```
for (n = 1; n < iNumOutlines; n++)
{
  // Select the main document
  QP.SelectDocument(iMainDocID);

  // Select the current bookmark
  iOutlineID = QP.GetOutlineID(n);

  // Get the bookmark title
  DocName = DocName + QP.OutlineTitle(iOutlineID) + "\n";
}

// Display a message box with all the bookmarks listed

MsgBox(DocName);
```

PDF Fonts

Quick PDF Library provides support for a wide range of different fonts, enabling you to embed and subset fonts. Support for Unicode characters is also provided.

Add a TrueType font

Use the AddTrueTypeFont function to embed a TrueType font in a PDF.

```
/* Embed a TrueType font within a PDF */

// Use the AddTrueTypeFont function to add a font to
// the default blank document and get the return
// value which is the font ID.

fontID1 = QP.AddTrueTypeFont("Arial Rounded MT Bold", 1);

// Select the font using its font ID

QP.SelectFont(fontID1);

// Draw some text onto the document to see if
// everything is working OK.

QP.DrawText(100, 700, "Arial Rounded MT Bold");

// Repeat exercise to see what a couple of other
// fonts will look like as well.

fontID2 = QP.AddTrueTypeFont("Times New Roman", 1);
QP.SelectFont(fontID2);
QP.DrawText(100, 650, "Times New Roman");

fontID3 = QP.AddTrueTypeFont("Century Gothic", 1);
QP.SelectFont(fontID3);
QP.DrawText(100, 600, "Century Gothic");

// Save the new document to the output folder.

QP.SaveToFile("embedded_fonts.pdf");
```

Add a standard font

Use the AddStandardFont function to embed a standard font to a PDF.

```
/* Add a Windows standard font to a PDF */

// Use the AddStandardFont function to add a font to
// the default blank document and get the return
// value which is the font ID.

fontID1 = QP.AddStandardFont(0);

// Select the font using its font ID

QP.SelectFont(fontID1);

// Draw some text onto the document to see if
// everything is working OK.
```

```
QP.DrawText(100, 700, "Courier");

// Repeat exercise to see what a couple of other
// fonts will look like as well.

fontID2 = QP.AddStandardFont(1);
QP.SelectFont(fontID2);
QP.DrawText(100, 650, "CourierBold");

fontID3 = QP.AddStandardFont(2);
QP.SelectFont(fontID3);
QP.DrawText(100, 600, "CourierBoldOblique");

fontID4 = QP.AddStandardFont(3);
QP.SelectFont(fontID4);
QP.DrawText(100, 550, "Helvetica");

fontID5 = QP.AddStandardFont(4);
QP.SelectFont(fontID5);
QP.DrawText(100, 500, "HelveticaBold");

// Save the new document to the output folder.

QP.SaveToFile("embedded_standard_fonts.pdf");
```

Add a subsetted font

Use the AddSubsettedFont function to embed a subset of a font in a PDF. This means that only the font information for the specified characters will be embedded, reducing the size of the document.

```
/* Add subsetted text to a PDF */

// Our string of Unicode text goes here

drawstr = "Hello World";

// Add a subset font for the text string

QP.AddSubSettedFont("Verdana", 1, drawstr);

// Remap the string to ensure that the correct character
// codes are used.

substr = QP.GetSubsetstring(drawstr);

// Draw the Unicode text onto the page

QP.DrawText(100, 600, substr);

// Save the new file to disk

QP.SaveToFile("subsetted_text.pdf");
```

Add a subsetted font with Unicode text

Unicode text can be added to PDFs using the AddSubsettedFont function.

```
/* Add subsetted Unicode text to a PDF */
```

```
// Our string of Unicode text goes here

drawstr = "你好";

// Add a subset font for the text string

QP.AddSubSettedFont("Verdana", 7, drawstr);

// Remap the string to ensure that the correct character
// codes are used.

substr = QP.GetSubsetstring(drawstr);

// Draw the Unicode text onto the page

QP.DrawText(100, 600, substr);

// Save the new file to disk

QP.SaveToFile("unicode_text.pdf");
```

Add a Type 1 font

Use the AddType1Font function to embed PostScript Type 1 fonts in a PDF.

```
/* Add a Type 1 font to a PDF */

// Load and add the Type 1 font using the
// AddType1Font function. Read function
// description for full requirements.

FontID = QP.AddType1Font("Allandale.PFM");

// Select the font that we've just added

QP.SelectFont(FontID);

// Set page origin to top left

QP.SetOrigin(1);

// Draw some sample text

QP.DrawText(100, 100, "Hello World");

// Save the new file to disk

QP.SaveToFile("embedded_type1_font.pdf");
```

