pdf_output

Method Summary

byte[]	#addMetaData(data, metaData) Add metadata to the PDF, like Author
byte[]	#combinePDFDocuments(pdf_docs_bytearrays) Combine multiple PDF docs into one.
byte[]	#combineProtectedPDFDocuments(pdf_docs_bytearrays, pdf_docs_passwords) Combine multiple protected PDF docs into one.
byte[]	#convertPDFFormToPDFDocument(pdf_form, field_values) Convert a PDF form to a PDF document.
byte[]	#convertProtectedPDFFormToPDFDocument(pdf_form, pdf_password, field_values) Convert a protected PDF form to a PDF document.
byte[]	#encrypt(data, ownerPassword) Add password protection and security options to the PDF
byte[]	#encrypt(data, ownerPassword, userPassword) Add password protection and security options to the PDF
byte[]	#encrypt(data, ownerPassword, userPassword, allowAssembly, allowCopy, allowDegradedPrinting, allowFillIn, allowModifyAnnotations, allowModifyContents, allowPrinting, allowScreenreaders) Add password protection and security options to the PDF
byte[]	#encrypt(data, ownerPassword, userPassword, allowAssembly, allowCopy, allowDegradedPrinting, allowFillIn, allowModifyAnnotations, allowModifyContents, allowPrinting, allowScreenreaders, is128bit) Add password protection and security options to the PDF
byte[]	#encrypt(data, ownerPassword, userPassword, allowAssembly, allowCopy, allowDegradedPrinting, allowFillIn, allowModifyAnnotations, allowModifyContents, allowPrinting, allowScreenreaders, is128bit, metaData) Add password protection and security options to the PDF
byte[]	#endMetaPrintJob() Ends a previously started meta print job.
java.awt.print.PrinterJob	#getPDFPrinter() Returns a PDF printer that can be used in print calls.
java.awt.print.PrinterJob	#getPDFPrinter(filename) Returns a PDF printer that can be used in print calls.
Number	#getPagesPrinted() Returns the number of pages printed by the last print call done in the context of a meta print job.
Number	#getTotalPagesPrinted() Returns the total number of pages printed in the context of a meta print job.
Number	#insertFontDirectory(path) Add a directory that should be searched for fonts.
byte[]	#numberPages(data) Add pages numbers to a PDF
byte[]	#numberPages(data, fontSize, locationX, locationY, font, hexColor) Add pages numbers to a PDF
byte[]	#overlay(data, forOverlay) Add some PDF based content over a PDF
byte[]	#overlay(data, forOverlay, isOver) Add some PDF based content over a PDF
byte[]	#overlay(data, forOverlay, isOver, pages) Add some PDF based content over a PDF
byte[]	#overlay(data, forOverlay, pages) Add some PDF based content over a PDF
byte[]	#overlayText(data, text) Add text over every page at a 45 degree angle
byte[]	#overlayText(data, text, locationX, locationY, isOver, fontSize, font, hexColor) Add text over every page at a 45 degree angle
Boolean	#startMetaPrintJob() Used for printing multiple things into the same PDF document.
Boolean	#startMetaPrintJob(filename) Used for printing multiple things into the same PDF document.
byte[]	#watermark(data, image) Add an image as a watermark on every page, or the pages specified as a parameter
byte[]	#watermark(data, image, locationX, locationY, isOver) Add an image as a watermark on every page, or the pages specified as a parameter
byte[]	#watermark(data, image, locationX, locationY, isOver, pages) Add an image as a watermark on every page, or the pages specified as a parameter

Method Details

addMetaData

byte[] **addMetaData**(data, metaData)
Add metadata to the PDF, like Author
Parameters\\\{byte[]} data – the PDF
{Object} metaData – a JavaScript object (Scriptable) that contains the metadata as property/value pairs
Returns
byte[] – the PDF with metaData added
Sample

```
// Add metadata to the PDF, like Author
var pdf = plugins.file.showFileOpenDialog();
if (pdf) {
        var data = plugins.file.readFile(pdf);
        var metaData = { Author: 'Servoy' };
        pdfResult = elements.customer_id.addMetaData(data, metaData);
}
```

combinePDFDocuments

byte[] **combinePDFDocuments**(pdf_docs_bytearrays)
Combine multiple PDF docs into one.
Parameters
{Object[]} pdf_docs_bytearrays – the array of documents to combine
Returns
byte[]
Sample

```
pdf_blob_column = combinePDFDocuments(new Array(pdf_blob1,pdf_blob2,pdf_blob3));
```

