UNIMINUTO
Corporación Universitaria Minuto de Dios

**Tema**

Explicación código formulario

**Presenta**

Jhon Gabriel Silva Tibaduiza

**Docente**

ALONSO GUEVARA PEREZ

**Asignatura**

Programación web
NRC: 7478

Colombia, Bogotá Abril 28 de 2020.

# Explicación código formulario

## Index

Creación de los campos donde las personas van a poner su información, el diseño de este formulario fue implementado con css, y la manera de descargar el formulario por medio de pdf fue por medio de js.

```
<!DOCTYPE html>

<html>

<head>

        <title>Formulario parcial</title>

        <script src="js/jspdf.js"></script>

        <script src="js/jquery-2.1.3.js"></script>

        <script src="js/pdfFromHTML.js"></script>

        <script type="js/foto.js"></script>

        <link rel="stylesheet" href="css/style.css">

</head>

<body>

        <div id="HTMLtoPDF">

        <form id="pago">

                <!--Creacion de las tres capas que conforman

                        el formulario deseado empezando por los

                        datos de la persona

                -->

                <fieldset>

                <legend>Datos de la persona</legend>

                <ol>

                <li>

                        <label for="nombre">Nombre</label>
```

```
            <input id="nombre" class="minusculas" name="nombre" type="text" placeholder="Escribe tu nombre completo" required autofocus>
        </li>
        <li>
            <label for="apellido">Apellidos</label>
            <input id="apellido" class="minusculas" name="nombre" type="text" placeholder="Escribe tu apellido completo" required autofocus>
        </li>
        <li>
            <label for="fecha">FechaNacimiento</label>
            <input id="fecha" name="FechaNacimiento" type="date" name="fecha">
        </li>
        <li>
            <label for="email">Email</label>
            <input id="email" class="minusculas" name="email" type="email" placeholder="ejemplo@um.es" required>
        </li>
        <li>
        <label for="telefono">Teléfono</label>
        <input id="telefono" name="telefono" type="tel" placeholder="Ej. 4537896" required >
        </li>
        <li>
        <label for="Edad">Edad</label>
        <input id="Edad" name="Edad" type="text" readonly="readonly">
        </li>
        </ol>
        </fieldset>
        <!--Creacion de la segunda parte del formulario en el cual se
            piden los datos adiccionales de la persona
        -->
        <fieldset>
        <legend>Datos adicionales</legend>
        <ol>
        <li>
        <label for="so">Genero</label>
        <select>
            <option value=""selected="selected">- selecciona -</option>
            <option value="Masculino">Masculino</option>
            <option value="Femenino">Femenino</option>
            <option value="Otro">Otro</option>
        </select>
        </li>
        <li>
        <label for="Direccion">Direccion</label>
        <input id="Direccion" name="Direccion" type="text"
placeholder="Direccion" required>
        </li>
        </ol>
        </fieldset>
        ------------------------------------------------
        <fieldset>
            <legend>Foto de la persona</legend>
            <form method="post" action="upload.php"
enctype="multipart/form-data"
            id="uploadForm">
```

```
             <input type="file" name="file" id="file" />

        </fieldset>

        <!--Creacion de la parte final donde se crean dos botones:

            -Boton validacion que valida que los datos sean correctos

            -Boton descargar formulario en modo pdf

        -->

        -----------------------------------------------

        <fieldset>

            <legend>Botones de validacion y descarga</legend>

        <button type="submit">Validar informacion</button>

        <button type="submit" href="#" onclick="HTMLtoPDF()">Guardar formulario como PDF</button>

        </fieldset>

        </form>

    </form>

</div>

</body>

</html>
```

**Css**

```
form#pago {

            margin:auto;

            background: Green;

            color: white;

            font-size: 17px;

            padding: 30px;

            width: 470px;

            height: 500px;
```

```
border:solid 10px LightSteelBlue;
border-radius: 5px;
-webkit-border-radius: 5px;
-moz-border-radius: 5px;
}

form#pago fieldset {
border: none;
}

form#pago select{
font-size:12px;
background: yellow;
border:solid 1px Green;
border-radius: 3px;
-webkit-border-radius: 3px;
-moz-border-radius: 3px;
float: right;
margin-right: 15px;
width: 200px;
}

form#pago input{
font-size:12px;
background: yellow;
border:solid 1px Green;
border-radius: 3px;
```

```
            -webkit-border-radius: 3px;

            -moz-border-radius: 3px;

            float: right;

            margin-right: 15px;

            width: 200px;

            }


            form#pago ol li {

            line-height: 25px;

            list-style: none;

            }


            form#pago [required]{

            border:solid 1px red;

            }


            input:required {

            outline: 1px solid red;

            }


            .minusculas{

                text-transform:lowercase;

            }
```

**Js**

```
var jsPDF = (function(global) {

    'use strict';
```

```
var pdfVersion = '1.3',
        pageFormats = { // Size in pt of various paper formats
                'a0'  : [2383.94, 3370.39], 'a1'  : [1683.78, 2383.94],
                'a2'  : [1190.55, 1683.78], 'a3'  : [ 841.89, 1190.55],
                'a4'  : [ 595.28,  841.89], 'a5'  : [ 419.53,  595.28],
                'a6'  : [ 297.64,  419.53], 'a7'  : [ 209.76,  297.64],
                'a8'  : [ 147.40,  209.76], 'a9'  : [ 104.88,  147.40],
                'a10' : [  73.70,  104.88], 'b0'  : [2834.65, 4008.19],
                'b1'  : [2004.09, 2834.65], 'b2'  : [1417.32, 2004.09],
                'b3'  : [1000.63, 1417.32], 'b4'  : [ 708.66, 1000.63],
                'b5'  : [ 498.90,  708.66], 'b6'  : [ 354.33,  498.90],
                'b7'  : [ 249.45,  354.33], 'b8'  : [ 175.75,  249.45],
                'b9'  : [ 124.72,  175.75], 'b10' : [  87.87,  124.72],
                'c0'  : [2599.37, 3676.54], 'c1'  : [1836.85, 2599.37],
                'c2'  : [1298.27, 1836.85], 'c3'  : [ 918.43, 1298.27],
                'c4'  : [ 649.13,  918.43], 'c5'  : [ 459.21,  649.13],
                'c6'  : [ 323.15,  459.21], 'c7'  : [ 229.61,  323.15],
                'c8'  : [ 161.57,  229.61], 'c9'  : [ 113.39,  161.57],
                'c10' : [  79.37,  113.39], 'dl'  : [ 311.81,  623.62],
                'letter'            : [612,   792],
                'government-letter' : [576,   756],
                'legal'             : [612,  1008],
                'junior-legal'      : [576,   360],
                'ledger'            : [1224,  792],
                'tabloid'           : [792,  1224],
                'credit-card'       : [153,   243]
        };
```

```
function PubSub(context) {
    var topics = {};

    this.subscribe = function(topic, callback, once) {
        if(typeof callback !== 'function') {
            return false;
        }

        if(!topics.hasOwnProperty(topic)) {
            topics[topic] = {};
        }

        var id = Math.random().toString(35);
        topics[topic][id] = [callback,!!once];

        return id;
    };

    this.unsubscribe = function(token) {
        for(var topic in topics) {
            if(topics[topic][token]) {
                delete topics[topic][token];
                return true;
            }
        }
```

```
            return false;
        };

        this.publish = function(topic) {
            if(topics.hasOwnProperty(topic)) {
                var args = Array.prototype.slice.call(arguments, 1), idr = [];

                for(var id in topics[topic]) {
                    var sub = topics[topic][id];
                    try {
                        sub[0].apply(context, args);
                    } catch(ex) {
                        if(global.console) {
                            console.error('jsPDF PubSub Error',
ex.message, ex);
                        }
                    }
                    if(sub[1]) idr.push(id);
                }
                if(idr.length) idr.forEach(this.unsubscribe);
            }
        };
    }

    /**
     * @constructor
     * @private
     */
    function jsPDF(orientation, unit, format, compressPdf) {
        var options = {};

        if (typeof orientation === 'object') {
            options = orientation;

            orientation = options.orientation;
            unit = options.unit || unit;
            format = options.format || format;
            compressPdf = options.compress || options.compressPdf || compressPdf;
        }

        // Default options
        unit       = unit || 'mm';
        format     = format || 'a4';
        orientation = ('' + (orientation || 'P')).toLowerCase();

        var format_as_string = ('' + format).toLowerCase(),
            compress = !!compressPdf && typeof Uint8Array === 'function',
            textColor          = options.textColor  || '0 g',
            drawColor          = options.drawColor  || '0 G',
            activeFontSize      = options.fontSize   || 16,
            lineHeightProportion = options.lineHeight || 1.15,
            lineWidth          = options.lineWidth  || 0.200025, // 2mm
            objectNumber =  2,  // 'n' Current object number
            outToPages   = !1,  // switches where out() prints. outToPages true = push to pages obj. outToPages false = doc builder content
```

```
                offsets      = [],  // List of offsets. Activated and reset by
buildDocument(). Pupulated by various calls buildDocument makes.
                fonts       = {},  // collection of font objects, where key is fontKey - a
dynamically created label for a given font.
                fontmap      = {},  // mapping structure fontName > fontStyle > font
key - performance layer. See addFont()
                activeFontKey,      // will be string representing the KEY of the font as
combination of fontName + fontStyle
                k,                  // Scale factor
                tmp,
                page = 0,
                currentPage,
                pages = [],
                pagedim = {},
                content = [],
                lineCapID = 0,
                lineJoinID = 0,
                content_length = 0,
                pageWidth,
                pageHeight,
                pageMode,
                zoomMode,
                layoutMode,
                documentProperties = {
                    'title'    : '',
                    'subject'  : '',
                    'author'   : '',
                    'keywords' : '',
                    'creator'  : ''
```

```
    },
    API = {},
    events = new PubSub(API),

/////////////////////
// Private functions
/////////////////////
f2 = function(number) {
    return number.toFixed(2); // Ie, %.2f
},
f3 = function(number) {
    return number.toFixed(3); // Ie, %.3f
},
padd2 = function(number) {
    return ('0' + parseInt(number)).slice(-2);
},
out = function(string) {
    if (outToPages) {
        /* set by beginPage */
        pages[currentPage].push(string);
    } else {
        // +1 for '\n' that will be used to join 'content'
        content_length += string.length + 1;
        content.push(string);
    }
},
newObject = function() {
```

```
            // Begin a new object
            objectNumber++;
            offsets[objectNumber] = content_length;
            out(objectNumber + ' 0 obj');
            return objectNumber;
        },
        putStream = function(str) {
            out('stream');
            out(str);
            out('endstream');
        },
        putPages = function() {
            var n,p,arr,i,deflater,adler32,adler32cs,wPt,hPt;

            adler32cs = global.adler32cs || jsPDF.adler32cs;
            if (compress && typeof adler32cs === 'undefined') {
                compress = false;
            }

            // outToPages = false as set in endDocument(). out() writes to
content.

            for (n = 1; n <= page; n++) {
                newObject();
                wPt = (pageWidth = pagedim[n].width) * k;
                hPt = (pageHeight = pagedim[n].height) * k;
                out('<</Type /Page');
                out('/Parent 1 0 R');
                out('/Resources 2 0 R');
                out('/MediaBox [0 0 ' + f2(wPt) + ' ' + f2(hPt) + ']');
                out('/Contents ' + (objectNumber + 1) + ' 0 R>>');
                out('endobj');

                // Page content
                p = pages[n].join('\n');
                newObject();
                if (compress) {
                    arr = [];
                    i = p.length;
                    while(i--) {
                        arr[i] = p.charCodeAt(i);
                    }
                    adler32 = adler32cs.from(p);
                    deflater = new Deflater(6);
                    deflater.append(new Uint8Array(arr));
                    p = deflater.flush();
                    arr = new Uint8Array(p.length + 6);
                    arr.set(new Uint8Array([120, 156])),
                    arr.set(p, 2);
                    arr.set(new Uint8Array([adler32 & 0xFF, (adler32 >> 8) & 0xFF, (adler32 >> 16) & 0xFF, (adler32 >> 24) & 0xFF]), p.length+2);
                    p = String.fromCharCode.apply(null, arr);
                    out('<</Length ' + p.length + ' /Filter [/FlateDecode]>>');
                } else {
                    out('<</Length ' + p.length + '>>');
```

```
        }
        putStream(p);
        out('endobj');
    }
    offsets[1] = content_length;
    out('1 0 obj');
    out('<</Type /Pages');
    var kids = '/Kids [';
    for (i = 0; i < page; i++) {
        kids += (3 + 2 * i) + ' 0 R ';
    }
    out(kids + ']');
    out('/Count ' + page);
    out('>>');
    out('endobj');
},
putFont = function(font) {
    font.objectNumber = newObject();
    out('<</BaseFont/' + font.PostScriptName + '/Type/Font');
    if (typeof font.encoding === 'string') {
        out('/Encoding/' + font.encoding);
    }
    out('/Subtype/Type1>>');
    out('endobj');
},
putFonts = function() {
    for (var fontKey in fonts) {
```

```
                if (fonts.hasOwnProperty(fontKey)) {
                    putFont(fonts[fontKey]);
                }
            }
        },
        putXobjectDict = function() {
            // Loop through images, or other data objects
            events.publish('putXobjectDict');
        },
        putResourceDictionary = function() {
            out('/ProcSet [/PDF /Text /ImageB /ImageC /ImageI]');
            out('/Font <<');

            // Do this for each font, the '1' bit is the index of the font
            for (var fontKey in fonts) {
                if (fonts.hasOwnProperty(fontKey)) {
                    out('/' + fontKey + ' ' + fonts[fontKey].objectNumber +
' 0 R');
                }
            }
            out('>>');
            out('/XObject <<');
            putXobjectDict();
            out('>>');
        },
        putResources = function() {
            putFonts();
            events.publish('putResources');
```

```
    // Resource dictionary
    offsets[2] = content_length;
    out('2 0 obj');
    out('<<');
    putResourceDictionary();
    out('>>');
    out('endobj');
    events.publish('postPutResources');
},
addToFontDictionary = function(fontKey, fontName, fontStyle) {
    // this is mapping structure for quick font key lookup.
    // returns the KEY of the font (ex: "F1") for a given
    // pair of font name and type (ex: "Arial". "Italic")
    if (!fontmap.hasOwnProperty(fontName)) {
        fontmap[fontName] = {};
    }
    fontmap[fontName][fontStyle] = fontKey;
},
/**
 * FontObject describes a particular font as member of an instnace of jsPDF
 *
 * It's a collection of properties like 'id' (to be used in PDF stream),
 * 'fontName' (font's family name), 'fontStyle' (font's style variant label)
 *
 * @class
 * @public
 * @property id {String} PDF-document-instance-specific label assinged to the font.
```

```
         * @property PostScriptName {String} PDF specification full name for the
font
         * @property encoding {Object} Encoding_name-to-Font_metrics_object
mapping.