Add a CJK font

Use the AddCJKFont function to add a Chinese, Japanese or Korean font to a PDF. Read the function description for full information.

```
/* Add a CJK font to a PDF */

// Add a CJK font and select it

FontID = QP.AddCJKFont(1);
QP.SelectFont(FontID);
```

```
// Specify text to be drawn

InputText = "中文文本"; // Chinese text

// Convert string of text to Unicode

UnicodeInputText = QP.ToPDFUnicode(InputText);

// Draw the string of text onto the PDF

QP.DrawText(100, 600, UnicodeInputText);

// Save the new file to disk

QP.SaveToFile("cjk_font.pdf");
```

Check PDF for font data

The HasFontResources function can be used to check a PDF for any font resources. If the PDF does not have any font resources then it can be assumed to be an image only PDF.

```
/* Check PDF for font resources */

// Load the PDF to test

QP.LoadFromFile("example.pdf");

// Check PDF for font resources. If this function returns 0
// then no font resources exist, but if it returns a non-zero value
// then font resources do exist.

HasFontResourcesResult = QP.HasFontResources();

if (HasFontResourcesResult == 0)
{
  MsgBox("No font resources found. Image only PDF.");
}
else
{
  MsgBox(HasFontResourcesResult + " font resource(s) found.");
}
```

Save embedded TrueType font data to file

Use the SaveFontToFile function to save embedded TrueType font data to file.

```
/* Save TrueType font data to file*/

// Load the PDF to extract TrueType font data from

QP.LoadFromFile("bookmarks.pdf");

// Locate all fonts in the PDF

FindFontsResult = QP.FindFonts();

// Loop through each font and save
// to file if the font is TrueType and embedded

for (i = 1; i < FindFontsResult; i++)
{
```

```
        FontID = QP.GetFontID(i);
        QP.SelectFont(FontID);
        if (QP.FontType() == 4)
        {
            QP.SaveFontToFile(QP.FontName() + ".ttf");
        }
    }
```

Some additional details:

1. First you need to load the PDF using the LoadFromFile function or one of the other LoadFrom* functions.
2. Then you need to count all of the fonts in the document using the FindFonts function. This function will return the total number of fonts in the document. Use this as an index to loop through each font in the document -- starting from 1 to the total number of fonts.
3. While looping through the index of the fonts, retrieve the fond ID and use this ID to select the font using the SelectFont function.
4. Only embedded TrueType fonts are supported by the SaveFontToFile function, so you need to use the FontType function to filter our all fonts that are not embedded TrueType fonts.
5. Finally, now you can save the embedded TrueType fonts to disk using the SaveFontToFile function as they are discovered in the loop

PDF Text

Quick PDF Library lets you draw text onto PDF files in a variety of different ways.

Draw text

Quick PDF Library lets you easily add simple text strings to PDF.

```
/* Draw a variety of different text on a new document */

// Set the origin for the drawing co-ordinates. In this case
// we'll draw the co-ordinates from the top left corner of the page.

QP.SetOrigin(1);

// Draw normal text

QP.DrawText(25, 25, "This text was drawn using the DrawText function.");

// Save the new file to the output folder

QP.SaveToFile("simple_text.pdf");
```

Draw styled text

Quick PDF Library gives you powerful control over styling your text with different fonts, text size and color and also the positing of the text on the page.

```
/* Draw a variety of different text on a new document */

// Set the origin for the drawing co-ordinates. In this case
// we'll draw the co-ordinates from the top left corner of the page.

QP.SetOrigin(1);

// Draw normal text

QP.DrawText(25, 25, "This text was drawn using the DrawText function.");

// Draw an arc of text

QP.DrawTextArc(150, 150, 100, 280, "This text was drawn using the DrawTextArc function.",
0);

// Draw text wrapped to a specified width

QP.DrawWrappedText(400, 50, 200, "This text was drawn using the DrawWrappedText function.
As you can see, the text automatically wraps when it exceeds the specified width.");

// Set the alignment of the text that we'll draw next

QP.SetTextAlign(2);

// Draw text in a text box. Specify width and height
// of the text box.

QP.DrawTextBox(350, 150, 200, 200, "This text was drawn using the DrawTextBox function.
Similar to the DrawText function except that the alignment can be specified.", 1);
```

```
// Change the alignment

QP.SetTextAlign(0);

// Draw rotated text

QP.DrawRotatedTextBox(300, 200, 200, 200, 90, "This text was drawn using the
DrawRotatedTextBox function.", 1);

// Draw text where each character has a space between it

QP.DrawSpacedText(15, 300, 10, "This text was drawn using the DrawSpacedText function.");

// Draw some more text

QP.DrawText(25, 25, "This text was drawn using the DrawText function.");

// Save the new file to the output folder

QP.SaveToFile("Text.pdf");
```

