls\gdiplus.dll&quot;.</th>
</tr>
</thead>
</table>

Return values

| 0 | The specified file could not be found |
| 1 | The GDI+ DLL file name was set successfully |
SetGDIPlusOptions
Rendering and printing

Description
Sets various options for the renderer when the GDI+ library is used. Options 10, 11 and 12 will override options 2 and 3 if they are set to anything other than 0.

Syntax
Dylib
int DPLSetGDIPlusOptions(int InstanceID, int OptionID, int NewValue);

Objective-C class
- (int)SetGDIPlusOptions:(int)OptionID :(int)NewValue;

Parameters

<table>
<thead>
<tr>
<th>OptionID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use of GDI+</td>
</tr>
<tr>
<td>1</td>
<td>Text/vector graphics smoothing</td>
</tr>
<tr>
<td>2</td>
<td>Interpolation</td>
</tr>
<tr>
<td>3</td>
<td>Image smoothing</td>
</tr>
<tr>
<td>4</td>
<td>Process null paths</td>
</tr>
<tr>
<td>5</td>
<td>Mono threshold</td>
</tr>
<tr>
<td>6</td>
<td>DLL piggyback</td>
</tr>
<tr>
<td>7</td>
<td>DLL startup</td>
</tr>
<tr>
<td>8</td>
<td>DLL suppress background thread</td>
</tr>
<tr>
<td>9</td>
<td>Enhance thin lines</td>
</tr>
<tr>
<td>10</td>
<td>GDIPlus SmoothingMode</td>
</tr>
<tr>
<td>11</td>
<td>GDIPlus InterpolationMode</td>
</tr>
<tr>
<td>12</td>
<td>GDIPlus PixelOffsetMode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use GDI+</td>
</tr>
<tr>
<td>1</td>
<td>Use GDI+ (default)</td>
</tr>
<tr>
<td>0</td>
<td>No smoothing</td>
</tr>
<tr>
<td>1</td>
<td>Smooth text and vector graphics (default)</td>
</tr>
<tr>
<td>0</td>
<td>Standard</td>
</tr>
<tr>
<td>1</td>
<td>Accurate (default)</td>
</tr>
<tr>
<td>0</td>
<td>Ignore</td>
</tr>
<tr>
<td>1</td>
<td>Process (default)</td>
</tr>
<tr>
<td>0</td>
<td>No thresholding (default)</td>
</tr>
<tr>
<td>1..255</td>
<td>Threshold level</td>
</tr>
<tr>
<td>0</td>
<td>DLL piggyback</td>
</tr>
<tr>
<td>7</td>
<td>DLL startup</td>
</tr>
<tr>
<td>0</td>
<td>DLL suppress background thread</td>
</tr>
<tr>
<td>0</td>
<td>Do not allow</td>
</tr>
<tr>
<td>1</td>
<td>Allow (reuse existing DLL instance)</td>
</tr>
<tr>
<td>0</td>
<td>Never call</td>
</tr>
<tr>
<td>1</td>
<td>Don't call if piggybacking on existing DLL</td>
</tr>
<tr>
<td>2</td>
<td>Always call</td>
</tr>
<tr>
<td>0</td>
<td>No (do not suppress)</td>
</tr>
<tr>
<td>1</td>
<td>Yes (suppress background thread)</td>
</tr>
<tr>
<td>0</td>
<td>Do nothing (default)</td>
</tr>
<tr>
<td>1 - 9</td>
<td>Thicken lines smaller than 1 device unit to thickness specified</td>
</tr>
<tr>
<td>0</td>
<td>smDefault, 1 = smHighSpeed, 2 = smHighQuality, 3 = smNone, 4 = smAntiAlias</td>
</tr>
<tr>
<td>0</td>
<td>imDefault, 1 = imLowQuality, 2 = imHighQuality, 3 = imBilinear, 4 = imBicubic, 5 = NearestNeighbor</td>
</tr>
<tr>
<td>0</td>
<td>imHighQualityBilinear, 7 = imHighQualityBicubic</td>
</tr>
<tr>
<td>0</td>
<td>poDefault, 1 = poHighSpeed, 2 = poHighQuality</td>
</tr>
<tr>
<td>4</td>
<td>poNone, 4 = poHalf</td>
</tr>
</tbody>
</table>
SetHTMLBoldFont

Text, HTML text

Description

 Specifies the font to use when a <b> or <strong> tag is encountered when using DrawHTMLText or DrawHTMLTextBox.

Syntax

**Dylib**

```
int DPLSetHTMLBoldFont(int InstanceID, wchar_t * FontSet, int FontID);
```

**Objective-C class**

```
- (int)SetHTMLBoldFont:(NSString *)FontSet :(int)FontID;
```

Parameters

| FontSet | The name of the font set to use. For this version of the library it should always be "Default". |
| FontID  | The ID of the font to use |

Return values

| 0       | The specified FontID is not a valid font |
| 1       | The font was set successfully |
SetHTMLBoldItalicFont

Text, HTML text

Description

Specifies the font to use when both `<b>` or `<strong>` and `<em>` or `<i>` tags are encountered when using `DrawHTMLText` or `DrawHTMLTextBox`.

Syntax

**Dylib**

```c
int DPLSetHTMLBoldItalicFont(int InstanceID, wchar_t * FontSet, int FontID);
```

**Objective-C class**

```c
-(int)SetHTMLBoldItalicFont:(NSString *)FontSet :(int)FontID;
```

Parameters

| FontSet | The name of the font set to use. For this version of the library it should always be "Default". |
| FontID  | The ID of the font to use |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FontID is not a valid font</td>
</tr>
<tr>
<td>1</td>
<td>The font was set successfully</td>
</tr>
</tbody>
</table>
SetHTMLItalicFont
Text, HTML text

Description
Specifies the font to use when an <em> or <i> tag is encountered when using DrawHTMLText or DrawHTMLTextBox.

Syntax
Dylib

```c
int DPLSetHTMLItalicFont(int InstanceID, wchar_t * FontSet, int FontID);
```

Objective-C class

```c
-(int)SetHTMLItalicFont:(NSString *)FontSet :(int)FontID;
```

Parameters

<table>
<thead>
<tr>
<th>FontSet</th>
<th>The name of the font set to use. For this version of the library it should always be &quot;Default&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontID</td>
<td>The ID of the font to use</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FontID is not a valid font</td>
</tr>
<tr>
<td>1</td>
<td>The font was set successfully</td>
</tr>
</tbody>
</table>
**SetHTMLNormalFont**

**Text, HTML text**

**Description**

Specifies the default font for text drawn using **DrawHTMLText** or **DrawHTMLTextBox**.

**Syntax**

**Dylib**

```c
int DPLSetHTMLNormalFont(int InstanceID, wchar_t * FontSet, int FontID);
```

**Objective-C class**

```objective-c
- (int)SetHTMLNormalFont:(NSString *)FontSet :(int)FontID;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontSet</td>
<td>The name of the font set to use. For this version of the library it should always be &quot;Default&quot;.</td>
</tr>
<tr>
<td>FontID</td>
<td>The ID of the font to use</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified FontID is not a valid font</td>
</tr>
<tr>
<td>1</td>
<td>The font was set successfully</td>
</tr>
</tbody>
</table>
SetHeaderCommentsFromString

Document properties

Description

Allows a binary string to be added between the file header and first objects. The string should start with a % character to indicate that it is a comment.

Syntax

**Dylib**

```c
int DPLSetHeaderCommentsFromString(int InstanceID, char * Source);
```

**Objective-C class**

```objective-c
- (int)SetHeaderCommentsFromString:(NSData *)Source;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>The new comments</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The header comments could not be set</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
**SetImageAsMask**  
*Image handling, Page layout*

**Description**

This function must be called to prepare the image before it is used as a mask for another image. The mask image must be a grayscale image, and be either 1-bit or 8-bit. Depending on your needs you may want to call `ReverseImage` which will reverse the effects of the mask. A soft-mask is just a normal image, so if you have an image setup as a stencil mask and no longer want it to be a mask just change it to a soft mask image (MaskType = 2).

**Syntax**

**Dylib**

```c
int DPLSetImageAsMask(int InstanceID, int MaskType);
```

**Objective-C class**

```c
-(int)SetImageAsMask:(int)MaskType;
```

**Parameters**

<table>
<thead>
<tr>
<th>MaskType</th>
<th>The type of mask to set this image as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stencil mask (only 1-bit images)</td>
</tr>
<tr>
<td>2</td>
<td>Soft mask (1-bit and 8-bit images)</td>
</tr>
</tbody>
</table>
**SetImageMask**

**Image handling, Page layout**

**Description**

Sets the mask for the selected image. This can be used to make parts of an image transparent when it is drawn with the `DrawImage` or `DrawScaledImage` functions.

The color range specified will become transparent. This works best with losslessly compressed images such as BMP or TIFF.

JPEG images use a lossy compression, so the image mask may cause halo effects.

The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color. For monochrome images only the red component will be used.

**Syntax**

**Dylib**

```c
int DPLSetImageMask(int InstanceID, double FromRed, double FromGreen,
                     double FromBlue, double ToRed, double ToGreen, double ToBlue);
```

**Objective-C class**

```objective-c
- (int)SetImageMask:(double)FromRed :(double)FromGreen :(double)FromBlue
               :(double)ToRed :(double)ToGreen :(double)ToBlue;
```

**Parameters**

- **FromRed**: The red component of the starting color for the mask
- **FromGreen**: The green component of the starting color for the mask
- **FromBlue**: The blue component of the starting color for the mask
- **ToRed**: The red component of the ending color for the mask
- **ToGreen**: The green component of the ending color for the mask
- **ToBlue**: The blue component of the ending color for the mask

**Return values**

- **0**: No image was selected
- **1**: The image mask was set successfully
SetImageMaskCMYK

Image handling, Color, Page layout

Description

Sets the mask for the selected image. This can be used to make parts of an image transparent when it is drawn with the DrawImage or DrawScaledImage functions. The color range specified will become transparent. Use this function when the image you added is a CMYK image. Use the SetImageMask function for RGB images. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLSetImageMaskCMYK(int InstanceID, double FromC, double FromM,
                        double FromY, double FromK, double ToC, double ToM,
                        double ToY, double ToK);
```

Objective-C class

```objc
- (int)SetImageMaskCMYK:(double)FromC :(double)FromM :(double)FromY
             :(double)FromK :(double)ToC :(double)ToM :(double)ToY
             :(double)ToK;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromC</td>
<td>The cyan component of the starting color for the mask</td>
</tr>
<tr>
<td>FromM</td>
<td>The magenta component of the starting color for the mask</td>
</tr>
<tr>
<td>FromY</td>
<td>The yellow component of the starting color for the mask</td>
</tr>
<tr>
<td>FromK</td>
<td>The black component of the starting color for the mask</td>
</tr>
<tr>
<td>ToC</td>
<td>The red component of the ending color for the mask</td>
</tr>
<tr>
<td>ToM</td>
<td>The magenta component of the ending color for the mask</td>
</tr>
<tr>
<td>ToY</td>
<td>The yellow component of the ending color for the mask</td>
</tr>
<tr>
<td>ToK</td>
<td>The black component of the ending color for the mask</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image was selected</td>
</tr>
<tr>
<td>1</td>
<td>The image mask was set successfully</td>
</tr>
</tbody>
</table>
SetImageMaskFromImage

Image handling, Page layout

Description

Use this function to use another image as a transparency mask for the selected image. The mask image must be a grayscale image. If it is not specifically prepared it will be added as a soft mask which only works with Acrobat 5.0 and later. If it is specially prepared using the `setImageAsMask` function you can choose whether the image will be a stencil mask (which will work with Acrobat 4.0 and later) or a soft mask (which will only work with Acrobat 5.0 and later). Remember that soft masks and stencil masks treat opaque and transparent in an opposite fashion. You may want to call `ReverseImage` on your mask image to ensure consistent results. For compatibility with Acrobat 6.0 and later it is important to set the transparency group for the page to ensure RGB colors in your image are not converted to CMYK yielding strange results. Use the `SetPageTransparencyGroup` function for this. To avoid problems with Acrobat 4.0 you may want to remove the /Decode array from the mask image. This can be achieved with the `ReverseImage` function setting the Reset parameter to 0.

Syntax

**Dylib**

```c
int DPLSetImageMaskFromImage(int InstanceID, int ImageID);
```

**Objective-C class**

```c
- (int)SetImageMaskFromImage:(int)ImageID;
```

Parameters

<table>
<thead>
<tr>
<th>ImageID</th>
<th>The ID of the image to use as the mask</th>
</tr>
</thead>
</table>
SetImageOptional
Image handling, Content Streams and Optional Content Groups

Description
Links the specified image to an optional content group. This allows the image to be selectively shown in Acrobat 6 or later.

Syntax

**Dylib**

```c
int DPLSetImageOptional(int InstanceID, int OptionalContentGroupID);
```

**Objective-C class**

```objc
-(int)SetImageOptional:(int)OptionalContentGroupID;
```

Parameters

<table>
<thead>
<tr>
<th><strong>OptionalContentGroupID</strong></th>
<th>An ID returned by the <strong>NewOptionalContentGroup</strong>, <strong>GetOptionalContentGroupID</strong> or <strong>GetOptionalContentConfigOrderItemID</strong> functions</th>
</tr>
</thead>
</table>
SetImageResolution

Image handling

Description

Sets the horizontal and vertical resolution of the selected image as well as the resolution units. These values are used by the FitImage function.

Syntax

Dylib

```c
int DPLSetImageResolution(int InstanceID, int Horizontal, int Vertical, int Units);
```

Objective-C class

```objective-c
- (int)SetImageResolution:(int)Horizontal :(int)Vertical :(int)Units;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Horizontal</strong></td>
<td>The new horizontal resolution of the image</td>
</tr>
<tr>
<td><strong>Vertical</strong></td>
<td>The new vertical resolution of the image</td>
</tr>
</tbody>
</table>
| **Units** | 0 = Unknown  
1 = No units, values specify the aspect ratio  
2 = Dots per inch (DPI)  
3 = Dots per centimetre (DPCM) |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image was selected</td>
</tr>
<tr>
<td>1</td>
<td>The resolution of the image was set successfully</td>
</tr>
</tbody>
</table>
SetInformation

Document properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Set the properties of the selected document.

For the CreationDate and ModDate (modification date) properties, the format of the date should be:

\[ D:YYYYMMDDHHmmSSOHH'mm' \]

where

- YYYY shall be the year
- MM shall be the month (01?12)
- DD shall be the day (01?31)
- HH shall be the hour (00?23)
- mm shall be the minute (00?59)
- SS shall be the second (00?59)

O shall be the relationship of local time to Universal Time (UT) using a +, - or Z character

HH followed by APOSTROPHE (U+0027) ('') shall be the absolute value of the offset from UT in hours (00?23)

mm followed by an optional APOSTROPHE (U+0027) ('') shall be the absolute value of the offset from UT in minutes (00?59)

Syntax

**Dylib**

```c
int DPLSetInformation(int InstanceID, int Key, wchar_t * NewValue);
```

**Objective-C class**

```c
- (int)SetInformation:(int)Key :(NSString *)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>The property to set:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PDF Version</td>
</tr>
<tr>
<td>1</td>
<td>Author</td>
</tr>
<tr>
<td>2</td>
<td>Title</td>
</tr>
<tr>
<td>3</td>
<td>Subject</td>
</tr>
<tr>
<td>4</td>
<td>Keywords</td>
</tr>
<tr>
<td>5</td>
<td>Creator</td>
</tr>
<tr>
<td>6</td>
<td>Producer</td>
</tr>
<tr>
<td>7</td>
<td>CreationDate</td>
</tr>
<tr>
<td>8</td>
<td>ModDate</td>
</tr>
<tr>
<td>9</td>
<td>XMP dc:subject</td>
</tr>
</tbody>
</table>

| NewValue | The new value of the specified property. |

Return values

| 0 | The specified information could not be set. Use the LastErrorCode function to determine the reason for failure. |
| 1 | The specified information was set successfully |
SetJPEGQuality

Rendering and printing

Description

Sets the quality for any JPEG images produced by the library.

Syntax

**Dylib**

```c
int DPLSetJPEGQuality(int InstanceID, int Quality);
```

**Objective-C class**

```c
-(int)SetJPEGQuality:(int)Quality;
```

Parameters

<table>
<thead>
<tr>
<th>Quality</th>
<th>A number between 1 and 100 indicating the quality of the image. The higher the value, the better the image quality, but the larger the file size. The lower the value, the smaller the resulting file size, but at the expense of picture quality.</th>
</tr>
</thead>
</table>
SetJavaScriptMode

Document properties, JavaScript

Description

This function allows you to control the way JavaScript is stored in the document.

Syntax

Dylib

```c
int DPLSetJavaScriptMode(int InstanceID, int JSMode);
```

Objective-C class

```objective-c
- (int)SetJavaScriptMode:(int)JSMode;
```

Parameters

<table>
<thead>
<tr>
<th>JSMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Store JavaScript in a stream</td>
</tr>
<tr>
<td>2</td>
<td>Store JavaScript in a compressed stream</td>
</tr>
<tr>
<td>Anything else</td>
<td>Store JavaScript as a string (default)</td>
</tr>
</tbody>
</table>
SetKerning
Text, Fonts

Description
Sets the amount of kerning for the specified character pair.

Syntax
Dylib

```c
int DPLSetKerning(int InstanceID, wchar_t * CharPair, int Adjustment);
```

Objective-C class

```objective-c
- (int)SetKerning:(NSString *)CharPair :(int)Adjustment;
```

Parameters

<table>
<thead>
<tr>
<th>CharPair</th>
<th>A two-character string containing the characters making the kerning pair, for example &quot;AW&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjustment</td>
<td>The amount to reduce the space between the kerning pair by. This is the same value as shown in graphics programs such as Adobe Illustrator. A value of 1000 is the same as the height of the text.</td>
</tr>
</tbody>
</table>

Return values

| 0 | The kerning could not be set. Either the CharPair was not 2 characters in length, or a font has not been selected. |
| 1 | The kerning for the specified pair of characters was set successfully |
SetLineCap
Vector graphics

Description
Sets the line cap style for subsequently drawn lines.

Syntax

**Dylib**
```
int DPLSetLineCap(int InstanceID, int LineCap);
```

**Objective-C class**
```
-(int)SetLineCap:(int)LineCap;
```

Parameters

<table>
<thead>
<tr>
<th><strong>LineCap</strong></th>
<th>The line cap style to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Butt</td>
</tr>
<tr>
<td>1</td>
<td>Round</td>
</tr>
<tr>
<td>2</td>
<td>Projecting square cap</td>
</tr>
</tbody>
</table>

Return values

| **0** | The LineCap parameter was not valid |
| **1** | The line cap style was set successfully |
SetLineColor

Vector graphics, Color

Description

Sets the outline color for any subsequently drawn graphics. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

**Dylib**

```c
int DPLSetLineColor(int InstanceID, double Red, double Green, double Blue);
```

**Objective-C class**

```objc
-(int)SetLineColor:(double)Red :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>Red</th>
<th>The red component of the color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>
SetLineColorCMYK
Vector graphics, Color

Description

Sets the outline color of subsequently drawn graphics. Similar to the SetLineColor function, but the color components are specified in CMYK mode (Cyan, Magenta, Yellow and Black). The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

**Dylib**

```c
int DPLSetLineColorCMYK(int InstanceID, double C, double M, double Y, double K);
```

**Objective-C class**

```objective-c
- (int)SetLineColorCMYK:(double)C :(double)M :(double)Y :(double)K;
```

Parameters

<table>
<thead>
<tr>
<th>C</th>
<th>The cyan component of the color</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>The magenta component of the color</td>
</tr>
<tr>
<td>Y</td>
<td>The yellow component of the color</td>
</tr>
<tr>
<td>K</td>
<td>The black component of the color</td>
</tr>
</tbody>
</table>
SetLineColorSep

Vector graphics, Color

Description

Sets the outline color of subsequently drawn graphics. Similar to the SetFillColor function, but a tint of a separation color added with the AddSeparationColor function is used.

Syntax

**Dylib**

```c
int DPLSetLineColorSep(int InstanceID, wchar_t * ColorName, double Tint);
```

**Objective-C class**

```c
-(int)SetLineColorSep:(NSString *)ColorName :(double)Tint;
```

Parameters

<table>
<thead>
<tr>
<th><strong>ColorName</strong></th>
<th>The name of the separation color that was used with the AddSeparationColor function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tint</strong></td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th><strong>0</strong></th>
<th>The separation color name could not be found</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>The line color was set successfully</td>
</tr>
</tbody>
</table>
SetLineDash

Vector graphics

Description

Sets the outline dash pattern for subsequently drawn graphics.

Calling this function with either parameter set to zero will return to a solid line style for subsequently drawn graphics.

Syntax

Dylib

int DPLSetLineDash(int InstanceID, double DashOn, double DashOff);

Objective-C class

- (int)SetLineDash:(double)DashOn :(double)DashOff;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DashOn</td>
<td>The width of the dashes</td>
</tr>
<tr>
<td>DashOff</td>
<td>The width of the space between the dashes</td>
</tr>
</tbody>
</table>
SetLineDashEx

Vector graphics

Description

Sets the outline dash pattern for subsequently drawn graphics. The dash pattern can be specified with a series of numeric values as per the PDF specification.

Calling this function with an empty string for the DashValues parameter will return to a solid line style for subsequently drawn graphics.

Syntax

Dylib

```c
int DPLSetLineDashEx(int InstanceID, wchar_t * DashValues);
```

Objective-C class

```objective-c
- (int)SetLineDashEx:(NSString *)DashValues;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DashValues</td>
<td>The dash pattern to use.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The value of the DashValues parameter was not valid. It should be a list of numeric values separated by spaces. For example &quot;1 1 5 1&quot;.</td>
</tr>
<tr>
<td>1</td>
<td>The dash pattern was set successfully.</td>
</tr>
</tbody>
</table>
SetLineJoin

Vector graphics

Description

Sets the line join style for subsequently drawn graphics.