         * @name FontObject
         */
        addFont = function(PostScriptName, fontName, fontStyle, encoding) {
            var fontKey = 'F' + (Object.keys(fonts).length + 1).toString(10),
            // This is FontObject
            font = fonts[fontKey] = {
                'id'             : fontKey,
                'PostScriptName' : PostScriptName,
                'fontName'       : fontName,
                'fontStyle'      : fontStyle,
                'encoding'       : encoding,
                'metadata'       : {}
            };
            addToFontDictionary(fontKey, fontName, fontStyle);
            events.publish('addFont', font);

            return fontKey;
        },
        addFonts = function() {

            var HELVETICA     = "helvetica",
                TIMES         = "times",
                COURIER       = "courier",
                NORMAL        = "normal",
                BOLD          = "bold",
                ITALIC        = "italic",
                BOLD_ITALIC   = "bolditalic",
                encoding      = 'StandardEncoding',
                standardFonts = [
                    ['Helvetica', HELVETICA, NORMAL],
                    ['Helvetica-Bold', HELVETICA, BOLD],
                    ['Helvetica-Oblique', HELVETICA, ITALIC],
                    ['Helvetica-BoldOblique', HELVETICA, BOLD_ITALIC],
                    ['Courier', COURIER, NORMAL],
                    ['Courier-Bold', COURIER, BOLD],
                    ['Courier-Oblique', COURIER, ITALIC],
                    ['Courier-BoldOblique', COURIER, BOLD_ITALIC],
                    ['Times-Roman', TIMES, NORMAL],
                    ['Times-Bold', TIMES, BOLD],
                    ['Times-Italic', TIMES, ITALIC],
                    ['Times-BoldItalic', TIMES, BOLD_ITALIC]
                ];

            for (var i = 0, l = standardFonts.length; i < l; i++) {
                var fontKey = addFont(
                        standardFonts[i][0],
                        standardFonts[i][1],
                        standardFonts[i][2],
                        encoding);


                // adding aliases for standard fonts, this time matching the
capitalization
```

```
                var parts = standardFonts[i][0].split('-');
                addToFontDictionary(fontKey, parts[0], parts[1] || '');
            }
            events.publish('addFonts', { fonts : fonts, dictionary : fontmap });
        },
        SAFE = function __safeCall(fn) {
            fn.foo = function __safeCallWrapper() {
                try {
                    return fn.apply(this, arguments);
                } catch (e) {
                    var stack = e.stack || '';
                    if(~stack.indexOf(' at ')) stack = stack.split(" at ")[1];
                    var m = "Error in function " +
stack.split("\n")[0].split('<')[0] + ": " + e.message;
                    if(global.console) {
                        global.console.error(m, e);
                        if(global.alert) alert(m);
                    } else {
                        throw new Error(m);
                    }
                }
            };
            fn.foo.bar = fn;
            return fn.foo;
        },
        to8bitStream = function(text, flags) {
        /**
         * PDF 1.3 spec:
                 * "For text strings encoded in Unicode, the first two bytes must be 254
followed by
                 * 255, representing the Unicode byte order marker, U+FEFF. (This
sequence conflicts
                 * with the PDFDocEncoding character sequence thorn ydieresis, which is
unlikely
                 * to be a meaningful beginning of a word or phrase.) The remainder of the
                 * string consists of Unicode character codes, according to the UTF-16
encoding
                 * specified in the Unicode standard, version 2.0. Commonly used Unicode
values
                 * are represented as 2 bytes per character, with the high-order byte
appearing first
                 * in the string."
                 *
                 * In other words, if there are chars in a string with char code above 255, we
                 * recode the string to UCS2 BE - string doubles in length and BOM is
prepended.
                 *
                 * HOWEVER!
                 * Actual *content* (body) text (as opposed to strings used in document
properties etc)
                 * does NOT expect BOM. There, it is treated as a literal GID (Glyph ID)
                 *
                 * Because of Adobe's focus on "you subset your fonts!" you are not
supposed to have
                 * a font that maps directly Unicode (UCS2 / UTF16BE) code to font GID, but
you could
                 * fudge it with "Identity-H" encoding and custom CIDtoGID map that
mimics Unicode
```

```
                 * code page. There, however, all characters in the stream are treated as
GIDs,
                 * including BOM, which is the reason we need to skip BOM in content text
(i.e. that
                 * that is tied to a font).
                 *
                 * To signal this "special" PDFEscape / to8bitStream handling mode,
                 * API.text() function sets (unless you overwrite it with manual values
                 * given to API.text(.., flags) )
                 * flags.autoencode = true
                 * flags.noBOM = true
                 *
                 *
=====================================================================
==========
                 * `flags` properties relied upon:
                 *   .sourceEncoding = string with encoding label.
                 *              "Unicode" by default. = encoding of the incoming text.
                 *              pass some non-existing encoding name
                 *              (ex: 'Do not touch my strings! I know what I am doing.')
                 *              to make encoding code skip the encoding step.
                 *   .outputEncoding = Either valid PDF encoding name
                 *              (must be supported by jsPDF font metrics, otherwise no
encoding)
                 *              or a JS object, where key = sourceCharCode, value =
outputCharCode
                 *              missing keys will be treated as: sourceCharCode ===
outputCharCode
                 *   .noBOM
                 *      See comment higher above for explanation for why this is important
```

```
         *   .autoencode
         *       See comment higher above for explanation for why this is important
         */

            var i,l,sourceEncoding,encodingBlock,outputEncoding,newtext,isUnicode,ch,bch;

            flags = flags || {};
            sourceEncoding = flags.sourceEncoding || 'Unicode';
            outputEncoding = flags.outputEncoding;

            // This 'encoding' section relies on font metrics format
            // attached to font objects by, among others,
            // "Willow Systems' standard_font_metrics plugin"
            // see jspdf.plugin.standard_font_metrics.js for format
            // of the font.metadata.encoding Object.
            // It should be something like
            //   .encoding = {'codePages':['WinANSI....'], 'WinANSI...':{code:code, ...}}
            //   .widths = {0:width, code:width, ..., 'fof':divisor}
            //   .kerning = {code:{previous_char_code:shift, ..., 'fof':-divisor},...}
            if ((flags.autoencode || outputEncoding) &&
                fonts[activeFontKey].metadata &&
                fonts[activeFontKey].metadata[sourceEncoding] &&
                fonts[activeFontKey].metadata[sourceEncoding].encoding) {
                encodingBlock = fonts[activeFontKey].metadata[sourceEncoding].encoding;
                // each font has default encoding. Some have it clearly
defined.
                if (!outputEncoding && fonts[activeFontKey].encoding) {
                    outputEncoding = fonts[activeFontKey].encoding;
                }

                // Hmmm, the above did not work? Let's try again, in
different place.
                if (!outputEncoding && encodingBlock.codePages) {
                    outputEncoding = encodingBlock.codePages[0]; //
let's say, first one is the default
                }

                if (typeof outputEncoding === 'string') {
                    outputEncoding = encodingBlock[outputEncoding];
                }
                // we want output encoding to be a JS Object, where
                // key = sourceEncoding's character code and
                // value = outputEncoding's character code.
                if (outputEncoding) {
                    isUnicode = false;
                    newtext = [];
                    for (i = 0, l = text.length; i < l; i++) {
                        ch = outputEncoding[text.charCodeAt(i)];
                        if (ch) {
                            newtext.push(
                                String.fromCharCode(ch));
                        } else {
                            newtext.push(
                                text[i]);
                        }

                        // since we are looping over chars anyway,
might as well
                        // check for residual unicodeness
                        if (newtext[i].charCodeAt(0) >> 8) {
                            /* more than 255 */
                            isUnicode = true;
                        }
                    }
                    text = newtext.join('');
                }
            }

            i = text.length;
            // isUnicode may be set to false above. Hence the triple-equal to
undefined
            while (isUnicode === undefined && i !== 0) {
                if (text.charCodeAt(i - 1) >> 8) {
                    /* more than 255 */
                    isUnicode = true;
                }
                i--;
            }
            if (!isUnicode) {
                return text;
            }

            newtext = flags.noBOM ? [] : [254, 255];
            for (i = 0, l = text.length; i < l; i++) {
                ch = text.charCodeAt(i);
                bch = ch >> 8; // divide by 256
                if (bch >> 8) {
                    /* something left after dividing by 256 second time */
                    throw new Error("Character at position " + i + " of
string '"
                        + text + "' exceeds 16bits. Cannot be encoded
into UCS-2 BE");
                }
                newtext.push(bch);
                newtext.push(ch - (bch << 8));
            }
            return String.fromCharCode.apply(undefined, newtext);
        },
        pdfEscape = function(text, flags) {
            /**
             * Replace '/', '(', and ')' with pdf-safe versions
             *
             * Doing to8bitStream does NOT make this PDF display unicode text.
For that
             * we also need to reference a unicode font and embed it - royal
pain in the rear.
             *
             * There is still a benefit to to8bitStream - PDF simply cannot handle
16bit chars,
             * which JavaScript Strings are happy to provide. So, while we still
cannot display
             * 2-byte characters property, at least CONDITIONALLY converting
(entire string containing)
             * 16bit chars to (USC-2-BE) 2-bytes per char + BOM streams we
ensure that entire PDF
             * is still parseable.
             * This will allow immediate support for unicode in document
properties strings.
             */
            return to8bitStream(text, flags).replace(/\\/g, '\\\\').replace(/\(/g,
'\\(').replace(/\)/g, '\\)');
        },
        putInfo = function() {
            out('/Producer (jsPDF ' + jsPDF.version + ')');
            for(var key in documentProperties) {
                if(documentProperties.hasOwnProperty(key) &&
documentProperties[key]) {
                    out('/'+key.substr(0,1).toUpperCase() + key.substr(1)
                        +' (' + pdfEscape(documentProperties[key]) +
')');
                }
            }
            var created  = new Date(),
                tzoffset = created.getTimezoneOffset(),
                tzsign   = tzoffset < 0 ? '+' : '-',
                tzhour   = Math.floor(Math.abs(tzoffset / 60)),
                tzmin    = Math.abs(tzoffset % 60),
                tzstr    = [tzsign, padd2(tzhour), "'", padd2(tzmin), "'"].join('');
            out(['/CreationDate (D:',
                    created.getFullYear(),
                    padd2(created.getMonth() + 1),
                    padd2(created.getDate()),
                    padd2(created.getHours()),
                    padd2(created.getMinutes()),
                    padd2(created.getSeconds()), tzstr, ')'].join(''));
        },
        putCatalog = function() {
            out('/Type /Catalog');
            out('/Pages 1 0 R');
            // PDF13ref Section 7.2.1
            if (!zoomMode) zoomMode = 'fullwidth';
            switch(zoomMode) {
                case 'fullwidth'  : out('/OpenAction [3 0 R /FitH null]');
break;
                case 'fullheight' : out('/OpenAction [3 0 R /FitV null]');
break;
                case 'fullpage'   : out('/OpenAction [3 0 R /Fit]');          break;
                case 'original'   : out('/OpenAction [3 0 R /XYZ null null 1]');
break;
                default:
                    var pcn = '' + zoomMode;
                    if (pcn.substr(pcn.length-1) === '%')
                        zoomMode = parseInt(zoomMode) / 100;
                    if (typeof zoomMode === 'number') {
                        out('/OpenAction [3 0 R /XYZ null null
'+f2(zoomMode)+']');
                    }
            }
            if (!layoutMode) layoutMode = 'continuous';
            switch(layoutMode) {
                case 'continuous' : out('/PageLayout /OneColumn');      break;
                case 'single'     : out('/PageLayout /SinglePage');     break;
                case 'two':
                case 'twoleft'    : out('/PageLayout /TwoColumnLeft');  break;
                case 'tworight'   : out('/PageLayout /TwoColumnRight');
break;
            }
            if (pageMode) {
                /**
                 * A name object specifying how the document should be
displayed when opened:
                 * UseNone      : Neither document outline nor thumbnail
images visible -- DEFAULT
                 * UseOutlines  : Document outline visible
                 * UseThumbs    : Thumbnail images visible
                 * FullScreen   : Full-screen mode, with no menu bar, window
controls, or any other window visible
                 */
                out('/PageMode /' + pageMode);
            }
            events.publish('putCatalog');
        },
        putTrailer = function() {
            out('/Size ' + (objectNumber + 1));
            out('/Root ' + objectNumber + ' 0 R');
            out('/Info ' + (objectNumber - 1) + ' 0 R');
        },
        beginPage = function(width,height) {
            // Dimensions are stored as user units and converted to points on output
            var orientation = typeof height === 'string' && height.toLowerCase();
            if (typeof width === 'string') {
                var format = width.toLowerCase();
                if (pageFormats.hasOwnProperty(format)) {
                    width  = pageFormats[format][0] / k;
                    height = pageFormats[format][1] / k;
                }
            }
            if (Array.isArray(width)) {
                height = width[1];
                width = width[0];
            }
            if (orientation) {
                switch(orientation.substr(0,1)) {
                    case 'l': if (height > width ) orientation = 's'; break;
                    case 'p': if (width > height ) orientation = 's'; break;
                }
                if (orientation === 's') { tmp = width; width = height; height = tmp; }
            }
            outToPages = true;
            pages[++page] = [];
            pagedim[page] = {
                width  : Number(width)  || pageWidth,
                height : Number(height) || pageHeight
```

```
    };
    _setPage(page);
},
_addPage = function() {
    beginPage.apply(this, arguments);
    // Set line width
    out(f2(lineWidth * k) + ' w');
    // Set draw color
    out(drawColor);
    // resurrecting non-default line caps, joins
    if (lineCapID !== 0) {
        out(lineCapID + ' J');
    }
    if (lineJoinID !== 0) {
        out(lineJoinID + ' j');
    }
    events.publish('addPage', { pageNumber : page });
},
_setPage = function(n) {
    if (n > 0 && n <= page) {
        currentPage = n;
        pageWidth = pagedim[n].width;
        pageHeight = pagedim[n].height;
    }
},
/**
 * Returns a document-specific font key - a label assigned to a
```

```
         * font name + font type combination at the time the font was added
         * to the font inventory.
         *
         * Font key is used as label for the desired font for a block of text
         * to be added to the PDF document stream.
         * @private
         * @function
         * @param fontName {String} can be undefined on "falthy" to indicate "use 
current"
         * @param fontStyle {String} can be undefined on "falthy" to indicate "use 
current"
         * @returns {String} Font key.
         */
        getFont = function(fontName, fontStyle) {
            var key;


            fontName  = fontName  !== undefined ? fontName  : 
fonts[activeFontKey].fontName;
            fontStyle = fontStyle !== undefined ? fontStyle : 
fonts[activeFontKey].fontStyle;


            try {
             // get a string like 'F3' - the KEY corresponding tot he font + type 
combination.
                key = fontmap[fontName][fontStyle];
            } catch (e) {}


            if (!key) {
                throw new Error("Unable to look up font label for font '" +
fontName + "', '"
                    + fontStyle + "'. Refer to getFontList() for available
fonts.");
            }
            return key;
        },
        buildDocument = function() {

            outToPages = false; // switches out() to content
            objectNumber = 2;
            content = [];
            offsets = [];

            // putHeader()
            out('%PDF-' + pdfVersion);

            putPages();

            putResources();

            // Info
            newObject();
            out('<<');
            putInfo();
            out('>>');
            out('endobj');
```

```
// Catalog
newObject();
out('<<');
putCatalog();
out('>>');
out('endobj');

// Cross-ref
var o = content_length, i, p = "0000000000";
out('xref');
out('0 ' + (objectNumber + 1));
out(p+' 65535 f ');
for (i = 1; i <= objectNumber; i++) {
        out((p + offsets[i]).slice(-10) + ' 00000 n ');
}
// Trailer
out('trailer');
out('<<');
putTrailer();
out('>>');
out('startxref');
out(o);
out('%%EOF');

outToPages = true;

return content.join('\n');
```

```
        },
        getStyle = function(style) {
            // see path-painting operators in PDF spec
            var op = 'S'; // stroke
            if (style === 'F') {
                op = 'f'; // fill
            } else if (style === 'FD' || style === 'DF') {
                op = 'B'; // both
            } else if (style === 'f' || style === 'f*' || style === 'B' || style === 'B*')
{
                /*
                Allow direct use of these PDF path-painting operators:
                - f     fill using nonzero winding number rule
                - f*    fill using even-odd rule
                - B     fill then stroke with fill using non-zero winding
number rule
                - B*    fill then stroke with fill using even-odd rule
                */
                op = style;
            }
            return op;
        },
        getArrayBuffer = function() {
            var data = buildDocument(), len = data.length,
                ab = new ArrayBuffer(len), u8 = new Uint8Array(ab);

            while(len--) u8[len] = data.charCodeAt(len);
            return ab;
```

```
},
getBlob = function() {
    return new Blob([getArrayBuffer()], { type : "application/pdf" });
},
/**
 * Generates the PDF document.