combineProtectedPDFDocuments

byte[] **combineProtectedPDFDocuments**(pdf_docs_bytearrays, pdf_docs_passwords)
Combine multiple protected PDF docs into one.
Parameters
{Object[]} pdf_docs_bytearrays – the array of documents to combine
{Object[]} pdf_docs_passwords – an array of passwords to use
Returns
byte[]
Sample

```
pdf_blob_column = combineProtectedPDFDocuments(new Array(pdf_blob1,pdf_blob2,pdf_blob3), new Array
(pdf_blob1_pass,pdf_blob2_pass,pdf_blob3_pass));
```

convertPDFFormToPDFDocument

byte[] **convertPDFFormToPDFDocument**(pdf_form, field_values)
Convert a PDF form to a PDF document.
Parameters\\\{byte[]} pdf_form – the PDF Form to convert
{Object} field_values – the values to use
Returns
byte[]
Sample

```
var pdfform = plugins.file.readFile('c:/temp/1040a-form.pdf');
//var field_values = plugins.file.readFile('c:/temp/1040a-data.fdf');//read adobe fdf values or
var field_values = new Array()//construct field values
field_values[0] = 'f1-1=John C.J.'
field_values[1] = 'f1-2=Longlasting'
var result_pdf_doc = plugins.pdf_output.convertPDFFormToPDFDocument(pdfform, field_values)
if (result_pdf_doc != null)
{
        plugins.file.writeFile('c:/temp/1040a-flatten.pdf', result_pdf_doc)
}
```

convertProtectedPDFFormToPDFDocument

byte[] **convertProtectedPDFFormToPDFDocument**(pdf_form, pdf_password, field_values)
Convert a protected PDF form to a PDF document.

Parameters\\\{byte[]} pdf_form – the PDF Form to convert
{String} pdf_password – the password to use
{Object} field_values – the field values to use
Returns
byte[]
Sample

```
var pdfform = plugins.file.readFile('c:/temp/1040a-form.pdf');
//var field_values = plugins.file.readFile('c:/temp/1040a-data.fdf');//read adobe fdf values or
var field_values = new Array()//construct field values
field_values[0] = 'f1-1=John C.J.'
field_values[1] = 'f1-2=Longlasting'
var result_pdf_doc = plugins.pdf_output.convertProtectedPDFFormToPDFDocument(pdfform, 'pdf_password',
field_values)
if (result_pdf_doc != null)
{
        plugins.file.writeFile('c:/temp/1040a-flatten.pdf', result_pdf_doc)
}
```

encrypt
byte[] **encrypt**(data, ownerPassword)
Add password protection and security options to the PDF
Parameters\\\{byte[]} data – the PDF
{String} ownerPassword – the owner password
Returns
byte[] – the encrypted PDF
Sample

```
// Add password protection and security options to the PDF
// NOTE: Passwords are case sensitive
var unEncryptedFile = plugins.file.showFileOpenDialog();
if (unEncryptedFile) {
        var data = plugins.file.readFile(unEncryptedFile);
        encryptedResult = elements.customer_id.encrypt(data, 'secretPassword', 'secretUserPassword', false,
false, false, false, false, false, false, false, true);
}
```

encrypt
byte[] **encrypt**(data, ownerPassword, userPassword)
Add password protection and security options to the PDF
Parameters\\\{byte[]} data – the PDF
{String} ownerPassword – the owner password
{String} userPassword – the user password
Returns
byte[] – the encrypted PDF
Sample

```
// Add password protection and security options to the PDF
// NOTE: Passwords are case sensitive
var unEncryptedFile = plugins.file.showFileOpenDialog();
if (unEncryptedFile) {
        var data = plugins.file.readFile(unEncryptedFile);
        encryptedResult = elements.customer_id.encrypt(data, 'secretPassword', 'secretUserPassword', false,
false, false, false, false, false, false, false, true);
}
```

encrypt
byte[] **encrypt**
(data, ownerPassword, userPassword, allowAssembly, allowCopy, allowDegradedPrinting, allowFillIn, allowModifyAnnotations, allowModifyContents, allowPrinting, allowScreenreaders)
Add password protection and security options to the PDF

