pdf-FieldMerge JavaScript Referenz

JavaScript API zur Einbindung eigener Abfragen

Folgende Methoden können genutzt werden:

appl.getFieldCount();	gibt die Anzahl der Felder im Pdf Dokument zurück
appl.getValue(int row);	gibt den Feldinhalt zurück
appl.getDataValue(int col);	gibt den Wert einer Zelle aus der Datentabelle zurück
appl.getTypeValue(int row);	gibt den Feldtyp zurück
appl.setValue(Object value, int row);	setzt den Feldinhalt
appl.selectRow(int row);	markiert die Zeile in der Tabelle des Hauptfensters
appl.getName(int row);	gibt den Feldnamen zurück
appl.getRow(String name);	gibt die Zeile mit dem Feldnamen zurück
appl.getCheckValueX(int row);	gibt zurück, ob Schreibgeschützt selektiert ist
appl.getCheckValueR(int row);	gibt zurück, ob Erforderlich selektiert ist
appl.getCheckValueF(int row);	gibt zurück, ob Flatten selektiert ist
appl.getCheckValueH(int row);	gibt zurück, ob Unsichtbar selektiert ist
appl.getCheckValueI(int row);	gibt zurück, ob Image Flag selektiert ist
appl.setCheckValueX(boolean check, int row);	setzt Feld auf Schreibgeschützt
appl.setCheckValueR(boolean check, int row);	setzt Feld auf Erforderlich
appl.setCheckValueF(boolean check, int row);	setzt Feld auf Flatten
appl.setCheckValueH(boolean check, int row);	setzt Feld auf Unsichtbar
appl.setCheckValueI(boolean check, int row);	setzt Feld auf Image Flag in Buttons
appl.setToolTip(String name, String text);	setzt ToolTip für ein Feld
appl.setBackColor(String name, String color);	setzt Feldfarbe (null = keine Farbe)
appl.setBorderColor(String name, String color);	setzt Feldrahmenfarbe (null = kein Rahmen)
appl.setTextColor(String name, String color);	setzt Textfarbe
appl.setTextSize(String name, int size);	setzt Textgröße (0 = Auto)
appl.setColumnAutoSize(String name, String col, int size);	setzt Textgröße (0 = Auto) (enthält *compare_ref*())
appl.compare_ref(String field_name, String col_name);	gibt true zurück, wenn für den Feldnamen eine Spalten-Referenz vorhanden ist.
appl.printd(String format, Date date);	Formatiert ein Datum (JavaScript Date Objekt)
appl.printds(String format, String pattern, String value);	Formatiert vorhandenen Datumstring.

Parameter:
format : vorhandenes Datumsformat, z.B. "dd.mm.yyyy" für "01.01.1999"
pattern : gewünschtes Format, z.B. "dd. mmmm yyyy" für "01. Januar 1999"
value : vorhandener Datumstring eines Feldes

```
appl.createTextField(String doc, int page, String name, [String text,] int x, int y, int w, int h, boolean multi,
                     int align, String script);
```

Erzeugt ein neues Textfeld
Parameter:
doc : Datei-Name des Templates. Die Funktion wird nur ausgeführt, wenn ein Template mit diesem Namen gerade "dran" ist.
page : >0: dieses Feld auf Seite page einfügen
=0: dieses Feld auf jeder Seite einfügen. Felder haben gleichen Namen.
-1: dieses Feld auf jeder Seite einfügen. Feldname wird erweitert um -p[Seite im Template]
-2: dieses Feld auf jeder Seite einfügen. Für dynamische Tabellen-Templates gleichen Namens. Feldname wird erweitert um -p[aktuelle Seitenzahl im Ausgabe-Pdf + Seite im Template]
name : Feldname
[text] : Feldinhalt (optional)
x : x-Koordinate des Feldes (in Point)
y : y-Koordinate des Feldes (in Point). Gemessen vom unteren Seitenrand bis Unterkante Feld
w : Breite des Feldes (in Point).
h : Höhe des Feldes (in Point).
multi : Feld ist mehrzeiliges Textfeld.
align : Text Ausrichtung. 0 = linksbündig, 1 = zentriert, 2 = rechtsbündig.
script : Kalkulations JavaScript oder null