 *
 * If `type` argument is undefined, output is raw body of resulting PDF returned as a string.
 *
 * @param {String} type A string identifying one of the possible output types.
 * @param {Object} options An object providing some additional signalling to PDF generator.
 * @function
 * @returns {jsPDF}
 * @methodOf jsPDF#
 * @name output
 */
output = SAFE(function(type, options) {
    var datauri = ('' + type).substr(0,6) === 'dataur'
        ? 'data:application/pdf;base64,'+btoa(buildDocument()):0;

    switch (type) {
        case undefined:
            return buildDocument();
        case 'save':
            if (navigator.getUserMedia) {
```

```
                        if (global.URL === undefined
                        || global.URL.createObjectURL === undefined)
{
                            return
API.output('dataurlnewwindow');
                        }
                    }
                    saveAs(getBlob(), options);
                    if(typeof saveAs.unload === 'function') {
                        if(global.setTimeout) {
                            setTimeout(saveAs.unload,911);
                        }
                    }
                    break;
                case 'arraybuffer':
                    return getArrayBuffer();
                case 'blob':
                    return getBlob();
                case 'bloburi':
                case 'bloburl':
                    // User is responsible of calling revokeObjectURL
                    return global.URL &&
global.URL.createObjectURL(getBlob()) || void 0;
                case 'datauristring':
                case 'dataurlstring':
                    return datauri;
                case 'dataurlnewwindow':
                    var nW = global.open(datauri);
                    if (nW || typeof safari === "undefined") return nW;
                    /* pass through */
                case 'datauri':
                case 'dataurl':
                    return global.document.location.href = datauri;
                default:
                    throw new Error('Output type "' + type + '" is not supported.');
            }
            // @TODO: Add different output options
        });

        switch (unit) {
            case 'pt':  k = 1;          break;
            case 'mm':  k = 72 / 25.4;  break;
            case 'cm':  k = 72 / 2.54;  break;
            case 'in':  k = 72;         break;
            case 'px':  k = 96 / 72;    break;
            case 'pc':  k = 12;         break;
            case 'em':  k = 12;         break;
            case 'ex':  k = 6;          break;
            default:
                throw ('Invalid unit: ' + unit);
        }

        //---------------------------------------
        // Public API
```

```
/**
 * Object exposing internal API to plugins
 * @public
 */
API.internal = {
    'pdfEscape' : pdfEscape,
    'getStyle' : getStyle,
    /**
     * Returns {FontObject} describing a particular font.
     * @public
     * @function
     * @param fontName {String} (Optional) Font's family name
     * @param fontStyle {String} (Optional) Font's style variation name (Example:"Italic")
     * @returns {FontObject}
     */
    'getFont' : function() {
        return fonts[getFont.apply(API, arguments)];
    },
    'getFontSize' : function() {
        return activeFontSize;
    },
    'getLineHeight' : function() {
        return activeFontSize * lineHeightProportion;
    },
    'write' : function(string1 /*, string2, string3, etc */) {
        out(arguments.length === 1 ? string1 : Array.prototype.join.call(arguments, ' '));
```

```
            },
            'getCoordinateString' : function(value) {
                return f2(value * k);
            },
            'getVerticalCoordinateString' : function(value) {
                return f2((pageHeight - value) * k);
            },
            'collections' : {},
            'newObject' : newObject,
            'putStream' : putStream,
            'events' : events,
            // ratio that you use in multiplication of a given "size" number to arrive to 'point'
            // units of measurement.
            // scaleFactor is set at initialization of the document and calculated against the stated
            // default measurement units for the document.
            // If default is "mm", k is the number that will turn number in 'mm' into 'points' number.
            // through multiplication.
            'scaleFactor' : k,
            'pageSize' : {
                get width() {
                    return pageWidth
                },
                get height() {
                    return pageHeight
                }
```

```
    },
    'output' : function(type, options) {
        return output(type, options);
    },
    'getNumberOfPages' : function() {
        return pages.length - 1;
    },
    'pages' : pages
};

/**
 * Adds (and transfers the focus to) new page to the PDF document.
 * @function
 * @returns {jsPDF}
 *
 * @methodOf jsPDF#
 * @name addPage
 */
API.addPage = function() {
    _addPage.apply(this, arguments);
    return this;
};
API.setPage = function() {
    _setPage.apply(this, arguments);
    return this;
};
API.setDisplayMode = function(zoom, layout, pmode) {
		zoomMode   = zoom;
		layoutMode = layout;
		pageMode   = pmode;
		return this;
	},


	/**
	 * Adds text to page. Supports adding multiline text when 'text' argument is an Array of Strings.
	 *
	 * @function
	 * @param {String|Array} text String or array of strings to be added to the page. Each line is shifted one line down per font, spacing settings declared before this call.
	 * @param {Number} x Coordinate (in units declared at inception of PDF document) against left edge of the page
	 * @param {Number} y Coordinate (in units declared at inception of PDF document) against upper edge of the page
	 * @param {Object} flags Collection of settings signalling how the text must be encoded. Defaults are sane. If you think you want to pass some flags, you likely can read the source.
	 * @returns {jsPDF}
	 * @methodOf jsPDF#
	 * @name text
	 */
	API.text = function(text, x, y, flags, angle) {
		/**
		 * Inserts something like this into PDF
		 *   BT
		 *    /F1 16 Tf  % Font name + size
 *    16 TL % How many units down for next line in multiline text
 *    0 g % color
 *    28.35 813.54 Td % position
 *    (line one) Tj
 *    T* (line two) Tj
 *    T* (line three) Tj
 *   ET
 */
function ESC(s) {
        s = s.split("\t").join(Array(options.TabLen||9).join(" "));
        return pdfEscape(s, flags);
}

// Pre-August-2012 the order of arguments was function(x, y, text, flags)
// in effort to make all calls have similar signature like
//   function(data, coordinates... , miscellaneous)
// this method had its args flipped.
// code below allows backward compatibility with old arg order.
if (typeof text === 'number') {
        tmp = y;
        y = x;
        x = text;
        text = tmp;
}

// If there are any newlines in text, we assume
// the user wanted to print multiple lines, so break the
// text up into an array.  If the text is already an array,
// we assume the user knows what they are doing.
if (typeof text === 'string' && text.match(/[\n\r]/)) {
    text = text.split(/\r\n|\r|\n/g);
}
if (typeof flags === 'number') {
    angle = flags;
    flags = null;
}
var xtra = '',mode = 'Td', todo;
if (angle) {
    angle *= (Math.PI / 180);
    var c = Math.cos(angle),
    s = Math.sin(angle);
    xtra = [f2(c), f2(s), f2(s * -1), f2(c), ''].join(" ");
    mode = 'Tm';
}
flags = flags || {};
if (!('noBOM' in flags))
    flags.noBOM = true;
if (!('autoencode' in flags))
    flags.autoencode = true;

if (typeof text === 'string') {
    text = ESC(text);
} else if (text instanceof Array) {
    // we don't want to destroy  original text array, so cloning it
```

```
                var sa = text.concat(), da = [], len = sa.length;
                // we do array.join('text that must not be PDFescaped")
                // thus, pdfEscape each component separately
                while (len--) {
                    da.push(ESC(sa.shift()));
                }
                var linesLeft = Math.ceil((pageHeight - y) * k / (activeFontSize * lineHeightProportion));
                if (0 <= linesLeft && linesLeft < da.length + 1) {
                    todo = da.splice(linesLeft-1);
                }
                text = da.join(") Tj\nT* (");
            } else {
                throw new Error('Type of text must be string or Array. "' + text + '" is not recognized.');
            }
            // Using "'" ("go next line and render text" mark) would save space but would complicate our rendering code, templates

            // BT .. ET does NOT have default settings for Tf. You must state that explicitely every time for BT .. ET
            // if you want text transformation matrix (+ multiline) to work reliably (which reads sizes of things from font declarations)
            // Thus, there is NO useful, *reliable* concept of "default" font for a page.
            // The fact that "default" (reuse font used before) font worked before in basic cases is an accident
            // - readers dealing smartly with brokenness of jsPDF's markup.
            out(
                'BT\n/' +
                activeFontKey + ' ' + activeFontSize + ' Tf\n' +     // font face,
style, size
                (activeFontSize * lineHeightProportion) + ' TL\n' +  // line
spacing
                textColor +
                '\n' + xtra + f2(x * k) + ' ' + f2((pageHeight - y) * k) + ' ' + mode
+ '\n(' +
                text +
                ') Tj\nET');

            if (todo) {
                this.addPage();
                this.text( todo, x, activeFontSize * 1.7 / k);
            }

            return this;
        };

        API.lstext = function(text, x, y, spacing) {
            for (var i = 0, len = text.length ; i < len; i++, x += spacing)
this.text(text[i], x, y);
        };

        API.line = function(x1, y1, x2, y2) {
            return this.lines([[x2 - x1, y2 - y1]], x1, y1);
        };

        API.clip = function() {
            // By patrick-roberts, github.com/MrRio/jsPDF/issues/328
            // Call .clip() after calling .rect() with a style argument of null

            out('W') // clip

            out('S') // stroke path; necessary for clip to work

        };


        /**

         * Adds series of curves (straight lines or cubic bezier curves) to canvas, starting at `x`, `y` coordinates.

         * All data points in `lines` are relative to last line origin.

         * `x`, `y` become x1,y1 for first line / curve in the set.

         * For lines you only need to specify [x2, y2] - (ending point) vector against x1, y1 starting point.

         * For bezier curves you need to specify [x2,y2,x3,y3,x4,y4] - vectors to control points 1, 2, ending point. All vectors are against the start of the curve - x1,y1.

         *

         * @example .lines([[2,2],[-2,2],[1,1,2,2,3,3],[2,1]], 212,110, 10) // line, line, bezier curve, line

         * @param {Array} lines Array of *vector* shifts as pairs (lines) or sextets (cubic bezier curves).

         * @param {Number} x Coordinate (in units declared at inception of PDF document) against left edge of the page

         * @param {Number} y Coordinate (in units declared at inception of PDF document) against upper edge of the page

         * @param {Number} scale (Defaults to [1.0,1.0]) x,y Scaling factor for all vectors. Elements can be any floating number Sub-one makes drawing smaller. Over-one grows the drawing. Negative flips the direction.

         * @param {String} style A string specifying the painting style or null.  Valid styles include: 'S' [default] - stroke, 'F' - fill,  and 'DF' (or 'FD') -  fill then stroke. A null value postpones setting the style so that a shape may be composed using multiple method calls. The last drawing method call used to define the shape should not have a null style argument.
```

```
         * @param {Boolean} closed If true, the path is closed with a straight line from the end of the last curve to the starting point.
         * @function
         * @returns {jsPDF}
         * @methodOf jsPDF#
         * @name lines
         */
        API.lines = function(lines, x, y, scale, style, closed) {
            var scalex,scaley,i,l,leg,x2,y2,x3,y3,x4,y4;

            // Pre-August-2012 the order of arguments was function(x, y, lines, scale, style)
            // in effort to make all calls have similar signature like
            //   function(content, coordinateX, coordinateY , miscellaneous)
            // this method had its args flipped.
            // code below allows backward compatibility with old arg order.
            if (typeof lines === 'number') {
                tmp = y;
                y = x;
                x = lines;
                lines = tmp;
            }

            scale = scale || [1, 1];

            // starting point
            out(f3(x * k) + ' ' + f3((pageHeight - y) * k) + ' m ');
            scalex = scale[0];

            scaley = scale[1];

            l = lines.length;

            //, x2, y2 // bezier only. In page default measurement "units", *after* scaling

            //, x3, y3 // bezier only. In page default measurement "units", *after* scaling

            // ending point for all, lines and bezier. . In page default measurement "units", *after* scaling

            x4 = x; // last / ending point = starting point for first item.

            y4 = y; // last / ending point = starting point for first item.


            for (i = 0; i < l; i++) {

                leg = lines[i];

                if (leg.length === 2) {

                    // simple line

                    x4 = leg[0] * scalex + x4; // here last x4 was prior ending point

                    y4 = leg[1] * scaley + y4; // here last y4 was prior ending point

                    out(f3(x4 * k) + ' ' + f3((pageHeight - y4) * k) + ' l');

                } else {

                    // bezier curve

                    x2 = leg[0] * scalex + x4; // here last x4 is prior ending point

                    y2 = leg[1] * scaley + y4; // here last y4 is prior ending point

                    x3 = leg[2] * scalex + x4; // here last x4 is prior ending point
                    y3 = leg[3] * scaley + y4; // here last y4 is prior ending
point
                    x4 = leg[4] * scalex + x4; // here last x4 was prior
ending point
                    y4 = leg[5] * scaley + y4; // here last y4 was prior
ending point
                    out(
                        f3(x2 * k) + ' ' +
                        f3((pageHeight - y2) * k) + ' ' +
                        f3(x3 * k) + ' ' +
                        f3((pageHeight - y3) * k) + ' ' +
                        f3(x4 * k) + ' ' +
                        f3((pageHeight - y4) * k) + ' c');
                }
            }

            if (closed) {
                out(' h');
            }

            // stroking / filling / both the path
            if (style !== null) {
                out(getStyle(style));
            }
            return this;
        };

        /**
```

```
         * Adds a rectangle to PDF
         *
         * @param {Number} x Coordinate (in units declared at inception of PDF document) against left edge of the page
         * @param {Number} y Coordinate (in units declared at inception of PDF document) against upper edge of the page
         * @param {Number} w Width (in units declared at inception of PDF document)
         * @param {Number} h Height (in units declared at inception of PDF document)
         * @param {String} style A string specifying the painting style or null.  Valid styles include: 'S' [default] - stroke, 'F' - fill,  and 'DF' (or 'FD') -  fill then stroke. A null value postpones setting the style so that a shape may be composed using multiple method calls. The last drawing method call used to define the shape should not have a null style argument.
         * @function
         * @returns {jsPDF}
         * @methodOf jsPDF#
         * @name rect
         */
        API.rect = function(x, y, w, h, style) {
            var op = getStyle(style);
            out([
                    f2(x * k),
                    f2((pageHeight - y) * k),
                    f2(w * k),
                    f2(-h * k),
                    're'
                ].join(' '));
            if (style !== null) {
                out(getStyle(style));
            }

            return this;
        };

        /**
         * Adds a triangle to PDF
         *
         * @param {Number} x1 Coordinate (in units declared at inception of PDF document) against left edge of the page
         * @param {Number} y1 Coordinate (in units declared at inception of PDF document) against upper edge of the page
         * @param {Number} x2 Coordinate (in units declared at inception of PDF document) against left edge of the page
         * @param {Number} y2 Coordinate (in units declared at inception of PDF document) against upper edge of the page
         * @param {Number} x3 Coordinate (in units declared at inception of PDF document) against left edge of the page
         * @param {Number} y3 Coordinate (in units declared at inception of PDF document) against upper edge of the page
         * @param {String} style A string specifying the painting style or null.  Valid styles include: 'S' [default] - stroke, 'F' - fill,  and 'DF' (or 'FD') -  fill then stroke. A null value postpones setting the style so that a shape may be composed using multiple method calls. The last drawing method call used to define the shape should not have a null style argument.