Parameters\\\{byte[]} data – the PDF
{String} ownerPassword – the owner password
{String} userPassword – the user password
{Boolean} allowAssembly – whether to set the allow assembly permission
{Boolean} allowCopy – whether to set the allow copy permission
{Boolean} allowDegradedPrinting – whether to set the allow degraded printing permission
{Boolean} allowFillIn – whether to set the allow fill in permission
{Boolean} allowModifyAnnotations – whether to set the allow modify annotations permission
{Boolean} allowModifyContents – whether to set the allow modify contents permission
{Boolean} allowPrinting – whether to set the allow printing permission
{Boolean} allowScreenreaders – whether to set the allow screen readers permission

Returns
byte[] – the encrypted PDF

Sample

```
// Add password protection and security options to the PDF
// NOTE: Passwords are case sensitive
var unEncryptedFile = plugins.file.showFileOpenDialog();
if (unEncryptedFile) {
        var data = plugins.file.readFile(unEncryptedFile);
        encryptedResult = elements.customer_id.encrypt(data, 'secretPassword', 'secretUserPassword', false,
false, false, false, false, false, false, false, true);
}
```

encrypt

byte[] **encrypt**
(data, ownerPassword, userPassword, allowAssembly, allowCopy, allowDegradedPrinting, allowFillIn, allowModifyAnnotations, allowModifyContents, allowPrinting, allowScreenreaders, is128bit)

Add password protection and security options to the PDF

Parameters\\\{byte[]} data – the PDF
{String} ownerPassword – the owner password
{String} userPassword – the user password
{Boolean} allowAssembly – whether to set the allow assembly permission
{Boolean} allowCopy – whether to set the allow copy permission
{Boolean} allowDegradedPrinting – whether to set the allow degraded printing permission
{Boolean} allowFillIn – whether to set the allow fill in permission
{Boolean} allowModifyAnnotations – whether to set the allow modify annotations permission
{Boolean} allowModifyContents – whether to set the allow modify contents permission
{Boolean} allowPrinting – whether to set the allow printing permission
{Boolean} allowScreenreaders – whether to set the allow screen readers permission
{Boolean} is128bit – whether to use 128-bit encryption

Returns
byte[] – the encrypted PDF

Sample

```
// Add password protection and security options to the PDF
// NOTE: Passwords are case sensitive
var unEncryptedFile = plugins.file.showFileOpenDialog();
if (unEncryptedFile) {
        var data = plugins.file.readFile(unEncryptedFile);
        encryptedResult = elements.customer_id.encrypt(data, 'secretPassword', 'secretUserPassword', false,
false, false, false, false, false, false, false, true);
}
```

encrypt

byte[] **encrypt**
(data, ownerPassword, userPassword, allowAssembly, allowCopy, allowDegradedPrinting, allowFillIn, allowModifyAnnotations, allowModifyContents, allowPrinting, allowScreenreaders, is128bit, metaData)

Add password protection and security options to the PDF

Parameters\\\{byte[]} data – the PDF
{String} ownerPassword – the owner password
{String} userPassword – the user password
{Boolean} allowAssembly – whether to set the allow assembly permission
{Boolean} allowCopy – whether to set the allow copy permission
{Boolean} allowDegradedPrinting – whether to set the allow degraded printing permission
{Boolean} allowFillIn – whether to set the allow fill in permission
{Boolean} allowModifyAnnotations – whether to set the allow modify annotations permission
{Boolean} allowModifyContents – whether to set the allow modify contents permission
{Boolean} allowPrinting – whether to set the allow printing permission
{Boolean} allowScreenreaders – whether to set the allow screen readers permission
{Boolean} is128bit – whether to use 128-bit encryption
{Object} metaData – a JavaScript object (Scriptable) that contains the metadata as property/value pairs

Returns
byte[] – the encrypted PDF

Sample

```
// Add password protection and security options to the PDF
// NOTE: Passwords are case sensitive
var unEncryptedFile = plugins.file.showFileOpenDialog();
if (unEncryptedFile) {
        var data = plugins.file.readFile(unEncryptedFile);
        encryptedResult = elements.customer_id.encrypt(data, 'secretPassword', 'secretUserPassword', false,
false, false, false, false, false, false, false, true);
}
```

endMetaPrintJob

byte[] **endMetaPrintJob**()

Ends a previously started meta print job. For meta print jobs that were stored in memory, not in a file on disk, also returns the content of the generated PDF document.