Add HTML text

Quick PDF Library provides some HTML text functions which give you greater control over the layout and styling of your text. See Appendix A of the Function Reference for a full list of the HTML tags that are available for use with Quick PDF Library.

```
/* Use HTML text to make styling and laying out text easier */

// Set the origin for the drawing co-ordinates. In this case
// we'll draw the co-ordinates from the top left corner of the page.

QP.SetOrigin(1);

// Draw a bullet list using DrawHTMLText

QP.DrawHTMLText(100, 100, 200, "<ul><li>Item 1</li><li>Item 2</li><li>Item 3</li><li>Item
4</li><li>Item 5</li></ul>");

// Draw a paragraph of text in a text box
// with some font and italic text

QP.DrawHTMLTextBox(200, 300, 200, 200, "<p>This is a text box. I can make some of it
<b>bold</b> and some <i>italic</i> using HTML tags.</p>");

// Save the new file to the output folder

QP.SaveToFile("html_text.pdf");
```

PDF Text Extraction

Extracting text from a PDF can at times be a difficult task. If a PDF has been scanned from paper and no OCR process has been performed then the page in a PDF might display what looks like text but in the objects in the PDF no text actually exists, just an image. You can use the HasFontResources function to check if a PDF has any text. Extracting text from a PDF that does indeed contain text objects is a relatively straight forward process using one of the text extraction functions.

Extract text

Simple text extraction where the text of a PDF is output in a human readable format into plain text is very simple using either the GetPageText, ExtractFilePageText or DAExtractPageText functions.

```
/* Extract text from a PDF */

// Load the test file and iterate through each page in the
// PDF and append it to a text file.

strInputFilePath = "apply_ fingerprint.pdf";

QP.LoadFromFile(strInputFilePath);

iNumPages = QP.PageCount(); // Calculate the number of pages

strText = "";
nPage = 0;

for(nPage = 1; nPage<=iNumPages; nPage++)
{
  strText = strText + QP.ExtractFilePageText(strInputFilePath, "", nPage, 0);
}

// Write all the data to a file
s = oFSO.CreateTextFile("extracted_text.txt", 1);
s.Writeline(strText);
s.Close();
```

Extract text advanced

Advanced text extraction where the text, along with co-ordinates, font information is returned as a CSV string is possible using the GetPageText, ExtractFilePageText or DAExtractPageText functions. Using options 3 or 4 of the ExtractOptions parameter you can have text returned as individual words or chunks.

```
/* Extract advanced text from a PDF */

// Load the file and iterate through each page in the
// PDF and append it to a text file.

strInputFilePath = "apply_ fingerprint.pdf";

QP.LoadFromFile(strInputFilePath);

iNumPages = QP.PageCount(); // Calculate the number of pages
```

```
strText = "";
nPage = 0;

for(nPage = 1; nPage<=iNumPages; nPage++)
{
  strText = strText + QP.ExtractFilePageText(strInputFilePath, "", nPage, 4);
}

// Write all the data to a file

s = oFSO.CreateTextFile("extracted_text.txt", 1);
s.Writeline(strText);
s.Close();
```

PDF Images

Quick PDF Library provides extensive support for adding images to PDF files, extracting images from PDF files, replacing images in PDF files and converting images to PDF and PDFs to images.