         * @function
         * @returns {jsPDF}
         * @methodOf jsPDF#
         * @name triangle
```

```
 */
API.triangle = function(x1, y1, x2, y2, x3, y3, style) {
    this.lines(
        [
            [x2 - x1, y2 - y1], // vector to point 2
            [x3 - x2, y3 - y2], // vector to point 3
            [x1 - x3, y1 - y3]// closing vector back to point 1
        ],
        x1,
        y1, // start of path
        [1, 1],
        style,
        true);
    return this;
};

/**
 * Adds a rectangle with rounded corners to PDF
 *
 * @param {Number} x Coordinate (in units declared at inception of PDF document) against left edge of the page
 * @param {Number} y Coordinate (in units declared at inception of PDF document) against upper edge of the page
 * @param {Number} w Width (in units declared at inception of PDF document)
 * @param {Number} h Height (in units declared at inception of PDF document)
 * @param {Number} rx Radius along x axis (in units declared at inception of PDF document)
```

```
         * @param {Number} rx Radius along y axis (in units declared at inception of
PDF document)
         * @param {String} style A string specifying the painting style or null.  Valid
styles include: 'S' [default] - stroke, 'F' - fill,  and 'DF' (or 'FD') -  fill then stroke. A null value
postpones setting the style so that a shape may be composed using multiple method calls.
The last drawing method call used to define the shape should not have a null style
argument.
         * @function
         * @returns {jsPDF}
         * @methodOf jsPDF#
         * @name roundedRect
         */
        API.roundedRect = function(x, y, w, h, rx, ry, style) {
            var MyArc = 4 / 3 * (Math.SQRT2 - 1);
            this.lines(
                [
                    [(w - 2 * rx), 0],
                    [(rx * MyArc), 0, rx, ry - (ry * MyArc), rx, ry],
                    [0, (h - 2 * ry)],
                    [0, (ry * MyArc),  - (rx * MyArc), ry, -rx, ry],
                    [(-w + 2 * rx), 0],
                    [ - (rx * MyArc), 0, -rx,  - (ry * MyArc), -rx, -ry],
                    [0, (-h + 2 * ry)],
                    [0,  - (ry * MyArc), (rx * MyArc), -ry, rx, -ry]
                ],
                x + rx,
                y, // start of path
                [1, 1],
                style);
            return this;
        };

        /**
         * Adds an ellipse to PDF
         *
         * @param {Number} x Coordinate (in units declared at inception of PDF document) against left edge of the page
         * @param {Number} y Coordinate (in units declared at inception of PDF document) against upper edge of the page
         * @param {Number} rx Radius along x axis (in units declared at inception of PDF document)
         * @param {Number} rx Radius along y axis (in units declared at inception of PDF document)
         * @param {String} style A string specifying the painting style or null.  Valid styles include: 'S' [default] - stroke, 'F' - fill,  and 'DF' (or 'FD') -  fill then stroke. A null value postpones setting the style so that a shape may be composed using multiple method calls. The last drawing method call used to define the shape should not have a null style argument.
         * @function
         * @returns {jsPDF}
         * @methodOf jsPDF#
         * @name ellipse
         */
        API.ellipse = function(x, y, rx, ry, style) {
            var lx = 4 / 3 * (Math.SQRT2 - 1) * rx,
                ly = 4 / 3 * (Math.SQRT2 - 1) * ry;

            out([
                    f2((x + rx) * k),
                    f2((pageHeight - y) * k),
                    'm',
                    f2((x + rx) * k),
                    f2((pageHeight - (y - ly)) * k),
                    f2((x + lx) * k),
                    f2((pageHeight - (y - ry)) * k),
                    f2(x * k),
                    f2((pageHeight - (y - ry)) * k),
                    'c'
                ].join(' '));
            out([
                    f2((x - lx) * k),
                    f2((pageHeight - (y - ry)) * k),
                    f2((x - rx) * k),
                    f2((pageHeight - (y - ly)) * k),
                    f2((x - rx) * k),
                    f2((pageHeight - y) * k),
                    'c'
                ].join(' '));
            out([
                    f2((x - rx) * k),
                    f2((pageHeight - (y + ly)) * k),
                    f2((x - lx) * k),
                    f2((pageHeight - (y + ry)) * k),
                    f2(x * k),
                    f2((pageHeight - (y + ry)) * k),
                    'c'
```

```
                ].join(' '));
            out([
                    f2((x + lx) * k),
                    f2((pageHeight - (y + ry)) * k),
                    f2((x + rx) * k),
                    f2((pageHeight - (y + ly)) * k),
                    f2((x + rx) * k),
                    f2((pageHeight - y) * k),
                    'c'
                ].join(' '));

            if (style !== null) {
                out(getStyle(style));
            }

            return this;
        };

        /**
         * Adds an circle to PDF
         *
         * @param {Number} x Coordinate (in units declared at inception of PDF document) against left edge of the page
         * @param {Number} y Coordinate (in units declared at inception of PDF document) against upper edge of the page
         * @param {Number} r Radius (in units declared at inception of PDF document)
         * @param {String} style A string specifying the painting style or null.  Valid styles include: 'S' [default] - stroke, 'F' - fill,  and 'DF' (or 'FD') -  fill then stroke. A null value
```

```
postpones setting the style so that a shape may be composed using multiple method calls. The last drawing method call used to define the shape should not have a null style argument.
		 * @function
		 * @returns {jsPDF}
		 * @methodOf jsPDF#
		 * @name circle
		 */
		API.circle = function(x, y, r, style) {
			return this.ellipse(x, y, r, r, style);
		};

		/**
		 * Adds a properties to the PDF document
		 *
		 * @param {Object} A property_name-to-property_value object structure.
		 * @function
		 * @returns {jsPDF}
		 * @methodOf jsPDF#
		 * @name setProperties
		 */
		API.setProperties = function(properties) {
			// copying only those properties we can render.
			for (var property in documentProperties) {
				if (documentProperties.hasOwnProperty(property) && properties[property]) {
					documentProperties[property] = properties[property];
				}
```

```
	}
	return this;
};

/**
 * Sets font size for upcoming text elements.
 *
 * @param {Number} size Font size in points.
 * @function
 * @returns {jsPDF}
 * @methodOf jsPDF#
 * @name setFontSize
 */
API.setFontSize = function(size) {
	activeFontSize = size;
	return this;
};

/**
 * Sets text font face, variant for upcoming text elements.
 * See output of jsPDF.getFontList() for possible font names, styles.
 *
 * @param {String} fontName Font name or family. Example: "times"
 * @param {String} fontStyle Font style or variant. Example: "italic"
 * @function
 * @returns {jsPDF}
 * @methodOf jsPDF#
```

```
         * @name setFont
         */
        API.setFont = function(fontName, fontStyle) {
            activeFontKey = getFont(fontName, fontStyle);
            // if font is not found, the above line blows up and we never go
further
            return this;
        };

        /**
         * Switches font style or variant for upcoming text elements,
         * while keeping the font face or family same.
         * See output of jsPDF.getFontList() for possible font names, styles.
         *
         * @param {String} style Font style or variant. Example: "italic"
         * @function
         * @returns {jsPDF}
         * @methodOf jsPDF#
         * @name setFontStyle
         */
        API.setFontStyle = API.setFontType = function(style) {
            activeFontKey = getFont(undefined, style);
            // if font is not found, the above line blows up and we never go
further
            return this;
        };

        /**
```

```
         * Returns an object - a tree of fontName to fontStyle relationships
available to
         * active PDF document.
         *
         * @public
         * @function
         * @returns {Object} Like {'times':['normal', 'italic', ... ], 'arial':['normal',
'bold', ... ], ... }
         * @methodOf jsPDF#
         * @name getFontList
         */
        API.getFontList = function() {
            // TODO: iterate over fonts array or return copy of fontmap instead
in case more are ever added.
            var list = {},fontName,fontStyle,tmp;

            for (fontName in fontmap) {
                if (fontmap.hasOwnProperty(fontName)) {
                    list[fontName] = tmp = [];
                    for (fontStyle in fontmap[fontName]) {
                        if
(fontmap[fontName].hasOwnProperty(fontStyle)) {
                            tmp.push(fontStyle);
                        }
                    }
                }
            }

            return list;
```

```
    };

    /**
     * Sets line width for upcoming lines.
     *
     * @param {Number} width Line width (in units declared at inception of PDF document)
     * @function
     * @returns {jsPDF}
     * @methodOf jsPDF#
     * @name setLineWidth
     */
    API.setLineWidth = function(width) {
        out((width * k).toFixed(2) + ' w');
        return this;
    };

    /**
     * Sets the stroke color for upcoming elements.
     *
     * Depending on the number of arguments given, Gray, RGB, or CMYK
     * color space is implied.
     *
     * When only ch1 is given, "Gray" color space is implied and it
     * must be a value in the range from 0.00 (solid black) to to 1.00 (white)
     * if values are communicated as String types, or in range from 0 (black)
     * to 255 (white) if communicated as Number type.
     * The RGB-like 0-255 range is provided for backward compatibility.
```

```
        *
        * When only ch1,ch2,ch3 are given, "RGB" color space is implied and each
        * value must be in the range from 0.00 (minimum intensity) to to 1.00
        * (max intensity) if values are communicated as String types, or
        * from 0 (min intensity) to to 255 (max intensity) if values are
communicated
        * as Number types.
        * The RGB-like 0-255 range is provided for backward compatibility.
        *
        * When ch1,ch2,ch3,ch4 are given, "CMYK" color space is implied and each
        * value must be a in the range from 0.00 (0% concentration) to to
        * 1.00 (100% concentration)
        *
        * Because JavaScript treats fixed point numbers badly (rounds to
        * floating point nearest to binary representation) it is highly advised to
        * communicate the fractional numbers as String types, not JavaScript
Number type.
        *
        * @param {Number|String} ch1 Color channel value
        * @param {Number|String} ch2 Color channel value
        * @param {Number|String} ch3 Color channel value
        * @param {Number|String} ch4 Color channel value
        *
        * @function
        * @returns {jsPDF}
        * @methodOf jsPDF#
        * @name setDrawColor
        */
        API.setDrawColor = function(ch1, ch2, ch3, ch4) {
            var color;
            if (ch2 === undefined || (ch4 === undefined && ch1 === ch2 ===
ch3)) {
                // Gray color space.
                if (typeof ch1 === 'string') {
                    color = ch1 + ' G';
                } else {
                    color = f2(ch1 / 255) + ' G';
                }
            } else if (ch4 === undefined) {
                // RGB
                if (typeof ch1 === 'string') {
                    color = [ch1, ch2, ch3, 'RG'].join(' ');
                } else {
                    color = [f2(ch1 / 255), f2(ch2 / 255), f2(ch3 / 255),
'RG'].join(' ');
                }
            } else {
                // CMYK
                if (typeof ch1 === 'string') {
                    color = [ch1, ch2, ch3, ch4, 'K'].join(' ');
                } else {
                    color = [f2(ch1), f2(ch2), f2(ch3), f2(ch4), 'K'].join(' ');
                }
            }

            out(color);
            return this;
        };

        /**
         * Sets the fill color for upcoming elements.
         *
         * Depending on the number of arguments given, Gray, RGB, or CMYK
         * color space is implied.
         *
         * When only ch1 is given, "Gray" color space is implied and it
         * must be a value in the range from 0.00 (solid black) to to 1.00 (white)
         * if values are communicated as String types, or in range from 0 (black)
         * to 255 (white) if communicated as Number type.
         * The RGB-like 0-255 range is provided for backward compatibility.
         *
         * When only ch1,ch2,ch3 are given, "RGB" color space is implied and each
         * value must be in the range from 0.00 (minimum intensity) to to 1.00
         * (max intensity) if values are communicated as String types, or
         * from 0 (min intensity) to to 255 (max intensity) if values are
communicated
         * as Number types.
         * The RGB-like 0-255 range is provided for backward compatibility.
         *
         * When ch1,ch2,ch3,ch4 are given, "CMYK" color space is implied and each
         * value must be a in the range from 0.00 (0% concentration) to to
         * 1.00 (100% concentration)
         *
         * Because JavaScript treats fixed point numbers badly (rounds to
         * floating point nearest to binary representation) it is highly advised to
         * communicate the fractional numbers as String types, not JavaScript Number type.
         *
         * @param {Number|String} ch1 Color channel value
         * @param {Number|String} ch2 Color channel value
         * @param {Number|String} ch3 Color channel value
         * @param {Number|String} ch4 Color channel value
         *
         * @function
         * @returns {jsPDF}
         * @methodOf jsPDF#
         * @name setFillColor
         */
        API.setFillColor = function(ch1, ch2, ch3, ch4) {
            var color;

            if (ch2 === undefined || (ch4 === undefined && ch1 === ch2 === ch3)) {
                // Gray color space.
                if (typeof ch1 === 'string') {
                    color = ch1 + ' g';
                } else {
                    color = f2(ch1 / 255) + ' g';
                }
            } else if (ch4 === undefined) {
                // RGB
                if (typeof ch1 === 'string') {
                    color = [ch1, ch2, ch3, 'rg'].join(' ');
                } else {
                    color = [f2(ch1 / 255), f2(ch2 / 255), f2(ch3 / 255), 'rg'].join(' ');
                }
            } else {
                // CMYK
                if (typeof ch1 === 'string') {
                    color = [ch1, ch2, ch3, ch4, 'k'].join(' ');
                } else {
                    color = [f2(ch1), f2(ch2), f2(ch3), f2(ch4), 'k'].join(' ');
                }
            }

            out(color);
            return this;
        };

        /**
         * Sets the text color for upcoming elements.
         * If only one, first argument is given,
         * treats the value as gray-scale color value.
         *
         * @param {Number} r Red channel color value in range 0-255 or {String} r color value in hexadecimal, example: '#FFFFFF'
         * @param {Number} g Green channel color value in range 0-255
         * @param {Number} b Blue channel color value in range 0-255
         * @function
```

```
 * @returns {jsPDF}
 * @methodOf jsPDF#
 * @name setTextColor
 */
API.setTextColor = function(r, g, b) {
    if ((typeof r === 'string') && /^#[0-9A-Fa-f]{6}$/.test(r)) {
        var hex = parseInt(r.substr(1), 16);
        r = (hex >> 16) & 255;
        g = (hex >> 8) & 255;
        b = (hex & 255);
    }

    if ((r === 0 && g === 0 && b === 0) || (typeof g === 'undefined')) {
        textColor = f3(r / 255) + ' g';
    } else {
        textColor = [f3(r / 255), f3(g / 255), f3(b / 255), 'rg'].join(' ');
    }
    return this;
};

/**
 * Is an Object providing a mapping from human-readable to
 * integer flag values designating the varieties of line cap
 * and join styles.