Returns
byte[]

Sample

```
//to print multiple forms to one pdf document (on file system).
var success = plugins.pdf_output.startMetaPrintJob('c:/temp/out.pdf')
if (success)
{
        forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
        application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
        forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
        application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
plugins.pdf_output.endMetaPrintJob()

//to print multiple forms to one pdf document (to store in dataprovider).
var success = plugins.pdf_output.startMetaPrintJob()
if (success)
{
        forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
        application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
        forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
        application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
mediaDataProvider = plugins.pdf_output.endMetaPrintJob()
```

getPDFPrinter

java.awt.print.PrinterJob **getPDFPrinter**()

Returns a PDF printer that can be used in print calls. Returns the last started meta print job.

Returns
java.awt.print.PrinterJob

Sample

```
//to print current record without printdialog to pdf file in temp dir.
controller.print(true,false,plugins.pdf_output.getPDFPrinter());
```

getPDFPrinter

java.awt.print.PrinterJob **getPDFPrinter**(filename)

Returns a PDF printer that can be used in print calls. The PDF printer that generates a PDF into the specified file is returned.

Parameters

{String} filename – the file name

Returns

java.awt.print.PrinterJob

Sample

```
//to print current record without printdialog to pdf file in temp dir.
controller.print(true,false,plugins.pdf_output.getPDFPrinter('c:/temp/out.pdf'));
```

getPagesPrinted

Number **getPagesPrinted**()

Returns the number of pages printed by the last print call done in the context of a meta print job.

Returns

Number

Sample

```
//to print multiple forms to one pdf document (on file system).
var success = plugins.pdf_output.startMetaPrintJob('c:/temp/out.pdf')
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
plugins.pdf_output.endMetaPrintJob()

//to print multiple forms to one pdf document (to store in dataprovider).
var success = plugins.pdf_output.startMetaPrintJob()
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
mediaDataProvider = plugins.pdf_output.endMetaPrintJob()
```

getTotalPagesPrinted

Number **getTotalPagesPrinted**()

Returns the total number of pages printed in the context of a meta print job. Call this method before ending the meta print job.

Returns

Number

Sample

```
//to print multiple forms to one pdf document (on file system).
var success = plugins.pdf_output.startMetaPrintJob('c:/temp/out.pdf')
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
plugins.pdf_output.endMetaPrintJob()

//to print multiple forms to one pdf document (to store in dataprovider).
var success = plugins.pdf_output.startMetaPrintJob()
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
mediaDataProvider = plugins.pdf_output.endMetaPrintJob()
```

insertFontDirectory

Number **insertFontDirectory**(path)

Add a directory that should be searched for fonts. Call this only in the context of an active meta print job.

Parameters

{String} path – the path to use

Returns

Number

Sample

```
//Insert font directories for font embedding.
//You must create an MetaPrintJob before using it.
plugins.pdf_output.insertFontDirectory('c:/Windows/Fonts');
plugins.pdf_output.insertFontDirectory('c:/WinNT/Fonts');
plugins.pdf_output.insertFontDirectory('/Library/Fonts');
```

numberPages

byte[] **numberPages**(data)

Add pages numbers to a PDF

Parameters\\\{byte[]} data – the PDF

Returns

byte[] – the PDF with numbered pages

Sample

```
// Add pages numbers to a PDF
var unNumberedFile = plugins.file.showFileOpenDialog();
if (unNumberedFile) {
	var data = plugins.file.readFile(unNumberedFile);
	pageNumberedPdf = elements.customer_id.numberPages(data, 12, 520, 30, 'Courier', '#ff0033');
}
```

numberPages

byte[] **numberPages**(data, fontSize, locationX, locationY, font, hexColor)

Add pages numbers to a PDF

Parameters\\\{byte[]} data – the PDF

{Number} fontSize – the font size to use

{Number} locationX – the x location of the numbers

{Number} locationY – the y location of the numbers

{String} font – the font to use

{String} hexColor – the font color to use

Returns
byte[] – the PDF with numbered pages
Sample

```
// Add pages numbers to a PDF
var unNumberedFile = plugins.file.showFileOpenDialog();
if (unNumberedFile) {
        var data = plugins.file.readFile(unNumberedFile);
        pageNumberedPdf = elements.customer_id.numberPages(data, 12, 520, 30, 'Courier', '#ff0033');
}
```