Add image to PDF

Use the AddImageFromFile and DrawImage functions to add images to PDF files. Supported image types: BMP, TIFF, JPEG, PNG, GIF, WMF and EMF.

```
/* Add an image to an existing PDF document */

// Load a file from disk. We'll place the image
// onto this file.

QP.LoadFromFile("example.pdf");

// Load your image into memory

QP.AddImageFromFile("example.bmp", 0);

// Get width and height of the image

lWidth = QP.ImageWidth();
lHeight = QP.ImageHeight();

// Draw the image onto the page using the specified width/height

QP.DrawImage(250, 450, lWidth, lHeight);

// Save the updated file to the output folder

QP.SaveToFile("pdf_with_image.pdf");
```

Extract image from PDF to file

Use the SaveImageToFile function to extract embedded images from PDF files to disk as JPG, BMP or TIFF images.

```
/* Extract an image from a PDF to file */

// Load the PDF

QP.LoadFromFile("example.pdf")

// Find all images in the PDF

ImagesFound = QP.FindImages()

// Get the image ID for the first image
// found in the PDF

ImageID = QP.GetImageID(1)

// Select the image using its ID

QP.SelectImage(ImageID)
```

```
// Determine the embedded images
// file type and then save to disk

ImageTypeFound = QP.ImageType();

if (ImageTypeFound == 0)
{
MsgBox("No image is selected");
}
else if (ImageTypeFound == 1)
{
QP.SaveImageToFile("embedded_image.jpg");
}
else if (ImageTypeFound == 2)
{
QP.SaveImageToFile("embedded_image.bmp");
}
else if (ImageTypeFound == 3)
{
QP.SaveImageToFile("embedded_image.tiff");
}
```

Replace an image

Use the ReplaceImage function to replace an old image in a PDF with a new image.

```
/* Replace an image in a PDF */

// Load the PDF

QP.LoadFromFile("example.pdf")

// Find all images in the PDF

ImagesFound = QP.FindImages()

// Get the image ID for the first image
// found in the PDF

OriginalImageID = QP.GetImageID(1)

// Add new image to the PDF

NewImageID = QP.AddImageFromFile("example.bmp", 0);

// Replace the old image with the new image

QP.ReplaceImage(OriginalImageID, NewImageID);

// Save the updated PDF

QP.SaveToFile("image_replaced.pdf");
```

Convert EMF to PDF

Use the ImportEMFFromFile function to convert EMF files to PDF.

```
/* EMF to PDF conversion */

// Load your image into memory
```

```
eImageID = QP.ImportEMFFromFile"tiger.emf", 0, 0);

// Select the imported image

QP.SelectImage(eImageID);

// Get width, height of the image

lWidth = QP.ImageWidth();
lHeight = QP.ImageHeight();

// Reformat the size of the page in the selected document

QP.SetPageDimensions(lWidth, lHeight);

// Draw the image onto the page using the specified width/height

QP.DrawImage(0, lHeight, lWidth, lHeight);

// Store the updated PDF where you like

QP.SaveToFile("tiger_emf.pdf");
```

Convert Image to PDF

Use the AddImageFromFile, SetPageDimensions and DrawImage functions to convert an image to PDF. Supported image types: BMP, TIFF, JPEG, PNG, GIF, WMF and EMF.

```
/* Convert an image to a PDF */

// Load a sample image into memory

QP.AddImageFromFile("sample123.jpg", 0);

// Get the width and height of the image

lWidth = QP.ImageWidth();
lHeight = QP.ImageHeight();

// Reformat the size of the page in the selected document

QP.SetPageDimensions(lWidth, lHeight);

// Draw the image onto the page using the specified width/height

QP.DrawImage(0, lHeight, lWidth, lHeight);

// Store the updated PDF where you like

QP.SaveToFile("sample123.pdf");
```

Convert PDF to Image

Converting a PDF to an image is easy using the RenderDocumentToFile function or any of the other functions that start with Render*.

```
/* PDF to image conversion */

// Load the 'debenu final tm.pdf' sample file
```

```
QP.LoadFromFile("example.pdf");

// Calculate the number of pages

iNumPages = QP.PageCount();

// Render each page of the document to a separate file.
// To view the images open the output folder.

QP.RenderDocumentToFile(72, 1, iNumPages, 0,"example.bmp");
```