 *
 * @returns {Object}
 * @fieldOf jsPDF#
 * @name CapJoinStyles
 */
API.CapJoinStyles = {
    0 : 0,
    'butt' : 0,
    'but' : 0,
    'miter' : 0,
    1 : 1,
    'round' : 1,
    'rounded' : 1,
    'circle' : 1,
    2 : 2,
    'projecting' : 2,
    'project' : 2,
    'square' : 2,
    'bevel' : 2
};

/**
 * Sets the line cap styles
 * See {jsPDF.CapJoinStyles} for variants
 *
 * @param {String|Number} style A string or number identifying the type of line cap
 * @function
 * @returns {jsPDF}
 * @methodOf jsPDF#
 * @name setLineCap
```

```
     */
    API.setLineCap = function(style) {
        var id = this.CapJoinStyles[style];
        if (id === undefined) {
            throw new Error("Line cap style of '" + style + "' is not recognized. See or extend .CapJoinStyles property for valid styles");
        }
        lineCapID = id;
        out(id + ' J');

        return this;
    };

    /**
     * Sets the line join styles
     * See {jsPDF.CapJoinStyles} for variants
     *
     * @param {String|Number} style A string or number identifying the type of line join
     * @function
     * @returns {jsPDF}
     * @methodOf jsPDF#
     * @name setLineJoin
     */
    API.setLineJoin = function(style) {
        var id = this.CapJoinStyles[style];
        if (id === undefined) {
            throw new Error("Line join style of '" + style + "' is not recognized. See or extend .CapJoinStyles property for valid styles");
        }
        lineJoinID = id;
        out(id + ' j');

        return this;
    };

    // Output is both an internal (for plugins) and external function
    API.output = output;

    /**
     * Saves as PDF document. An alias of jsPDF.output('save', 'filename.pdf')
     * @param  {String} filename The filename including extension.
     *
     * @function
     * @returns {jsPDF}
     * @methodOf jsPDF#
     * @name save
     */
    API.save = function(filename) {
        API.output('save', filename);
    };

    // applying plugins (more methods) ON TOP of built-in API.
    // this is intentional as we allow plugins to override
    // built-ins
    for (var plugin in jsPDF.API) {
        if (jsPDF.API.hasOwnProperty(plugin)) {
            if (plugin === 'events' && jsPDF.API.events.length) {
                (function(events, newEvents) {

                    // jsPDF.API.events is a JS Array of Arrays
                    // where each Array is a pair of event name,
handler
                    // Events were added by plugins to the jsPDF
instantiator.
                    // These are always added to the new instance
and some ran
                    // during instantiation.
                    var eventname,handler_and_args,i;

                    for (i = newEvents.length - 1; i !== -1; i--) {
                        // subscribe takes 3 args: 'topic',
function, runonce_flag
                        // if undefined, runonce is false.
                        // users can attach callback directly,
                        // or they can attach an array with
[callback, runonce_flag]
                        // that's what the "apply" magic is for
below.
                        eventname = newEvents[i][0];
                        handler_and_args = newEvents[i][1];
                        events.subscribe.apply(
                            events,
                            [eventname].concat(
                                typeof handler_and_args
=== 'function' ?

    [handler_and_args] : handler_and_args));
                    }
                }(events, jsPDF.API.events));
            } else {
                API[plugin] = jsPDF.API[plugin];
            }
        }
    }

    //////////////////////////////////////////////////////
    // continuing initialization of jsPDF Document object
    //////////////////////////////////////////////////////
    // Add the first page automatically
    addFonts();
    activeFontKey = 'F1';
    _addPage(format, orientation);

    events.publish('initialized');
    return API;
    }

    /**
     * jsPDF.API is a STATIC property of jsPDF class.
     * jsPDF.API is an object you can add methods and properties to.
     * The methods / properties you add will show up in new jsPDF objects.
     *
     * One property is prepopulated. It is the 'events' Object. Plugin authors can add topics,
     * callbacks to this object. These will be reassigned to all new instances of jsPDF.
     * Examples:
     * jsPDF.API.events['initialized'] = function(){ 'this' is API object }
     * jsPDF.API.events['addFont'] = function(added_font_object){ 'this' is API object }
     *
     * @static
     * @public
     * @memberOf jsPDF
     * @name API
     *
     * @example
     * jsPDF.API.mymethod = function(){
     *   // 'this' will be ref to internal API object. see jsPDF source
     *   // , so you can refer to built-in methods like so:
     *   //     this.line(....)
     *   //     this.text(....)
     * }
     * var pdfdoc = new jsPDF()
     * pdfdoc.mymethod() // <- !!!!!!
     */
    jsPDF.API = {events:[]};
    jsPDF.version = "1.0.272-debug 2014-09-29T15:09:diegocr";

    if (typeof define === 'function' && define.amd) {
        define('jsPDF', function() {
            return jsPDF;
        });
    } else {
        global.jsPDF = jsPDF;
    }
    return jsPDF;
}(typeof self !== "undefined" && self || typeof window !== "undefined" && window || this));
/**
 * jsPDF addHTML PlugIn
 * Copyright (c) 2014 Diego Casorran
 *
 * Licensed under the MIT License.
 * http://opensource.org/licenses/mit-license
 */

(function (jsPDFAPI) {
    'use strict';

    /**
     * Renders an HTML element to canvas object which added as an image to the PDF
     *
     * This PlugIn requires html2canvas: https://github.com/niklasvh/html2canvas
     *          OR rasterizeHTML: https://github.com/cburgmer/rasterizeHTML.js
     *
     * @public
     * @function
     * @param element {Mixed} HTML Element, or anything supported by html2canvas.
     * @param x {Number} starting X coordinate in jsPDF instance's declared units.
     * @param y {Number} starting Y coordinate in jsPDF instance's declared units.
     * @param options {Object} Additional options, check the code below.
     * @param callback {Function} to call when the rendering has finished.
     *
     * NOTE: Every parameter is optional except 'element' and 'callback', in such
     *       case the image is positioned at 0x0 covering the whole PDF document
     *       size. Ie, to easily take screenshoots of webpages saving them to PDF.
     */
    jsPDFAPI.addHTML = function (element, x, y, options, callback) {
        'use strict';

        if(typeof html2canvas === 'undefined' && typeof rasterizeHTML === 'undefined')
            throw new Error('You need either '
                +'https://github.com/niklasvh/html2canvas'
                +' or https://github.com/cburgmer/rasterizeHTML.js');

        if(typeof x !== 'number') {
            options = x;
            callback = y;
        }

        if(typeof options === 'function') {
            callback = options;
            options = null;
        }


        var I = this.internal, K = I.scaleFactor, W = I.pageSize.width, H =
I.pageSize.height;


        options = options || {};
        options.onrendered = function(obj) {
            x = parseInt(x) || 0;
            y = parseInt(y) || 0;
            var dim = options.dim || {};
            var h = dim.h || 0;
            var w = dim.w || Math.min(W,obj.width/K) - x;

            var format = 'JPEG';
            if(options.format)
                format = options.format;

            if(obj.height > H && options.pagesplit) {
                var crop = function() {
                    var cy = 0;
                    while(1) {
                        var canvas =
document.createElement('canvas');
                        canvas.width = Math.min(W*K,obj.width);
                        canvas.height = Math.min(H*K,obj.height-cy);
                        var ctx = canvas.getContext('2d');

    ctx.drawImage(obj,0,cy,obj.width,canvas.height,0,0,canvas.width,canvas.height);
```

```
                        var args = [canvas,
x,cy?0:y,canvas.width/K,canvas.height/K, format,null,'SLOW'];
                        this.addImage.apply(this, args);
                        cy += canvas.height;
                        if(cy >= obj.height) break;
                        this.addPage();
                    }
                    callback(w,cy,null,args);
                }.bind(this);
                if(obj.nodeName === 'CANVAS') {
                    var img = new Image();
                    img.onload = crop;
                    img.src = obj.toDataURL("image/png");
                    obj = img;
                } else {
                    crop();
                }
            } else {
                var alias = Math.random().toString(35);
                var args = [obj, x,y,w,h, format,alias,'SLOW'];

                this.addImage.apply(this, args);

                callback(w,h,alias,args);
            }
        }.bind(this);

        if(typeof html2canvas !== 'undefined' && !options.rstz) {
            return html2canvas(element, options);
        }

        if(typeof rasterizeHTML !== 'undefined') {
            var meth = 'drawDocument';
            if(typeof element === 'string') {
                meth = /^http/.test(element) ? 'drawURL' : 'drawHTML';
            }
            options.width = options.width || (W*K);
            return rasterizeHTML[meth](element, void 0,
options).then(function(r) {
                options.onrendered(r.image);
            }, function(e) {
                callback(null,e);
            });
        }

        return null;
    };
})(jsPDF.API);
/** @preserve
 * jsPDF addImage plugin
 * Copyright (c) 2012 Jason Siefken, https://github.com/siefkenj/
 *      2013 Chris Dowling, https://github.com/gingerchris
 *      2013 Trinh Ho, https://github.com/ineedfat
 *      2013 Edwin Alejandro Perez, https://github.com/eaparango
 *      2013 Norah Smith, https://github.com/burnburnrocket
 *      2014 Diego Casorran, https://github.com/diegocr
 *          2014 James Robb, https://github.com/jamesbrobb
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
 * LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

;(function(jsPDFAPI) {
        'use strict'


        var namespace = 'addImage_',
                supported_image_types = ['jpeg', 'jpg', 'png'];
```

```
    // Image functionality ported from pdf.js
    var putImage = function(img) {

        var objectNumber = this.internal.newObject()
        , out = this.internal.write
        , putStream = this.internal.putStream

        img['n'] = objectNumber

        out('<</Type /XObject')
        out('/Subtype /Image')
        out('/Width ' + img['w'])
        out('/Height ' + img['h'])
        if (img['cs'] === this.color_spaces.INDEXED) {
            out('/ColorSpace [/Indexed /DeviceRGB '
                    // if an indexed png defines more than one colour with transparency, we've created a smask
                    + (img['pal'].length / 3 - 1) + ' ' + ('smask' in img ? objectNumber + 2 : objectNumber + 1)
                    + ' 0 R]');
        } else {
            out('/ColorSpace /' + img['cs']);
            if (img['cs'] === this.color_spaces.DEVICE_CMYK) {
                out('/Decode [1 0 1 0 1 0 1 0]');
            }
        }
        out('/BitsPerComponent ' + img['bpc']);
        if ('f' in img) {
            out('/Filter /' + img['f']);
        }
        if ('dp' in img) {
            out('/DecodeParms <<' + img['dp'] + '>>');
        }
        if ('trns' in img && img['trns'].constructor == Array) {
            var trns = '',
                i = 0,
                len = img['trns'].length;
            for (; i < len; i++)
                trns += (img['trns'][i] + ' ' + img['trns'][i] + ' ');
            out('/Mask [' + trns + ']');
        }
        if ('smask' in img) {
            out('/SMask ' + (objectNumber + 1) + ' 0 R');
        }
        out('/Length ' + img['data'].length + '>>');

        putStream(img['data']);

        out('endobj');

        // Soft mask
        if ('smask' in img) {
            var dp = '/Predictor 15 /Colors 1 /BitsPerComponent ' + img['bpc'] + ' /Columns ' + img['w'];
            var smask = {'w': img['w'], 'h': img['h'], 'cs': 'DeviceGray', 'bpc': img['bpc'], 'dp': dp, 'data': img['smask']};
            if ('f' in img)
                smask.f = img['f'];
            putImage.call(this, smask);
        }

     //Palette
        if (img['cs'] === this.color_spaces.INDEXED) {

            this.internal.newObject();
            //out('<< /Filter / ' + img['f'] +' /Length ' + img['pal'].length + '>>');
            //putStream(zlib.compress(img['pal']));
            out('<< /Length ' + img['pal'].length + '>>');
            putStream(this.arrayBufferToBinaryString(new Uint8Array(img['pal'])));
            out('endobj');
        }
    }
    , putResourcesCallback = function() {
        var images = this.internal.collections[namespace + 'images']
        for ( var i in images ) {
            putImage.call(this, images[i])
        }
    }
    , putXObjectsDictCallback = function(){
        var images = this.internal.collections[namespace + 'images']
        , out = this.internal.write
        , image
        for (var i in images) {
            image = images[i]
            out(
                '/I' + image['i']
                , image['n']
                , '0'
                , 'R'
            )
        }
    }
    , checkCompressValue = function(value) {
        if(value && typeof value === 'string')
            value = value.toUpperCase();
        return value in jsPDFAPI.image_compression ? value : jsPDFAPI.image_compression.NONE;
    }
    , getImages = function() {
        var images = this.internal.collections[namespace + 'images'];
        //first run, so initialise stuff
        if(!images) {
            this.internal.collections[namespace + 'images'] = images = {};
            this.internal.events.subscribe('putResources', putResourcesCallback);
            this.internal.events.subscribe('putXobjectDict', putXObjectsDictCallback);
        }
        return images;
    }
    , getImageIndex = function(images) {
        var imageIndex = 0;

        if (images){
            // this is NOT the first time this method is ran on this instance of jsPDF object.
            imageIndex = Object.keys ?