appl.createPageFields(String doc, String name, String text, int x, int y, int w, int h, int align);
Erzeugt ein neues Textfeld speziell für Seitennummerierung. Kann auch mehrfach pro Seite gesetzt werden.
Parameter:
doc : Datei-Name des Templates. Die Funktion wird nur ausgeführt, wenn ein Template mit diesem Namen gerade "dran" ist.
name : Feldname. Wird automatisch für jede Seite umbenannt, damit keine "Durchschlagsfelder" entstehen. Bei mehrfach-Plazierung auf der Seite muss sich der Name von weiteren Seitenfeldern unterscheiden.
text : Text für die Nummerierung. Beispiel: "Seite %1 von %2" oder "Seite %1". %1 wird durch die aktuelle Seitennummer ersetzt und %2 (optional) durch die Anzahl aller Seiten.
x : x-Koordinate des Feldes (in Point)
y : y-Koordinate des Feldes (in Point). Gemessen vom unteren Seitenrand bis Unterkante Feld
w : Breite des Feldes (in Point).
h : Höhe des Feldes (in Point).
align : Text Ausrichtung. 0 = linksbündig, 1 = zentriert, 2 = rechtsbündig.

appl.addContent(String doc, int page, String content, int x, int y, int align, String font, int style, int size, String color);
Funktion fügt eine Textzeile in das Dokument ein.
Parameter:
doc : Name des Templates
page : Seite im Template
content : Text, der eingefügt werden soll
x : x-Koordinate des Textes (in Point)
y : y-Koordinate des Textes (in Point). Gemessen vom unteren Seitenrand bis Unterkante Text
align : Text Ausrichtung. 0 = linksbündig, 1 = zentriert, 2 = rechtsbündig.
font : Schriftart
style : Schriftstil (0 = normal, 1 = fett, 2 = kursiv, 3 = fett-kursiv)
size : Schriftgröße
color : Schriftfarbe

appl.addImage(String doc, int page, String image, int x, int y, float scale);
Funktion fügt ein Bild in das Dokument ein.
doc : Name des Templates
page : Seite im Template
image : Name der Image Datei
x : x-Koordinate des Textes (in Point)
y : y-Koordinate des Textes (in Point). Gemessen vom unteren Seitenrand bis Unterkante Feld
scale : Skalierung (Standard = 1.0)

setFormat(String doc, String name, int type, int sep, int neg, int dez, String symbol, boolean prep)
Diese Funktion formatiert einen String als Zahlenformat.
Parameter:
doc : Name des Templates
name : Name der Datenspalte
type : 1 = Zahl. Andere Formate wie Datum oder Zeit können mit appl.printd() oder appl.printds() erzeugt werden.
sep : Trennzeichen
0 = 1,234.56
1 = 1234.56
2 = 1.234,56
3 = 1234,56
neg : Negativ-Darstellung (nur für Zahlen)
0 = -1.234,56
1 = 1.234,56 (Rot)
2 = (1.234,56)
3 = (1.234,56) (Rot)
dez : Anzahl der Nachkommastellen (0 - 10)
symbol : Symbol-Typen für Währungsdarstellung (null = kein Symbol)
prep : Symbol vor Zahl (true) oder dahinter (false)

addBarcode(String doc, int page, int type, int add, boolean txtUp, boolean guards, boolean chkSum,
boolean chkSTxt, String txt1, String txt2, int x, int y, String Bcolor, String Tcolor, int align, int rot)

Funktion fügt einen Barcode in das Dokument ein.
Parameter:

doc	: Name des Templates
page	: Seite im Template
type	: Barcode Typ
add	: Typ des Zusatzcodes (optional, Text steht dann in txt2), Standard = 0
txtUp	: (optional) Text über dem Code = true, darunter = false (Standard)
guards	: (optional) Begrenzungen nach unten verlängern = true (Standard) sonst false. Nur für EAN Codes.
chkSum	: (optional) Prüfsumme berechnen = true sonst false (Standard)
chkSTxt	: (optional) Prüfsumme im Text mit anzeigen = true sonst false (Standard)
txt1	: Barcode als Text
txt2	: (optional) Zusatzcode wenn add >0 oder Anzeigetext für CODE-128-R
x	: x-Koordinate des Barcodes (in Point)
y	: y-Koordinate des Barcodes (in Point). Gemessen vom unteren Seitenrand bis Unterkante Barcode
Bcolor	: (optional) Farbe des Barcodes. Standard = schwarz
Tcolor	: (optional) Textfarbe. Standard = schwarz
align	: (optional) Ausrichtung (0 = linksbündig, 1 = zentriert, 2 = rechtsbündig)
rot	: (optional) Barcode drehen (zulässige Werte: 0 (Standard), 90, 180, 270). Nur in Verbindung mit align 1.