PDF Color

Quick PDF Library provides you with extensive support for controlling the color of text and vector graphics that you add to PDF files.

```
/* Set the color for a variety of different objects -- text, vector graphics, etc */

// Specify the page size

QP.SetPageSize('A4');

// Specify the corner of the page where co-ordinates should start.
// In this example we'll specify the top left corner.

QP.SetOrigin(1);

// Set the line color and width

QP.SetLineColor(255, 0, 255);
QP.SetLineWidth(0.5);

// Set fill color and then draw a circle

QP.SetFillColor(0, 0, 255);
QP.DrawCircle(250, 600, 100, 2);

// Draw some text and specify various
// different colors for the text.

QP.DrawText(25, 25, "This text was drawn using the DrawText function.");

QP.SetTextColor(.4, .5, 0);
QP.DrawTextArc(150, 150, 100, 280, "This text was drawn using the DrawTextArc function.",
0);

QP.SetTextColor(0, .3, 0);
QP.DrawWrappedText(400, 50, 200, "This text was drawn using the DrawWrappedText function.
As you can see, the text automatically wraps when it exceeds the specified width.");

QP.SetTextAlign(2);
QP.SetTextColor(0, .1, .4);
QP.DrawTextBox(350, 150, 200, 200, "This text was drawn using the DrawTextBox function.
Similar to the DrawText function except that the alignment can be specified.", 1);

QP.SetTextAlign(0);
QP.DrawRotatedTextBox(300, 200, 200, 200, 90, "This text was drawn using the
DrawRotatedTextBox function.", 1);
QP.SetTextColor(.6, .11, .44);

QP.DrawSpacedText(15, 300, 10, "This text was drawn using the DrawSpacedText function.");
QP.DrawText(25, 25, "This text was drawn using the DrawText function.");

// Save the new document to disk

QP.SaveToFile("Color.pdf");
```

PDF Vector Graphics

Quick PDF Library provides extensive support for vector graphics enabling you to draw various different shapes onto your PDF files. In this example we draw multiple boxes and circles with a variety of different colors.

```
/* Draw some shapes on a new PDF */

// Set the page size for the new PDF

QP.SetPageSize('A4');

// Set the origin for the drawing co-ordinates
// to be the top left corner of the page.

QP.SetOrigin(1);

// Set the measurement unit that we'll use.
// In this example we'll use millimetres.

QP.SetMeasurementUnits(1);

// Set the color and the width for the line

QP.SetLineColor(255, 0, 0);
QP.SetLineWidth(0.5);

// Create a variety of boxes and circles
// using a loop.

for (x = 0; x <= 9; x++)
{
  for (y = 0; y <= 10; y++)
  {
    QP.DrawBox(5 + x * 20, 5 + y * 25, 20, 25, 0);
    QP.SetFillColor(0, 192, 0);
    QP.DrawCircle(15 + x * 20, 17.5 + y * 25, 8, 2);
  }
}

// Save the new file to the output folder

QP.SaveToFile("Shapes.pdf");
```

PDF Optional Content Groups (aka Acrobat Layers)

*Please note: optional content groups are what Acrobat and Adobe Reader refer to as layers. Optional content groups is the term used in the PDF specification. See the **API Overview** section for a more detailed analysis.*

Quick PDF Library provides extensive support for working with optional content groups in a variety of different ways, from creating new OCGs to editing or deleting existing OCGs.

Create OCGs

Use the NewOptionalContentGroup function to create optional content groups and add content to them.

```
/* These script shows you how to create multiple layers in a document. Layers are
technically called Optional Content Groups (OCGs). */

// Create four new optional content groups

OCG1 = QP.NewOptionalContentGroup("Layer 1");
OCG2 = QP.NewOptionalContentGroup("Layer 2");
OCG3 = QP.NewOptionalContentGroup("Layer 3");
OCG4 = QP.NewOptionalContentGroup("Layer 4");

// Select the page that you want the layers to be
// associated with.

QP.SelectPage(1);

// Specify top left corner for starting point
// of all drawing functions.

QP.SetOrigin(1);

// Add layer 1

QP.NewLayer();
QP.SelectLayer(1);
QP.DrawText(100, 100, "Layer 1");
QP.SetLayerOptional(OCG1);
QP.SetOptionalContentGroupVisible(OCG1, 1);

// Add layer 2

QP.NewLayer();
QP.SelectLayer(2);
QP.DrawText(200, 100, "Layer 2");
QP.SetLayerOptional(OCG2);
QP.SetOptionalContentGroupVisible(OCG2, 1);

// Add layer 3

QP.NewLayer();
QP.SelectLayer(3);
QP.DrawText(300, 100, "Layer 3");
QP.SetLayerOptional(OCG3);
QP.SetOptionalContentGroupVisible(OCG3, 1);

// Add layer 4

QP.NewLayer();
QP.SelectLayer(4);
QP.DrawText(400, 100, "Layer 4");
QP.SetLayerOptional(OCG4);
QP.SetOptionalContentGroupVisible(OCG4, 1);

// Save file to disk with new layers

QP.SaveToFile("option_content_groups.pdf");
```