            Object.keys(images).length :
            (function(o){
                var i = 0
                for (var e in o){if(o.hasOwnProperty(e)){ i++ }}
                return i
            })(images)
        }

        return imageIndex;
    }
    , notDefined = function(value) {
        return typeof value === 'undefined' || value === null;
    }
    , generateAliasFromData = function(data) {
        return typeof data === 'string' && jsPDFAPI.sHashCode(data);
    }
    , doesNotSupportImageType = function(type) {
        return supported_image_types.indexOf(type) === -1;
    }
    , processMethodNotEnabled = function(type) {
        return typeof jsPDFAPI['process' + type.toUpperCase()] !== 'function';
    }
    , isDOMElement = function(object) {
        return typeof object === 'object' && object.nodeType === 1;
    }
    , createDataURIFromElement = function(element, format, angle) {

        //if element is an image which uses data url defintion, just return the
dataurl
        if (element.nodeName === 'IMG' && element.hasAttribute('src')) {
            var src = ''+element.getAttribute('src');
            if (!angle && src.indexOf('data:image/') === 0) return src;

            // only if the user doesn't care about a format
            if (!format && /\.png(?:[?#].*)?$/i.test(src)) format = 'png';
        }

        if(element.nodeName === 'CANVAS') {
            var canvas = element;
        } else {
            var canvas = document.createElement('canvas');
            canvas.width = element.clientWidth || element.width;
            canvas.height = element.clientHeight || element.height;

            var ctx = canvas.getContext('2d');
            if (!ctx) {
                throw ('addImage requires canvas to be supported by
browser.');
            }
            if (angle) {
                var x, y, b, c, s, w, h, to_radians = Math.PI/180,
angleInRadians;

                if (typeof angle === 'object') {
                    x = angle.x;
                    y = angle.y;
                    b = angle.bg;
                    angle = angle.angle;
                }
                angleInRadians = angle*to_radians;
                c = Math.abs(Math.cos(angleInRadians));
                s = Math.abs(Math.sin(angleInRadians));
                w = canvas.width;
                h = canvas.height;
                canvas.width = h * s + w * c;
                canvas.height = h * c + w * s;

                if (isNaN(x)) x = canvas.width / 2;
                if (isNaN(y)) y = canvas.height / 2;

                ctx.clearRect(0,0,canvas.width, canvas.height);
                ctx.fillStyle = b || 'white';
                ctx.fillRect(0, 0, canvas.width, canvas.height);
                ctx.save();
                ctx.translate(x, y);

                ctx.rotate(angleInRadians);

                ctx.drawImage(element, -(w/2), -(h/2));

                ctx.rotate(-angleInRadians);

                ctx.translate(-x, -y);

                ctx.restore();

            } else {

                ctx.drawImage(element, 0, 0, canvas.width, canvas.height);

            }

        }

        return canvas.toDataURL((''+format).toLowerCase() == 'png' ? 'image/png' :
'image/jpeg');

    }

    ,checkImagesForAlias = function(alias, images) {

        var cached_info;

        if(images) {

            for(var e in images) {

                if(alias === images[e].alias) {

                    cached_info = images[e];

                    break;

                }

            }

        }

        return cached_info;

    }

    ,determineWidthAndHeight = function(w, h, info) {

        if (!w && !h) {

            w = -96;
```

```
			h = -96;
		}
		if (w < 0) {
			w = (-1) * info['w'] * 72 / w / this.internal.scaleFactor;
		}
		if (h < 0) {
			h = (-1) * info['h'] * 72 / h / this.internal.scaleFactor;
		}
		if (w === 0) {
			w = h * info['w'] / info['h'];
		}
		if (h === 0) {
			h = w * info['h'] / info['w'];
		}

		return [w, h];
	}
	, writeImageToPDF = function(x, y, w, h, info, index, images) {
		var dims = determineWidthAndHeight.call(this, w, h, info),
			coord = this.internal.getCoordinateString,
			vcoord = this.internal.getVerticalCoordinateString;

		w = dims[0];
		h = dims[1];

		images[index] = info;
	this.internal.write(
		'q'
		, coord(w)
		, '0 0'
		, coord(h) // TODO: check if this should be shifted by vcoord
		, coord(x)
		, vcoord(y + h)
		, 'cm /I'+info['i']
		, 'Do Q'
	)
};

/**
 * COLOR SPACES
 */
jsPDFAPI.color_spaces = {
	DEVICE_RGB:'DeviceRGB',
	DEVICE_GRAY:'DeviceGray',
	DEVICE_CMYK:'DeviceCMYK',
	CAL_GREY:'CalGray',
	CAL_RGB:'CalRGB',
	LAB:'Lab',
	ICC_BASED:'ICCBased',
	INDEXED:'Indexed',
	PATTERN:'Pattern',
	SEPERATION:'Seperation',
	DEVICE_N:'DeviceN'
```

```
};

/**
 * DECODE METHODS
 */
jsPDFAPI.decode = {
	DCT_DECODE:'DCTDecode',
	FLATE_DECODE:'FlateDecode',
	LZW_DECODE:'LZWDecode',
	JPX_DECODE:'JPXDecode',
	JBIG2_DECODE:'JBIG2Decode',
	ASCII85_DECODE:'ASCII85Decode',
	ASCII_HEX_DECODE:'ASCIIHexDecode',
	RUN_LENGTH_DECODE:'RunLengthDecode',
	CCITT_FAX_DECODE:'CCITTFaxDecode'
};

/**
 * IMAGE COMPRESSION TYPES
 */
jsPDFAPI.image_compression = {
	NONE: 'NONE',
	FAST: 'FAST',
	MEDIUM: 'MEDIUM',
	SLOW: 'SLOW'
};
```

```
        jsPDFAPI.sHashCode = function(str) {
                return Array.prototype.reduce &&
str.split("").reduce(function(a,b){a=((a<<5)-a)+b.charCodeAt(0);return a&a},0);
        };

        jsPDFAPI.isString = function(object) {
                return typeof object === 'string';
        };

        /**
         * Strips out and returns info from a valid base64 data URI
         * @param {String[dataURI]} a valid data URI of format 'data:[<MIME-
type>][;base64],<data>'
         * @returns an Array containing the following
         * [0] the complete data URI
         * [1] <MIME-type>
         * [2] format - the second part of the mime-type i.e 'png' in 'image/png'
         * [4] <data>
         */
        jsPDFAPI.extractInfoFromBase64DataURI = function(dataURI) {
                return /^data:([\w]+?\/([\w]+?));base64,(.+?)$/g.exec(dataURI);
        };

        /**
         * Check to see if ArrayBuffer is supported
         */
        jsPDFAPI.supportsArrayBuffer = function() {
                return typeof ArrayBuffer !== 'undefined' && typeof Uint8Array !==
'undefined';
        };

        /**
         * Tests supplied object to determine if ArrayBuffer
         * @param {Object[object]}
         */
        jsPDFAPI.isArrayBuffer = function(object) {
                if(!this.supportsArrayBuffer())
             return false;
                return object instanceof ArrayBuffer;
        };

        /**
         * Tests supplied object to determine if it implements the ArrayBufferView
(TypedArray) interface
         * @param {Object[object]}
         */
        jsPDFAPI.isArrayBufferView = function(object) {
                if(!this.supportsArrayBuffer())
             return false;
                if(typeof Uint32Array === 'undefined')
                        return false;
                return (object instanceof Int8Array ||
                                object instanceof Uint8Array ||
                                (typeof Uint8ClampedArray !== 'undefined' && object
instanceof Uint8ClampedArray) ||
                object instanceof Int16Array ||
                object instanceof Uint16Array ||
                object instanceof Int32Array ||
                object instanceof Uint32Array ||
                object instanceof Float32Array ||
                object instanceof Float64Array );
    };

    /**
     * Exactly what it says on the tin
     */
    jsPDFAPI.binaryStringToUint8Array = function(binary_string) {
        /*
         * not sure how efficient this will be will bigger files. Is there a native method?
         */
        var len = binary_string.length;
       var bytes = new Uint8Array( len );
       for (var i = 0; i < len; i++) {
          bytes[i] = binary_string.charCodeAt(i);
       }
       return bytes;
    };

    /**
     * @see this discussion
     * http://stackoverflow.com/questions/6965107/converting-between-strings-and-arraybuffers
     *
     * As stated, i imagine the method below is highly inefficent for large files.
     *
     * Also of note from Mozilla,
     *
     * "However, this is slow and error-prone, due to the need for multiple conversions (especially if the binary data is not actually byte-format data, but, for example, 32-bit integers or floats)."
     *
     * https://developer.mozilla.org/en-US/Add-ons/Code_snippets/StringView
     *
     * Although i'm strugglig to see how StringView solves this issue? Doesn't appear to be a direct method for conversion?
     *
     * Async method using Blob and FileReader could be best, but i'm not sure how to fit it into the flow?
     */
    jsPDFAPI.arrayBufferToBinaryString = function(buffer) {
        if(this.isArrayBuffer(buffer))
            buffer = new Uint8Array(buffer);

       var binary_string = '';
       var len = buffer.byteLength;
       for (var i = 0; i < len; i++) {
          binary_string += String.fromCharCode(buffer[i]);
       }
       return binary_string;
       /*
```

```
             * Another solution is the method below - convert array buffer straight to base64
and then use atob
             */
                  //return atob(this.arrayBufferToBase64(buffer));
         };


         /**
          * Converts an ArrayBuffer directly to base64
          *
          * Taken from here
          *
          * http://jsperf.com/encoding-xhr-image-data/31
          *
          * Need to test if this is a better solution for larger files
          *
          */
         jsPDFAPI.arrayBufferToBase64 = function(arrayBuffer) {
                  var base64    = ''
                  var encodings =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'


                  var bytes         = new Uint8Array(arrayBuffer)
                  var byteLength    = bytes.byteLength
                  var byteRemainder = byteLength % 3
                  var mainLength    = byteLength - byteRemainder


                  var a, b, c, d
                  var chunk
```

```
        // Main loop deals with bytes in chunks of 3
        for (var i = 0; i < mainLength; i = i + 3) {
            // Combine the three bytes into a single integer
            chunk = (bytes[i] << 16) | (bytes[i + 1] << 8) | bytes[i + 2]

            // Use bitmasks to extract 6-bit segments from the triplet
            a = (chunk & 16515072) >> 18 // 16515072 = (2^6 - 1) << 18
            b = (chunk & 258048)   >> 12 // 258048   = (2^6 - 1) << 12
            c = (chunk & 4032)     >>  6 // 4032     = (2^6 - 1) << 6
            d = chunk & 63               // 63       = 2^6 - 1

            // Convert the raw binary segments to the appropriate ASCII
encoding
            base64 += encodings[a] + encodings[b] + encodings[c] +
encodings[d]
        }

        // Deal with the remaining bytes and padding
        if (byteRemainder == 1) {
            chunk = bytes[mainLength]

            a = (chunk & 252) >> 2 // 252 = (2^6 - 1) << 2

            // Set the 4 least significant bits to zero
            b = (chunk & 3)   << 4 // 3   = 2^2 - 1

            base64 += encodings[a] + encodings[b] + '=='
        } else if (byteRemainder == 2) {
            chunk = (bytes[mainLength] << 8) | bytes[mainLength + 1]

            a = (chunk & 64512) >> 10 // 64512 = (2^6 - 1) << 10
            b = (chunk & 1008)  >>  4 // 1008  = (2^6 - 1) << 4

            // Set the 2 least significant bits to zero
            c = (chunk & 15)    <<  2 // 15    = 2^4 - 1

            base64 += encodings[a] + encodings[b] + encodings[c] + '='
        }

        return base64
    };

    jsPDFAPI.createImageInfo = function(data, wd, ht, cs, bpc, f, imageIndex, alias, dp,
trns, pal, smask) {
        var info = {
                alias:alias,
                w : wd,
                h : ht,
                cs : cs,
                bpc : bpc,
                i : imageIndex,
                data : data
                // n: objectNumber will be added by putImage code
            };
        if(f) info.f = f;
        if(dp) info.dp = dp;
        if(trns) info.trns = trns;
        if(pal) info.pal = pal;
        if(smask) info.smask = smask;

        return info;
    };

    jsPDFAPI.addImage = function(imageData, format, x, y, w, h, alias, compression, rotation) {
        'use strict'

        if(typeof format !== 'string') {
            var tmp = h;
            h = w;
            w = y;
            y = x;
            x = format;
            format = tmp;
        }

        if (typeof imageData === 'object' && !isDOMElement(imageData) && "imageData" in imageData) {
            var options = imageData;

            imageData = options.imageData;
            format = options.format || format;
            x = options.x || x || 0;

            y = options.y || y || 0;

            w = options.w || w;

            h = options.h || h;

            alias = options.alias || alias;

            compression = options.compression || compression;

            rotation = options.rotation || options.angle || rotation;

        }


        if (isNaN(x) || isNaN(y))

        {

            console.error('jsPDF.addImage: Invalid coordinates', arguments);

            throw new Error('Invalid coordinates passed to jsPDF.addImage');

        }


        var images = getImages.call(this), info;


        if (!(info = checkImagesForAlias(imageData, images))) {

            var dataAsBinaryString;


            if(isDOMElement(imageData))

                imageData = createDataURIFromElement(imageData, format,
rotation);


            if(notDefined(alias))

                alias = generateAliasFromData(imageData);


            if (!(info = checkImagesForAlias(alias, images))) {
                if(this.isString(imageData)) {

                    var base64Info =
this.extractInfoFromBase64DataURI(imageData);

                    if(base64Info) {

                        format = base64Info[2];
                        imageData = atob(base64Info[3]);//convert to
binary string

                    } else {

                        if (imageData.charCodeAt(0) === 0x89 &&
                            imageData.charCodeAt(1) === 0x50 &&
                            imageData.charCodeAt(2) === 0x4e &&
                            imageData.charCodeAt(3) === 0x47  )
format = 'png';
                    }
                }
                format = (format || 'JPEG').toLowerCase();

                if(doesNotSupportImageType(format))
                    throw new Error('addImage currently only supports
formats ' + supported_image_types + ', not \''+format+'\'');

                if(processMethodNotEnabled(format))
```

```
                    throw new Error('please ensure that the plugin for \''+format+'\' support is added');

                /**
                 * need to test if it's more efficent to convert all binary strings
                 * to TypedArray - or should we just leave and process as string?
                 */
                if(this.supportsArrayBuffer()) {
                    dataAsBinaryString = imageData;
                    imageData = this.binaryStringToUint8Array(imageData);
                }

                info = this['process' + format.toUpperCase()](
                    imageData,
                    getImageIndex(images),
                    alias,
                    checkCompressValue(compression),
                    dataAsBinaryString
                );

                if(!info)
                    throw new Error('An unkwown error occurred whilst processing the image');
            }
        }
        writeImageToPDF.call(this, x, y, w, h, info, info.i, images);

        return this
    };

    /**
     * JPEG SUPPORT
     **/

    //takes a string imgData containing the raw bytes of
    //a jpeg image and returns [width, height]
    //Algorithm from: http://www.64lines.com/jpeg-width-height
    var getJpegSize = function(imgData) {
        'use strict'
        var width, height, numcomponents;
        // Verify we have a valid jpeg header 0xff,0xd8,0xff,0xe0,?,?,'J','F','I','F',0x00
        if (!imgData.charCodeAt(0) === 0xff ||
            !imgData.charCodeAt(1) === 0xd8 ||
            !imgData.charCodeAt(2) === 0xff ||
            !imgData.charCodeAt(3) === 0xe0 ||
            !imgData.charCodeAt(6) === 'J'.charCodeAt(0) ||
            !imgData.charCodeAt(7) === 'F'.charCodeAt(0) ||
            !imgData.charCodeAt(8) === 'I'.charCodeAt(0) ||
            !imgData.charCodeAt(9) === 'F'.charCodeAt(0) ||
            !imgData.charCodeAt(10) === 0x00) {
                throw new Error('getJpegSize requires a binary string jpeg
file')
        }
        var blockLength = imgData.charCodeAt(4)*256 + imgData.charCodeAt(5);
        var i = 4, len = imgData.length;
        while ( i < len ) {
            i += blockLength;
            if (imgData.charCodeAt(i) !== 0xff) {
                throw new Error('getJpegSize could not find the size of the
image');
            }
            if (imgData.charCodeAt(i+1) === 0xc0 || //(SOF) Huffman  - Baseline
DCT
              imgData.charCodeAt(i+1) === 0xc1 || //(SOF) Huffman  - Extended
sequential DCT
              imgData.charCodeAt(i+1) === 0xc2 || // Progressive DCT (SOF2)
              imgData.charCodeAt(i+1) === 0xc3 || // Spatial (sequential)
lossless (SOF3)
              imgData.charCodeAt(i+1) === 0xc4 || // Differential sequential
DCT (SOF5)
              imgData.charCodeAt(i+1) === 0xc5 || // Differential progressive
DCT (SOF6)
              imgData.charCodeAt(i+1) === 0xc6 || // Differential spatial (SOF7)
              imgData.charCodeAt(i+1) === 0xc7) {
                height = imgData.charCodeAt(i+5)*256 +
imgData.charCodeAt(i+6);
                width = imgData.charCodeAt(i+7)*256 +
imgData.charCodeAt(i+8);
     numcomponents = imgData.charCodeAt(i+9);
                return [width, height, numcomponents];
            } else {
                i += 2;
                blockLength = imgData.charCodeAt(i)*256 +
imgData.charCodeAt(i+1)
            }
        }
    }
    , getJpegSizeFromBytes = function(data) {

        var hdr = (data[0] << 8) | data[1];

        if(hdr !== 0xFFD8)
            throw new Error('Supplied data is not a JPEG');

        var len = data.length,
            block = (data[4] << 8) + data[5],
            pos = 4,
            bytes, width, height, numcomponents;

        while(pos < len) {
            pos += block;
            bytes = readBytes(data, pos);
            block = (bytes[2] << 8) + bytes[3];
            if((bytes[1] === 0xC0 || bytes[1] === 0xC2) && bytes[0] === 0xFF &&
block > 7) {
                bytes = readBytes(data, pos + 5);
                width = (bytes[2] << 8) + bytes[3];
                height = (bytes[0] << 8) + bytes[1];
     numcomponents = bytes[4];
```

```
                return {width:width, height:height, numcomponents:
numcomponents};
            }

            pos+=2;
        }

        throw new Error('getJpegSizeFromBytes could not find the size of the
image');
    }
    , readBytes = function(data, offset) {
        return data.subarray(offset, offset+ 5);
    };

    jsPDFAPI.processJPEG = function(data, index, alias, compression,
dataAsBinaryString) {
        'use strict'
        var colorSpace = this.color_spaces.DEVICE_RGB,
            filter = this.decode.DCT_DECODE,
            bpc = 8,
            dims;

        if(this.isString(data)) {
            dims = getJpegSize(data);
            return this.createImageInfo(data, dims[0], dims[1], dims[3] == 1 ?