Syntax

Dylib

    int DPLSetLineJoin(int InstanceID, int LineJoin);

Objective-C class

    -(int)SetLineJoin:(int)LineJoin;

Parameters

<table>
<thead>
<tr>
<th>LineJoin</th>
<th>The line join style to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Miter join</td>
</tr>
<tr>
<td>1</td>
<td>Round join</td>
</tr>
<tr>
<td>2</td>
<td>Bevel join</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The LineJoin parameter was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The line join style was set successfully</td>
</tr>
</tbody>
</table>
SetLineShader

Vector graphics, Path definition and drawing, Color

Description

Sets the outline color to the specified shader for subsequently drawn graphics.

Syntax

Dylib

    int DPLSetLineShader(int InstanceID, wchar_t * ShaderName);

Objective-C class

    - (int)SetLineShader:(NSString *)ShaderName;

Parameters

| ShaderName | The shader name that was used when the shader was created. |

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The shader could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The shader outline was setup correctly</td>
</tr>
</tbody>
</table>
SetLineWidth

Vector graphics

Description

Sets the outline width for any subsequently drawn shapes.

Syntax

Dylib

```c
int DPLSetLineWidth(int InstanceID, double LineWidth);
```

Objective-C class

```objective-c
-(int)SetLineWidth:(double)LineWidth;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineWidth</td>
<td>The width to use</td>
</tr>
</tbody>
</table>
SetMarkupAnnotStyle

Color, Annotations and hotspot links

Description

Sets the background color and transparency of a text markup annotation.

Syntax

Dylib

```c
int DPLSetMarkupAnnotStyle(int InstanceID, int Index, double Red,
                           double Green, double Blue, double Transparency);
```

Objective-C class

```objc
- (int)SetMarkupAnnotStyle:(int)Index :(double)Red :(double)Green
            :(double)Blue :(double)Transparency;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the annotation. The first annotation on the page has an index of 1.</td>
</tr>
<tr>
<td>Red</td>
<td>The red component of the color</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>
| Transparency | The amount of transparency to apply  
0 = No transparency
50 = 50% transparency
100 = Invisible                 |
SetMeasureDictBoundsCount
Measurement and coordinate units

Description
Sets the number of items in the Bounds array of a measurement dictionary.

Syntax
Dylib

```c
int DPLSetMeasureDictBoundsCount(int InstanceID, int MeasureDictID,
int NewCount);
```

Objective-C class

```objectivec
- (int)SetMeasureDictBoundsCount:(int)MeasureDictID :(int)NewCount;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeasureDictID</td>
<td>A value returned from the <code>GetImageMeasureDict</code> function</td>
</tr>
<tr>
<td>NewCount</td>
<td>The new number of items in the list. Must be a multiple of 2.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetMeasureDictBoundsItem

Measurement and coordinate units

Description

Sets the value of an item in the Bounds array of a measurement dictionary.

Syntax

**Dylib**

```c
int DPLSetMeasureDictBoundsItem(int InstanceID, int MeasureDictID, int ItemIndex, double NewValue);
```

**Objective-C class**

```objective-c
- (int)SetMeasureDictBoundsItem:(int)MeasureDictID :(int)ItemIndex :(double)NewValue;
```

Parameters

- **MeasureDictID**
  A value returned from the `getImageMeasureDict` function
- **ItemIndex**
  The index of the item to set. The first item has an index of 1.
- **NewValue**
  The new value of the item.

Return values

- **0**
  The MeasureDictID parameter was incorrect or the ItemIndex parameter was out of range
- **1**
  Success
SetMeasureDictCoordinateSystem

Measurement and coordinate units

Description
Sets the coordinate system of a measurement dictionary.

Syntax

**Dylib**

```
int DPLSetMeasureDictCoordinateSystem(int InstanceID, int MeasureDictID,  
int CoordinateSystemID);
```

**Objective-C class**

```
- (int)SetMeasureDictCoordinateSystem:(int)MeasureDictID  
:(int)CoordinateSystemID;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MeasureDictID</strong></td>
<td>A value returned from the <strong>GetImageMeasureDict</strong> function</td>
</tr>
<tr>
<td><strong>CoordinateSystemID</strong></td>
<td>1 = Rectilinear coordinate system (RL)</td>
</tr>
<tr>
<td></td>
<td>2 = Geospatial coordinate system (GEO)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect or the CoordinateSystemID parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetMeasureDictGPTSCount

Measurement and coordinate units

Description

Sets the number of items in the GPTS array of a measurement dictionary.

Syntax

**Dylib**

```c
int DPLSetMeasureDictGPTSCount(int InstanceID, int MeasureDictID,
                               int NewCount);
```

**Objective-C class**

```objective-c
-(int)SetMeasureDictGPTSCount:(int)MeasureDictID :(int)NewCount;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeasureDictID</td>
<td>A value returned from the <code>GetImageMeasureDict</code> function</td>
</tr>
<tr>
<td>NewCount</td>
<td>The new number of items in the list. Must be a multiple of 2.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
**SetMeasureDictGPTSItem**

**Measurement and coordinate units**

**Description**

Sets the value of an item in the GPTS array of a measurement dictionary.

**Syntax**

**Dylib**

```c
int DPLSetMeasureDictGPTSItem(int InstanceID, int MeasureDictID,
                               int ItemIndex, double NewValue);
```

**Objective-C class**

```c
- (int)SetMeasureDictGPTSItem:(int)MeasureDictID :(int)ItemIndex
                         :(double)NewValue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeasureDictID</td>
<td>A value returned from the <a href="#">GetImageMeasureDict</a> function</td>
</tr>
<tr>
<td>ItemIndex</td>
<td>The index of the item to set. The first item has an index of 1.</td>
</tr>
<tr>
<td>NewValue</td>
<td>The new value of the item.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect or the ItemIndex parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetMeasureDictLPTSCount

Measurement and coordinate units

Description

Sets the number of items in the LPTS array of a measurement dictionary.

Syntax

**Dylib**

```c
int DPLSetMeasureDictLPTSCount(int InstanceID, int MeasureDictID, int NewCount);
```

**Objective-C class**

```objective-c
- (int)SetMeasureDictLPTSCount:(int)MeasureDictID :(int)NewCount;
```

Parameters

- **MeasureDictID**
  A value returned from the `GetImageMeasureDict` function

- **NewCount**
  The new number of items in the list. Must be a multiple of 2.

Return values

- **0**
  The MeasureDictID parameter was incorrect

- **1**
  Success
**SetMeasureDictLPTSItem**

**Measurement and coordinate units**

**Description**

Sets the value of an item in the LPTS array of a measurement dictionary.

**Syntax**

**Dylib**

```c
int DPLSetMeasureDictLPTSItem(int InstanceID, int MeasureDictID, int ItemIndex, double NewValue);
```

**Objective-C class**

```objective-c
- (int)SetMeasureDictLPTSItem:(int)MeasureDictID :(int)ItemIndex :(double)NewValue;
```

**Parameters**

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>A value returned from the GetImageMeasureDict function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemIndex</td>
<td>The index of the item to set. The first item has an index of 1.</td>
</tr>
<tr>
<td>NewValue</td>
<td>The new value of the item.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The MeasureDictID parameter was incorrect or the ItemIndex parameter was out of range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetMeasureDictPDU

Measurement and coordinate units

Description

Sets the page display units of a measurement dictionary.

Syntax

Dylib

```c
int DPLSetMeasureDictPDU(int InstanceID, int MeasureDictID,
int LinearUnit, int AreaUnit, int AngularUnit);
```

Objective-C class

```c
- (int)SetMeasureDictPDU:(int)MeasureDictID :(int)LinearUnit
 :(int)AreaUnit :(int)AngularUnit;
```

Parameters

<table>
<thead>
<tr>
<th>MeasureDictID</th>
<th>A value returned from the GetImageMeasureDict function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinearUnit</td>
<td>1 = M (a meter)</td>
</tr>
<tr>
<td></td>
<td>2 = KM (a kilometer)</td>
</tr>
<tr>
<td></td>
<td>3 = FT (an international foot)</td>
</tr>
<tr>
<td></td>
<td>4 = USFT (a U.S. Survey foot)</td>
</tr>
<tr>
<td></td>
<td>5 = MI (an international mile)</td>
</tr>
<tr>
<td></td>
<td>6 = NM (an international nautical mile)</td>
</tr>
<tr>
<td>AreaUnit</td>
<td>1 = SQM (a square meter)</td>
</tr>
<tr>
<td></td>
<td>2 = HA (a hectare = 10,000 square meters)</td>
</tr>
<tr>
<td></td>
<td>3 = SQKM (a square kilometer)</td>
</tr>
<tr>
<td></td>
<td>4 = SQFT (a square foot)</td>
</tr>
<tr>
<td></td>
<td>5 = A (an acre)</td>
</tr>
<tr>
<td></td>
<td>6 = SQMI (a square mile)</td>
</tr>
</tbody>
</table>

AngularUnit

| AngularUnit   | 1 = DEG (a degree)                                    |
|               | 2 = GRD (a grad = 0.9 degrees)                        |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The MeasureDictID parameter was incorrect or one of the other parameters was out of range.</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetMeasurementUnits

Measurement and coordinate units

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Set the units to use for all measurements given to and returned from the library. Default user space is exactly 1/72 inches per unit, which is approximately the same as a "point", a unit used in the printing industry. 25.4 millimetres is one inch.

**Syntax**

**Dylib**

```c
int DPLSetMeasurementUnits(int InstanceID, int MeasurementUnits);
```

**Objective-C class**

```c
- (int)SetMeasurementUnits:(int)MeasurementUnits;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>MeasurementUnits</strong></th>
<th>The units to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default user space</td>
</tr>
<tr>
<td>1</td>
<td>Millimetres</td>
</tr>
<tr>
<td>2</td>
<td>Inches</td>
</tr>
<tr>
<td>Anything else</td>
<td>Default user space</td>
</tr>
</tbody>
</table>
SetNeedAppearances

Form fields

Description
Sets the value of the document's "NeedAppearances" key. Setting this to 1 (True) will instruct the PDF viewer to create the appearances for the form fields when the document is loaded. The document must have at least one form field for this function to have any effect.

Syntax

**Dylib**

```c
int DPLSetNeedAppearances(int InstanceID, int NewValue);
```

**Objective-C class**

```c
-(int)SetNeedAppearances:(int)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Set NeedAppearances to False</td>
</tr>
<tr>
<td>1</td>
<td>Set NeedAppearances to True</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The document does not have any form fields</td>
</tr>
<tr>
<td>1</td>
<td>The NeedAppearances flag was set successfully</td>
</tr>
</tbody>
</table>
SetObjectFromString

Miscellaneous functions

Description

Sets the raw PDF object data for the specified object number. This is for advanced use only.

Syntax

Dylib

```c
int DPLSetObjectFromString(int InstanceID, int ObjectNumber,
char * Source);
```

Objective-C class

```objective-c
-(int)SetObjectFromString:(int)ObjectNumber :(NSData *)Source;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectNumber</td>
<td>The number of the object to update. The first object is numbered 1 and the last object has an object number equal to the result of the <code>GetObjectCount</code> function.</td>
</tr>
<tr>
<td>Source</td>
<td>The raw PDF object data to associate with the specified object.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The ObjectNumber parameter was out of bounds.</td>
</tr>
<tr>
<td>1</td>
<td>The specified object was updated successfully.</td>
</tr>
</tbody>
</table>
SetOpenActionDestination

Document properties

Description

This function allows the opening page and zoom factor to be set for the selected document.

Syntax

Dylib

```c
int DPLSetOpenActionDestination(int InstanceID, int OpenPage, int Zoom);
```

Objective-C class

```objective-c
- (int)SetOpenActionDestination:(int)OpenPage :(int)Zoom;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenPage</td>
<td>The page number to jump to when the document is opened</td>
</tr>
</tbody>
</table>
| Zoom      | The zoom percentage to use when the document is opened: 0..1600 = percentage zoom  
-1 = Fit in window  
-2 = Fit width |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The open action could not be set</td>
</tr>
<tr>
<td>1</td>
<td>The open action was set successfully</td>
</tr>
</tbody>
</table>
SetOpenActionDestinationFull

Document properties

Description

This function allows the opening page and various sizing/positioning values to be set for the selected document.

Syntax

Dylib

```c
int DPLSetOpenActionDestinationFull(int InstanceID, int OpenPage,
int Zoom, int DestType, double Left, double Top, double Right,
double Bottom);
```

Objective-C class

```c
-(int)SetOpenActionDestinationFull:(int)OpenPage :(int)Zoom
:(int)DestType :(double)Left :(double)Top :(double)Right
:(double)Bottom;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenPage</td>
<td>The page number to jump to when the document is opened</td>
</tr>
<tr>
<td>Zoom</td>
<td>The zoom percentage to use when the document is opened, valid values from 0 to 6400. Only used for DestType = 1, should be set to 0 for other DestTypes.</td>
</tr>
<tr>
<td>DestType</td>
<td>1 = &quot;XYZ&quot; - the target page is positioned at the point specified by the Left and Top parameters. The Zoom parameter specifies the zoom percentage. 2 = &quot;Fit&quot; - the entire page is zoomed to fit the window. None of the other parameters are used and should be set to zero. 3 = &quot;FitH&quot; - the page is zoomed so that the entire width of the page is visible. The height of the page may be greater or less than the height of the window. The page is positioned at the vertical position specified by the Top parameter. 4 = &quot;FitV&quot; - the page is zoomed so that the entire height of the page can be seen. The width of the page may be greater or less than the width of the window. The page is positioned at the horizontal position specified by the Left parameter. 5 = &quot;FitR&quot; - the page is zoomed so that a certain rectangle on the page is visible. The Left, Top, Right and Bottom parameters define the rectangular area on the page. 6 = &quot;FitB&quot; - the page is zoomed so that it's bounding box is visible. 7 = &quot;FitBH&quot; - the page is positioned vertically at the position specified by the Top parameter. The page is zoomed so that the entire width of the page's bounding box is visible. 8 = &quot;FitBV&quot; - the page is positioned at the horizontal position specified by the Left parameter. The page is zoomed just enough to fit the entire height of the bounding box into the window.</td>
</tr>
<tr>
<td>Left</td>
<td>The horizontal position used by DestType = 1, 4, 5 and 8</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical position used by DestType = 1, 3, 5 and 7</td>
</tr>
<tr>
<td>Right</td>
<td>The horizontal position of the righthand edge of the rectangle. Used by DestType = 5</td>
</tr>
<tr>
<td>Bottom</td>
<td>The horizontal position of the bottom of the rectangle. Used by DestType = 5</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The open action destination could not be set. The usually indicates that the Zoom or DestType parameters are out of range.</td>
</tr>
<tr>
<td>1</td>
<td>The open action destination was set successfully.</td>
</tr>
</tbody>
</table>
**SetOpenActionJavaScript**

**Document properties, JavaScript**

**Description**

Use this function to run a block of JavaScript as the document is opened.

**Syntax**

**Dylib**

```c
int DPLSetOpenActionJavaScript(int InstanceID, wchar_t * JavaScript);
```

**Objective-C class**

```objc
- (int)SetOpenActionJavaScript:(NSString *)JavaScript;
```

**Parameters**

| JavaScript | The JavaScript to use for this action. |

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>The JavaScript could not be added</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The JavaScript was added successfully</td>
</tr>
</tbody>
</table>
SetOpenActionMenu

Document properties

Description

Specifies an Acrobat menu item to execute when the document is first loaded.

Syntax

**Dylib**

```c
int DPLSetOpenActionMenu(int InstanceID, wchar_t * MenuItem);
```

**Objective-C class**

```objc
-(int)SetOpenActionMenu:(NSString *)MenuItem;
```

Parameters

<table>
<thead>
<tr>
<th>MenuItem</th>
<th>The menu item which should be executed, for example &quot;print&quot;</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The open action could not be set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The open action was set successfully</td>
</tr>
</tbody>
</table>
SetOptionalContentConfigLocked

Content Streams and Optional Content Groups

Description

This function is used to lock an optional content group as defined by the specified optional content configuration dictionary.

Syntax

**Dylib**

```c
int DPLSetOptionalContentConfigLocked(int InstanceID,
                                        int OptionalContentConfigID, int OptionalContentGroupID,
                                        int NewLocked);
```

**Objective-C class**

```objective-c
- (int)SetOptionalContentConfigLocked:(int)OptionalContentConfigID
                                :(int)OptionalContentGroupID :(int)NewLocked;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptionalContentConfigID</td>
<td>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</td>
</tr>
<tr>
<td>OptionalContentGroupID</td>
<td>An ID returned by the <code>NewOptionalContentGroup</code>, <code>GetOptionalContentGroupID</code> or <code>GetOptionalContentConfigOrderItemID</code> functions</td>
</tr>
<tr>
<td>NewLocked</td>
<td>0 = Unlocked 1 = Locked</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The optional content group could not be locked</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetOptionalContentConfigState

Content Streams and Optional Content Groups

Description

This function is used to set the state of an optional content group as defined by the specified optional content configuration dictionary.

All optional content configuration dictionaries have a base state (either ON, OFF or Unchanged) and two membership arrays called /ON and /OFF.

A reference to the optional content group is added to the appropriate /ON or /OFF array (or removed from either array) depending on the value of the base state.

A particular optional content group can only be set to Unchanged if the base state of the optional content configuration dictionary is Unchanged.

The base state of the default optional content configuration dictionary (accessed by setting OptionalContentConfigID to 1) is always ON, so optional content groups in this configuration dictionary can only be set to ON or OFF.

Syntax

Dylib

```c
int DPLSetOptionalContentConfigState(int InstanceID,
int OptionalContentConfigID, int OptionalContentGroupID,
int NewState);
```

Objective-C class

```objective-c
- (int)SetOptionalContentConfigState:(int)OptionalContentConfigID
   :(int)OptionalContentGroupID :(int)NewState;
```

Parameters

<table>
<thead>
<tr>
<th>OptionalContentConfigID</th>
<th>The first default optional content configuration dictionary has an ID of 1. Higher numbers are used for other optional content configuration dictionaries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptionalContentGroupID</td>
<td>An ID returned by the NewOptionalContentGroup, GetOptionalContentGroupID or GetOptionalContentConfigOrderItemID functions</td>
</tr>
<tr>
<td>NewState</td>
<td>Specifies the state that the optional content group in this configuration dictionary should be set to: 1 = Set to ON 2 = Set to OFF 3 = Set to unchanged (if possible)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The state could not be set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The state was set successfully</td>
</tr>
</tbody>
</table>
SetOptionalContentGroupName
Content Streams and Optional Content Groups

Description
Sets the name of the specified optional content group.

Syntax

Dylib

```c
int DPLSetOptionalContentGroupName(int InstanceID,
    int OptionalContentGroupID, wchar_t * NewGroupName);
```

Objective-C class

```objc
- (int)SetOptionalContentGroupName:(int)OptionalContentGroupID
    :(NSString *)NewGroupName;
```

Parameters

<table>
<thead>
<tr>
<th>OptionalContentGroupID</th>
<th>An ID returned by the <code>NewOptionalContentGroup</code>, <code>GetOptionalContentGroupID</code> or <code>GetOptionalContentConfigOrderItemID</code> functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewGroupName</td>
<td>The new name for the OCG</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The name could not be set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The OCG's name was set successfully</td>
</tr>
</tbody>
</table>
**SetOptionalContentGroupPrintable**

**Content Streams and Optional Content Groups**

**Description**

This function allows an optional content group to be marked as visible or invisible when the document is printed.

**Syntax**

**Dylib**

```c
int DPLSetOptionalContentGroupPrintable(int InstanceID,
                                        int OptionalContentGroupID, int Printable);
```

**Objective-C class**

```objective-c
- (int)SetOptionalContentGroupPrintable:(int)OptionalContentGroupID
  :(int)Printable;
```

**Parameters**

<table>
<thead>
<tr>
<th>OptionalContentGroupID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An ID returned by the <a href="#">NewOptionalContentGroup</a>, <a href="#">GetOptionalContentGroupID</a> or <a href="#">GetOptionalContentConfigOrderItemID</a> functions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Printable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not printed</td>
</tr>
<tr>
<td>1</td>
<td>Printed</td>
</tr>
</tbody>
</table>
### SetOptionalContentGroupVisible

#### Content Streams and Optional Content Groups

#### Description

This function allows an optional content group to be marked as visible or invisible when the document is opened.

#### Syntax

**Dylib**

```c
int DPLSetOptionalContentGroupVisible(int InstanceID,
int OptionalContentGroupID, int Visible);
```

**Objective-C class**

```c
- (int)SetOptionalContentGroupVisible:(int)OptionalContentGroupID
  :(int)Visible;
```

#### Parameters

<table>
<thead>
<tr>
<th>OptionalContentGroupID</th>
<th>An ID returned by the <code>NewOptionalContentGroup</code>, <code>GetOptionalContentGroupID</code> or <code>GetOptionalContentConfigOrderItemID</code> functions</th>
</tr>
</thead>
</table>
| Visible                | 0 = Not visible  
1 = Visible                                                              |

#### Return values

<table>
<thead>
<tr>
<th>Non-zero</th>
<th>An ID that can be used as the OptionalContentGroupID parameter with the other optional content group functions</th>
</tr>
</thead>
</table>
SetOrigin

Measurement and coordinate units

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Sets the origin for all subsequent drawing operations.

Syntax

**Dylib**

```c
int DPLSetOrigin(int InstanceID, int Origin);
```

**Objective-C class**

```objective-c
-(int)SetOrigin:(int)Origin;
```

Parameters

<table>
<thead>
<tr>
<th>Origin</th>
<th>Specifies which page corner to use for the origin:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bottom left (default)</td>
</tr>
<tr>
<td>1</td>
<td>Top left</td>
</tr>
<tr>
<td>2</td>
<td>Top right</td>
</tr>
<tr>
<td>3</td>
<td>Bottom right</td>
</tr>
<tr>
<td>Anything else</td>
<td>Bottom left</td>
</tr>
</tbody>
</table>
SetOutlineColor

Color, Outlines

Description

Sets the color of an outline item (bookmark). The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLSetOutlineColor(int InstanceID, int OutlineID, double Red, double Green, double Blue);
```

Objective-C class

```c
- (int)SetOutlineColor:(int)OutlineID :(double)Red :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>The ID of the outline as returned by the <code>NewOutline</code> function. Alternatively, use the <code>GetOutlineID</code> function to get a valid outline ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>The red component of the color</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>

Return values

| 0          | The Outline ID provided was invalid                                                                  |
| 1          | The color of the outline item was set successfully                                                  |
SetOutlineDestination

Outlines

Description

Sets the destination that an outline item (bookmark) points to.