Control visibility and printability of OCGs

Use the SetOptionalContentGroupVisible and SetOptionalContentGroupPrintable functions to control the visibility and printability settings of OCGs.

```
/* Create OCGs with differing visibility and printability settings */

// Create four new optional content groups

OCG1 = QP.NewOptionalContentGroup("Layer 1");
OCG2 = QP.NewOptionalContentGroup("Layer 2");
OCG3 = QP.NewOptionalContentGroup("Layer 3");
OCG4 = QP.NewOptionalContentGroup("Layer 4");

// Select the page that you want the layers to be
// associated with.

QP.SelectPage(1);

// Specify top left corner for starting point
// of all drawing functions.

QP.SetOrigin(1);

// Add OCG  1

QP.NewLayer();
QP.SelectLayer(1);
QP.DrawText(100, 100, "OCG  1");
QP.SetLayerOptional(OCG1);
QP.SetOptionalContentGroupVisible(OCG1, 1); // Set this OCG to be visible

// Add OCG  2

QP.NewLayer();
QP.SelectLayer(2);
QP.DrawText(200, 100, "OCG 2");
QP.SetLayerOptional(OCG2);
QP.SetOptionalContentGroupVisible(OCG2, 0); // Set this OCG to not be visible

// Add OCG  3

QP.NewLayer();
QP.SelectLayer(3);
QP.DrawText(300, 100, "OCG 3");
QP.SetLayerOptional(OCG3);
QP.SetOptionalContentGroupVisible(OCG3, 1); // Set this OCG to be visible
QP.SetOptionalContentGroupPrintable(OCG3, 1);  // Set this OCG to be printable

// Add OCG  4

QP.NewLayer();
QP.SelectLayer(4);
QP.DrawText(400, 100, "OCG 4");
QP.SetLayerOptional(OCG4);
QP.SetOptionalContentGroupVisible(OCG4, 1); // Set this OCG to be visible
QP.SetOptionalContentGroupPrintable(OCG4, 0);  // Set this OCG to be not printable

// Save file to disk with new OCGs

QP.SaveToFile("ocgs.pdf");
```

Remove OCGs

Use the OptionalContentGroupCount, GetOptionalContentGroupID and DeleteOptionalContentGroup functions to delete optional content groups from your PDFs.

```
/* Delete all optional content groups */

// Load the PDF that contains the OCGs

QP.LoadFromFile("ocgs.pdf");

// Count OCGs

OCGCount = QP.OptionalContentGroupCount();

// Loop through each OCG and delete it

for(i = 1; i <= OCGCount; i++)
{
  OCGID = QP.GetOptionalContentGroupID(i);
  QP.DeleteOptionalContentGroup(OCGID);
}

// Save file to disk with new layers

QP.SaveToFile("no_ocgs.pdf");
```

PDF Content Streams

Content streams are the primary means for describing the appearance of pages and other graphical elements. In Quick PDF Library content streams are called layers.

Combine content streams

Use the CombineLayers function to combine all content streams for the selected page into one content stream.

```
/* Combine all layers for each page in a PDF */

// Load PDF to process

QP.LoadFromFile("example.pdf");

// Count pages

int xPageCount = QP.PageCount();

// Go through each page and combine layers

for (int i = 1; i <= xPageCount; i++)
{
    QP.SelectPage(i);
    QP.CombineLayers();
}

// Save the updated file

QP.SaveToFile("example_updated.pdf");
```

Remove shared content streams

Use the RemoveSharedLayers function to ensure that none of the pages in the selected document share any content streams.

```
/* Remove shared content streams */

// Load the PDF
QP.LoadFromFile(“example.pdf”);

// Remove shared content streams

QP.RemoveSharedLayers();

// Save the updated PDF

QP.SaveToFile(“example_updated.pdf”);
```

Encapsulate layers

Use the EncapsulateLayer function to surround the current layer with "save graphics state" and "restore graphics state" operators.

```
/* Encapsulate content streams in a PDF */

// Load the PDF

QP.LoadFromFile("example.pdf");

// Count pages

xPageCount = QP.PageCount();

// Go through each page and encapsulate layers

for (int i = 1; i <= xPageCount; i++)
{
    QP.SelectPage(i);
    int xLayerCount = QP.LayerCount();

    for (int x = 1; x <= xLayerCount; x++)
    {
        QP.SelectLayer(x);
        QP.EncapsulateLayer();
    }
}

// Save the updated file

QP.SaveToFile("example_updated.pdf");
```

PDF Attachments

PDF files can contain fully embedded files. These embedded files are accessible from the Attachments menu in conforming PDF readers. There is no limitation on the types of files which can be embedded, although recent versions of Adobe Acrobat have not permitted executable depending on the security settings specified in the preferences.