this.color_spaces.DEVICE_GRAY:colorSpace, bpc, filter, index, alias);
        }

        if(this.isArrayBuffer(data))
```

```
            data = new Uint8Array(data);

        if(this.isArrayBufferView(data)) {

            dims = getJpegSizeFromBytes(data);

            // if we already have a stored binary string rep use that
            data = dataAsBinaryString || this.arrayBufferToBinaryString(data);

            return this.createImageInfo(data, dims.width, dims.height,
dims.numcomponents == 1 ? this.color_spaces.DEVICE_GRAY:colorSpace, bpc, filter,
index, alias);
        }

        return null;
    };

    jsPDFAPI.processJPG = function(/*data, index, alias, compression,
dataAsBinaryString*/) {
        return this.processJPEG.apply(this, arguments);
    }

})(jsPDF.API);
(function (jsPDFAPI) {
    'use strict';

    jsPDFAPI.autoPrint = function () {
        'use strict'
				var refAutoPrintTag;


				this.internal.events.subscribe('postPutResources', function () {
					refAutoPrintTag = this.internal.newObject()
						this.internal.write("<< /S/Named /Type/Action /N/Print >>", "endobj");
				});


				this.internal.events.subscribe("putCatalog", function () {
					this.internal.write("/OpenAction " + refAutoPrintTag + " 0" + " R");
				});
				return this;
		};
})(jsPDF.API);
/** ====================================================================
 * jsPDF Cell plugin
 * Copyright (c) 2013 Youssef Beddad, youssef.beddad@gmail.com
 *		2013 Eduardo Menezes de Morais, eduardo.morais@usp.br
 *		2013 Lee Driscoll, https://github.com/lsdriscoll
 *		2014 Juan Pablo Gaviria, https://github.com/juanpgaviria
 *		2014 James Hall, james@parall.ax
 *		2014 Diego Casorran, https://github.com/diegocr
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
 * LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 * ====================================================================
 */

(function (jsPDFAPI) {
    'use strict';
    /*jslint browser:true */
    /*global document: false, jsPDF */

    var fontName,
        fontSize,
        fontStyle,
        padding = 3,
        margin = 13,
        headerFunction,
        lastCellPos = { x: undefined, y: undefined, w: undefined, h: undefined, ln: undefined },
        pages = 1,
        setLastCellPosition = function (x, y, w, h, ln) {
            lastCellPos = { 'x': x, 'y': y, 'w': w, 'h': h, 'ln': ln };
        },
        getLastCellPosition = function () {
            return lastCellPos;
        },
        NO_MARGINS = {left:0, top:0, bottom: 0};

    jsPDFAPI.setHeaderFunction = function (func) {
        headerFunction = func;
    };


    jsPDFAPI.getTextDimensions = function (txt) {
        fontName = this.internal.getFont().fontName;
        fontSize = this.table_font_size || this.internal.getFontSize();
        fontStyle = this.internal.getFont().fontStyle;
        // 1 pixel = 0.264583 mm and 1 mm = 72/25.4 point
        var px2pt = 0.264583 * 72 / 25.4,
            dimensions,
            text;

        text = document.createElement('font');
        text.id = "jsPDFCell";
        text.style.fontStyle = fontStyle;
        text.style.fontName = fontName;
        text.style.fontSize = fontSize + 'pt';
        text.textContent = txt;

        document.body.appendChild(text);

        dimensions = { w: (text.offsetWidth + 1) * px2pt, h: (text.offsetHeight + 1) * px2pt};

        document.body.removeChild(text);

        return dimensions;
    };

    jsPDFAPI.cellAddPage = function () {
        var margins = this.margins || NO_MARGINS;

        this.addPage();

        setLastCellPosition(margins.left, margins.top, undefined, undefined);
        //setLastCellPosition(undefined, undefined, undefined, undefined, undefined);
        pages += 1;
    };

    jsPDFAPI.cellInitialize = function () {
        lastCellPos = { x: undefined, y: undefined, w: undefined, h: undefined, ln: undefined };
        pages = 1;
    };
```

```
    jsPDFAPI.cell = function (x, y, w, h, txt, ln, align) {
        var curCell = getLastCellPosition();

        // If this is not the first cell, we must change its position
        if (curCell.ln !== undefined) {
            if (curCell.ln === ln) {
                //Same line
                x = curCell.x + curCell.w;
                y = curCell.y;
            } else {
                //New line
                var margins = this.margins || NO_MARGINS;
                if ((curCell.y + curCell.h + h + margin) >= this.internal.pageSize.height - margins.bottom) {
                    this.cellAddPage();
                    if (this.printHeaders && this.tableHeaderRow) {
                        this.printHeaderRow(ln, true);
                    }
                }
                //We ignore the passed y: the lines may have diferent heights
                y = (getLastCellPosition().y + getLastCellPosition().h);

            }
        }

        if (txt[0] !== undefined) {
            if (this.printingHeaderRow) {
                this.rect(x, y, w, h, 'FD');
            } else {
                this.rect(x, y, w, h);
            }
            if (align === 'right') {
                if (txt instanceof Array) {
                    for(var i = 0; i<txt.length; i++) {
                        var currentLine = txt[i];
                        var textSize = this.getStringUnitWidth(currentLine) * this.internal.getFontSize();
                        this.text(currentLine, x + w - textSize - padding, y + this.internal.getLineHeight()*(i+1));
                    }
                }
            } else {
                this.text(txt, x + padding, y + this.internal.getLineHeight());
            }
        }
        setLastCellPosition(x, y, w, h, ln);
        return this;
    };

    /**
     * Return the maximum value from an array
     * @param array
     * @param comparisonFn
     * @returns {*}
     */
    jsPDFAPI.arrayMax = function (array, comparisonFn) {
        var max = array[0],
            i,
            ln,
            item;

        for (i = 0, ln = array.length; i < ln; i += 1) {
            item = array[i];

            if (comparisonFn) {
                if (comparisonFn(max, item) === -1) {
                    max = item;
                }
            } else {
                if (item > max) {
                    max = item;
                }
            }
        }

        return max;
    };

    /**
     * Create a table from a set of data.
     * @param {Integer} [x] : left-position for top-left corner of table
     * @param {Integer} [y] top-position for top-left corner of table
     * @param {Object[]} [data] As array of objects containing key-value pairs corresponding to a row of data.
     * @param {String[]} [headers] Omit or null to auto-generate headers at a performance cost


     * @param {Object} [config.printHeaders] True to print column headers at the top of every page

     * @param {Object} [config.autoSize] True to dynamically set the column widths to match the widest cell value

     * @param {Object} [config.margins] margin values for left, top, bottom, and width

     * @param {Object} [config.fontSize] Integer fontSize to use (optional)

     */


    jsPDFAPI.table = function (x,y, data, headers, config) {

      if (!data) {

        throw 'No data for PDF table';

      }


      var headerNames = [],

        headerPrompts = [],

        header,

        i,

        ln,

        cln,

        columnMatrix = {},

        columnWidths = {},

        columnData,

        column,

        columnMinWidths = [],

        j,
        tableHeaderConfigs = [],
        model,
        jln,
        func,

      //set up defaults. If a value is provided in config, defaults will be overwritten:
         autoSize       = false,
         printHeaders    = true,
         fontSize       = 12,
         margins        = NO_MARGINS;

         margins.width = this.internal.pageSize.width;

      if (config) {
      //override config defaults if the user has specified non-default behavior:
        if(config.autoSize === true) {
            autoSize = true;
        }
        if(config.printHeaders === false) {
            printHeaders = false;
        }
        if(config.fontSize){
            fontSize = config.fontSize;
        }
        if(config.margins){
            margins = config.margins;
        }
```

```
}

/**
 * @property {Number} lnMod
 * Keep track of the current line number modifier used when creating cells
 */
this.lnMod = 0;
lastCellPos = { x: undefined, y: undefined, w: undefined, h: undefined, ln: undefined },
pages = 1;

this.printHeaders = printHeaders;
this.margins = margins;
this.setFontSize(fontSize);
this.table_font_size = fontSize;

// Set header values
if (headers === undefined || (headers === null)) {
    // No headers defined so we derive from data
    headerNames = Object.keys(data[0]);

} else if (headers[0] && (typeof headers[0] !== 'string')) {
    var px2pt = 0.264583 * 72 / 25.4;

    // Split header configs into names and prompts
    for (i = 0, ln = headers.length; i < ln; i += 1) {
        header = headers[i];
        headerNames.push(header.name);
    headerPrompts.push(header.prompt);
    columnWidths[header.name] = header.width *px2pt;
  }

} else {
  headerNames = headers;
}

if (autoSize) {
  // Create a matrix of columns e.g., {column_title: [row1_Record, row2_Record]}
  func = function (rec) {
    return rec[header];
  };

  for (i = 0, ln = headerNames.length; i < ln; i += 1) {
    header = headerNames[i];

    columnMatrix[header] = data.map(
      func
    );

    // get header width
    columnMinWidths.push(this.getTextDimensions(headerPrompts[i] || header).w);
    column = columnMatrix[header];

    // get cell widths
    for (j = 0, cln = column.length; j < cln; j += 1) {
          columnData = column[j];

          columnMinWidths.push(this.getTextDimensions(columnData).w);

        }


        // get final column width

        columnWidths[header] = jsPDFAPI.arrayMax(columnMinWidths);

      }

    }


    // -- Construct the table


    if (printHeaders) {

      var lineHeight = this.calculateLineHeight(headerNames, columnWidths,
headerPrompts.length?headerPrompts:headerNames);


      // Construct the header row

      for (i = 0, ln = headerNames.length; i < ln; i += 1) {

        header = headerNames[i];

        tableHeaderConfigs.push([x, y, columnWidths[header], lineHeight,
String(headerPrompts.length ? headerPrompts[i] : header)]);

      }


      // Store the table header config

      this.setTableHeaderRow(tableHeaderConfigs);


      // Print the header for the start of the table

      this.printHeaderRow(1, false);

    }
    // Construct the data rows
    for (i = 0, ln = data.length; i < ln; i += 1) {
      var lineHeight;
      model = data[i];
      lineHeight = this.calculateLineHeight(headerNames, columnWidths, model);

      for (j = 0, jln = headerNames.length; j < jln; j += 1) {
        header = headerNames[j];
        this.cell(x, y, columnWidths[header], lineHeight, model[header], i + 2,
header.align);
      }
    }
    this.lastCellPos = lastCellPos;
    this.table_x = x;
    this.table_y = y;
    return this;
  };
  /**
   * Calculate the height for containing the highest column
   * @param {String[]} headerNames is the header, used as keys to the data
   * @param {Integer[]} columnWidths is size of each column
   * @param {Object[]} model is the line of data we want to calculate the height of
   */
  jsPDFAPI.calculateLineHeight = function (headerNames, columnWidths, model) {
    var header, lineHeight = 0;
    for (var j = 0; j < headerNames.length; j++) {
      header = headerNames[j];
      model[header] = this.splitTextToSize(String(model[header]), columnWidths[header]
- padding);

      var h = this.internal.getLineHeight() * model[header].length + padding;

      if (h > lineHeight)

        lineHeight = h;

    }

    return lineHeight;

  };


  /**

   * Store the config for outputting a table header

   * @param {Object[]} config

   * An array of cell configs that would define a header row: Each config matches the
config used by jsPDFAPI.cell

   * except the ln parameter is excluded

   */

  jsPDFAPI.setTableHeaderRow = function (config) {

    this.tableHeaderRow = config;

  };


  /**

   * Output the store header row

   * @param lineNumber The line number to output the header at

   */

  jsPDFAPI.printHeaderRow = function (lineNumber, new_page) {

    if (!this.tableHeaderRow) {

      throw 'Property tableHeaderRow does not exist.';

    }
```

```
var tableHeaderCell,
   tmpArray,
   i,
   ln;

this.printingHeaderRow = true;
if (headerFunction !== undefined) {
   var position = headerFunction(this, pages);
   setLastCellPosition(position[0], position[1], position[2], position[3], -1);
}
this.setFontStyle('bold');
var tempHeaderConf = [];
for (i = 0, ln = this.tableHeaderRow.length; i < ln; i += 1) {
   this.setFillColor(200,200,200);

   tableHeaderCell = this.tableHeaderRow[i];
   if (new_page) {
      tableHeaderCell[1] = this.margins && this.margins.top || 0;
      tempHeaderConf.push(tableHeaderCell);
   }
   tmpArray = [].concat(tableHeaderCell);
   this.cell.apply(this, tmpArray.concat(lineNumber));
}
if (tempHeaderConf.length > 0){
   this.setTableHeaderRow(tempHeaderConf);
}
```

```
        this.setFontStyle('normal');
        this.printingHeaderRow = false;
    };

})(jsPDF.API);
/** @preserve
 * jsPDF fromHTML plugin. BETA stage. API subject to change. Needs browser
 * Copyright (c) 2012 Willow Systems Corporation, willow-systems.com
 *               2014 Juan Pablo Gaviria, https://github.com/juanpgaviria
 *               2014 Diego Casorran, https://github.com/diegocr
 *               2014 Daniel Husar, https://github.com/danielhusar
 *               2014 Wolfgang Gassler, https://github.com/woolfg
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
 * LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 * ====================================================================
 */

(function (jsPDFAPI) {
        var clone,
        DrillForContent,
        FontNameDB,
        FontStyleMap,
        TextAlignMap,
        FontWeightMap,
        FloatMap,
        ClearMap,
        GetCSS,
        PurgeWhiteSpace,
        Renderer,
        ResolveFont,
        ResolveUnitedNumber,
        UnitedNumberMap,
        elementHandledElsewhere,
        images,
        loadImgs,
        checkForFooter,
        process,
tableToJson;
clone = (function () {
    return function (obj) {
        Clone.prototype = obj;
        return new Clone()
    };
    function Clone() {}
})();
PurgeWhiteSpace = function (array) {
    var fragment,
    i,
    l,
    lTrimmed,
    r,
    rTrimmed,
    trailingSpace;
    i = 0;
    l = array.length;
    fragment = void 0;
    lTrimmed = false;
    rTrimmed = false;
    while (!lTrimmed && i !== l) {
        fragment = array[i] = array[i].trimLeft();
        if (fragment) {
            lTrimmed = true;
        }
        i++;
```

```
		}
		i = l - 1;
		while (l && !rTrimmed && i !== -1) {
			fragment = array[i] = array[i].trimRight();
			if (fragment) {
				rTrimmed = true;
			}
			i--;
		}
		r = /\s+$/g;
		trailingSpace = true;
		i = 0;
		while (i !== l) {
			fragment = array[i].replace(/\s+/g, " ");
			if (trailingSpace) {
				fragment = fragment.trimLeft();
			}
			if (fragment) {
				trailingSpace = r.test(fragment);
			}
			array[i] = fragment;
			i++;
		}
		return array;
	};
	Renderer = function (pdf, x, y, settings) {
		this.pdf = pdf;
```

```
        this.x = x;
        this.y = y;
        this.settings = settings;
        //list of functions which are called after each element-rendering process
        this.watchFunctions = [];
        this.init();
        return this;
    };
    ResolveFont = function (css_font_family_string) {
        var name,
        part,
        parts;
        name = void 0;
        parts = css_font_family_string.split(",");
        part = parts.shift();
        while (!name && part) {
            name = FontNameDB[part.trim().toLowerCase()];
            part = parts.shift();
        }
        return name;
    };
    ResolveUnitedNumber = function (css_line_height_string) {

        //IE8 issues
        css_line_height_string = css_line_height_string === "auto" ? "0px" : css_line_height_string;
        if (css_line_height_string.indexOf("em") > -1 && !isNaN(Number(css_line_height_string.replace("em", "")))) {
            css_line_height_string =
Number(css_line_height_string.replace("em", "")) * 18.719 + "px";
        }
        if (css_line_height_string.indexOf("pt") > -1 &&
!isNaN(Number(css_line_height_string.replace("pt", "")))) {
            css_line_height_string =
Number(css_line_height_string.replace("pt", "")) * 1.333 + "px";