Syntax

Dylib

```c
int DPLSetOutlineDestination(int InstanceID, int OutlineID, int DestPage, double DestPosition);
```

Objective-C class

```objective-c
-(int)SetOutlineDestination:(int)OutlineID :(int)DestPage :(double)DestPosition;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutlineID</td>
<td>The ID of the outline as returned by the <code>NewOutline</code> function. Alternatively, use the <code>GetOutlineID</code> function to get a valid outline ID.</td>
</tr>
<tr>
<td>DestPage</td>
<td>The page number that this outline item links to</td>
</tr>
<tr>
<td>DestPosition</td>
<td>The vertical position of the page that this outline item links to. Jumping to the bottom of the page will result in the following page being shown. If possible link to the top of the page.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Outline ID provided was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The destination of the outline item was set successfully</td>
</tr>
</tbody>
</table>
SetOutlineDestinationFull

Outlines

Description

Sets the destination of an outline item (bookmark) to a specific position and zoom percentage.

Syntax

**Dylib**

```c
int DPLSetOutlineDestinationFull(int InstanceID, int OutlineID, 
int DestPage, int Zoom, int DestType, double Left, double Top, 
double Right, double Bottom);
```

**Objective-C class**

```c
- (int)SetOutlineDestinationFull:(int)OutlineID :(int)DestPage :(int)Zoom 
:(int)DestType :(double)Left :(double)Top :(double)Right 
:(double)Bottom;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OutlineID</strong></td>
<td>The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID.</td>
</tr>
<tr>
<td><strong>DestPage</strong></td>
<td>The page number that this outline item links to</td>
</tr>
<tr>
<td><strong>Zoom</strong></td>
<td>The zoom percentage to use when the outline destination is opened, valid values from 0 to 6400. Only used for DestType = 1, should be set to 0 for other DestTypes.</td>
</tr>
<tr>
<td><strong>DestType</strong></td>
<td>The Target page is positioned at the point specified by the Left and Top parameters. The Zoom parameter specifies the zoom percentage. 1 = &quot;XYZ&quot; - the target page is positioned at the point specified by the Left and Top parameters. The Zoom parameter specifies the zoom percentage. 2 = &quot;Fit&quot; - the entire page is zoomed to fit the window. None of the other parameters are used and should be set to zero. 3 = &quot;FitH&quot; - the page is zoomed so that the entire width of the page is visible. The height of the page may be greater or less than the height of the window. The page is positioned at the vertical position specified by the Top parameter. 4 = &quot;FitV&quot; - the page is zoomed so that the entire height of the page can be seen. The width of the page may be greater or less than the width of the window. The page is positioned at the horizontal position specified by the Left parameter. 5 = &quot;FitR&quot; - the page is zoomed so that a certain rectangle on the page is visible. The Left, Top, Right and Bottom parameters define the rectangular area on the page. 6 = &quot;FitB&quot; - the page is zoomed so that its bounding box is visible. 7 = &quot;FitBH&quot; - the page is positioned vertically at the position specified by the Top parameter. The page is zoomed so that the entire width of the page's bounding box is visible. 8 = &quot;FitBV&quot; - the page is positioned at the horizontal position specified by the Left parameter. The page is zoomed just enough to fit the entire height of the bounding box into the window.</td>
</tr>
<tr>
<td><strong>Left</strong></td>
<td>The horizontal position used by DestType = 1, 4, 5 and 8</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The vertical position used by DestType = 1, 3, 5 and 7</td>
</tr>
<tr>
<td><strong>Right</strong></td>
<td>The horizontal position of the righthand edge of the rectangle. Used by DestType = 5</td>
</tr>
<tr>
<td><strong>Bottom</strong></td>
<td>The horizontal position of the bottom of the rectangle. Used by DestType = 5</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The outline destination could not be set. The usually indicates that the Zoom or DestType parameters are out of range or the OutlineID is invalid.</td>
</tr>
<tr>
<td>1</td>
<td>The outline destination was set successfully</td>
</tr>
</tbody>
</table>
SetOutlineDestinationZoom

Outlines

Description

Sets the destination of an outline item (bookmark) to a specific position on a page, and sets the zoom percentage of the document.

Syntax

Dylib

```c
int DPLSetOutlineDestinationZoom(int InstanceID, int OutlineID,
int DestPage, double DestPosition, int Zoom);
```

Objective-C class

```objective-c
- (int)SetOutlineDestinationZoom:(int)OutlineID :(int)DestPage :
(double)DestPosition :(int)Zoom;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutlineID</td>
<td>The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID.</td>
</tr>
<tr>
<td>DestPage</td>
<td>The page number that this outline should link to</td>
</tr>
<tr>
<td>DestPosition</td>
<td>The vertical position on the page that the outline should link to. Specifying a point at the bottom of the page will result in the next page being shown - it is better to link to a point at the top of the page.</td>
</tr>
<tr>
<td>Zoom</td>
<td>The zoom factor to show the target page at: 0..1600 = Zoom percentage -1 = Fit in window -2 = Fit width</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The OutlineID was invalid</td>
</tr>
<tr>
<td>1</td>
<td>Destination set successfully</td>
</tr>
</tbody>
</table>
SetOutlineJavaScript

JavaScript, Outlines

Description

Specifies the JavaScript to run when the user clicks on the outline item (bookmark).

Syntax

Dylib

int DPLSetOutlineJavaScript(int InstanceID, int OutlineID, wchar_t * JavaScript);

Objective-C class

- (int)SetOutlineJavaScript:(int)OutlineID :(NSString *)JavaScript;

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>The ID of the outline as returned by the NewOutline function. Alternatively, use the GetOutlineID function to get a valid outline ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>The JavaScript to execute.</td>
</tr>
</tbody>
</table>

Return values

| 0          | The OutlineID was invalid                                                                                   |
| 1          | The JavaScript action was successfully added to the outline ID                                             |
SetOutlineNamedDestination
Annotations and hotspot links, Outlines

Description
Sets the destination of the specified outline item (bookmark) to a named destination.

Syntax

**Dylib**

```c
int DPLSetOutlineNamedDestination(int InstanceID, int OutlineID, wchar_t * DestName);
```

**Objective-C class**

```c
-(int)SetOutlineNamedDestination:(int)OutlineID :(NSString *)DestName;
```

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>The ID of the outline as returned by the <strong>NewOutline</strong> function. Alternatively, use the <strong>GetOutlineID</strong> function to get a valid outline ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestName</td>
<td>The named destination.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The OutlineID was invalid</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetOutlineOpenFile

Outlines

Description

Sets the outline item (bookmark) to open a file when it is clicked.

Syntax

Dylib

```c
int DPLSetOutlineOpenFile(int InstanceID, int OutlineID, wchar_t * FileName);
```

Objective-C class

```objc
- (int)SetOutlineOpenFile:(int)OutlineID :(NSString *)FileName;
```

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>The ID of the outline as returned by the <code>NewOutline</code> function. Alternatively, use the <code>GetOutlineID</code> function to get a valid outline ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The file to open when the outline is clicked. This should be in a specific format. Back slashes should be converted to forward slashes and the drive, if any, should be specified as just the drive letter between forward slashes without a colon. For example, the file &quot;c:\my documents\hello.pdf&quot; should be specified as &quot;/c/my documents/hello.pdf&quot;. Relative path names are valid, including paths that include the &quot;..&quot; operator to move up a directory.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The OutlineID was invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The outline destination was set successfully</td>
</tr>
</tbody>
</table>
SetOutlineRemoteDestination

Description

Sets the outline item (bookmark) to open another PDF when it is clicked. The opening page number and various sizing/positioning values can be specified.

Note: because the page size of the target document is not known, all positions are specified in points measured from the bottom left corner of the opening page.

Syntax

Dylib

```c
int DPlSetOutlineRemoteDestination(int InstanceID, int OutlineID, wchar_t * FileName, int OpenPage, int Zoom, int DestType, double PntLeft, double PntTop, double PntRight, double PntBottom, int NewWindow);
```

Objective-C class

```c
- (int)SetOutlineRemoteDestination:(int)OutlineID :(NSString *)FileName :(int)OpenPage :(int)Zoom :(int)DestType :(double)PntLeft :(double)PntTop :(double)PntRight :(double)PntBottom :(int)NewWindow;
```

Parameters

**OutlineID**
The ID of the outline as returned by the `NewOutline` function. Alternatively, use the `GetOutlineID` function to get a valid outline ID.

**FileName**
The filename of the PDF document to open when the outline is clicked. This should be in a specific format. Backslashes should be converted to forward slashes and the drive, if any, should be specified as just the drive letter between forward slashes without a colon. For example, the file "c:\my documents\hello.pdf" should be specified as "/c/my documents/hello.pdf". Relative path names are valid, including paths that include the ".." operator to move up a directory.

**OpenPage**
The page number to jump to when the target document is opened. The first page has an index of zero (0).

**Zoom**
The zoom percentage to use when the document is opened, valid values from 0 to 6400. Only used for DestType = 1, should be set to 0 for other DestTypes.

**DestType**
- 1 = "XYZ" - the target page is positioned at the point specified by the Left and Top parameters. The Zoom parameter specifies the zoom percentage.
- 2 = "Fit" - the entire page is zoomed to fit the window. None of the other parameters are used and should be set to zero.
- 3 = "FitH" - the page is zoomed so that the entire width of the page is visible. The height of the page may be greater or less than the height of the window. The page is positioned at the vertical position specified by the Top parameter.
- 4 = "FitV" - the page is zoomed so that the entire height of the page can be seen. The width of the page may be greater or less than the width of the window. The page is positioned at the horizontal position specified by the Left parameter.
- 5 = "FitR" - the page is zoomed so that a certain rectangle on the page is visible. The Left, Top, Right and Bottom parameters define the rectangular area on the page.
- 6 = "FitB" - the page is zoomed so that it's bounding box is visible.
- 7 = "FitBH" - the page is positioned vertically at the position specified by the Top parameter. The page is zoomed just enough to fit the entire height of the bounding box into the window.
- 8 = "FitBV" - the page is positioned at the horizontal position specified by the Left parameter. The page is zoomed just enough to fit the entire height of the bounding box into the window.

**PntLeft**
The horizontal position used by DestType = 1, 4, 5 and 8. The position is specified in points measured from the bottom left corner of the target document's page.

**PntTop**
The vertical position used by DestType = 1, 3, 5 and 7. The position is specified in points measured from the bottom left corner of the target document's page.

**PntRight**
The horizontal position of the righthand edge of the rectangle. Used by DestType = 5. The position is specified in points measured from the bottom left corner of the target document's page.

**PntBottom**
The horizontal position of the bottom of the rectangle. Used by DestType = 5. The position is specified in points measured from the bottom left corner of the target document's page.

**NewWindow**
- 0 = Replace the current document with the target document
- 1 = Open the target document in a new window unless the user has specified a different preference in the PDF viewer

Return values

- 0: The OutlineID was invalid
- 1: The outline destination was set successfully
SetOutlineStyle

Outlines

Description

Sets the way an outline item (bookmark) is displayed.

Syntax

**Dylib**

```c
int DPLSetOutlineStyle(int InstanceID, int OutlineID, int SetItalic, int SetBold);
```

**Objective-C class**

```c
- (int)SetOutlineStyle:(int)OutlineID :(int)SetItalic :(int)SetBold;
```

Parameters

<table>
<thead>
<tr>
<th>OutlineID</th>
<th>The ID of the outline as returned by the <code>NewOutline</code> function. Alternatively, use the <code>GetOutlineID</code> function to get a valid outline ID.</th>
</tr>
</thead>
</table>
| SetItalic | 0 = Normal  
|           | 1 = Italic                                                                                              |
| SetBold   | 0 = Normal  
|           | 1 = Bold                                                                                                |

Return values

| 0 | The Outline ID provided was invalid |
| 1 | The style of the outline item was set successfully |
## SetOutlineTitle

### Outlines

Sets the title of an outline item (bookmark).

### Syntax

**Dylib**

```c
int DPLSetOutlineTitle(int InstanceID, int OutlineID, wchar_t * NewTitle);
```

**Objective-C class**

```objective-c
-(int)SetOutlineTitle:(int)OutlineID :(NSString *)NewTitle;
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OutlineID</strong></td>
<td>The ID of the outline as returned by the <strong>NewOutline</strong> function. Alternatively, use the <strong>GetOutlineID</strong> function to get a valid outline ID.</td>
</tr>
<tr>
<td><strong>NewTitle</strong></td>
<td>The new title for the outline item.</td>
</tr>
</tbody>
</table>

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Outline ID provided was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The title of the outline item was set successfully</td>
</tr>
</tbody>
</table>
SetOutlineWebLink

Outlines

Description

Specifies an internet link that should be opened when the user clicks on the outline item (bookmark).

Syntax

Dylib

```c
int DPLSetOutlineWebLink(int InstanceID, int OutlineID, wchar_t * Link);
```

Objective-C class

```objective-c
- (int)SetOutlineWebLink:(int)OutlineID :(NSString *)Link;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutlineID</td>
<td>The ID of the outline as returned by the <code>NewOutline</code> function. Alternatively, use the <code>GetOutlineID</code> function to get a valid outline ID.</td>
</tr>
<tr>
<td>Link</td>
<td>The URL to link to. Some examples: &quot;<a href="http://www.example.com/">http://www.example.com/</a>&quot; &quot;<a href="mailto:info@example.com">mailto:info@example.com</a>&quot;</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The OutlineID was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The web link action was added to the outline item successfully</td>
</tr>
</tbody>
</table>
SetOverprint

Vector graphics, Page layout

Description

Sets the overprint parameter of the graphics state for subsequently drawn text and graphics.

Syntax

**Dylib**

```c
int DPLSetOverprint(int InstanceID, int StrokingOverprint,
                    int OtherOverprint, int OverprintMode);
```

**Objective-C class**

```objc
-(int)SetOverprint:(int)StrokingOverprint :(int)OtherOverprint
            :(int)OverprintMode;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StrokingOverprint</strong></td>
<td>Controls overprint for stroking operations:</td>
</tr>
<tr>
<td></td>
<td>0 = Turn overprint off</td>
</tr>
<tr>
<td></td>
<td>1 = Turn overprint on</td>
</tr>
<tr>
<td><strong>OtherOverprint</strong></td>
<td>Controls overprint for non-stroking operations:</td>
</tr>
<tr>
<td></td>
<td>0 = Turn overprint off</td>
</tr>
<tr>
<td></td>
<td>1 = Turn overprint on</td>
</tr>
<tr>
<td><strong>OverprintMode</strong></td>
<td>Sets the interpretation of a tint value of 0.0 for a color component in a</td>
</tr>
<tr>
<td></td>
<td>DeviceCMYK colour space.</td>
</tr>
<tr>
<td></td>
<td>0 = Default behaviour</td>
</tr>
<tr>
<td></td>
<td>1 = Nonzero overprint mode</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An error occurred. One or more of the parameters were out of range.</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetPDFAMode

Document properties

Description
Sets up the document for PDF/A standards compliance mode.

Syntax

Dylib

```c
int DPLSetPDFAMode(int InstanceID, int NewMode);
```

Objective-C class

```objective-c
- (int)SetPDFAMode:(int)NewMode;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewMode</td>
<td>2 = PDF/A-1b</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalid NewMode parameter</td>
</tr>
<tr>
<td>1</td>
<td>The compliance mode was set successfully</td>
</tr>
</tbody>
</table>
SetPNGTransparencyColor

Image handling, Color

Description
Sets the RGB color to use as the transparency mask in PNG images that are generated by the rendering functions.

Syntax

Dylib

```c
int DPLSetPNGTransparencyColor(int InstanceID, int RedByte, int GreenByte, int BlueByte);
```

Objective-C class

```objective-c
- (int)SetPNGTransparencyColor:(int)RedByte :(int)GreenByte :(int)BlueByte;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedByte</td>
<td>The red component</td>
</tr>
<tr>
<td>GreenByte</td>
<td>The green component</td>
</tr>
<tr>
<td>BlueByte</td>
<td>The blue component</td>
</tr>
</tbody>
</table>
SetPageActionMenu

Page properties

Description

Specifies a menu item to run when the document is first opened.

Syntax

Dylib

```c
int DPLSetPageActionMenu(int InstanceID, wchar_t * MenuItem);
```

Objective-C class

```objective-c
- (int)SetPageActionMenu:(NSString *)MenuItem;
```

Parameters

| MenuItem | The MenuItem to call, for example "Print" |

Return values

| 0         | The open action could not be set, there is a problem with the document |
| 1         | The page open action was set successfully |
SetPageBox

Page properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Sets the dimensions of the selected page's boundary rectangles.

The MediaBox represents the physical medium of the page.
The CropBox represents the visible region of the page, the contents will be clipped to this region.
The BleedBox is similar to the CropBox, but is the rectangle used in a production environment.
The TrimBox indicates the intended dimensions of the finished page after trimming, and the ArtBox defines the extent of the page's meaningful content as intended by the page's creator.

Syntax

Dylib

```c
int DPLSetPageBox(int InstanceID, int BoxType, double Left, double Top, double Width, double Height);
```

Objective-C class

```c
- (int)SetPageBox:(int)BoxType :(double)Left :(double)Top :(double)Width :(double)Height;
```

Parameters

<table>
<thead>
<tr>
<th>BoxType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MediaBox</td>
</tr>
<tr>
<td>2</td>
<td>CropBox</td>
</tr>
<tr>
<td>3</td>
<td>BleedBox</td>
</tr>
<tr>
<td>4</td>
<td>TrimBox</td>
</tr>
<tr>
<td>5</td>
<td>ArtBox</td>
</tr>
</tbody>
</table>

| Left    | The horizontal co-ordinate of the left edge of the rectangle |
| Top     | The vertical co-ordinate of the top edge of the rectangle  |
| Width   | The width of the rectangle  |
| Height  | The height of the rectangle  |
SetPageContentFromString

**Page properties, Page layout, Page manipulation**

**Description**

This function allows the content of the selected PDF page to be set. This is for advanced use only! If incorrect information is put into the page's content stream then the PDF file may not load correctly with Acrobat or any other PDF viewer.

In previous versions of Quick PDF Library this function would only set the content of the selected content stream part.

From version 8.11 this function sets the content of the entire page resulting in a single content stream part. The `SetContentStreamFromString` function can be used to set the PDF page description commands of the content stream part selected with the `SelectContentStream` function.

**Syntax**

**Dylib**

```c
int DPLSetPageContentFromString(int InstanceID, char * Source);
```

**Objective-C class**

```c
- (int)SetPageContentFromString:(NSData *)Source;
```

**Parameters**

| Source | The new contents of the page |
SetPageDimensions

Page properties, Page layout

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Set the size of the selected page.

This function updates the MediaBox entry which represents the physical medium of the page and will only affect content subsequently added to the page. This function does not resize the already existing content of the page.

Syntax

Dylib

```c
int DPLSetPageDimensions(int InstanceID, double NewPageWidth, double NewPageHeight);
```

Objective-C class

```objc
```

Parameters

- **NewPageWidth**
  - The new width of the page

- **NewPageHeight**
  - The new height of the page

Return values

- **0**
  - The page size could not be set. This should never occur.

- **1**
  - The page was resized successfully
SetPageLayout

Document properties

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Sets the initial page layout of the document.

Syntax

Dylib

```c
int DPLSetPageLayout(int InstanceID, int NewPageLayout);
```

Objective-C class

```objective-c
- (int)SetPageLayout:(int)NewPageLayout;
```

Parameters

<table>
<thead>
<tr>
<th>NewPageLayout</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Single page</td>
</tr>
<tr>
<td>1</td>
<td>One column</td>
</tr>
<tr>
<td>2</td>
<td>Two columns, odd-numbered pages on left</td>
</tr>
<tr>
<td>3</td>
<td>Two columns, odd-numbered pages on right</td>
</tr>
<tr>
<td>4</td>
<td>Two pages, odd-numbered pages on left</td>
</tr>
<tr>
<td>5</td>
<td>Two pages, odd-numbered pages on right</td>
</tr>
<tr>
<td>6</td>
<td>No preference (setting removed from document)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page mode could not be set</td>
</tr>
<tr>
<td>1</td>
<td>The page mode was set successfully</td>
</tr>
</tbody>
</table>
SetPageMode

**Document properties**

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

**Description**

Sets the initial page mode of the document.

**Syntax**

**Dylib**

```c
int DPLSetPageMode(int InstanceID, int NewPageMode);
```

**Objective-C class**

```objc
- (int)SetPageMode:(int)NewPageMode;
```

**Parameters**

<table>
<thead>
<tr>
<th>NewPageMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal view</td>
</tr>
<tr>
<td>1</td>
<td>Show the outlines pane</td>
</tr>
<tr>
<td>2</td>
<td>Show the thumbnails pane</td>
</tr>
<tr>
<td>3</td>
<td>Show the document in full screen mode</td>
</tr>
<tr>
<td>4</td>
<td>Optional content group panel visible</td>
</tr>
<tr>
<td>5</td>
<td>Attachments panel visible</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The page mode could not be set</td>
</tr>
<tr>
<td>1</td>
<td>The page mode was set successfully</td>
</tr>
</tbody>
</table>
SetPageSize

Page properties, Page layout

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Use this function to set the current page to a named size, for example "A4" or "Letter Landscape".

Syntax

Dylib

    int DPLSetPageSize(int InstanceID, wchar_t * PaperName);

Objective-C class

    - (int)SetPageSize:(NSString *)PaperName;

Parameters

<table>
<thead>
<tr>
<th>PaperName</th>
<th>The name of the paper, one of the following: A0 to A10, B0 to B10, ISOB0 to ISOB10, C0 to C7, DL, Letter, Legal, Statement, Tabloid, Ledger, Executive, Folio. You can make a landscape page by adding the word Landscape after the paper name, for example &quot;A3 Landscape&quot;.</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The specified paper name was not valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The page was resized successfully</td>
</tr>
</tbody>
</table>
SetPageThumbnail

Page manipulation

Description

Sets the selected image as the "thumbnail" for the selected page.