Embed a file in a PDF

Use the EmbedFile function to embed a file within a PDF.

```
/* Add a file attachment to a PDF */

// Load the PDF

QP.LoadFromFile("example.pdf");

// Embed a Word document (can be any media type) in
// selected document.

QP.EmbedFile("Link to embedded file...", "example.docx", "application/msword");

// Save the updated file to disk

QP.SaveToFile(example_updated.pdf");
```

Count embedded files in a PDF

Use the EmbeddedFileCount function to count the number of embedded files in a PDF.

```
/* Count embedded files in a PDF */

// Load the PDF

QP.LoadFromFile("example.pdf");

// Count the embedded files

embeddedFiles = QP.EmbeddedFileCount();

// Display number of embedded files

MsgBox("Number of Embedded Files: " + embeddedFiles);
```

Extract embedded file from PDF

Use the GetEmbeddedFileContentToFile function to extract the content of an embedded file from a PDF.

```
/* Extract embedded file from a PDF*/

// Load PDF with embedded file

QP.LoadFromFile("pdf_with_embedded_file.pdf");

// Extract embedded file content to file on disk

QP.GetEmbeddedFileContentToFile(1, "JavaScript.docx");
```

PDF Barcodes

Quick PDF Library provides support for Code39 (or Code 3 of 9), EAN-13, Code128, PostNet and Interleaved 2 of 5 barcodes.

Draw a barcode

Use the DrawBarcode function to add barcodes to your PDF.

```
/* Draw a variety of different barcodes on a new page in a PDF */

// Set the origin for the co-ordinates to be
// the top left corner of the page.

QP.SetOrigin(1);

// Draw three different barcodes

QP.DrawBarCode(25, 25, 150, 100, "MyBarcode256", 1, 0);
QP.DrawBarCode(225, 50, 100, 600, "MyBarcode257/RC", 1, 0);
QP.DrawBarCode(350, 50, 200, 150, "MyBarcode258", 3, 0);

// Save the new file

QP.SaveToFile("Barcodes.pdf");
```

Draw text under a barcode

Use the GetBarcodeWidth, DrawBarcode and Drawtext functions to add a barcode and draw text underneath it.

```
/* Draw text under a barcode */

// Add the Helvetica standard font

QP.AddStandardFont(4);

// Get the width of the barcode with a bar width of 1 unit

BarcodeWidth = QP.GetBarcodeWidth(1, "12345", 1);

// Draw the barcode

QP.DrawBarcode(100, 500, BarcodeWidth, 100, "12345", 1, 0);

// Center the text

QP.SetTextAlign(1);

// Draw the text at 10pt

QP.SetTextSize(10);
QP.DrawText(100 + BarcodeWidth / 2, 500 - 200 - 5 - QP.GetTextHeight(), "12345");

// Save the updated PDF

QP.SaveToFile("text_under_barcode.pdf");
```

PDF Metadata

Quick PDF Library lets you control metadata in PDF files as the Title, Author, Subject and Keyword properties, as well as custom metadata.

Set document properties

Use the SetInformation function to add Title, Author, Subject and Keyword properties to a PDF.

```
/* Add standard document information (metadata) to a document. */

// Custom information can be added using the

QP.SetInformation(0, "1.9");
QP.SetInformation(1, "John Smith");
QP.SetInformation(2, "The Life and Times of John Smith");
QP.SetInformation(3, "A very special book");
QP.SetInformation(4, "book, paperback, ebook");
QP.SetInformation(5, "Mother Earth");
QP.SetInformation(6, "Humanity");

// Draw some explanatory text onto the blank document.

QP.DrawText(100, 700, "Select Ctrl+D and then click on the Description tab to see the
document information.");

// Save the new file

QP.SaveToFile("document_information.pdf");
```

Get document properties

Use the GetInformation function to get document properties from PDF files.

```
/* Get and display the document properties (metadata) for a document */

// Load a PDF

QP.LoadFromFile("document properties.pdf");

// Extract information from the document

DocInformation = "";
DocInformation = DocInformation + "PDF Version: " + QP.GetInformation(0) + "\n";
DocInformation = DocInformation + "Author: " + QP.GetInformation(1) + "\n";
DocInformation = DocInformation + "Title: " + QP.GetInformation(2) + "\n";
DocInformation = DocInformation + "Subject: " + QP.GetInformation(3) + "\n";
DocInformation = DocInformation + "Keywords: " + QP.GetInformation(4) + "\n";
DocInformation = DocInformation + "Creator: " + QP.GetInformation(5) + "\n";
DocInformation = DocInformation + "Producer: " + QP.GetInformation(6) + "\n";
DocInformation = DocInformation + "Creation date: " + QP.GetInformation(7) + "\n";
DocInformation = DocInformation + "Modification date: " + QP.GetInformation(8) + "\n";

// Display the information to the user

MsgBox(DocInformation);
```