        }

        var normal,
        undef,
        value;
        undef = void 0;
        normal = 16.00;
        value = UnitedNumberMap[css_line_height_string];
        if (value) {
            return value;
        }
        value = {
            "xx-small"  :  9,
            "x-small"   : 11,
            small       : 13,
            medium      : 16,
            large       : 19,
            "x-large"   : 23,
            "xx-large"  : 28,
            auto        :  0
        }[{ css_line_height_string : css_line_height_string }];
        if (value !== undef) {
            return UnitedNumberMap[css_line_height_string] = value / normal;
        }
        if (value = parseFloat(css_line_height_string)) {
            return UnitedNumberMap[css_line_height_string] = value / normal;
        }
        value = css_line_height_string.match(/([\d\.]+)(px)/);
        if (value.length === 3) {
            return UnitedNumberMap[css_line_height_string] =
parseFloat(value[1]) / normal;
        }
        return UnitedNumberMap[css_line_height_string] = 1;
    };
    GetCSS = function (element) {
        var css,tmp,computedCSSElement;
        computedCSSElement = (function (el) {
            var compCSS;
            compCSS = (function (el) {
                if (document.defaultView &&
document.defaultView.getComputedStyle) {
                    return document.defaultView.getComputedStyle(el,
null);
                } else if (el.currentStyle) {
                    return el.currentStyle;
                } else {
                    return el.style;
                }
            })(el);
            return function (prop) {
                prop = prop.replace(/-\D/g, function (match) {
                    return match.charAt(1).toUpperCase();
                });
                return compCSS[prop];
            };
        })(element);
        css = {};
        tmp = void 0;
        css["font-family"] = ResolveFont(computedCSSElement("font-family")) || "times";
        css["font-style"] = FontStyleMap[computedCSSElement("font-style")] || "normal";
        css["text-align"] = TextAlignMap[computedCSSElement("text-align")] || "left";
        tmp = FontWeightMap[computedCSSElement("font-weight")] || "normal";
        if (tmp === "bold") {
            if (css["font-style"] === "normal") {
                css["font-style"] = tmp;
            } else {
                css["font-style"] = tmp + css["font-style"];
            }
        }
        css["font-size"] = ResolveUnitedNumber(computedCSSElement("font-size")) || 1;
        css["line-height"] = ResolveUnitedNumber(computedCSSElement("line-height")) || 1;
        css["display"] = (computedCSSElement("display") === "inline" ? "inline" :
"block");


        tmp = (css["display"] === "block");
        css["margin-top"]     = tmp &&
ResolveUnitedNumber(computedCSSElement("margin-top"))     || 0;
        css["margin-bottom"]  = tmp &&
ResolveUnitedNumber(computedCSSElement("margin-bottom"))  || 0;
        css["padding-top"]    = tmp &&
ResolveUnitedNumber(computedCSSElement("padding-top"))    || 0;
        css["padding-bottom"] = tmp &&
ResolveUnitedNumber(computedCSSElement("padding-bottom")) || 0;
        css["margin-left"]    = tmp &&
ResolveUnitedNumber(computedCSSElement("margin-left"))    || 0;
        css["margin-right"]   = tmp &&
ResolveUnitedNumber(computedCSSElement("margin-right"))   || 0;
        css["padding-left"]   = tmp &&
ResolveUnitedNumber(computedCSSElement("padding-left"))   || 0;
        css["padding-right"]  = tmp &&
ResolveUnitedNumber(computedCSSElement("padding-right"))  || 0;


        //float and clearing of floats
        css["float"] = FloatMap[computedCSSElement("cssFloat")] || "none";
        css["clear"] = ClearMap[computedCSSElement("clear")] || "none";
        return css;
    };
    elementHandledElsewhere = function (element, renderer, elementHandlers) {
        var handlers,
        i,
        isHandledElsewhere,
        l,
        t;
        isHandledElsewhere = false;
        i = void 0;
        l = void 0;
        t = void 0;
        handlers = elementHandlers["#" + element.id];
        if (handlers) {
            if (typeof handlers === "function") {
                isHandledElsewhere = handlers(element, renderer);
            } else {
                i = 0;
                l = handlers.length;
                while (!isHandledElsewhere && i !== l) {
                    isHandledElsewhere = handlers[i](element, renderer);
                    i++;
                }
            }
        }
        handlers = elementHandlers[element.nodeName];
        if (!isHandledElsewhere && handlers) {
            if (typeof handlers === "function") {
                isHandledElsewhere = handlers(element, renderer);
            } else {
                i = 0;
                l = handlers.length;
                while (!isHandledElsewhere && i !== l) {
                    isHandledElsewhere = handlers[i](element, renderer);
                    i++;
                }
            }
        }
        return isHandledElsewhere;
    };
    tableToJson = function (table, renderer) {
        var data,
        headers,
        i,
        j,
        rowData,
        tableRow,
        table_obj,
        table_with,
        cell,
        l;
        data = [];
        headers = [];
        i = 0;
        l = table.rows[0].cells.length;
        table_with = table.clientWidth;
        while (i < l) {
            cell = table.rows[0].cells[i];
            headers[i] = {
                name : cell.textContent.toLowerCase().replace(/\s+/g, ''),
                prompt : cell.textContent.replace(/\r?\n/g, ''),
                width : (cell.clientWidth / table_with) *
renderer.pdf.internal.pageSize.width
            };
            i++;
        }
        i = 1;
        while (i < table.rows.length) {
            tableRow = table.rows[i];
            rowData = {};
            j = 0;
            while (j < tableRow.cells.length) {
                rowData[headers[j].name] =
tableRow.cells[j].textContent.replace(/\r?\n/g, '');
                j++;
            }
            data.push(rowData);
            i++;
        }
        return table_obj = {
            rows : data,
            headers : headers
        };
    };
    var SkipNode = {
        SCRIPT   : 1,
        STYLE    : 1,
        NOSCRIPT : 1,
    OBJECT   : 1,
    EMBED    : 1,
    SELECT   : 1
};
var listCount = 1;
DrillForContent = function (element, renderer, elementHandlers) {
    var cn,
    cns,
    fragmentCSS,
    i,
    isBlock,
    l,
    px2pt,
    table2json,
    cb;
    cns = element.childNodes;
    cn = void 0;
    fragmentCSS = GetCSS(element);
    isBlock = fragmentCSS.display === "block";
    if (isBlock) {
        renderer.setBlockBoundary();
        renderer.setBlockStyle(fragmentCSS);
    }
    px2pt = 0.264583 * 72 / 25.4;
    i = 0;
    l = cns.length;
    while (i < l) {
            cn = cns[i];
            if (typeof cn === "object") {

                //execute all watcher functions to e.g. reset floating
                renderer.executeWatchFunctions(cn);

                /*** HEADER rendering **/
                if (cn.nodeType === 1 && cn.nodeName === 'HEADER') {
                    var header = cn;
                    //store old top margin
                    var oldMarginTop = renderer.pdf.margins_doc.top;
                    //subscribe for new page event and render header first on every page
                    renderer.pdf.internal.events.subscribe('addPage', function (pageInfo) {
                        //set current y position to old margin
                        renderer.y = oldMarginTop;
                        //render all child nodes of the header element
                        DrillForContent(header, renderer, elementHandlers);
                        //set margin to old margin + rendered header + 10 space to prevent overlapping
                        //important for other plugins (e.g. table) to start rendering at correct position after header
                        renderer.pdf.margins_doc.top = renderer.y + 10;
                        renderer.y += 10;
                    }, false);
                }
                if (cn.nodeType === 8 && cn.nodeName === "#comment") {
                    if (~cn.textContent.indexOf("ADD_PAGE")) {
                        renderer.pdf.addPage();
                        renderer.y = renderer.pdf.margins_doc.top;
                    }

                } else if (cn.nodeType === 1 && !SkipNode[cn.nodeName]) {
                    /*** IMAGE RENDERING ***/
                    var cached_image;
                    if (cn.nodeName === "IMG") {
                        var url = cn.getAttribute("src");
                        cached_image =
images[renderer.pdf.sHashCode(url) || url];
                    }
                    if (cached_image) {
                        if ((renderer.pdf.internal.pageSize.height -
renderer.pdf.margins_doc.bottom < renderer.y + cn.height) && (renderer.y >
renderer.pdf.margins_doc.top)) {
                            renderer.pdf.addPage();
                            renderer.y =
renderer.pdf.margins_doc.top;
                            //check if we have to set back some
values due to e.g. header rendering for new page
                            renderer.executeWatchFunctions(cn);
                        }

                        var imagesCSS = GetCSS(cn);
                        var imageX = renderer.x;
                        var fontToUnitRatio = 12 /
renderer.pdf.internal.scaleFactor;


                        //define additional paddings, margins which
have to be taken into account for margin calculations
                        var additionalSpaceLeft = (imagesCSS["margin-
left"] + imagesCSS["padding-left"])*fontToUnitRatio;
                        var additionalSpaceRight =
(imagesCSS["margin-right"] + imagesCSS["padding-right"])*fontToUnitRatio;
                        var additionalSpaceTop = (imagesCSS["margin-
top"] + imagesCSS["padding-top"])*fontToUnitRatio;
                        var additionalSpaceBottom =
(imagesCSS["margin-bottom"] + imagesCSS["padding-bottom"])*fontToUnitRatio;


                        //if float is set to right, move the image to the
right border
                        //add space if margin is set
                        if (imagesCSS['float'] !== undefined &&
imagesCSS['float'] === 'right') {
                            imageX += renderer.settings.width -
cn.width - additionalSpaceRight;
                        } else {
                            imageX +=  additionalSpaceLeft;
                        }


                        renderer.pdf.addImage(cached_image,
imageX, renderer.y + additionalSpaceTop, cn.width, cn.height);
                        cached_image = undefined;
                        //if the float prop is specified we have to float
the text around the image
```

```
                        if (imagesCSS['float'] === 'right' ||
imagesCSS['float'] === 'left') {
                            //add functiont to set back coordinates
after image rendering

    renderer.watchFunctions.push((function(diffX , thresholdY, diffWidth, el) {
                                //undo drawing box adaptions
which were set by floating
                                if (renderer.y >= thresholdY) {
                                    renderer.x += diffX;
                                    renderer.settings.width
+= diffWidth;
                                    return true;
                                } else if(el && el.nodeType ===
1 && !SkipNode[el.nodeName] && renderer.x+el.width > (renderer.pdf.margins_doc.left +
renderer.pdf.margins_doc.width)) {
                                    renderer.x += diffX;
                                    renderer.y = thresholdY;
                                    renderer.settings.width
+= diffWidth;
                                    return true;
                                } else {
                                    return false;
                                }
                            }).bind(this, (imagesCSS['float'] ===
'left') ? -cn.width-additionalSpaceLeft-additionalSpaceRight : 0,
renderer.y+cn.height+additionalSpaceTop+additionalSpaceBottom, cn.width));
                            //reset floating by clear:both divs
                            //just set cursorY after the floating
element
```

```
    renderer.watchFunctions.push((function(yPositionAfterFloating, pages, el) {
                                if (renderer.y <
yPositionAfterFloating && pages === renderer.pdf.internal.getNumberOfPages()) {
                                    if (el.nodeType === 1 &&
GetCSS(el).clear === 'both') {
                                        renderer.y =
yPositionAfterFloating;
                                        return true;
                                    } else {
                                        return false;
                                    }
                                } else {
                                    return true;
                                }
                            }).bind(this, renderer.y+cn.height,
renderer.pdf.internal.getNumberOfPages()));

                            //if floating is set we decrease the
available width by the image width
                            renderer.settings.width -=
cn.width+additionalSpaceLeft+additionalSpaceRight;
                            //if left just add the image width to the
X coordinate
                            if (imagesCSS['float'] === 'left') {
                                renderer.x +=
cn.width+additionalSpaceLeft+additionalSpaceRight;
                            }
                        } else {
                        //if no floating is set, move the rendering
cursor after the image height
```

```
                            renderer.y += cn.height +
additionalSpaceBottom;
                        }

                    /*** TABLE RENDERING ***/
                    } else if (cn.nodeName === "TABLE") {
                        table2json = tableToJson(cn, renderer);
                        renderer.y += 10;
                        renderer.pdf.table(renderer.x, renderer.y,
table2json.rows, table2json.headers, {
                            autoSize : false,
                            printHeaders : true,
                            margins : renderer.pdf.margins_doc
                        });
                        renderer.y = renderer.pdf.lastCellPos.y +
renderer.pdf.lastCellPos.h + 20;
                    } else if (cn.nodeName === "OL" || cn.nodeName ===
"UL") {
                        listCount = 1;
                        if (!elementHandledElsewhere(cn, renderer,
elementHandlers)) {
                            DrillForContent(cn, renderer,
elementHandlers);
                        }
                        renderer.y += 10;
                    } else if (cn.nodeName === "LI") {
                        var temp = renderer.x;
                        renderer.x += cn.parentNode.nodeName ===
"UL" ? 22 : 10;

                        renderer.y += 3;
                        if (!elementHandledElsewhere(cn, renderer,
elementHandlers)) {
                            DrillForContent(cn, renderer,
elementHandlers);
                        }
                        renderer.x = temp;
                    } else if (cn.nodeName === "BR") {
                        renderer.y += fragmentCSS["font-size"] *
renderer.pdf.internal.scaleFactor;
                    } else {
                        if (!elementHandledElsewhere(cn, renderer,
elementHandlers)) {
                            DrillForContent(cn, renderer,
elementHandlers);
                        }
                    }
                } else if (cn.nodeType === 3) {
                    var value = cn.nodeValue;
                    if (cn.nodeValue && cn.parentNode.nodeName ===
"LI") {
                        if (cn.parentNode.parentNode.nodeName ===
"OL") {
                            value = listCount++ + '. ' + value;
                        } else {
                            var fontPx = fragmentCSS["font-size"] *
16;
                            var radius = 2;
                            if (fontPx > 20) {
                                radius = 3;
                            }
                            cb = function (x, y) {
                                this.pdf.circle(x, y, radius, 'FD');
                            };
                        }
                    }
                    renderer.addText(value, fragmentCSS);
                } else if (typeof cn === "string") {
                    renderer.addText(cn, fragmentCSS);
                }
            }
            i++;
        }
```