Syntax

**Dylib**

```c
int DPLSetPageThumbnail(int InstanceID);
```

**Objective-C class**

`- (int)SetPageThumbnail`

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No image was selected</td>
</tr>
<tr>
<td>1</td>
<td>The thumbnail was set successfully</td>
</tr>
</tbody>
</table>
SetPageTransparencyGroup

Vector graphics, Text, Page layout

Description

Allows the transparency group for the page to be set. Whenever image are used as masks for other images the page transparency group should be set to ensure consistent results across different versions of PDF viewers.

Syntax

Dylib

```
int DPLSetPageTransparencyGroup(int InstanceID, int CS, int Isolate, int Knockout);
```

Objective-C class

```
-(int)SetPageTransparencyGroup:(int)CS :(int)Isolate :(int)Knockout;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **CS**    | The color space to use:  
  1 = RGB  
  2 = CMYK |
| **Isolate** | This parameter has no effect and is reserved for future use. It should always be set to 0. |
| **Knockout** | Indicates whether items added to the page are drawn over each other or "knocked out" of the page. In knockout mode a "hole" is made through existing objects on the page in the shape of the new object. The new object is then drawn against the background.  
  0 = Do not knockout  
  1 = Knockout |
SetPrecision

Measurement and coordinate units

Description

Use this function to set the precision of numerical values stored in the PDF document. Setting the precision to a lower number will reduce the size of the generated file, while a higher precision will result in a larger file, although objects and graphics will be more accurately positioned. The default precision is 4.

Syntax

Dylib

```c
int DPLSetPrecision(int InstanceID, int NewPrecision);
```

Objective-C class

```c
-(int)SetPrecision:(int)NewPrecision;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewPrecision</td>
<td>The precision to use for subsequent drawing operations. A value from 2 to 8.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The precision specified was out of range</td>
</tr>
<tr>
<td>1</td>
<td>The precision was set successfully</td>
</tr>
</tbody>
</table>
SetPrinterDevModeFromString

Rendering and printing

Description

Sets the printer DEVMODE structure for the next printing operation using the value retrieved from a prior call to `GetPrinterDevModeToString`.

Syntax

**Dylib**

```c
int DPLSetPrinterDevModeFromString(int InstanceID, char * Source);
```

**Objective-C class**

```objective-c
- (int)SetPrinterDevModeFromString:(NSData *)Source;
```

Parameters

| Source | A value returned from the `GetPrinterDevModeToString` function. |
SetRenderCropType
Rendering and printing

Description
Sets the page boundary to use as the cropping area for rendering.

Syntax
Dylib
int DPLSetRenderCropType(int InstanceID, int NewCropType);

Objective-C class
- (int)SetRenderCropType:(int)NewCropType;

Parameters

<table>
<thead>
<tr>
<th>NewCropType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MediaBox</td>
</tr>
<tr>
<td>2</td>
<td>CropBox</td>
</tr>
<tr>
<td>3</td>
<td>BleedBox</td>
</tr>
<tr>
<td>4</td>
<td>TrimBox</td>
</tr>
<tr>
<td>5</td>
<td>ArtBox</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The NewCropType parameter was out of range.</td>
</tr>
<tr>
<td>1</td>
<td>The rendering crop type was set successfully.</td>
</tr>
</tbody>
</table>
SetRenderDCErasePage

Rendering and printing

Description

By default the `RenderPageToDC` and `DARenderPageToDC` functions fill the page area with solid white background before rendering the page contents. This function can be used to suppress the background allowing the page contents to be drawn over any existing content in the supplied device context.

Syntax

Dylib

```c
int DPLSetRenderDCErasePage(int InstanceID, int NewErasePage);
```

Objective-C class

```objc
- (int)SetRenderDCErasePage:(int)NewErasePage;
```

Parameters

<table>
<thead>
<tr>
<th>NewErasePage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No page background is drawn</td>
</tr>
<tr>
<td>1</td>
<td>The page area is filled with a solid white background before rendering</td>
</tr>
</tbody>
</table>
SetRenderDCOffset

Rendering and printing

Description

Sets the position on the device context that the RenderPageToDC and DARenderPageToDC functions use for the top-left corner of the rendered output.

Syntax

Dylib

```c
int DPLSetRenderDCOffset(int InstanceID, int NewOffsetX, int NewOffsetY);
```

Objective-C class

```c
-(int)SetRenderDCOffset:(int)NewOffsetX :(int)NewOffsetY;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewOffsetX</td>
<td>The horizontal offset measured in pixels</td>
</tr>
<tr>
<td>NewOffsetY</td>
<td>The vertical offset measured in pixels</td>
</tr>
</tbody>
</table>
SetRenderOptions

Rendering and printing

Description

Sets various options for the renderer.

Syntax

**Dylib**

```c
int DPLSetRenderOptions(int InstanceID, int OptionID, int NewValue);
```

**Objective-C class**

```c
-(int)SetRenderOptions:(int)OptionID :(int)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>OptionID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Render Formfields</td>
</tr>
<tr>
<td>2</td>
<td>Render Annotations</td>
</tr>
<tr>
<td>3</td>
<td>Render Formfields only</td>
</tr>
<tr>
<td>4</td>
<td>Gamma Correction</td>
</tr>
<tr>
<td>5</td>
<td>ICCBased colorspaces</td>
</tr>
<tr>
<td>6</td>
<td>Progress HWND</td>
</tr>
<tr>
<td>7</td>
<td>Progress Message</td>
</tr>
<tr>
<td>8</td>
<td>Progress Data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Don't render formfields</td>
</tr>
<tr>
<td>1</td>
<td>Render formfields (default)</td>
</tr>
</tbody>
</table>

For **RenderAnnotations**:

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Don't render annotations</td>
</tr>
<tr>
<td>1</td>
<td>Render annotations (default)</td>
</tr>
</tbody>
</table>

For **RenderFormFieldsOnly**:

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Render the page including formfields (default)</td>
</tr>
<tr>
<td>1</td>
<td>Only render the formfields</td>
</tr>
</tbody>
</table>

For **UseGammaCorrection**:

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turn off CMYK gamma correction</td>
</tr>
<tr>
<td>1</td>
<td>Use CMYK Gamma correction (default)</td>
</tr>
</tbody>
</table>

For **Ignore ICCBased colorspaces**:

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Render using ICCBased colorspaces</td>
</tr>
<tr>
<td>1</td>
<td>Ignore ICCBased colorspace corrections</td>
</tr>
</tbody>
</table>

For **progress options**:

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>
SetRenderScale
Rendering and printing

Description

Applies a non-integer scaling to the DPI parameter of subsequent calls to any of the rendering functions.

For example, if the render scale is set to 0.1 and the RenderPageToFile function is called with the DPI parameter set to 125, the resulting image will be rendered with an effective DPI of 12.5.

Syntax

**Dylib**

```c
int DPLSetRenderScale(int InstanceID, double NewScale);
```

**Objective-C class**

```objective-c
- (int)SetRenderScale:(double)NewScale;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewScale</td>
<td>The new render scale</td>
</tr>
</tbody>
</table>
**SetScale**

**Measurement and coordinate units**

**Description**

Scales the co-ordinate system for all subsequent drawing operations. A scale factor of 1 is equivalent to calling `SetMeasurementUnits(0)` which sets the measurement units to be Points. A scale factor of (72 / 25.4) is equivalent to calling `SetMeasurementUnits(1)` which sets the measurement units to be millimetres.

**Syntax**

**Dylib**

```c
int DPLSetScale(int InstanceID, double NewScale);
```

**Objective-C class**

```c
- (int)SetScale:(double)NewScale;
```

**Parameters**

| NewScale        | The scale factor to use |
SetSignProcessField

Security and Signatures

Description

Sets the signature field to use for a digital signature process.
If a field with a specified name is not found a new signature field will be added with the given name. The new field will be invisible (zero width and height) and will be attached to the first page in the document. Call SetSignProcessFieldBounds to set location and size of new form field and SetSignProcessFieldPage to set the page it is placed on.

Syntax

**Dylib**

```c
int DPLSetSignProcessField(int InstanceID, int SignProcessID, wchar_t * SignatureFieldName);
```

**Objective-C class**

```c
-(int)SetSignProcessField:(int)SignProcessID:(NSString *)SignatureFieldName;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SignProcessID</strong></td>
<td>A value returned by the <code>NewSignProcessFromFile</code>, <code>NewSignProcessFromStream</code> or <code>NewSignProcessFromString</code> functions.</td>
</tr>
<tr>
<td><strong>SignatureFieldName</strong></td>
<td>The name of the signature form field</td>
</tr>
</tbody>
</table>
SetSignProcessFieldBounds

Security and Signatures

Description

Sets the location and size of the signature field in the specified digital signature process.

Syntax

Dylib

```c
int DPLSetSignProcessFieldBounds(int InstanceID, int SignProcessID,
      double Left, double Top, double Width, double Height);
```

Objective-C class

```c
-(int)SetSignProcessFieldBounds:(int)SignProcessID :(double)Left
      :(double)Top :(double)Width :(double)Height;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>The horizontal coordinate of the left edge of the area measured in points from the left edge of the media box.</td>
</tr>
<tr>
<td>Top</td>
<td>The vertical coordinate of the top edge of the area measured in points from the bottom edge of the media box.</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the area measured in points.</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the area measured in points.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalid SignProcessID parameter</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
### SetSignProcessFieldImageFromFile

**Security and Signatures**

**Description**

Sets the image to use for a visual signature field in the specified digital signature process.

The *SetSignProcessFieldBounds* function can be used to specify the location and size of the signature field.

**Syntax**

**Dylib**

```c
int DPLSetSignProcessFieldImageFromFile(int InstanceID, int SignProcessID, wchar_t * ImageFileName, int Options);
```

**Objective-C class**

```
-(int)SetSignProcessFieldImageFromFile:(int)SignProcessID :(NSString *)ImageFileName :(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SignProcessID</strong></td>
<td>A value returned by the <em>NewSignProcessFromFile</em>, <em>NewSignProcessFromStream</em> or <em>NewSignProcessFromString</em> functions.</td>
</tr>
<tr>
<td><strong>ImageFileName</strong></td>
<td>The path and file name of the image to use for the visual signature.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Image could not be added</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetSignProcessFieldPage

Security and Signatures

**Description**

Specifies the page number where the new signature field will be placed. By default the signature field will be attached to the first page in the document.

If the field name specified by `SetSignProcessField` already exists then a call to this function will be ignored and the field will remain on the page it is currently attached to.

**Syntax**

**Dylib**

```c
int DPLSetSignProcessFieldPage(int InstanceID, int SignProcessID, int SignaturePage);
```

**Objective-C class**

```c
- (int)SetSignProcessFieldPage:(int)SignProcessID :(int)SignaturePage;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SignProcessID</strong></td>
<td>A value returned by the <code>NewSignProcessFromFile</code>, <code>NewSignProcessFromStream</code> or <code>NewSignProcessFromString</code> functions.</td>
</tr>
<tr>
<td><strong>SignaturePage</strong></td>
<td>The number of the page that the signature should appear on.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The <code>SignProcessID</code> parameter is invalid</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
**SetSignProcessInfo**

**Security and Signatures**

**Description**

Sets the signing information for a digital signature process. This information includes the reason for signing, the location and contact info. The supplied details will be displayed by the PDF viewer when the signature has been validated.

**Syntax**

**Dylib**

```c
int DPLSetSignProcessInfo(int InstanceID, int SignProcessID, wchar_t * Reason, wchar_t * Location, wchar_t * ContactInfo);
```

**Objective-C class**

```objc
- (int)SetSignProcessInfo:(int)SignProcessID :(NSString *)Reason :(NSString *)Location :(NSString *)ContactInfo;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason</td>
<td>The reason for signing</td>
</tr>
<tr>
<td>Location</td>
<td>The location that the signing was done</td>
</tr>
<tr>
<td>ContactInfo</td>
<td>The contact information of the signer</td>
</tr>
</tbody>
</table>
SetSignProcessKeyset
Security and Signatures

Description
Sets the MS Crypto API keyset value.

Syntax
Dylib

```c
int DPLSetSignProcessKeyset(int InstanceID, int SignProcessID, int KeysetID);
```

Objective-C class

```c
-(int)SetSignProcessKeyset:(int)SignProcessID :(int)KeysetID;
```

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>KeysetID</td>
<td>1 = CRYPT_USER_KEYSET (Default)</td>
</tr>
<tr>
<td></td>
<td>2 = CRYPT_MACHINE_KEYSET</td>
</tr>
</tbody>
</table>

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalid SignProcessID parameter or KeysetID out of range</td>
</tr>
<tr>
<td>1</td>
<td>Signature process keyset was set successfully</td>
</tr>
</tbody>
</table>
SetSignProcessPFXFromFile

Security and Signatures

Description

Sets a file to use as the digital identity for a digital signature process. The file should be in PKCS #12 format, also known as a PFX file, and contain a private key as well as an X.509 certificate. PFX files are usually protected with a password.

Syntax

Dylib

```c
int DPLSetSignProcessPFXFromFile(int InstanceID, int SignProcessID, wchar_t * PFXFileName, wchar_t * PFXPassword);
```

Objective-C class

```objc
-(int)SetSignProcessPFXFromFile:(int)SignProcessID : (NSString *)PFXFileName : (NSString *)PFXPassword;
```

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PFXFileName</td>
<td>The path and name of the PFX signature file (PKCS #12 format).</td>
</tr>
<tr>
<td>PFXPassword</td>
<td>The password to open the PFX file.</td>
</tr>
</tbody>
</table>

SetSignProcessPassthrough

Description

Sets the signature process to passthrough mode.
In this mode, the PDF is prepared using a placeholder for the signature data. The user can then replace this placeholder with the signature data of their choice using the GetSignProcessByteRange function to determine the byte range that the signature hashing should be calculated over.

Syntax

Dylib

```c
int DPLSetSignProcessPassthrough(int InstanceID, int SignProcessID, int SignatureLength);
```

Objective-C class

```objc
- (int)SetSignProcessPassthrough:(int)SignProcessID :(int)SignatureLength;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignatureLength</td>
<td>The length in bytes of the raw binary signature data. This value will be doubled when allocating the space in the PDF as the signature data should be written using hex encoding (two characters per raw byte).</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The signature could not be set into passthrough mode.</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetSignProcessSubFilter

Security and Signatures

Description

Sets the SubFilter entry for a digital signature process, specifying the encoding of the signature value.

Syntax

Dylib

```c
int DPLSetSignProcessSubFilter(int InstanceID, int SignProcessID, int SubFilter);
```

Objective-C class

```c
-(int)SetSignProcessSubFilter:(int)SignProcessID :(int)SubFilter;
```

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SubFilter</td>
<td>1 = adbe.pkcs7.sha1  2 = adbe.pkcs7.detached</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The SignProcessID parameter was invalid or the SubFilter parameter was out of range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
SetTabOrderMode
Form fields, Annotations and hotspot links

Description

This function sets the default tabbing order mode for the currently selected page for all of the annotations including the formfields when tabbing in a PDF viewer.

If you use SetFormFieldTabOrder to define a custom tabbing order then you should set the tabbing order to 'S'tructure mode.

Syntax

Dylib

```c
int DPLSetTabOrderMode(int InstanceID, wchar_t * Mode);
```

Objective-C class

```c
- (int)SetTabOrderMode:(NSString *)Mode;
```

Parameters

<table>
<thead>
<tr>
<th>Mode</th>
<th>The mode string</th>
</tr>
</thead>
<tbody>
<tr>
<td>'S'</td>
<td>Structure mode - use the order of the Annots and/or Formfields as they are defined</td>
</tr>
<tr>
<td>'R'</td>
<td>Row Mode - Left to right, top to bottom order</td>
</tr>
<tr>
<td>'C'</td>
<td>Column Mode - Top to bottom, left to right order</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The tabbing mode was not set correctly</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>
Sets the color of the specified table border using the RGB color space.

**Syntax**

**Dylib**

```c
int DPLSetTableBorderColor(int InstanceID, int TableID, int BorderIndex,
                               double Red, double Green, double Blue);
```

**Objective-C class**

```objective-c
-(int)SetTableBorderColor:(int)TableID :(int)BorderIndex :(double)Red
                    :(double)Green :(double)Blue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TableID</strong></td>
<td>A TableID returned by the <code>CreateTable</code> function</td>
</tr>
</tbody>
</table>
| **BorderIndex** | 0 = All borders  
1 = Left  
2 = Top  
3 = Right  
4 = Bottom |
| **Red** | The red component of the color, a value from 0 to 1 |
| **Green** | The green component of the color, a value from 0 to 1 |
| **Blue** | The blue component of the color, a value from 0 to 1 |
**SetTableBorderColorCMYK**

*Color, Page layout*

**Description**

Sets the color of the specified table border using the CMYK color space.

**Syntax**

**Dylib**

```
int DPLSetTableBorderColorCMYK(int InstanceID, int TableID,
                                int BorderIndex, double C, double M, double Y, double K);
```

**Objective-C class**

```
-(int)SetTableBorderColorCMYK:(int)TableID :(int)BorderIndex :(double)C
                          :(double)M :(double)Y :(double)K;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TableID</strong></td>
<td>A TableID returned by the <em>CreateTable</em> function</td>
</tr>
</tbody>
</table>
| **BorderIndex** | 0 = All borders  
1 = Left  
2 = Top  
3 = Right  
4 = Bottom |
| **C** | The cyan component of the color, a value from 0 to 1 |
| **M** | The magenta component of the color, a value from 0 to 1 |
| **Y** | The yellow component of the color, a value from 0 to 1 |
| **K** | The black component of the color, a value from 0 to 1 |
SetTableBorderWidth

Description

Sets the width of the specified table border.

Syntax

Dylib

```c
int DPLSetTableBorderWidth(int InstanceID, int TableID, int BorderIndex, double NewWidth);
```

Objective-C class

```objc
- (int)SetTableBorderWidth:(int)TableID :(int)BorderIndex :(double)NewWidth;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <strong>CreateTable</strong> function</td>
</tr>
<tr>
<td>BorderIndex</td>
<td>0 = All borders</td>
</tr>
<tr>
<td></td>
<td>1 = Left</td>
</tr>
<tr>
<td></td>
<td>2 = Top</td>
</tr>
<tr>
<td></td>
<td>3 = Right</td>
</tr>
<tr>
<td></td>
<td>4 = Bottom</td>
</tr>
<tr>
<td>NewWidth</td>
<td>The new width of the specified table border</td>
</tr>
</tbody>
</table>
**SetTableCellAlignment**

**Page layout**

**Description**

Sets the vertical and horizontal alignment of one or more cells.

**Syntax**

**Dylib**

```c
int DPLSetTableCellAlignment(int InstanceID, int TableID, int FirstRow,
                               int FirstColumn, int LastRow, int LastColumn,
                               int NewCellAlignment);
```

**Objective-C class**

```objc
- (int)SetTableCellAlignment:(int)TableID :(int)FirstRow :(int)FirstColumn
  :(int)LastRow :(int)LastColumn :(int)NewCellAlignment;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <code>CreateTable</code> function</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
</tbody>
</table>
| NewCellAlignment | 0 = top left  
|               | 1 = top center  
|               | 2 = top right  
|               | 3 = middle left  
|               | 4 = middle center  
|               | 5 = middle right  
|               | 6 = bottom left  
|               | 7 = bottom center  
|               | 8 = bottom right            |
**SetTableCellBackgroundColor**  
*Color, Page layout*

**Description**

Sets the background color of one or more cells using the RGB color space.

**Syntax**

**Dylib**

```c
int DPLSetTableCellBackgroundColor(int InstanceID, int TableID,
    int FirstRow, int FirstColumn, int LastRow, int LastColumn,
    double Red, double Green, double Blue);
```

**Objective-C class**

```c
- (int)SetTableCellBackgroundColor:(int)TableID :(int)FirstRow
    :(int)FirstColumn :(int)LastRow :(int)LastColumn :(double)Red
    :(double)Green :(double)Blue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TableID</strong></td>
<td>A TableID returned by the <code>CreateTable</code> function</td>
</tr>
<tr>
<td><strong>FirstRow</strong></td>
<td>The the number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td><strong>FirstColumn</strong></td>
<td>The the number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td><strong>LastRow</strong></td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td><strong>LastColumn</strong></td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td><strong>Red</strong></td>
<td>The red component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td><strong>Green</strong></td>
<td>The green component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td><strong>Blue</strong></td>
<td>The blue component of the color, a value from 0 to 1</td>
</tr>
</tbody>
</table>
SetTableCellBackgroundColorCMYK

**Color, Page layout**

**Description**

Sets the background color of one or more cells using the CMYK color space.

**Syntax**

**Dylib**

```c
int DPLSetTableCellBackgroundColorCMYK(int InstanceID, int TableID,
int FirstRow, int FirstColumn, int LastRow, int LastColumn,
double C, double M, double Y, double K);
```

**Objective-C class**

```objective-c
- (int)SetTableCellBackgroundColorCMYK:(int)TableID :(int)FirstRow
 :(int)FirstColumn :(int)LastRow :(int)LastColumn :(double)C
 :(double)M :(double)Y :(double)K;
```

**Parameters**

| **TableID** | A TableID returned by the `CreateTable` function |
| **FirstRow** | The number of the first row to set. Top row is row number 1. |
| **FirstColumn** | The number of the first column to set. Left most column is column number 1. |
| **LastRow** | The number of the final row to set |
| **LastColumn** | The number of the final column to set |
| **C** | The cyan component of the color, a value from 0 to 1 |
| **M** | The magenta component of the color, a value from 0 to 1 |
| **Y** | The yellow component of the color, a value from 0 to 1 |
| **K** | The black component of the color, a value from 0 to 1 |
SetTableCellBorderColor

**Color, Page layout**

**Description**

Sets the color of one or more cell borders using the RGB color space.