Set custom metadata

Use the SetCustomInformation function to add custom metadata to PDF files.

```
/* Add custom information (metadata) to documents. */

// Custom information can be added using the
// SetCustomInformation function. Create your
// own Key and Value and call the function as
// many times as you require.

QP.SetCustomInformation("FirstName", "John");
QP.SetCustomInformation("LastName", "Smith");
QP.SetCustomInformation("Coolness Level", "Very Cool");

// Draw some explanatory text onto the blank document.

QP.DrawText(100, 700, "Select Ctrl+D and then click on the Custom tab to see the custom
information.");

// Save the new file

QP.SaveToFile("custom_information.pdf");
```

Get custom metadata

Use the GetCustomInformation to retrieve custom metadata from PDF files.

```
/* Get custom information (metadata) from PDFs. */

QP.LoadFromFile("custom_information.pdf");

// Custom information/metadata can be retrieved
// from PDF files using the GetCustomInformation function.
// Custom metadata is stored in name/value pairs,
// so in order to retrieve metadata you must know the
// name of the entry that you want to retrieve.

myCustomInfo1 = QP.GetCustomInformation("FirstName");
myCustomInfo2 = QP.GetCustomInformation("LastName");
myCustomInfo3 = QP.GetCustomInformation("Coolness Level");

// Display the custom metadata that we've just retrieved

MsgBox(myCustomInfo1);
MsgBox(myCustomInfo2);
MsgBox(myCustomInfo3);
```

Misc

This section contains a variety of useful information that doesn't fall into any other category.

PDF Resources

There are many PDF resources available on the web, but we've compiled a few of the really good ones for you below to give you a head start. It isn't necessary to know all about the technical details of PDF when you work with our library, so this is just for the curious.

- Planet PDF -- PDF news, tips and in-depth articles

- Planet PDF Forum -- PDF discussion forum
- Planet PDF Q&A -- PDF questions and PDF answers
- PDF Tutorials by Jim King -- easy to follow technical presentations on PDF
- Inside PDF -- a blog from Jim King, a Senior Scientist at Adobe Systems
- PDF basics -- articles on the basic building blocks of PDF
- PDF Specification -- PDF Reference and Adobe Extensions to the PDF Specification

There are many more PDF resources available on the web, but these are some of the best. There are also some more specific resources dealing with PDF available at Planet PDF.

- Introduction to Acrobat & PDF
- Accessible PDF
- PDF Color
- PDF Preflight
- Introduction to Acrobat Development
- Introduction to Acrobat JavaScript

Useful 3rd Party Applications

These are just some useful applications that we use in our day to day testing.

- AsTiffTagViewer -- a free TIFF tag (code, data type, count, value) viewer application
- Quick PDF Tools - application for manipulating PDFs, built using Quick PDF Library
- ARTS PDF Workshop -
- tiffinfo.exe (uses libTIFF) - analyze TIFF images, discussed on our blog
- JPEGsnoop -- JPEG file decoding utility
- Nitro PDF Reader -- free PDF viewer with some form filling and other advanced features
- Foxit PDF Reader - free lightweight PDF viewer
- Adobe Acrobat -- Acrobat is a very useful tool if you work with PDF every day
- PDF CanOpener -- this is an Acrobat plug-in for COS level manipulation of PDFs
- GPL Ghostscript -- software based on interpreter for PostScript and PDF
- GSview -- graphical interface for Ghostscript, display PDF and PostScript
- Primo PDF -- open source PDF printer driver
- EMF Explorer -- EMF/WMF previewing, conversion and printing
- Enfocus Browser -- browse internal objects and dictionaries of PDF files
- Notepad++ -- plain text editor useful for opening PDF files in plain text
- AMP Font Viewer -- useful font manager for keeping track of installed fonts

You will notice a couple of different PDF viewers on this list, we always find it is a good idea to test your PDF files in a variety of different PDF viewers because you never know what PDF viewer your client is going to be using.