Barcode Typen:

1. EAN-8
2. EAN-13
3. UPC-A
4. UPC-E
5. ADD-2 (Zusatzcode 2-stellig)
6. ADD-5 (Zusatzcode 5-stellig)
7. POSTNET
8. PLANET
9. CODE-128
10. CODE-128-R (text1 = Barcode, text2 = Anzeigetext. Erweiteter Zeichensatz)
11. CODABAR
12. CODE-39
13. CODE-39-EXT
14. ITF (Interleave 2/5)
15. DATAMATRIX
16. QRCODE

Zulässige minimale Funktionsaufrufe bei Nutzung der Standardwerte:

addBarcode(String doc, int page, int type, String txt1, int x, int y);
addBarcode(String doc, int page, int type, String txt1, int x, int y, int align);
addBarcode(String doc, int page, int type, String txt1, int x, int y, int align, int rot);
addBarcode(String doc, int page, int type, String txt1, int x, int y, String Bcolor, String Tcolor, int align);
addBarcode(String doc, int page, int type, String txt1, int x, int y, String Bcolor, String Tcolor, int align, int rot);
addBarcode(String doc, int page, int type, int add, String txt1, String txt2, int x, int y, int align);
addBarcode(String doc, int page, int type, int add, String txt1, String txt2, int x, int y, int align, int rot);

Farben definieren:

Farben können entweder mit Standards angegeben werden oder als RGB String (r,g,b)
Standardfarben sind:
"white", "lightGray", "gray", "darkGray", "black", "red", "pink",
"orange", "yellow", "green", "magenta", "cyan", "blue"

Funktionen gemäß Acrobat JavaScript util.print...

Folgende Methode kann genutzt werden:

appl.printfa(String cFormat, Object arguments);

Formatiert ein oder mehrere Argumente als Text. Diese Methode konvertiert und formatiert eingehende Argumente in einen Ergebnis-String gemäß der Parameter im Format-String (format).
Der Format-String besteht aus zwei Arten von Objekten:

- Gewöhnliche Zeichen, die in den Ergebnis-String kopiert werden.
- Umwandlung Spezifikationen, die jeweils für Konvertierung und Formatierung des nächstfolgenden Arguments in printfa sorgen.

Jede Umwandlung Spezifikation ist wie folgt aufgebaut:

%[,nDecSep][cFlags][nWidth][.nPrecision]cConvChar

Parameter:

cFormat	Der zu verwendende Format-String.
arguments	Das/die optionale(n) Argument(e), welche die Daten enthalten, die an Stelle der angegebenen %-Tags eingefügt werden soll(en). Die Anzahl der optionalen Argumente muß gleich der Anzahl der %-Tags sein.

Die folgende Tabelle beschreibt die einzelnen Komponenten einer Umwandlungs Spezifikation.

nDecSep Ein Komma, gefolgt von einer Ziffer um das Dezimal-/Tausendertrennzeichen zu bestimmen:
- **0** = Komma getrennt, Punkt als Dezimal Zeichen
- **1** = Kein Trennzeichen, Punkt als Dezimal Zeichen
- **2** = Punkt getrennt, Komma als Dezimal Zeichen
- **3** = Kein Trennzeichen, Komma als Dezimal Zeichen

cFlags Gilt nur für numerische Konvertierungen und besteht aus einer Anzahl von Zeichen (in beliebiger Reihenfolge) welche die Spezifikation ändern:
- **+** = Eine Zahl wird immer mit Vorzeichen dargestellt.
- **space** = Ist das erste Zeichen kein Vorzeichen, wird ein Leerzeichen vorangestellt
- **0** = Füllung mit vorangestellten Nullen.
- **#** = Alternative Ausgabe Form. Für f, hat die Ausgabe immer einen Dezimal Punkt.

nWidth Minimale Feldbreite. Das umgewandelte Argument formatiert mindestens diese Anzahl Zeichen, einschließlich der Vorzeichen und Dezimalpunkt, und kann größer sein, wenn nötig. Wenn das konvertierte Argument weniger Zeichen hat als die Feldbreite, wird es auf der linken Seite um die Feldbreite aufgefüllt. Die Füllung ist im allgemeinen ein Leerzeichen, aber 0 wenn der Null-Auffüllung Schalter vorhanden ist (cFlags enthält 0).

nPrecision Ein Punkt (.) gefolgt von einer Zahl welche die Anzahl der Nachkommastellen angibt.

cConvChar Gibt an, wie das Argument zu interpretieren ist:
- **d** = Ganzzahl (Abschneiden bei Bedarf)
- **f** = Dezimal Zahl
- **s** = String (Text)
- **x** = Ganzzahl (Abschneiden bei Bedarf) und in vorzeichenlose hexadezimale Ausgabe formatiert.

Diese Funktion ist identisch mit appl.printf(String cFormat, Object argument); die allerdings nur einen Ersetzungswert **argument** übernimmt.