**Syntax**

**Dylib**

```c
int DPLSetTableCellBorderColor(int InstanceID, int TableID, int FirstRow,
int FirstColumn, int LastRow, int LastColumn, int BorderIndex,
double Red, double Green, double Blue);
```

**Objective-C class**

```objective-c
- (int)SetTableCellBorderColor:(int)TableID :(int)FirstRow
 :(int)FirstColumn :(int)LastRow :(int)LastColumn
 :(int)BorderIndex :(double)Red :(double)Green :(double)Blue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <strong>CreateTable</strong> function</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td>BorderIndex</td>
<td>0 = All borders 1 = Left 2 = Top 3 = Right 4 = Bottom</td>
</tr>
<tr>
<td>Red</td>
<td>The red component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color, a value from 0 to 1</td>
</tr>
</tbody>
</table>
SetTableCellBorderColorCMYK

**Color, Page layout**

**Description**

Sets the color of one or more cell borders using the CMYK color space.

**Syntax**

**Dylib**

```c
int DPLSetTableCellBorderColorCMYK(int InstanceID, int TableID, int FirstRow, int FirstColumn, int LastRow, int LastColumn, int BorderIndex, double C, double M, double Y, double K);
```

**Objective-C class**

```c
-(int)SetTableCellBorderColorCMYK:(int)TableID :(int)FirstRow :(int)FirstColumn :(int)LastRow :(int)LastColumn :(int)BorderIndex :(double)C :(double)M :(double)Y :(double)K;
```

**Parameters**

<table>
<thead>
<tr>
<th>TableID</th>
<th>A TableID returned by the CreateTable function</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstRow</td>
<td>The number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td>BorderIndex</td>
<td>0 = All borders  1 = Left  2 = Top  3 = Right  4 = Bottom</td>
</tr>
<tr>
<td>C</td>
<td>The cyan component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td>M</td>
<td>The magenta component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td>Y</td>
<td>The yellow component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td>K</td>
<td>The black component of the color, a value from 0 to 1</td>
</tr>
</tbody>
</table>
SetTableCellBorderWidth

Description
Sets the width of one or more cell borders.

Syntax
Dylib

```c
int DPLSetTableCellBorderWidth(int InstanceID, int TableID, int FirstRow,
                               int FirstColumn, int LastRow, int LastColumn, int BorderIndex,
                               double NewWidth);
```

Objective-C class

```c
-(int)SetTableCellBorderWidth:(int)TableID :(int)FirstRow
                          :(int)FirstColumn :(int)LastRow :(int)LastColumn
                          :(int)BorderIndex :(double)NewWidth;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the CreateTable function</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The the number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The the number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td>BorderIndex</td>
<td>0 = All borders 1 = Left 2 = Top 3 = Right 4 = Bottom</td>
</tr>
<tr>
<td>NewWidth</td>
<td>The new width of the specified border</td>
</tr>
</tbody>
</table>
SetTableCellContent

Description

Sets the content of the specified cell. The content will be drawn with the equivalent of the `DrawHTMLText` function, prefixed with the necessary paragraph alignment, font size and font color tags.

Syntax

**Dylib**

```c
int DPLSetTableCellContent(int InstanceID, int TableID, int RowNumber,
                          int ColumnNumber, wchar_t * HTMLText);
```

**Objective-C class**

```objective-c
- (int)SetTableCellContent:(int)TableID :(int)RowNumber :(int)ColumnNumber
                         :(NSString *)HTMLText;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TableID</strong></td>
<td>A TableID returned by the <code>CreateTable</code> function</td>
</tr>
<tr>
<td><strong>RowNumber</strong></td>
<td>The row number of the cell. Top row is row number 1.</td>
</tr>
<tr>
<td><strong>ColumnNumber</strong></td>
<td>The column number of the cell. Left most column is column number 1.</td>
</tr>
<tr>
<td><strong>HTMLText</strong></td>
<td>The HTML text to place into the specified cell</td>
</tr>
</tbody>
</table>
SetTableCellPadding

Description

Sets the padding of one or more cells. The padding is the distance from the cell boundary to the text contents. The padding is set on the side of the specified border.

Syntax

Dylib

```c
int DPLSetTableCellPadding(int InstanceID, int TableID, int FirstRow, int FirstColumn, int LastRow, int LastColumn, int BorderIndex, double NewPadding);
```

Objective-C class

```objc
- (int)SetTableCellPadding:(int)TableID :(int)FirstRow :(int)FirstColumn :(int)LastRow :(int)LastColumn :(int)BorderIndex :(double)NewPadding;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <code>CreateTable</code> function</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The the number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The the number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td>BorderIndex</td>
<td>0 = All borders 1 = Left 2 = Top 3 = Right 4 = Bottom</td>
</tr>
<tr>
<td>NewPadding</td>
<td>The new padding on the side of the specified border</td>
</tr>
</tbody>
</table>
**SetTableCellTextColor**

**Color, Page layout**

**Description**

Sets the default text color of one or more cells using the RGB color space.

**Syntax**

**Dylib**

```c
int DPLSetTableCellTextColor(int InstanceID, int TableID, int FirstRow,
    int FirstColumn, int LastRow, int LastColumn, double Red,
    double Green, double Blue);
```

**Objective-C class**

```objc
-(int)SetTableCellTextColor:(int)TableID :(int)FirstRow :(int)FirstColumn
    :(int)LastRow :(int)LastColumn :(double)Red :(double)Green
    :(double)Blue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TableID</strong></td>
<td>A TableID returned by the <strong>CreateTable</strong> function</td>
</tr>
<tr>
<td><strong>FirstRow</strong></td>
<td>The number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td><strong>FirstColumn</strong></td>
<td>The number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td><strong>LastRow</strong></td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td><strong>LastColumn</strong></td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td><strong>Red</strong></td>
<td>The red component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td><strong>Green</strong></td>
<td>The green component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td><strong>Blue</strong></td>
<td>The blue component of the color, a value from 0 to 1</td>
</tr>
</tbody>
</table>
**SetColorTableCellTextColorCMYK**

**Color, Page layout**

**Description**

Sets the default text color of one or more cells using the CMYK color space.

**Syntax**

**Dylib**

```c
int DPLSetTableCellTextColorCMYK(int InstanceID, int TableID,
    int FirstRow, int FirstColumn, int LastRow, int LastColumn,
    double C, double M, double Y, double K);
```

**Objective-C class**

```objc
- (int)SetTableCellTextColorCMYK:(int)TableID :(int)FirstRow
    :(int)FirstColumn :(int)LastRow :(int)LastColumn :(double)C
    :(double)M :(double)Y :(double)K;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TableID</strong></td>
<td>A TableID returned by the <strong>CreateTable</strong> function</td>
</tr>
<tr>
<td><strong>FirstRow</strong></td>
<td>The the number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td><strong>FirstColumn</strong></td>
<td>The the number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td><strong>LastRow</strong></td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td><strong>LastColumn</strong></td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>The cyan component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>The magenta component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td>The yellow component of the color, a value from 0 to 1</td>
</tr>
<tr>
<td><strong>K</strong></td>
<td>The black component of the color, a value from 0 to 1</td>
</tr>
</tbody>
</table>
SetTableCellTextSize

Page layout

Description

Sets the default text size of one or more cells.

Syntax

Dylib

```c
int DPLSetTableCellTextSize(int InstanceID, int TableID, int FirstRow,
   int FirstColumn, int LastRow, int LastColumn,
   double NewTextSize);
```

Objective-C class

```objc
- (int)SetTableCellTextSize:(int)TableID :(int)FirstRow :(int)FirstColumn
   :(int)LastRow :(int)LastColumn :(double)NewTextSize;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <code>CreateTable</code> function</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td>NewTextSize</td>
<td>The new text size for the specified cell range</td>
</tr>
</tbody>
</table>
SetTableColumnWidth

Page layout

Description

Sets the width of one or more table columns.

Syntax

**Dylib**

```c
int DPLSetTableColumnWidth(int InstanceID, int TableID, int FirstColumn, int LastColumn, double NewWidth);
```

**Objective-C class**

```objc
- (int)SetTableColumnWidth:(int)TableID :(int)FirstColumn :(int)LastColumn :(double)NewWidth;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the <strong>CreateTable</strong> function</td>
</tr>
<tr>
<td>FirstColumn</td>
<td>The number of the first column to set. Left most column is column number 1.</td>
</tr>
<tr>
<td>LastColumn</td>
<td>The number of the final column to set</td>
</tr>
<tr>
<td>NewWidth</td>
<td>The new width of the specified columns</td>
</tr>
</tbody>
</table>
SetTableRowHeight

Description

Sets the height of one or more table rows. If the row height is set to zero (default) the row will autosize to the maximum height of the contents of all the cells in the row.

Syntax

Dylib

    int DPLSetTableRowHeight(int InstanceID, int TableID, int FirstRow, int LastRow, double NewHeight);

Objective-C class

    - (int)SetTableRowHeight:(int)TableID :(int)FirstRow :(int)LastRow :(double)NewHeight;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the CreateTable function</td>
</tr>
<tr>
<td>FirstRow</td>
<td>The the number of the first row to set. Top row is row number 1.</td>
</tr>
<tr>
<td>LastRow</td>
<td>The number of the final row to set</td>
</tr>
</tbody>
</table>
| NewHeight | 0 = auto size  
Non-zero = the new maximum height of the row |
**Description**

Sets a table to use thin border lines instead of bevelled edges. These lines appear as a single pixel width for all zoom levels.

The lines are drawn using the color specified by the Red, Green and Blue parameters.

**Syntax**

**Dylib**

```c
int DPLSetTableThinBorders(int InstanceID, int TableID, int ThinBorders, double Red, double Green, double Blue);
```

**Objective-C class**

```c
-(int)SetTableThinBorders:(int)TableID :(int)ThinBorders :(double)Red :(double)Green :(double)Blue;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TableID</strong></td>
<td>A TableID returned by the <strong>CreateTable</strong> function</td>
</tr>
</tbody>
</table>
| **ThinBorders** | 0 = Use bevelled edges (the default)  
1 = Use thin lines |
| **Red** | The red component of the color, a value from 0 to 1 |
| **Green** | The green component of the color, a value from 0 to 1 |
| **Blue** | The blue component of the color, a value from 0 to 1 |

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The table line style could not be set</td>
</tr>
<tr>
<td>1</td>
<td>The table line style was set successfully</td>
</tr>
</tbody>
</table>
SetTableThinBordersCMYK

Description

Sets a table to use thin border lines instead of bevelled edges. These lines appear as a single pixel width for all zoom levels.

The lines are drawn using the color specified by the C, M, Y and K parameters.

Syntax

Dylib

```c
int DPLSetTableThinBordersCMYK(int InstanceID, int TableID,
 int ThinBorders, double C, double M, double Y, double K);
```

Objective-C class

```objective-c
-(int)SetTableThinBordersCMYK:(int)TableID :(int)ThinBorders :(double)C
 :(double)M :(double)Y :(double)K;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableID</td>
<td>A TableID returned by the CreateTable function</td>
</tr>
</tbody>
</table>
| ThinBorders | 0 = Use bevelled edges (the default)  
                         1 = Use thin lines |
| C         | The cyan component of the color, a value from 0 to 1 |
| M         | The magenta component of the color, a value from 0 to 1 |
| Y         | The yellow component of the color, a value from 0 to 1 |
| K         | The black component of the color, a value from 0 to 1 |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The table line style could not be set</td>
</tr>
<tr>
<td>1</td>
<td>The table line style was set successfully</td>
</tr>
</tbody>
</table>
**SetTempFile**

**Miscellaneous functions**

**Description**

Specifies a temporary file which can be used during operations such as encryption. This allows large documents to be processed without running out of memory.

**Syntax**

**Dylib**

```c
int DPLSetTempFile(int InstanceID, wchar_t * FileName);
```

**Objective-C class**

```objective-c
- (int)SetTempFile:(NSString *)FileName;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileName</strong></td>
<td>The full path and file to use as a temporary file. This path must have write access by the running process. For example, &quot;c:\temp\pdftemp.dat&quot;.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The path specified was not valid. A temporary file could not be created.</td>
</tr>
<tr>
<td>1</td>
<td>The temporary file could be created successfully</td>
</tr>
</tbody>
</table>
SetTempPath

Miscellaneous functions

Description

Sets the folder to use for storage of temporary files which are generated by functions such as MergeFileList.

Syntax

**Dylib**

```c
int DPLSetTempPath(int InstanceID, wchar_t * NewPath);
```

**Objective-C class**

```objc
- (int)SetTempPath:(NSString *)NewPath;
```

Parameters

- **NewPath** The new folder to use. This folder must exist already, it will not be created.

Return values

- **0** The specified folder does not exists or does not have read/write access
- **1** The temporary path was set successfully
SetTextAlign

Text

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Set the alignment of subsequent text drawn with the DrawText, DrawWrappedText or DrawMultiLineText functions.

Syntax

**Dylib**

```c
int DPLSetTextAlign(int InstanceID, int TextAlign);
```

**Objective-C class**

```objective-c
- (int)SetTextAlign:(int)TextAlign;
```

Parameters

<table>
<thead>
<tr>
<th>TextAlign</th>
<th>The alignment of the text:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Left aligned (default)</td>
</tr>
<tr>
<td>1</td>
<td>Center aligned</td>
</tr>
<tr>
<td>2</td>
<td>Right aligned</td>
</tr>
<tr>
<td>3</td>
<td>Justified</td>
</tr>
<tr>
<td>4</td>
<td>Force justified</td>
</tr>
<tr>
<td>5</td>
<td>Last line justified</td>
</tr>
<tr>
<td>Anything else</td>
<td>Left aligned</td>
</tr>
</tbody>
</table>

"Justified" mode will not justify a line if it's the last line in a paragraph or if the line ends with a hard-break. "Force justified" will justify every line even if it's the last line or if it ends with a hard-break. "Last line justified" will not justify the last line of text, this is useful when different blocks of text are drawn one after the other.
SetTextCharSpacing

Description
Sets the amount of space to add between characters for subsequently drawn text.

Syntax

Dylib

```c
int DPLSetTextCharSpacing(int InstanceID, double CharSpacing);
```

Objective-C class

```objective-c
- (int)SetTextCharSpacing:(double)CharSpacing;
```

Parameters

| CharSpacing | The amount of extra space to add between characters |
SetTextColor

Text, Color

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Sets the color for any subsequently drawn text. The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

```c
int DPLSetTextColor(int InstanceID, double Red, double Green, double Blue);
```

Objective-C class

```objective-c
- (int)SetTextColor:(double)Red :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>Red</th>
<th>The red component of the color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>The green component of the color</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color</td>
</tr>
</tbody>
</table>

SetTextColorCMYK

Text, Color

Description

Sets the color for any subsequently drawn text. Similar to the SetTextColor function, but the color components are specified in the CMYK color space (Cyan, Magenta, Yellow, Black). The values of the color parameters range from 0 to 1, with 0 indicating 0% and 1 indicating 100% of the color.

Syntax

Dylib

int DPLSetTextColorCMYK(int InstanceID, double C, double M, double Y, double K);

Objective-C class

- (int)SetTextColorCMYK:(double)C :(double)M :(double)Y :(double)K;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>The cyan component of the color</td>
</tr>
<tr>
<td>M</td>
<td>The magenta component of the color</td>
</tr>
<tr>
<td>Y</td>
<td>The yellow component of the color</td>
</tr>
<tr>
<td>K</td>
<td>The black component of the color</td>
</tr>
</tbody>
</table>
SetTextColorSep

**Text, Color**

**Description**

Sets the color for any subsequently drawn text. Similar to the *SetTextColor* function, but a tint of a separation color added with the *AddSeparationColor* function is used.

**Syntax**

**Dylib**

```c
int DPLSetTextColorSep(int InstanceID, wchar_t * ColorName, double Tint);
```

**Objective-C class**

```objective-c
- (int)SetTextColorSep:(NSString *)ColorName :(double)Tint;
```

**Parameters**

<table>
<thead>
<tr>
<th>ColorName</th>
<th>The name of the separation color that was used with the <em>AddSeparationColor</em> function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tint</td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The separation color name could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The text color was set successfully</td>
</tr>
</tbody>
</table>
SetTextExtractionArea

**Text, Extraction**

**Description**

Sets the area for certain modes of text extraction. Any text that appears outside this area will be excluded from the results. This function has no effect on text extraction using modes 0 to 2.

From 8.13, this function sets the text extraction area for the selected document only. It also only affects the results of the **GetPageText** function.

To adjust the text extraction for the **ExtractFilePageText** and **DAExtractPageText** functions, use the new **DASetsTextExtractionArea** function.

The coordinate values passed into this function are specified using the units set with the **SetMeasurementUnits** function and the origin set with the **SetOrigin** function.

The area limitation can be removed by calling this function with a value of zero for both the Width and Height parameters.

**Syntax**

**Dylib**

```c
int DPLSetTextExtractionArea(int InstanceID, double Left, double Top, double Width, double Height);
```

**Objective-C class**

```c
- (int)SetTextExtractionArea:(double)Left :(double)Top :(double)Width :(double)Height;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left</strong></td>
<td>The horizontal coordinate of the left edge of the area</td>
</tr>
<tr>
<td><strong>Top</strong></td>
<td>The vertical coordinate of the top edge of the area</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The width of the area</td>
</tr>
<tr>
<td><strong>Height</strong></td>
<td>The height of the area</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The text extraction area was set successfully</td>
</tr>
<tr>
<td>2</td>
<td>The text extraction area was cleared</td>
</tr>
</tbody>
</table>
SetTextExtractionOptions

Text, Extraction

Description

Sets various options that affect the text extraction functionality.

From 8.13, this function sets the text extraction options for the selected document only. It also only affects the results of the `GetPageText` function.

To adjust the text extraction for the `ExtractFilePageText` and `DAExtractPageText` functions, use the new `DASetTextExtractionOptions` function.

Syntax

Dylib

```c
int DPLSetTextExtractionOptions(int InstanceID, int OptionID,
                                int NewValue);
```

Objective-C class

```c
-(int)SetTextExtractionOptions:(int)OptionID :(int)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>OptionID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use Font information matching when grouping to separate text blocks</td>
</tr>
<tr>
<td>2</td>
<td>Use Color information matching when grouping to separate text blocks</td>
</tr>
<tr>
<td>3</td>
<td>Use Block information matching when grouping to separate text blocks</td>
</tr>
<tr>
<td>4</td>
<td>Output CMYK color values</td>
</tr>
<tr>
<td>5</td>
<td>Sort text blocks based on top left position</td>
</tr>
<tr>
<td>6</td>
<td>Descenders from font metrics</td>
</tr>
<tr>
<td>7</td>
<td>Ignore overlaps</td>
</tr>
<tr>
<td>8</td>
<td>Ignore duplicates</td>
</tr>
<tr>
<td>9</td>
<td>Split on double space</td>
</tr>
<tr>
<td>10</td>
<td>Trim characters outside area</td>
</tr>
<tr>
<td>11</td>
<td>Alternative block matching</td>
</tr>
<tr>
<td>12</td>
<td>Ignore rotated text blocks</td>
</tr>
<tr>
<td>13</td>
<td>Trim leading and trailing whitespace from blocks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For OptionID = 1, 2, 3 and 6:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Use, 1 = Ignore</td>
</tr>
<tr>
<td>For OptionID = 4:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Show as RGB (default), 1 = Show as CMYK</td>
</tr>
<tr>
<td>For OptionID = 5:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not sort blocks (default), 1 = Sort blocks</td>
</tr>
<tr>
<td>For OptionID = 7, 8 and 12:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not ignore, 1 = Ignore</td>
</tr>
<tr>
<td>OptionID = 9:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not split on double space (default)</td>
</tr>
<tr>
<td>1</td>
<td>Split on double space</td>
</tr>
<tr>
<td>OptionID = 10:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not trim characters outside area (default)</td>
</tr>
<tr>
<td>1</td>
<td>Trim characters outside area</td>
</tr>
<tr>
<td>OptionID = 11:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Regular block matching</td>
</tr>
<tr>
<td>1</td>
<td>Alternative block matching</td>
</tr>
<tr>
<td>OptionID = 13:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Do not trim leading or trailing whitespace</td>
</tr>
<tr>
<td>1</td>
<td>Trim leading and trailing whitespace</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The OptionID or NewValue parameter was not valid</td>
</tr>
<tr>
<td>1</td>
<td>The text extraction option was set successfully</td>
</tr>
</tbody>
</table>
SetTextExtractionScaling

Text, Extraction

Description

Sets the scaling to use for the GetPageText function in Mode 7. This controls the number of rows and columns in the monospaced text output.

The setting is applied to the selected document only.

To adjust the text extraction for the ExtractFilePageText and DAExtractPageText functions, use the DASetTextExtractionScaling function.