overlay

byte[] **overlay**(data, forOverlay)
Add some PDF based content over a PDF
Parameters\\\{byte[]} data – the PDF
{byte[]} forOverlay – a PDF to use as overlay
Returns
byte[] – the PDF with added overlay
Sample

```
// Add some PDF based content over a PDF
var pages = new Array();
pages[0] = '1';
pages[1] = '3';
pages[2] = '5';
var input1 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select source file');
if (input1) {
        var data = plugins.file.readFile(input1);
        var input2 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select file for overlay');
        if (input2) {
                var data2 = plugins.file.readFile(input2);
                overlayedPdf = elements.customer_id.overlay( data, data2, false, pages );
                //overlayedPdf = elements.customer_id.overlay( data, data2 );
                //overlayedPdf = elements.customer_id.overlay( data, data2, false, null );
                //overlayedPdf = elements.customer_id.overlay( data, data2, pages );
        }
}
```

overlay

byte[] **overlay**(data, forOverlay, isOver)
Add some PDF based content over a PDF
Parameters\\\{byte[]} data – the PDF
{byte[]} forOverlay – a PDF to use as overlay
{Boolean} isOver – whether the overlay will be put over the content
Returns
byte[] – the PDF with added overlay

Sample

```
// Add some PDF based content over a PDF
var pages = new Array();
pages[0] = '1';
pages[1] = '3';
pages[2] = '5';
var input1 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select source file');
if (input1) {
        var data = plugins.file.readFile(input1);
        var input2 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select file for overlay');
        if (input2) {
                var data2 = plugins.file.readFile(input2);
                overlayedPdf = elements.customer_id.overlay( data, data2, false, pages );
                //overlayedPdf = elements.customer_id.overlay( data, data2 );
                //overlayedPdf = elements.customer_id.overlay( data, data2, false, null );
                //overlayedPdf = elements.customer_id.overlay( data, data2, pages );
        }
}
```

overlay

byte[] **overlay**(data, forOverlay, isOver, pages)

Add some PDF based content over a PDF

Parameters\\\{byte[]} data – the PDF
{byte[]} forOverlay – a PDF to use as overlay
{Boolean} isOver – whether the overlay will be put over the content
{String[]} pages – an array of page numbers to put the overlay on

Returns

byte[] – the PDF with added overlay

Sample

```
// Add some PDF based content over a PDF
var pages = new Array();
pages[0] = '1';
pages[1] = '3';
pages[2] = '5';
var input1 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select source file');
if (input1) {
        var data = plugins.file.readFile(input1);
        var input2 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select file for overlay');
        if (input2) {
                var data2 = plugins.file.readFile(input2);
                overlayedPdf = elements.customer_id.overlay( data, data2, false, pages );
                //overlayedPdf = elements.customer_id.overlay( data, data2 );
                //overlayedPdf = elements.customer_id.overlay( data, data2, false, null );
                //overlayedPdf = elements.customer_id.overlay( data, data2, pages );
        }
}
```

overlay

byte[] **overlay**(data, forOverlay, pages)

Add some PDF based content over a PDF

Parameters\\\{byte[]} data – the PDF
{byte[]} forOverlay – a PDF to use as overlay
{String[]} pages – an array of page numbers to put the overlay on

Returns

byte[] – the PDF with added overlay

Sample

```
// Add some PDF based content over a PDF
var pages = new Array();
pages[0] = '1';
pages[1] = '3';
pages[2] = '5';
var input1 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select source file');
if (input1) {
        var data = plugins.file.readFile(input1);
        var input2 = plugins.file.showFileOpenDialog(1,null,false,'pdf',null,'Select file for overlay');
        if (input2) {
                var data2 = plugins.file.readFile(input2);
                overlayedPdf = elements.customer_id.overlay( data, data2, false, pages );
                //overlayedPdf = elements.customer_id.overlay( data, data2 );
                //overlayedPdf = elements.customer_id.overlay( data, data2, false, null );
                //overlayedPdf = elements.customer_id.overlay( data, data2, pages );
        }
}
```

overlayText

byte[] **overlayText**(data, text)

Add text over every page at a 45 degree angle

Parameters\\\{byte[]} data – the PDF
{String} text – the text to use for the overlay

Returns
byte[] – the PDF with added overlay

Sample

```
// Add text over every page at a 45 degree angle\m
var pdf = plugins.file.showFileOpenDialog();
if (pdf) {
        var data = plugins.file.readFile(pdf);
        modifiedPdf = elements.customer_id.overlayText(data, 'DRAFT', 230, 430, true, 32, 'Helvetica',
'#33ff33');
}
```

overlayText

byte[] **overlayText**(data, text, locationX, locationY, isOver, fontSize, font, hexColor)

Add text over every page at a 45 degree angle

Parameters\\\{byte[]} data – the PDF
{String} text – the text to use for the overlay
{Number} locationX – the x location of the overlay
{Number} locationY – the y location of the overlay
{Boolean} isOver – whether to put the overlay over the content
{Number} fontSize – the font size to use
{String} font – the font to use
{String} hexColor – the font color to use