Syntax

Dylib

```c
int DPLSetTextExtractionScaling(int InstanceID, int Options, double Horizontal, double Vertical);
```

Objective-C class

```objc
-(int)SetTextExtractionScaling:(int)Options :(double)Horizontal :(double)Vertical;
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options</strong></td>
<td>Should always be set to 0. This indicates a scaling factor will be set for the Horizontal and Vertical parameters, with a default value of 5 for horizontal and 8 for vertical. Smaller values stretch the text out into more rows/columns.</td>
</tr>
<tr>
<td><strong>Horizontal</strong></td>
<td>The scaling to use for the horizontal axis in units defined by the Options parameter.</td>
</tr>
<tr>
<td><strong>Vertical</strong></td>
<td>The scaling to use for the vertical axis in units defined by the Options parameter.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Options parameter was not valid or a value less than 1 was used for the Horizontal or Vertical parameters.</td>
</tr>
<tr>
<td>1</td>
<td>Text extraction scaling was set successfully.</td>
</tr>
</tbody>
</table>
Sets the word gap ratio for the text extraction functionality.

From 8.13, this function sets the text extraction options for the selected document only. It affects the results of any of the text extraction function that use options 3, 4, 5, 6, 7 or 8.

To adjust the text extraction for the ExtractFilePageText and DAExtractPageText functions, use the new DASETTextExtractionWordGap function.

The word gap ratio is the maximum distance between two text blocks specified as the ratio of the horizontal distance between the blocks to the height of the text.

The default initial value is 0.7 and smaller values will allow closer distances between words.

**Syntax**

**Dylib**

```c
int DPLSetTextExtractionWordGap(int InstanceID, double NewWordGap);
```

**Objective-C class**

```c
- (int)SetTextExtractionWordGap:(double)NewWordGap;
```

**Parameters**

<table>
<thead>
<tr>
<th>NewWordGap</th>
<th>The new WordGap ratio</th>
</tr>
</thead>
</table>

**Return values**

| 1                           | The word gap ratio was set successfully. |
SetTextHighlight

Text

Description

Sets the text highlighting mode for subsequently drawn text.

Syntax

Dylib

```c
int DPLSetTextHighlight(int InstanceID, int Highlight);
```

Objective-C class

```objective-c
-(int)SetTextHighlight:(int)Highlight;
```

Parameters

<table>
<thead>
<tr>
<th>Highlight</th>
<th>The text highlighting mode to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Square</td>
</tr>
<tr>
<td>2</td>
<td>Rounded</td>
</tr>
</tbody>
</table>
SetTextHighlightColor

Text, Color

Description

Sets the color used to highlight text.

Syntax

Dylib

```c
int DPLSetTextHighlightColor(int InstanceID, double Red, double Green, double Blue);
```

Objective-C class

```c
- (int)SetTextHighlightColor:(double)Red :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Red</strong></td>
<td>A value between 0 and 1 indicating the amount of red to add to the highlight color. 0 indicates no red, 1 indicates maximum red.</td>
</tr>
<tr>
<td><strong>Green</strong></td>
<td>A value between 0 and 1 indicating the amount of green to add to the highlight color. 0 indicates no green, 1 indicates maximum green.</td>
</tr>
<tr>
<td><strong>Blue</strong></td>
<td>A value between 0 and 1 indicating the amount of blue to add to the highlight color. 0 indicates no blue, 1 indicates maximum blue.</td>
</tr>
</tbody>
</table>
**SetTextHighlightColorCMYK**

**Text, Color**

**Description**

Sets the color used to highlight text, but allows the color to be specified in the CMYK color space.

**Syntax**

**Dylib**

```c
int DPLSetTextHighlightColorCMYK(int InstanceID, double C, double M,
double Y, double K);
```

**Objective-C class**

```c
-(int)SetTextHighlightColorCMYK:(double)C :(double)M :(double)Y
:(double)K;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong></td>
<td>A value between 0 and 1 indicating the amount of cyan to add to the highlight color. 0 indicates no cyan, 1 indicates maximum cyan.</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>A value between 0 and 1 indicating the amount of magenta to add to the highlight color. 0 indicates no magenta, 1 indicates maximum magenta.</td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td>A value between 0 and 1 indicating the amount of yellow to add to the highlight color. 0 indicates no yellow, 1 indicates maximum yellow.</td>
</tr>
<tr>
<td><strong>K</strong></td>
<td>A value between 0 and 1 indicating the amount of black to add to the highlight color. 0 indicates no black, 1 indicates maximum black.</td>
</tr>
</tbody>
</table>
SetTextHighlightColorSep

**Text, Color**

**Description**

Sets the color used to highlight text. Similar to the `SetTextHighlightColor` function, but a tint of a separation color added with the `AddSeparationColor` function is used.

**Syntax**

**Dylib**

```c
int DPLSetTextHighlightColorSep(int InstanceID, wchar_t * ColorName, double Tint);
```

**Objective-C class**

```objc
- (int)SetTextHighlightColorSep:(NSString *)ColorName :(double)Tint;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ColorName</strong></td>
<td>The name of the separation color that was used with the <code>AddSeparationColor</code> function</td>
</tr>
<tr>
<td><strong>Tint</strong></td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The separation color name could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The text highlight color was set successfully</td>
</tr>
</tbody>
</table>
SetTextMode

**Text**

**Description**

Specifies the mode to draw subsequent text in. Modes 4 to 7 are used to add text to the clipping path. If one of these modes is selected and text is drawn onto the page, then subsequent items drawn onto the page will be clipped to the outline of the text.

**Syntax**

**Dylib**

```c
int DPLSetTextMode(int InstanceID, int TextMode);
```

**Objective-C class**

```c
- (int)SetTextMode:(int)TextMode;
```

**Parameters**

<table>
<thead>
<tr>
<th>TextMode</th>
<th>The text mode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Filled text (default)</td>
</tr>
<tr>
<td>1</td>
<td>Outline text</td>
</tr>
<tr>
<td>2</td>
<td>Fill then stroke text</td>
</tr>
<tr>
<td>3</td>
<td>Invisible text</td>
</tr>
<tr>
<td>4</td>
<td>Fill text and add to clipping path</td>
</tr>
<tr>
<td>5</td>
<td>Stroke text and add to clipping path</td>
</tr>
<tr>
<td>6</td>
<td>Fill then stroke text and add to clipping path</td>
</tr>
<tr>
<td>7</td>
<td>Add text to clipping path</td>
</tr>
<tr>
<td>Anything else</td>
<td>= Filled text (default)</td>
</tr>
</tbody>
</table>
**SetTextRise**

**Text**

**Description**

 Allows text to be positioned above or below the baseline. This is useful for superscript and subscript text.

**Syntax**

**Dylib**

```c
int DPLSetTextRise(int InstanceID, double Rise);
```

**Objective-C class**

```c
- (int)SetTextRise:(double)Rise;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rise</td>
<td>The amount to raise or lower subsequent text from the baseline. Positive values result in text that is higher than normal (superscript), negative values result in text that is lower than normal (subscript).</td>
</tr>
</tbody>
</table>
**SetTextScaling**

**Text**

**Description**

Sets the amount to scale text in the direction the text is written. This stretches all the characters in the font as well as the spacing between the characters.

**Syntax**

**Dylib**

```c
int DPLSetTextScaling(int InstanceID, double ScalePercentage);
```

**Objective-C class**

```objective-c
-(int)SetTextScaling:(double)ScalePercentage;
```

**Parameters**

<table>
<thead>
<tr>
<th>ScalePercentage</th>
<th>The percentage to scale the text by. Values less than 100 will result in narrower text. Values greater than 100 will result in wider text.</th>
</tr>
</thead>
</table>
SetTextShader
Vector graphics, Path definition and drawing, Color

Description

Sets the text color to the specified shader for subsequently drawn text.

Syntax

Dylib

```c
int DPLSetTextShader(int InstanceID, wchar_t * ShaderName);
```

Objective-C class

```objc
- (int)SetTextShader:(NSString *)ShaderName;
```

Parameters

| ShaderName     | The shader name that was used when the shader was created. |

Return values

| 0              | The shader could not be found |
| 1              | The text shader was setup correctly |
SetTextSize

Text

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Set the size of the text to use for any subsequently draw text. The text size is always measured in points, even if the measurement units have been changed with SetMeasurementUnits.

Syntax

Dylib

int DPLSetTextSize(int InstanceID, double TextSize);

Objective-C class

- (int)SetTextSize:(double)TextSize;

Parameters

| TextSize | The text size in points |

Return values

| 0        | A font has not been selected |
| 1        | The text size was set successfully |
SetTextSpacing

Description

Set the amount of space to add between each line for the DrawWrappedText, GetWrappedTextHeight and DrawMultiLineText functions.

Syntax

**Dylib**

```c
int DPLSetTextSpacing(int InstanceID, double Spacing);
```

**Objective-C class**

```objc
- (int)SetTextSpacing:(double)Spacing;
```

Parameters

<table>
<thead>
<tr>
<th><strong>Spacing</strong></th>
<th>The amount of space to add between each line</th>
</tr>
</thead>
</table>
SetTextUnderline

Text

This function is available in the Dylib Lite Edition of Debenu Quick PDF Library, see Appendix C.

Description

Sets the underline mode for subsequently drawn text.

Syntax

Dylib

int DPLSetTextUnderline(int InstanceID, int Underline);

Objective-C class

- (int)SetTextUnderline:(int)Underline;

Parameters

<table>
<thead>
<tr>
<th>Underline</th>
<th>The underline mode to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Single</td>
</tr>
<tr>
<td>2</td>
<td>Double</td>
</tr>
<tr>
<td>3</td>
<td>Strikeout</td>
</tr>
<tr>
<td>4</td>
<td>Over</td>
</tr>
</tbody>
</table>
SetTextUnderlineColor

Text, Color

Description
Sets the color used to draw the lines for subsequently drawn text that has an underline style.

Syntax

Dylib

```c
int DPLSetTextUnderlineColor(int InstanceID, double Red, double Green,
                             double Blue);
```

Objective-C class

```c
- (int)SetTextUnderlineColor:(double)Red :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>A value between 0 and 1 indicating the amount of red to add to the underline color. 0 indicates no red, 1 indicates maximum red.</td>
</tr>
<tr>
<td>Green</td>
<td>A value between 0 and 1 indicating the amount of green to add to the underline color. 0 indicates no green, 1 indicates maximum green.</td>
</tr>
<tr>
<td>Blue</td>
<td>A value between 0 and 1 indicating the amount of blue to add to the underline color. 0 indicates no blue, 1 indicates maximum blue.</td>
</tr>
</tbody>
</table>
**SetTextUnderlineColorCMYK**

**Text, Color**

**Description**

Sets the color used to draw the lines for subsequently drawn text that has an underline style, but allows the color to be set using the CMYK color space.

**Syntax**

**Dylib**

`int DPLSetTextUnderlineColorCMYK(int InstanceID, double C, double M, double Y, double K);`

**Objective-C class**

`- (int)SetTextUnderlineColorCMYK:(double)C :(double)M :(double)Y :(double)K;`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>A value between 0 and 1 indicating the amount of cyan to add to the underline color. 0 indicates no cyan, 1 indicates maximum cyan.</td>
</tr>
<tr>
<td>M</td>
<td>A value between 0 and 1 indicating the amount of magenta to add to the underline color. 0 indicates no magenta, 1 indicates maximum magenta.</td>
</tr>
<tr>
<td>Y</td>
<td>A value between 0 and 1 indicating the amount of yellow to add to the underline color. 0 indicates no yellow, 1 indicates maximum yellow.</td>
</tr>
<tr>
<td>K</td>
<td>A value between 0 and 1 indicating the amount of black to add to the underline color. 0 indicates no black, 1 indicates maximum black.</td>
</tr>
</tbody>
</table>
SetTextUnderlineColorSep

Text, Color

Description

Sets the color used to draw the lines for subsequently drawn text that has an underline style. Similar to the SetTextUnderlineColor function, but a tint of a separation color added with the AddSeparationColor function is used.

Syntax

Dylib

```
int DPLSetTextUnderlineColorSep(int InstanceID, wchar_t * ColorName, double Tint);
```

Objective-C class

```
- (int)SetTextUnderlineColorSep:(NSString *)ColorName :(double)Tint;
```

Parameters

<table>
<thead>
<tr>
<th>ColorName</th>
<th>The name of the separation color that was used with the AddSeparationColor function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tint</td>
<td>The amount of color to use. 0 indicates no color (white), 1 indicates maximum color.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The separation color name could not be found</td>
</tr>
<tr>
<td>1</td>
<td>The text underline color was set successfully</td>
</tr>
</tbody>
</table>
**SetTextUnderlineCustomDash**

**Text**

**Description**

Use this function to apply a dashed effect to the underlines added to subsequently drawn text.

**Syntax**

**Dylib**

```c
int DPLSetTextUnderlineCustomDash(int InstanceID, wchar_t * DashPattern,
                           double DashPhase);
```

**Objective-C class**

```c
- (int)SetTextUnderlineCustomDash:(NSString *)DashPattern
                           :(double)DashPhase;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DashPattern</strong></td>
<td>The dash pattern to use, for example &quot;10 5 0 5&quot;.</td>
</tr>
<tr>
<td><strong>DashPhase</strong></td>
<td>The dash phase. Usually set to zero.</td>
</tr>
</tbody>
</table>
SetTextUnderlineDash

Text

Description

Use this function to apply a dashed effect to the underlines added to subsequently drawn text.

Syntax

Dylib

```c
int DPLSetTextUnderlineDash(int InstanceID, double DashOn, double DashOff);
```

Objective-C class

```objective-c
- (int)SetTextUnderlineDash:(double)DashOn :(double)DashOff;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DashOn</strong></td>
<td>A factor to use for the solid parts of the dashed line. If a factor of 1 is used then the solid parts of the line will be the same width as the line. A factor of 3 would result in the solid parts of the dashed line being three times longer than the width of the line.</td>
</tr>
<tr>
<td><strong>DashOff</strong></td>
<td>A factor to use for the invisible parts of the dashed line. For example, if a factor of 2 is used then the invisible parts of the line will be twice as wide as the width of the line.</td>
</tr>
</tbody>
</table>
**SetTextUnderlineDistance**

**Text**

**Description**

Sets the distance of the underlines from the text for subsequently drawn text that has an underline style.

**Syntax**

**Dylib**

```c
int DPLSetTextUnderlineDistance(int InstanceID, double UnderlineDistance);
```

**Objective-C class**

```c
-(int)SetTextUnderlineDistance:(double)UnderlineDistance;
```

**Parameters**

<table>
<thead>
<tr>
<th>UnderlineDistance</th>
<th>The distance from the text to the underline</th>
</tr>
</thead>
</table>
SetTextUnderlineWidth

Description

Sets the width of the underlines for subsequently drawn text that has an underline style.

Syntax

**Dylib**

```c
int DPLSetTextUnderlineWidth(int InstanceID, double UnderlineWidth);
```

**Objective-C class**

```objc
- (int)SetTextUnderlineWidth:(double)UnderlineWidth;
```

Parameters

<table>
<thead>
<tr>
<th><strong>UnderlineWidth</strong></th>
<th>The width of the underline to use</th>
</tr>
</thead>
</table>
SetTextWordSpacing

**Text**

Description

Sets the amount of space to add between words for subsequently drawn text.

**Syntax**

**Dylib**

```c
int DPLSetTextWordSpacing(int InstanceID, double WordSpacing);
```

**Objective-C class**

```objective-c
- (int)SetTextWordSpacing:(double)WordSpacing;
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>WordSpacing</strong></th>
<th>The amount of extra space to add between words</th>
</tr>
</thead>
</table>
SetTransparency
Vector graphics, Text, Page layout

Description
Sets the transparency for all subsequently drawn text and graphics.

Syntax
Dylib

```c
int DPLSetTransparency(int InstanceID, int Transparency);
```

Objective-C class

```objective-c
-(int)SetTransparency:(int)Transparency;
```

Parameters

<table>
<thead>
<tr>
<th>Transparency</th>
<th>The amount of transparency to apply</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No transparency</td>
</tr>
<tr>
<td>50</td>
<td>50% transparency</td>
</tr>
<tr>
<td>100</td>
<td>Invisible</td>
</tr>
</tbody>
</table>

Return values

| 0           | The transparency specified was out of range |
| 1           | The transparency was set successfully     |
## SetViewerPreferences

### Document properties

#### Description

Sets the viewer preferences for the document.

For Option=7 to take effect, the initial page mode should be set to Full Screen using the `SetPageMode` function with the NewPageMode parameter set to 3.

#### Syntax

**Dylib**

```c
int DPLSetViewerPreferences(int InstanceID, int Option, int NewValue);
```

**Objective-C class**

```
- (int)SetViewerPreferences:(int)Option :(int)NewValue;
```

#### Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hide toolbar</td>
</tr>
<tr>
<td>2</td>
<td>Hide menubar</td>
</tr>
<tr>
<td>3</td>
<td>Hide window user interface</td>
</tr>
<tr>
<td>4</td>
<td>Resize window to first page size</td>
</tr>
<tr>
<td>5</td>
<td>Center window</td>
</tr>
<tr>
<td>6</td>
<td>Display document title</td>
</tr>
<tr>
<td>7</td>
<td>Page mode after full screen</td>
</tr>
<tr>
<td>8</td>
<td>Predominant text reading order</td>
</tr>
<tr>
<td>9</td>
<td>Display boundary for viewing</td>
</tr>
<tr>
<td>10</td>
<td>Clipping boundary for viewing</td>
</tr>
<tr>
<td>11</td>
<td>Display boundary for printing</td>
</tr>
<tr>
<td>12</td>
<td>Clipping boundary for printing</td>
</tr>
<tr>
<td>13</td>
<td>Default print dialog: scaling</td>
</tr>
<tr>
<td>14</td>
<td>Default print dialog: duplex</td>
</tr>
<tr>
<td>15</td>
<td>Default print dialog: auto paper tray</td>
</tr>
<tr>
<td>16</td>
<td>Default print dialog: number of copies</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NewValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For Option 1 to 6: 0=No, 1=Yes</td>
<td></td>
</tr>
<tr>
<td>For Option 7: 0=Normal view, 1=Show the outlines pane, 2=Show the thumbnails pane, 3=Show the layers pane</td>
<td></td>
</tr>
<tr>
<td>For Option 8: 0=Left to right, 1=Right to left</td>
<td></td>
</tr>
<tr>
<td>For Option 9 to 12: 0=MediaBox, 1=CropBox, 2=BleedBox, 3=TrimBox, 4=ArtBox</td>
<td></td>
</tr>
<tr>
<td>For Option 13: 0=None, 1=Application default</td>
<td></td>
</tr>
<tr>
<td>For Option 14: 0=Simplex, 1=Duplex flip short edge, 2=Duplex flip long edge</td>
<td></td>
</tr>
<tr>
<td>For Option 15: 0=No, 1=Yes</td>
<td></td>
</tr>
<tr>
<td>For Option 16: Any positive number</td>
<td></td>
</tr>
</tbody>
</table>

#### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The viewer preferences could not be set</td>
</tr>
<tr>
<td>1</td>
<td>The viewer preferences were set successfully</td>
</tr>
</tbody>
</table>
SetXFAFormFieldAccess

Form fields

Description

Sets the access flags of the specified XFA form field.

Syntax

Dylib

```c
int DPLSetXFAFormFieldAccess(int InstanceID, wchar_t * XFAFieldName,
                              int NewAccess);
```

Objective-C class

```objective-c
- (int)SetXFAFormFieldAccess:(NSString *)XFAFieldName :(int)NewAccess;
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFAFieldName</td>
<td>The name of the XFA field to work with</td>
</tr>
<tr>
<td>NewAccess</td>
<td>1 = Non interactive 2 = Open 3 = Protected 4 = Read only</td>
</tr>
</tbody>
</table>


SetXFAFormFieldBorderColor

Form fields, Color

Description

Sets the border color of the specified XFA form field.

Syntax

Dylib

```c
int DPLSetXFAFormFieldBorderColor(int InstanceID, wchar_t * XFAFieldName,
         double Red, double Green, double Blue);
```

Objective-C class

```c
- (int)SetXFAFormFieldBorderColor:(NSString *)XFAFieldName :(double)Red
   :(double)Green :(double)Blue;
```

Parameters

<table>
<thead>
<tr>
<th>XFAFieldName</th>
<th>The name of the XFA field to work with</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>The red component of the color, which should be a value between 0 and 1</td>
</tr>
<tr>
<td>Green</td>
<td>The green component of the color, which should be a value between 0 and 1</td>
</tr>
<tr>
<td>Blue</td>
<td>The blue component of the color, which should be a value between 0 and 1</td>
</tr>
</tbody>
</table>
SetXFAFormFieldBorderPresence

Form fields

Description

Sets the border style of the specified XFA form field.

Syntax

**Dylib**

```c
int DPLSetXFAFormFieldBorderPresence(int InstanceID, 
wchar_t * XFAFieldName, int NewPresence);
```

**Objective-C class**

```swift
- (int)SetXFAFormFieldBorderPresence:(NSString *)XFAFieldName 
:(int)NewPresence;
```

Parameters

<table>
<thead>
<tr>
<th><strong>XFAFieldName</strong></th>
<th>The name of the XFA field to work with</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NewPresence</strong></td>
<td>1 = Visible</td>
</tr>
<tr>
<td></td>
<td>2 = Invisible</td>
</tr>
<tr>
<td></td>
<td>3 = Hidden</td>
</tr>
</tbody>
</table>
SetXFAFormFieldBorderWidth

**Form fields**

**Description**

Sets the border width of the specified XFA form field.

**Syntax**

**Dylib**

```c
int DPLSetXFAFormFieldBorderWidth(int InstanceID, wchar_t * XFAFieldName, double BorderWidth);
```

**Objective-C class**

```objective-c
- (int)SetXFAFormFieldBorderWidth:(NSString *)XFAFieldName
  :(double)BorderWidth;
```

**Parameters**

<table>
<thead>
<tr>
<th>XFAFieldName</th>
<th>The name of the XFA field to work with</th>
</tr>
</thead>
<tbody>
<tr>
<td>BorderWidth</td>
<td>The desired width of the border</td>
</tr>
</tbody>
</table>
SetXFAFormFieldValue

Form fields

Description

Sets the value of the specified XFA form field.

Syntax

Dylib

```c
int DPLSetXFAFormFieldValue(int InstanceID, wchar_t * XFAFieldName,
    wchar_t * NewValue);
```

Objective-C class

```c
- (int)SetXFAFormFieldValue:(NSString *)XFAFieldName :(NSString *)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFAFieldName</td>
<td>The name of the XFA field to work with</td>
</tr>
<tr>
<td>NewValue</td>
<td>The new value for the XFA field</td>
</tr>
</tbody>
</table>
SetXFAFromString

Form fields

Description

Sets the document's XFA form data to the specified XML string.

Syntax

**Dylib**

```c
int DPLSetXFAFromString(int InstanceID, char * Source, int Options);
```

**Objective-C class**

```objc
- (int)SetXFAFromString:(NSData *)Source :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Source</th>
<th>The new XML string to store as the XFA form data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td>Reserved for future use. Should be set to 0.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The XFA form data could not be set, this usually means the document does not have an AcroForm dictionary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The XFA form data was set successfully.</td>
</tr>
</tbody>
</table>
SetupCustomPrinter
Rendering and printing

Description
Changes the properties of a custom printer created with the NewCustomPrinter function.

Syntax

**Dylib**

```c
int DPLSetupCustomPrinter(int InstanceID, wchar_t * CustomPrinterName, int Setting, int NewValue);
```

**Objective-C class**

```objective-c
-(int)SetupCustomPrinter:(NSString *)CustomPrinterName :(int)Setting :
(int)NewValue;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomPrinterName</td>
<td>A custom printer name, as returned by the NewCustomPrinter function</td>
</tr>
</tbody>
</table>
| Setting            | 1 = Paper size  
2 = Paper length  
3 = Paper width  
4 = Copies  
5 = Print quality  
6 = Color  
7 = Duplex  
8 = Collate  
9 = Default source (paper trays / bins)  
10 = Media type  
11 = Orientation |
| NewValue           | For paper size:  
1 to 68, DMPAPER_XXX (Win32 API DEVMODE data structure)  
For paper height and width:  
Size of paper in tenths of millimetres  
For copies:  
Number of copies  
For print quality:  
1 = high, 2 = medium, 3 = low, 4 = draft  
or an exact DPI, for example 600  
For color:  
1 = monochrome, 2 = color  
For duplex:  
1 = simplex, 2 = vertical duplex, 3 = horizontal duplex  
For collate:  
0 = no, 1 = yes  
For default source:  
1 to 15, DMBIN_XXX (Win32 API DEVMODE data structure)  
256 and higher for custom bins / paper trays, see the GetPrinterBins function  
For media type:  
1 = standard, 2 = transparency, 3 = glossy  
256 and higher for device-specific media  
For orientation:  
1 = portrait, 2 = landscape |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The custom printer could not be found, or the Settings or NewValue parameters were invalid</td>
</tr>
<tr>
<td>1</td>
<td>The custom printer settings were changed successfully</td>
</tr>
</tbody>
</table>
SignFile

Security and Signatures

Description

Applies a digital signature to a PDF document on disk.
The signing identity must be in PKCS#12 format containing a certificate and private key.

Syntax

Dylib

```c
int DPLSignFile(int InstanceID, wchar_t * InputFileName,
    wchar_t * OpenPassword, wchar_t * SignatureFieldName,
    wchar_t * OutputFileName, wchar_t * PFXFileName,
    wchar_t * PFXPassword, wchar_t * Reason, wchar_t * Location,
    wchar_t * ContactInfo);
```

Objective-C class

```objective-c
- (int)SignFile:(NSString *)InputFileName :(NSString *)OpenPassword 
    :(NSString *)SignatureFieldName :(NSString *)OutputFileName 
    :(NSString *)PFXFileName :(NSString *)PFXPassword 
    :(NSString *)Reason :(NSString *)Location 
    :(NSString *)ContactInfo;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFileName</td>
<td>The path and file name of the input PDF to sign.</td>
</tr>
<tr>
<td>OpenPassword</td>
<td>The optional password to open the input PDF if it is encrypted</td>
</tr>
<tr>
<td>SignatureFieldName</td>
<td>The name of the signature field to sign. If a field with this name does not</td>
</tr>
<tr>
<td></td>
<td>exist it will be created. This field cannot be blank.</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>The path and file name of the signed PDF that should be created. This</td>
</tr>
<tr>
<td></td>
<td>should be different to InputFileName.</td>
</tr>
<tr>
<td>PFXFileName</td>
<td>The path and name of the PKCS#12 certificate/private key file (.pfx file).</td>
</tr>
<tr>
<td>PFXPassword</td>
<td>The password to open the PFX file.</td>
</tr>
<tr>
<td>Reason</td>
<td>An optional string indicating the reason for signing.</td>
</tr>
<tr>
<td>Location</td>
<td>An optional string indicating the location that the signing was done.</td>
</tr>
<tr>
<td>ContactInfo</td>
<td>An optional string indicating the contact information of the signer.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The file was signed successfully</td>
</tr>
<tr>
<td>2</td>
<td>Input PDF not found</td>
</tr>
<tr>
<td>3</td>
<td>Input PDF cannot be read</td>
</tr>
<tr>
<td>4</td>
<td>Input PDF password incorrect</td>
</tr>
<tr>
<td>5</td>
<td>Certificate file not found</td>
</tr>
<tr>
<td>6</td>
<td>Certificate file is invalid</td>
</tr>
<tr>
<td>7</td>
<td>Incorrect certificate password</td>
</tr>
<tr>
<td>8</td>
<td>Unknown certificate format</td>
</tr>
<tr>
<td>9</td>
<td>No private key found in certificate file</td>
</tr>
<tr>
<td>10</td>
<td>Could not write output file</td>
</tr>
<tr>
<td>11</td>
<td>Could not apply signature</td>
</tr>
<tr>
<td>12</td>
<td>The signature field name was blank</td>
</tr>
</tbody>
</table>
**SplitPageText**

**Page manipulation**

**Description**

Splits the text and graphics on the current page into two layers. The graphics are placed into the bottom layer with the text in the top layer.

**Syntax**

**Dylib**

```c
int DPLSplitPageText(int InstanceID, int Options);
```

**Objective-C class**

```c
- (int)SplitPageText:(int)Options;
```

**Parameters**

<table>
<thead>
<tr>
<th>Options</th>
<th>This parameter is reserved for future use and should be set to zero</th>
</tr>
</thead>
</table>
StartPath

Vector graphics, Path definition and drawing

Description

Starts a multi-segment path.

Syntax

**Dylib**

```
int DPLStartPath(int InstanceID, double StartX, double StartY);
```

**Objective-C class**

```
- (int)StartPath:(double)StartX :(double)StartY;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StartX</strong></td>
<td>Horizontal co-ordinate of the point where the curve should start</td>
</tr>
<tr>
<td><strong>StartY</strong></td>
<td>Vertical co-ordinate of the point where the curve should start</td>
</tr>
</tbody>
</table>
StoreCustomDataFromFile

Document properties

Description
Saves custom data from a file into the PDF under a key name. This data can later be retrieved using `RetrieveCustomDataToString` or `RetrieveCustomDataToFile`. The storage type (string, stream or compressed stream) and location (Document Information Dictionary or Document Catalog) can be set. If the location is the Document Catalog any storage type can be used, but the key must have a special prefix assigned to you by Adobe. If the location is the Document Information Dictionary any key apart from the standard keys can be used, but only strings can be used.

Syntax

**Dylib**

```c
int DPLStoreCustomDataFromFile(int InstanceID, wchar_t * Key, wchar_t * FileName, int Location, int Options);
```

**Objective-C class**

```objective-c
- (int)StoreCustomDataFromFile:(NSString *)Key :(NSString *)FileName :(int)Location :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key</strong></td>
<td>The key to store the data under. If the location is the Document Information Dictionary then the key cannot be &quot;Author&quot;, &quot;Title&quot;, &quot;Subject&quot;, &quot;Keywords&quot;, &quot;Creator&quot; or &quot;Producer&quot;. Any other key can be used but keys should be chosen with care so they make sense to the user. If the location is the Document Catalog then the key must have a special prefix assigned to you by Adobe to avoid conflicts with other software.</td>
</tr>
<tr>
<td><strong>FileName</strong></td>
<td>The path and name of the file containing the data to store in the PDF under the specified key.</td>
</tr>
</tbody>
</table>
| **Location** | 1 = Store the data in the Document Information Dictionary  
2 = Store the data in the Document Catalog |
| **Options** | 0 = Store the data as a string (the only option available if the location is the Document Information Dictionary)  
1 = Store the data in a stream  
2 = Store the data in a compressed stream |

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The file containing the data could not be opened, or the Key parameter was invalid</td>
</tr>
<tr>
<td>1</td>
<td>The data was stored successfully</td>
</tr>
</tbody>
</table>
StoreCustomDataFromString

Document properties

Description

Saves custom data into the PDF under a key name. This data can later be retrieved using the 
RetrieveCustomDataToString, RetrieveCustomDataToVariant or 
RetrieveCustomDataToFile functions. The storage type (string, stream or compressed stream) 
and location (Document Information Dictionary or Document Catalog) can be set. If the location is 
the Document Catalog any storage type can be used, but the key must have a special prefix 
assigned to you by Adobe. If the location is the Document Information Dictionary any key apart 
from the standard keys can be used, but only strings can be used.

Syntax

Dylib

```c
int DPLStoreCustomDataFromString(int InstanceID, char * Key, 
char * NewValue, int Location, int Options);
```

Objective-C class

```objective-c
-(int)StoreCustomDataFromString:(NSData *)Key :(NSData *)NewValue 
:(int)Location :(int)Options;
```

Parameters

| Key       | The key to store the data under. If the location is the Document Information 
|           | Dictionary then the key cannot be "Author", "Title", "Subject", "Keywords", 
|           | "Creator" or "Producer". Any other key can be used but keys should be chosen 
|           | with care so they make sense to the user. If the location is the Document Catalog 
|           | then the key must have a special prefix assigned to you by Adobe to avoid 
|           | conflicts with other software. |
| NewValue  | The new value for the data |
| Location  | 1 = Store the data in the Document Information Dictionary 
|           | 2 = Store the data in the Document Catalog |
| Options   | 0 = Store the data as a string (the only option available if the location is the 
|           | Document Information Dictionary) 
|           | 1 = Store the data in a stream 
|           | 2 = Store the data in a compressed stream |

Return values

| 0         | The data could not be stored because the key name was a reserved name |
| 1         | The data was stored successfully |
Description

Returns the character length of the most recent string returned from the library by all functions that return Unicode (16-bit) strings.

The value returned is the number of 16-bit characters. So the total byte length will be twice that value.

A few functions return 8-bit strings, the `AnsiStringResultLength` function must be used to obtain the data length for those functions.

Syntax

```
Dylib

int DPLStringResultLength(int InstanceID);
```
TestTempPath

Miscellaneous functions

Description
Tests that folder used for storage of temporary files has read/write access by the process running the library.

Syntax

Dylib

```
int DPLTestTempPath(int InstanceID);
```

Objective-C class

```
-(int)TestTempPath
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The temporary path does not have read/write access</td>
</tr>
<tr>
<td>1</td>
<td>The temporary path is valid</td>
</tr>
</tbody>
</table>
TransformFile

Document manipulation, Miscellaneous functions

Description

Applies a transformation to a file allowing objects to be renumbered and reordered.
In certain cases this can result in a more compact cross reference table reducing the size of the PDF.

Syntax

Dylib

```c
int DPLTransformFile(int InstanceID, wchar_t * InputFileName,
                     wchar_t * Password, wchar_t * OutputFileName,
                     int TransformType, int Options);
```

Objective-C class

```objective-c
- (int)TransformFile:(NSString *)InputFileName :(NSString *)Password :
  :(NSString *)OutputFileName :(int)TransformType :(int)Options;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFileName</td>
<td>The path and file name of the input PDF to transform.</td>
</tr>
<tr>
<td>Password</td>
<td>The optional password to open the input PDF if it is encrypted</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>The path and file name of the signed PDF that should be created. This should be different to InputFileName.</td>
</tr>
</tbody>
</table>
| TransformType     | 1 = Renumber all objects writing them out in order
                     2 = Same as 1 but uses an xref stream                                  |
| Options           | Reserved for future use, should be set to zero.                            |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td>2</td>
<td>Input PDF not found</td>
</tr>
<tr>
<td>3</td>
<td>Input PDF cannot be read</td>
</tr>
<tr>
<td>4</td>
<td>Input PDF password incorrect</td>
</tr>
<tr>
<td>5</td>
<td>Could not write output file</td>
</tr>
</tbody>
</table>
UnlockKey

Miscellaneous functions

Description

Unlocks the library. The library must be unlocked using a registration key before it can be used.

Syntax

**Dylib**

```c
int DPLUnlockKey(int InstanceID, wchar_t * LicenseKey);
```

**Objective-C class**

```objective-c
- (int)UnlockKey:(NSString *)LicenseKey;
```

Parameters

| LicenseKey | The registration key |

Return values

| 0 | The library could not be unlocked |
| 1 | The library was unlocked successfully |
**Description**

Determine if the library has been unlocked. If the library has not been unlocked it cannot be used.

**Syntax**

**Dylib**

```c
int DPLUnlocked(int InstanceID);
```

**Objective-C class**

```objc
- (int)Unlocked
```

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The library has not been unlocked</td>
</tr>
<tr>
<td>1</td>
<td>The library has been unlocked</td>
</tr>
</tbody>
</table>
UpdateAndFlattenFormField

Description

Use this function to draw the visual appearance onto the page it is associated with. The form field will then be removed from the document and only it's appearance will remain - it will no longer be an interactive field.

If the field is flattened successfully the field index of subsequent form fields will be decreased by 1. The appearance stream of the form field will be generated before the form field is flattened. This behaviour is the same as the FlattenFormField function before version 9.11.

Syntax

Dylib

```c
int DPLUpdateAndFlattenFormField(int InstanceID, int Index);
```

Objective-C class

```c
-(int)UpdateAndFlattenFormField:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The form field could not be found or it was not possible to flatten the form field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The form field was flattened successfully</td>
</tr>
</tbody>
</table>
UpdateAppearanceStream

Form fields

Description

Generates an appearance stream for the form field. Appearance streams can be generated for text, pushbutton and choice form fields.

Syntax

**Dylib**

```c
int DPLUpdateAppearanceStream(int InstanceID, int Index);
```

**Objective-C class**

```c
- (int)UpdateAppearanceStream:(int)Index;
```

Parameters

<table>
<thead>
<tr>
<th>Index</th>
<th>The index of the form field to work with. The first form field has an index of 1.</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>The form field could not be found or an appearance stream could not be created for the specified field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The appearance stream for the specified form field was created successfully</td>
</tr>
</tbody>
</table>
**UpdateTrueTypeSubsettedFont**

**Text, Fonts**

**Description**
Updates the selected font with a new subset. This can only be done if the font was originally created using `AddTrueTypeSubsettedFont` using Options 2, 3, 4 or 5.

**Syntax**

**Dylib**

```c
int DPLUpdateTrueTypeSubsettedFont(int InstanceID, wchar_t * SubsetChars);
```

**Objective-C class**

```c
-(int)UpdateTrueTypeSubsettedFont:(NSString *)SubsetChars;
```

**Parameters**

| SubsetChars | The new list of characters to include in the font subset in addition to the existing characters. |

**Return values**

| 0 | Could not update the font subset |
| 1 | Success |
UseKerning

Text, Fonts

Description

Specifies whether to use kerning for text subsequently drawn using the DrawText and DrawRotatedText functions.

Syntax

**Dylib**

```c
int DPLUseKerning(int InstanceID, int Kern);
```

**Objective-C class**

```objc
-(int)UseKerning:(int)Kern;
```

Parameters

<table>
<thead>
<tr>
<th>Kern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not use kerning</td>
</tr>
<tr>
<td>1</td>
<td>Use kerning</td>
</tr>
<tr>
<td>2</td>
<td>Do not attempt to load kerning from TrueType fonts subsequently added to the document</td>
</tr>
</tbody>
</table>
UseUnsafeContentStreams
Content Streams and Optional Content Groups

Description

A page in a PDF document has one or more content stream parts that together contain all the PDF page description commands for the page.

This function specifies whether content stream parts that were not created by Quick PDF Library should be automatically re-used or not.

Syntax

**Dylib**

```c
int DPLUseUnsafeContentStreams(int InstanceID, int SafetyLevel);
```

**Objective-C class**

```objective-c
- (int)UseUnsafeContentStreams:(int)SafetyLevel;
```

Parameters

<table>
<thead>
<tr>
<th>SafetyLevel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Only re-use existing Quick PDF Library content stream parts (Default)</td>
</tr>
<tr>
<td>1</td>
<td>Re-use any content stream part</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The SafetyLevel parameter was out of range</td>
</tr>
<tr>
<td>1</td>
<td>The safety level was set successfully</td>
</tr>
</tbody>
</table>
Appendix A: Supported HTML tags

A limited HTML subset is supported:

- `<br>` to break onto a new line
- `<b>` or `<strong>` for bold
- `<i>` or `<em>` for italics
- `<sup>` and `<sub>` for superscript/subscript.
- `<u>` for underline
- `<u style="double">` for double underline
- `<u style="strikeout">` for strikeout (a line drawn through the text)
- `<u style="over">` for a line drawn above the text
- `<p align="left">` for left aligned paragraphs
- `<p align="center">` for centered paragraphs
- `<p align="justified">` for justified paragraphs
- `<ul>` and `<ol>` for ordered/unordered lists

```
<font
size="__"
color="__"
background="__"
roundback="yes/no"
mode="__"
outlinecolor="__"
outlinewidth="__pt"
>
<span background="__" roundback="yes/no">

The font size can be specified as a standard HTML size, or a point size such as "11.5pt", the outline width must be specified in points, for example "1.5pt".

Text and background colors can be specified in RGB using the standard HTML hexadecimal notation, for example "#3A498C". CMYK colors can be specified using eight hexadecimal values and omitting the #, for example "5C238F02".

If the roundback attribute is "yes", the background rectangles will be drawn with rounded edges.
Appendix B: Function groups

Annotations and hotspot links

- AddArcToPath
- AddFreeTextAnnotation
- AddLinkToDestination
- AddLinkToEmbeddedFile
- AddLinkToFile
- AddLinkToFileEx
- AddLinkToJavaScript
- AddLinkToLocalFile
- AddLinkToPage
- AddLinkToWeb
- AddNoteAnnotation
- AddSVGAnnotationFromFile
- AddSWFAnnotationFromFile
- AddStampAnnotation
- AddTextMarkupAnnotation
- AddU3DAnnotationFromFile
- AnnotationCount
- AttachAnnotToForm
- CheckPageAnnots
- CloneOutlineAction
- DeleteAnnotation
- DrawPostScriptXObject
- FlattenAnnot
- GetActionDest
- GetActionType
- GetActionURL
- GetAnnotActionID
- GetAnnotDbProperty
- GetAnnotDest
- GetAnnotIntProperty
- GetAnnotQuadCount
- GetAnnotSoundToFile
- GetAnnotSoundToString
- GetAnnotStrProperty
- GetBaseURL
- GetDestName
- GetDestPage
- GetDestType
- GetDestValue
- GetFormFieldActionID
- GetNamedDestination
- GetOutlineActionID
- GetTabOrderMode
- IsAnnotFormField
- NewDestination
- NewNamedDestination
- SetActionURL
- SetAnnotBorderColor
- SetAnnotBorderStyle
- SetAnnotContents
- SetAnnotDbProperty
- SetAnnotIntProperty
- SetAnnotQuadPoints
- SetAnnotRect
- SetAnnotStrProperty
- SetBaseURL
- SetDestProperties
- SetDestValue
- SetMarkupAnnotStyle
- SetOutlineNamedDestination
- SetTabOrderMode

Barcodes

- DrawBarcode
- DrawDataMatrixSymbol
- DrawIntelligentMailBarcode
- DrawPDF417Symbol
- DrawPDF417SymbolEx
- DrawQRCode

Color

- AddSeparationColor
- DAGetTextBlockColor
- DAGetTextBlockColorType
- GetFormFieldBackgroundColor
- GetFormFieldBackgroundColorType
- GetFormFieldBorderColor
- GetFormFieldBorderColorType
- GetFormFieldColor
- GetOutlineColor
- GetPageColorSpaces
- GetPageJavaScript
- GetTextBlockColor
- GetTextBlockColorType
- ImageFillColor
- NewRGBAxialShader
- NewTilingPatternFromCapturedPage
- SetAnnotBorderColor
- SetFillColor
- SetFillColorCMYK
- SetFillColorSep
- SetFillShader
- SetFillTilingPattern
- SetFormFieldBackgroundColor
- SetFormFieldBackgroundColorCMYK
- SetFormFieldBackgroundColorGray
- SetFormFieldBackgroundColorSep
- SetFormFieldBorderColor
- SetFormFieldBorderColorCMYK
- SetFormFieldBorderColorGray
- SetFormFieldBorderColorSep
- SetFormFieldColor
- SetFormFieldColorCMYK
- SetFormFieldColorSep
- SetImageMaskCMYK
- SetLineColor
- SetLineColorCMYK
- SetLineColorSep
- SetLineShader
- SetMarkupAnnotStyle
Color continued...

- SetOutlineColor
- SetPNGTransparencyColor
- SetTableBorderColor
- SetTableBorderColorCMYK
- SetTableCellBackgroundColor
- SetTableCellBackgroundColorCMYK
- SetTableCellBorderColor
- SetTableCellBorderColorCMYK
- SetTableCellTextColor
- SetTableCellTextColorCMYK
- SetTextColor
- SetTextColorCMYK
- SetTextColorSep
- SetTextHighlightColor
- SetTextHighlightColorCMYK
- SetTextShader
- SetTextUnderlineColor
- SetTextUnderlineColorCMYK
- SetTextUnderlineColorSep
- SetXFAFormFieldBorderColor

Content Streams and Optional Content Groups continued...