Returns
byte[] – the PDF with added overlay

Sample

```
// Add text over every page at a 45 degree angle\m
var pdf = plugins.file.showFileOpenDialog();
if (pdf) {
        var data = plugins.file.readFile(pdf);
        modifiedPdf = elements.customer_id.overlayText(data, 'DRAFT', 230, 430, true, 32, 'Helvetica',
'#33ff33');
}
```

startMetaPrintJob

Boolean **startMetaPrintJob**()

Used for printing multiple things into the same PDF document. Starts a meta print job and all print calls made before ending the meta print job will be done into the same PDF document. The PDF document is stored in memory and can be retrieved when ending the meta print job and can be saved, for example, into a dataprovider.

Returns
Boolean
Sample

```
//to print multiple forms to one pdf document (on file system).
var success = plugins.pdf_output.startMetaPrintJob('c:/temp/out.pdf')
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
plugins.pdf_output.endMetaPrintJob()

//to print multiple forms to one pdf document (to store in dataprovider).
var success = plugins.pdf_output.startMetaPrintJob()
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
mediaDataProvider = plugins.pdf_output.endMetaPrintJob()
```

startMetaPrintJob

Boolean **startMetaPrintJob**(filename)
Used for printing multiple things into the same PDF document. Starts a meta print job and all print calls made before ending the meta print job will be done into the same PDF document. The PDF document is generated in a File specified by the filename.
Parameters
{String} filename – the file name
Returns
Boolean
Sample

```
//to print multiple forms to one pdf document (on file system).
var success = plugins.pdf_output.startMetaPrintJob('c:/temp/out.pdf')
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
plugins.pdf_output.endMetaPrintJob()

//to print multiple forms to one pdf document (to store in dataprovider).
var success = plugins.pdf_output.startMetaPrintJob()
if (success)
{
	forms.form_one.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form one printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
	forms.form_two.controller.print(false,false,plugins.pdf_output.getPDFPrinter());
	application.output('form two printed ' + plugins.pdf_output.getPagesPrinted() + ' pages.');
}
application.output('total printed pages: ' + plugins.pdf_output.getTotalPagesPrinted());
mediaDataProvider = plugins.pdf_output.endMetaPrintJob()
```

watermark

byte[] **watermark**(data, image)
Add an image as a watermark on every page, or the pages specified as a parameter
Parameters\\\{byte[]} data – the PDF
{String} image – the path of an image to use

Returns
byte[] – the PDF with added watermak
Sample

```
// Add an image as a watermark on every page, or the pages specified as a parameter.
var pdf = plugins.file.showFileOpenDialog();
if (pdf) {
        var data = plugins.file.readFile(pdf);
        var image = plugins.file.showFileOpenDialog();
        modifiedPdf = elements.customer_id.watermark(data, image);
}
```

watermark

byte[] **watermark**(data, image, locationX, locationY, isOver)

Add an image as a watermark on every page, or the pages specified as a parameter

Parameters\\\{byte[]} data – the PDF
{String} image – the path of an image to use
{Number} locationX – the x location of the image
{Number} locationY – the y location of the image
{Boolean} isOver – whether to put over the content
Returns
byte[] – the PDF with added watermak
Sample

```
// Add an image as a watermark on every page, or the pages specified as a parameter.
var pdf = plugins.file.showFileOpenDialog();
if (pdf) {
        var data = plugins.file.readFile(pdf);
        var image = plugins.file.showFileOpenDialog();
        modifiedPdf = elements.customer_id.watermark(data, image);
}
```

watermark

byte[] **watermark**(data, image, locationX, locationY, isOver, pages)

Add an image as a watermark on every page, or the pages specified as a parameter

Parameters\\\{byte[]} data – the PDF
{String} image – the path of an image to use
{Number} locationX – the x location of the image
{Number} locationY – the y location of the image
{Boolean} isOver – whether to put over the content
{String[]} pages – an array of pages where to apply the watermark
Returns
byte[] – the PDF with added watermak
Sample

```
// Add an image as a watermark on every page, or the pages specified as a parameter.
var pdf = plugins.file.showFileOpenDialog();
if (pdf) {
        var data = plugins.file.readFile(pdf);
        var image = plugins.file.showFileOpenDialog();
        modifiedPdf = elements.customer_id.watermark(data, image);
}
```