- SetFormFieldOptional
- SetImageOptional
- SetOptionalContentView
- SetOptionalContentViewLocked
- SetOptionalContentViewState
- SetOptionalContentViewGroup
- SetOptionalContentViewGroupPrintable
- SetOptionalContentViewGroupVisible
- UseUnsafeContentStreams

Direct access functionality

- DAAppendFile
- DACapturePage
- DACapturePageEx
- DACloseFile
- DADrawCapturedPage
- DADrawRotatedCapturedPage
- DAEmbedFileStreams
- DAExtractPageText
- DAExtractPageTextBlocks
- DAFindPage
- DAGetAnnotationCount
- DAGetFormFieldCount
- DAGetFormFieldTitle
- DAGetFormFieldValue
- DAGetImageDataToString
- DAGetImageDblProperty
- DAGetImageIntProperty
- DAGetImageListCount
- DAGetInformation
- DAGetObjectCount
- DAGetObjectToString
- DAGetPageBox
- DAGetPageContentToString
- DAGetPageCount
- DAGetPageHeight
- DAGetPageImageList
- DAGetPageWidth
- DAGetTextBlockBound
- DAGetTextBlockCharWidth
- DAGetTextBlockColor
- DAGetTextBlockColorType
- DAGetTextBlockCount
- DAGetTextBlockFontName
- DAGetTextBlockFontSize
- DAGetTextBlockText
- DAHasPageBox
- DAHidePage
- DAMovePage
- DAOpenFile
- DAOpenFileReadOnly
- DAPageRotation
- DAReleaseImageList
### Direct access functionality
- DARemoveUsageRights
- DARenderPageToFile
- DARenderPageToString
- DARotatePage
- DASaveAsFile
- DASaveImageDataToFile
- DASetInformation
- DASetPageBox
- DASetPageSize
- DASetTextExtractionArea
- DASetTextExtractionOptions
- DASetTextExtractionScaling
- DASetTextExtractionWordGap
- DAShiftedHeader

### Document management
- AppendToFile
- BalancePageTree
- DAAppendFile
- DAOpenFile
- DAOpenFileReadOnly
- DASaveAsFile
- DAShiftedHeader
- DecryptFile
- DocumentCount
- GetDocumentFileName
- GetDocumentID
- GetDocumentRepaired
- InsertPages
- LoadFromFile
- LoadFromString
- MovePage
- NewDestination
- NewDocument
- RemoveDocument
- SaveToFile
- SaveToString
- SelectDocument
- SelectedDocument
- SetFindImagesMode

### Document manipulation
- CheckFileCompliance
- DAEmbedFileStreams
- DARemoveUsageRights
- ExtractFilePages
- ExtractFilePagesEx
- ExtractPageRanges
- MergeDocument
- MergeFileList
- MergeFileListFast
- MergeFiles
- RemoveUsageRights
- ReplaceFonts
- TransformFile

### Document properties
- AddEmbeddedFile
- AddFileAttachment
- AddGlobalJavaScript
- AddLinkToEmbeddedFile
- AnalyseFile
- CompressContent
- CompressFonts
- CompressImages
- DAGetInformation
- DAGetPageCount
- DASaveAsFile
- DASaveImageDataToFile
- DASetInformation
- DASetPageBox
- DASetPageSize
- DASetTextExtractionArea
- DASetTextExtractionOptions
- DASetTextExtractionScaling
- DASetTextExtractionWordGap
- DAShiftedHeader

- Decrypt
- DeleteAnalysis
- DocJavaScriptAction
- EmbedFile
- EmbeddedFileCount
- EncryptionAlgorithm
- EncryptionStatus
- EncryptionStrength
- FindFonts
- FindImages
- GetAnalysisInfo
- GetBaseURL
- GetCatalogInformation
- GetCustomInformation
- GetCustomKeys
- GetDocJavaScript
- GetDocumentFileSize
- GetDocumentIdentifier
- GetDocumentMetadata
- GetDocumentRepaired
- GetDocumentResourceList
- GetEmbeddedFileContentToFile
- GetEmbeddedFileContentToString
- GetEmbeddedFileCount
- GetEmbeddedFileID
- GetEmbeddedFileIntProperty
- GetEncryptionFingerprint
- GetEncryptionFingerprint
- GetFileMetadata
- GetGlobalJavaScript
- GetInformation
- GetMaxObjectNumber
- GetNamedDestination
- GetOpenActionDestination
- GetOpenActionJavaScript
- GetPageLayout
- GetPageMode
- GetViewerPreferences
- GlobalJavaScriptCount
- GlobalJavaScriptPackageName
- HasFontResources
- ImageCount
Document properties continued...

- Linearized
- NewPostScriptXObject
- PageCount
- RemoveCustomInformation
- RemoveEmbeddedFile
- RemoveGlobalJavaScript
- RemoveOpenAction
- RemoveUsageRights
- RemoveXFAEntries
- RetrieveCustomDataToFile
- RetrieveCustomDataToString
- SecurityInfo
- SetBaseURL
- SetCatalogInformation
- SetCustomInformation
- SetDocumentMetadata
- SetEmbeddedFileStrProperty
- SetHeaderCommentsFromString
- SetInformation
- SetJavaScriptMode
- SetOpenActionDestination
- SetOpenActionDestinationFull
- SetOpenActionJavaScript
- SetOpenActionMenu
- SetPDFAMode
- SetPageLayout
- SetPageMode
- SetViewerPreferences
- StoreCustomDataFromFile
- StoreCustomDataFromString

Extraction continued...

- ExtractPages
- GetPageText
- GetTextBlockBound
- GetTextBlockCharWidth
- GetTextBlockColor
- GetTextBlockColorType
- GetTextBlockCount
- GetTextBlockFontName
- GetTextBlockFontSize
- GetTextBlockText
- ReleaseTextBlocks
- SetTextExtractionArea
- SetTextExtractionOptions
- SetTextExtractionScaling
- SetTextExtractionWordGap

Fonts

- AddCJKFont
- AddFormFont
- AddOpenTypeFontFromFile
- AddStandardFont
- AddTrueTypeFontFromFile
- AddType1Font
- CharWidth
- CompressFonts
- DAGetTextBlockCharWidth
- DAGetTextBlockFontName
- DAGetTextBlockFontSize
- FindFonts
- FontCount
- FontFamily
- FontHasKerning
- FontName
- FontReference
- FontSize
- FontType
- GetFontEncoding
- GetFontFlags
- GetFontID
- GetFontObjectNumber
- GetFormFontCount
- GetFormFontName
- GetInstalledFontsByCharset
- GetInstalledFontsByCodePage
- GetKerning
- GetTextAscent
- GetTextBlockBound
- GetTextBlockCharWidth
- GetTextBlockFontName
- GetTextBlockFontSize
- GetTextBound
- GetTextDescent
- GetTextHeight
- GetTextSize
- GetTextWidth
Fonts continued...

- GetUnicodeCharactersFromEncoding
- HasFontResources
- NoEmbedFontListAdd
- NoEmbedFontListCount
- NoEmbedFontListGet
- NoEmbedFontListRemoveAll
- NoEmbedFontListRemoveIndex
- NoEmbedFontListRemoveName
- ReplaceFonts
- SaveFontToFile
- SelectFont
- SelectedFont
- SetFontEncoding
- SetFontFlags
- SetFormFieldStandardFont
- SetKerning
- UpdateTrueTypeSubsettedFont
- UseKerning

Form fields continued...

- GetFormFieldJavaScript
- GetFormFieldKidCount
- GetFormFieldKidTempIndex
- GetFormFieldMaxLen
- GetFormFieldNoExport
- GetFormFieldPage
- GetFormFieldPrintable
- GetFormFieldReadOnly
- GetFormFieldRequired
- GetFormFieldRichTextString
- GetFormFieldRotation
- GetFormFieldSubCount
- GetFormFieldSubDisplayName
- GetFormFieldSubName
- GetFormFieldTabOrder
- GetFormFieldTabOrderEx
- GetFormFieldTextFlags
- GetFormFieldTextSize
- GetFormFieldTitle
- GetFormFieldType
- GetFormFieldValue
- GetFormFieldValueByTitle
- GetFormFieldVisible
- GetFormFieldWebLink
- GetFormFontCount
- GetFormFontName
- GetTabOrderMode
- GetXFAFormFieldCount
- GetXFAFormFieldNames
- GetXFAFormFieldValue
- GetXFAToString
- IsAnnotFormField
- NewChildFormField
- NewFormField
- RemoveAppearanceStream
- RemoveFormFieldBackgroundColor
- RemoveFormFieldBorderColor
- RemoveFormFieldChoiceSub
- RemoveXFAEntries
- SetCharWidth
- SetFormFieldAlignment
- SetFormFieldAnnotFlags
- SetFormFieldBackgroundColor
- SetFormFieldBackgroundColorCMYK
- SetFormFieldBackgroundColorGray
- SetFormFieldBackgroundColorSep
- SetFormFieldBorderColor
- SetFormFieldBorderColorCMYK
- SetFormFieldBorderColor
- SetFormFieldBorderColorCMYK
- SetFormFieldBorderColorGray
- SetFormFieldBorderColorSep
- SetFormFieldBorderStyle
- SetFormFieldCalcOrder
- SetFormFieldCaption
- SetFormFieldCheckStyle
- SetFormFieldChildTitle
- SetFormFieldChoiceType
- SetFormFieldColor
- SetFormFieldComb
- SetFormFieldDefaultValue
- SetFormFieldDescription
- SetFormFieldFlags
- SetFormFieldFontSize
- SetFormFieldPreferredHeight
- SetFormFieldPreferredWidth
- SetFormFieldRichTextString
- SetFormFieldRotation
- SetFormFieldSrc
- SetFormFieldSubName
- SetFormFieldTabOrder
- SetFormFieldTabOrderEx
- SetFormFieldTextFlags
- SetFormFieldTextSize
- SetFormFieldTitle
- SetFormFieldType
- SetFormFieldValue
- SetFormFieldValueByTitle
- SetFormFieldVisible
- SetFormFieldWebLink
- SetFormFontCount
- SetFormFontName
- SetTabOrderMode
- SetXFAFormFieldCount
- SetXFAFormFieldNames
- SetXFAFormFieldValue
- SetXFAToString
- IsAnnotFormField
- NewChildFormField
- NewFormField
Form fields  continued...

SetFormFieldChildTitle
SetFormFieldChoiceSub
SetFormFieldChoiceType
SetFormFieldColor
SetFormFieldColorCMYK
SetFormFieldColorSep
SetFormFieldComb
SetFormFieldDefaultValue
SetFormFieldDescription
SetFormFieldFont
SetFormFieldHighlightMode
SetFormFieldIcon
SetFormFieldIconStyle
SetFormFieldMaxLen
SetFormFieldNoExport
SetFormFieldOptional
SetFormFieldPage
SetFormFieldPrintable
SetFormFieldReadOnly
SetFormFieldRequired
SetFormFieldResetAction
SetFormFieldRichTextString
SetFormFieldRotation
SetFormFieldSignatureImage
SetFormFieldStandardFont
SetFormFieldSubmitAction
SetFormFieldSubmitActionEx
SetFormFieldTabOrder
SetFormFieldTextFlags
SetFormFieldTextSize
SetFormFieldTitle
SetFormFieldValue
SetFormFieldValueByTitle
SetFormFieldVisible
SetNeedAppearances
SetTabOrderMode
SetXFAFormFieldAccess
SetXFAFormFieldBorderColor
SetXFAFormFieldBorderPresence
SetXFAFormFieldBorderWidth
SetXFAFormFieldValue
SetXFAFromString
UpdateAndFlattenFormField
UpdateAppearanceStream

HTML text  continued...

SetHTMLBoldItalicFont
SetHTMLItalicFont
SetHTMLNormalFont

Image handling

AddImageFromFile
AddImageFromFileOffset
AddImageFromString
AddSVGAnnotationFromFile
AddSWFAnnotationFromFile
AddU3DAnnotationFromFile
ClearImage
CompressImages
DAGetImageDataToString
DAGetImageDblProperty
DAGetImageIntProperty
DAGetImageListCount
DAGetPageImageList
DAReleaseImageList
DASaveImageDataToFile
DrawImage
DrawImageMatrix
DrawRotatedImage
DrawScaledImage
FindImages
FitImage
GetImageID
GetImageListCount
GetImageListItemDataToString
GetImageListItemDblProperty
GetImageListItemIntProperty
GetImagePageCount
GetImagePageCountFromString
GetPageImageList
ImageCount
ImageFillColor
ImageHeight
ImageHorizontalResolution
ImageResolutionUnits
ImageType
ImageVerticalResolution
ImageWidth
ImportEMFFromFile
ReleaseEMFFromFile
ReleaseImage
ReleaseImageList
RenderAsMultipageTIFFToFile
ReplaceImage
ReverseImage
SaveImageListItemDataToFile
SaveImageToFile
SaveImageToString
SelectImage
SelectedImage
SetBlendMode
SetFindImagesMode

HTML text

DrawHTMLText
DrawHTMLTextBox
DrawHTMLTextBoxMatrix
DrawHTMLTextMatrix
GetHTMLTextHeight
GetHTMLTextLineCount
GetHTMLTextWidth
SetHTMLBoldFont
Image handling  continued...

SetFormFieldSignatureImage
SetImageAsMask
SetImageMask
SetImageMaskCMYK
SetImageMaskFromImage
SetImageOptional
SetImageResolution
SetPNGTransparencyColor

JavaScript

AddGlobalJavaScript
AddLinkToJavaScript
DocJavaScriptAction
FormFieldJavaScriptAction
GetDocJavaScript
GetGlobalJavaScript
GetOpenActionJavaScript
GetOutlineJavaScript
GetPageJavaScript
GlobalJavaScriptCount
GlobalJavaScriptPackageName
PageJavaScriptAction
RemoveGlobalJavaScript
SetJavaScriptMode
SetOpenActionJavaScript
SetOutlineJavaScript

Measurement and coordinate units  continued...

MultiplyScale
SetCSDictEPSG
SetCSDictType
SetCSDictWKT
SetMeasureDictBoundsCount
SetMeasureDictBoundsItem
SetMeasureDictCoordinateSystem
SetMeasureDictGPTSCount
SetMeasureDictGPTSItem
SetMeasureDictLPTSCount
SetMeasureDictLPTSItem
SetMeasurementUnits
SetOrigin
SetPrecision
SetScale

Miscellaneous functions

AddToBuffer
AddToFileList
AnsiStringResultLength
CheckObjects
CheckPageAnnots
ClearFileList
CreateBuffer
CreateLibrary
CreateNewObject
LibraryVersion
LicenseInfo
NoEmbedFontListAdd
NoEmbedFontListCount
NoEmbedFontListGet
NoEmbedFontListRemoveAll
NoEmbedFontListRemoveIndex
NoEmbedFontListRemoveName
ReleaseBuffer
ReleaseLibrary
ReleaseStringList

Measurement and coordinate units

AddLGIDictToPage
DeletePageLGIDict
GetCSDictEPSG
GetCSDictType
GetCSDictWKT
GetImageMeasureDict
GetImagePtDataDict
GetMeasureDictBoundsCount
GetMeasureDictBoundsItem
GetMeasureDictCoordinateSystem
GetMeasureDictDCSDict
GetMeasureDictGCDict
GetMeasureDictGPTSCount
GetMeasureDictGPTSItem
GetMeasureDictLPTSCount
GetMeasureDictLPTSItem
GetMeasureDictPDU
GetOrigin
GetPageLGIDictContent
GetPageLGIDictCount
GetPageViewportPortCount
GetPageViewportPortID
GetViewportBBox
GetViewportMeasureDict
GetViewportPort
GetViewportPortDataDict

Measurement and coordinate units

AddLGIDictToPage
DeletePageLGIDict
GetCSDictEPSG
GetCSDictType
GetCSDictWKT
GetImageMeasureDict
GetImagePtDataDict
GetMeasureDictBoundsCount
GetMeasureDictBoundsItem
GetMeasureDictCoordinateSystem
GetMeasureDictDCSDict
GetMeasureDictGCDict
GetMeasureDictGPTSCount
GetMeasureDictGPTSItem
GetMeasureDictLPTSCount
GetMeasureDictLPTSItem
GetMeasureDictPDU
GetOrigin
GetPageLGIDictContent
GetPageLGIDictCount
GetPageViewportPortCount
GetPageViewportPortID
GetViewportBBox
GetViewportMeasureDict
GetViewportPort
GetViewportPortDataDict
### Miscellaneous functions continued...

- SetCompatibility
- SetObjectFromString
- SetTempFile
- SetTempPath
- StringResultLength
- TestTempPath
- TransformFile
- UnlockKey
- Unlocked

### Outlines

- CloneOutlineAction
- CloseOutline
- CompareOutlines
- GetFirstChildOutline
- GetFirstOutline
- GetNextOutline
- GetOutlineActionID
- GetOutlineColor
- GetOutlineDest
- GetOutlineID
- GetOutlineJavaScript
- GetOutlineObjectNumber
- GetOutlineOpenFile
- GetOutlinePage
- GetOutlineStyle
- GetOutlineWebLink
- GetParentOutline
- GetPrevOutline
- MoveOutlineAfter
- MoveOutlineBefore
- NewOutline
- NewStaticOutline
- OpenOutline
- OutlineCount
- OutlineTitle
- RemoveOutline
- SetOutlineColor
- SetOutlineDestination
- SetOutlineDestinationFull
- SetOutlineDestinationZoom
- SetOutlineJavaScript
- SetOutlineNamedDestination
- SetOutlineOpenFile
- SetOutlineRemoteDestination
- SetOutlineStyle
- SetOutlineTitle
- SetOutlineWebLink

### Page layout continued...

- AppendTableColumns
- AppendTableRows
- AppendText
- ApplyStyle
- BeginPageUpdate
- CreateTable
- DADrawCapturedPage
- DADrawRotatedCapturedPage
- DrawCapturedPage
- DrawCapturedPageMatrix
- DrawHTMLText
- DrawHTMLTextBox
- DrawHTMLTextBoxMatrix
- DrawImage
- DrawImageMatrix
- DrawMultiLineText
- DrawPostScriptXObject
- DrawRotatedCapturedPage
- DrawRotatedImage
- DrawRotatedMultiLineText
- DrawRotatedText
- DrawRotatedTextBox
- DrawRotatedTextBoxMatrix
- DrawRotatedBox
- DrawScaledImage
- DrawSpacedText
- DrawTableRows
- DrawText
- DrawTextArc
- DrawTextBox
- DrawTextBoxMatrix
- DrawWrappedText
- EndPageUpdate
- FitImage
- FitRotatedTextBox
- FitTextBox
- FlattenAnnot
- FlattenFormField
- GetBarcodeWidth
- GetTableCellDblProperty
- GetTableCellIntProperty
- GetTableCellStrProperty
- GetTableColumnCount
- GetTableLastDrawnRow
- GetTableRowCount
- GetTextAscent
- GetTextBound
- GetTextDescent
- GetTextHeight
- GetTextSize
- GetTextWidth
- GetWrappedText
- GetWrappedTextHeight
- GetWrappedTextLineCount
- ImageFillColor
- InsertTableColumns
Page layout  continued...

- InsertTableRows
- LoadState
- MergeTableCells
- ReplaceImage
- SaveState
- SelectImage
- SelectPage
- SelectedImage
- SelectedPage
- SetCapturedPageOptional
- SetCapturedPageTransparencyGroup
- SetImageAsMask
- SetImageMask
- SetImageMaskCMYK
- SetImageMaskFromImage
- SetOverprint
- SetPageContentFromString
- SetPageDimensions
- SetPageSize
- SetPageTransparencyGroup
- SetTableBorderColor
- SetTableBorderColorCMYK
- SetTableBorderWidth
- SetTableCellAlignment
- SetTableCellBackgroundColor
- SetTableCellBackgroundColorCMYK
- SetTableCellBorderColor
- SetTableCellBorderColorCMYK
- SetTableCellBorderWidth
- SetTableCellContent
- SetTableCellPadding
- SetTableCellTextColor
- SetTableCellTextColorCMYK
- SetTableCellTextSize
- SetTableColumnWidth
- SetTableRowHeight
- SetTableThinBorders
- SetTableThinBordersCMYK
- SetTransparency
- UpdateAndFlattenFormField

Page manipulation  continued...

- DeletePages
- DrawBox
- DrawRotatedBox
- DrawRotatedCapturedPage
- ExtractFilePageContentToString
- ExtractFilePages
- ExtractFilePagesEx
- ExtractPageRanges
- ExtractPages
- GetContentStreamToString
- GetPageContentToString
- GetPageText
- HidePage
- InsertPages
- MovePage
- NewPage
- NewPages
- NormalizePage
- ReplaceTag
- RotatePage
- SelectPage
- SelectedPage
- SetContentStreamFromString
- SetPageThumbnail
- SetPageText
- SplitPageText

Page properties

- AddLGIDictToPage
- AddLinkToDestination
- AddLinkToPage
- AddPageLabels
- BalancePageTree
- ClearPageLabels
- CompressPage
- DAGetPageBox
- DAGetPageContentToString
- DAGetPageHeight
- DAGetPageImageList
- DAGetPageWidth
- DAHasPageBox
- DAPageRotation
- DARotatePage
- DARotatePage
- DASetPageBox
- DASetPageSize
- DeletePageLGIDict
- ExtractFilePageText
- ExtractFilePageTextBlocks
- GetContentStreamToString
- GetPageBox
- GetPageColorSpaces
- GetPageContentToString
- GetPageImageList
- GetPageJavaScript
Vector graphics continued...

DrawPath
DrawPathEvenOdd
DrawQRCode
DrawRotatedBox
DrawRoundedBox
GetBarcodeWidth
ImportEMFFromFile
LoadState
MovePath
NewRGBAxialShader
NewTilingPatternFromCapturedPage
NoEmbedFontListAdd
NoEmbedFontListCount
NoEmbedFontListGet
NoEmbedFontListRemoveAll
NoEmbedFontListRemoveIndex
NoEmbedFontListRemoveName
SaveState
SetBlendMode
SetClippingPath
SetClippingPathEvenOdd
SetCustomLineDash
SetFillColor
SetFillColorCMYK
SetFillColorSep
SetFillShader
SetFillTilingPattern
SetLineCap
SetLineColor
SetLineColorCMYK
SetLineColorSep
SetLineDash
SetLineDashEx
SetLineJoin
SetLineShader
SetLineWidth
SetOverprint
SetPageTransparencyGroup
SetTextShader
SetTransparency
StartPath
Appendix C: Functions available in the Dylib Lite Edition

AddImageFromFile
AddLinkToWeb
AddStandardFont
DocumentCount
DrawQRCODE
DrawImage
DrawText
DrawTextBox
FindImages
GetInformation
GetPageBox
HasFontResources
ImageCount
ImageHeight
ImageWidth
LastError
Linearized
LoadFromFile
MergeDocument
NewDocument
NewPage
NormalizePage
PageCount
PageHeight
PageRotation
PageWidth
RemoveDocument
RotatePage
SaveToFile
SecurityInfo
SelectDocument
SelectedDocument
SelectFont
SelectImage
SelectPage
SetBaseURL
SetInformation
SetMeasurementUnits
SetOrigin
SetPageBox
SetPageDimensions
SetPageLayout
SetPageMode
SetPageSize
SetTextAlign
SetTextColor
SetTextSize
SetTextUnderline