ドキュメントはE●celかW●rd...まあ、PDFも可

はるか昔、太古の時代には、顧客への納入ドキュメントは手書きでした。
仕様変更など出ようものなら、ホワイトで消して書き直します。
プログラムを直すより遥かに手がかかります。
仕様変更が嫌いな、最大の理由です(^_^;

恐竜が絶滅した頃、ワードプロセッサという新しい種が現れました。
めっちゃ便利でしたが、やはり紙に出力して納品するので、手書きの頃より遥かにページ数が増えました。
もう殆ど、ファイル何冊納品するかを他社と競っていたような...
仕様変更など出ようものなら、全ページ出力し直して、全ページにページ数をスタンプして、穴を開けて綴じます。
プログラムを直すより遥かに手がかかります。
仕様変更が嫌いな、最大の理由です(^_^;

人類が出現した頃、顧客側でもワープロやスプレッドシートを表示することができるようになり、紙での納品は無くなりました(コンサルは、まだ棚いっぱいのファイルを収めていましたが。『全部読めってか?無理だろう』)。
しかし、ワープロやスプレッドシートで作成したファイルは、基本的に差分が取れないためバージョン管理がしにくい(できない)という、致命的な欠点があります。
そのせいで、変更履歴をコツコツと作成する羽目になります(電子データを使っているにもかかわらずです)。
プログラムを直すより遥かに手がかかります。
仕様変更が嫌いな、最大の理由です(^_^;

そして、やっと最近になってPDFでの納入ができるようになりました(「えっ今更」っと思ったあなた、あなたは幸せ者です)。
PDFでも、差分が取れないことは変わりませんが、なんと言っても、いろいろなソースから比較的きれいなPDFが作成できます。
特に、HTMLで作成したコンテンツからWebブラウザで(印刷処理で)簡単にPDFが出力できるようになったのは、非常に嬉しいところです。
HTMLはテキストなので、簡単に差分がとれます(Texとか思ったあなた、研究頑張ってください)。
それに、JavaScriptという強力な手段があるので、編集をサポートする機能を簡単に実現できます。
このコンテンツは、できるだけ編集内容を簡潔にして、ページ単位のドキュメントを作成ことを目的としています。
ツールの使用方法を説明している体裁ですが、ソースを見ていただければ多分何をやっているか理解できると思います。
興味があれば、参考にしてください。

1. できるだけ簡単に、ページ単位のドキュメントを作成する

こちらから、サンプルコンテンツを表示できます。...
下記は、そのソースです。
body 要素の直下を section 要素とし、その中にドキュメントの内容を記述します。
クラス属性に、”break_always” を記述すると、その直後で改ページを行います。
改ページの指示が無い場合、section 要素に指定されたページサイズに合わせて改ページを行います。

ページ単位のドキュメント用HTML

```
1:<body>
2: <section>
3:  <h1>ドキュメントタイトル</h1>
4:  <p>ここに、このドキュメントの概要を記述します。
5:  <h2>サブタイトル、問題点</h2>
6:  <p>ここに、考察する問題点を記述します。
7:  <h3>問題点其の壱</h3>
8:  <h4>（１）一点目</h4>
9:  <p>一点目の問題点
10:  <h3>問題点其の弐</h3>
11:  <p>二点目の問題点
12:  <table class=”break_always”>
13:   <tr><td>ここに表</td><td>備考</td></tr>
14:   <tr><td>ここに表</td><td>備考</td></tr>
15:  </table>
16:  <h2>サブタイトル、解決策</h2>
17:  <p>ここに、問題点の解決策を記述します。
18:  <h3>解決策其の壱</h3>
19:  <object type=”image/svg+xml” data=”sample.svg”></object>
20:  <h3>解決策其の弐</h3>
21:  <p>解決策
22: </section>
23:</body>
```

下記が、ヘッダ部でスクリプトとスタイルシートをロードしている箇所です。
必要な処理は、jsdoc.jsの中で全て行うため、ロード以外の処理は不要です。

ページ単位のドキュメント用HTML

```
1: <script type=”text/javascript”
2:  src=”../jslib/00.23.00.00/jsdoc.js”></script>
3: <link rel=”stylesheet”
4:  href=”../jslib/00.23.00.00/jsdoc.css” type=”text/css” />
```

1-1. 機能の紹介

(1)タイトルへの項番付与とインデント

h2 要素と h3 要素に項番を付与し、それ以降の要素をインデントさせます。
h2 要素の項番は、コンテンツ内で1から順にカウントアップしたものを付与します。
h3 要素の項番は、 h2 要素の出現毎に1から順にカウントアップしたものを付与します。
h2 要素の項番を、1以外から始める場合は、jsdoc.jsロード後に下記の様に設定してください。下記例では、項番を4から始めます。

項番の初期値を変更する

```
1:    <script type="text/javascript"
2:        src="../jslib/00.23.00.00/jsdoc.js"></script>
3:    <script type="text/javascript">
4:        com.yscjp.jsapp.jsdoc.chapterInit = 4;
5:    </script>
```

インデントは、デフォルトでは h2 要素の後に出現する要素の左側の開始位置を 6mm 右に移動し、 h3 要素の後に出現する要素の左側の開始位置を 12mm 右に移動させます。
デフォルト値は、下記のように定義されているので、これを書き換えることで変更することが出来ます。

インデントのデフォルト値

```
1:    com.yscjp.jsapp.jsdoc.indentUnit = 'mm';
2:    com.yscjp.jsapp.jsdoc.indentSizeH2 = 6;
3:    com.yscjp.jsapp.jsdoc.indentSizeH3 = 12;
```

単位(indentUnit)は、CSSに設定できる単位を指定してください。
変更を行う箇所は、上記の項番の初期値を変更するケースと同じです。

(2)ページ単位の分割と、ページ番号の付与

クラス属性に ”page” が指定されている section 要素が、ページの単位となります。
この要素に指定されているCSSの内容から、1ページに表示可能なサイズ(ピクセル単位)を取得して、自動改ページを行います。
デフォルトは、高さが 297mm 、幅が 210mm のA4縦で、余白が上 20mm 、下 10mm 、左右が 15mm となっています。
変更する場合は、jsdoc.cssのロード後に、 section 要素のパッディングを変更してください。
当ドキュメントでは、下記設定で下の余白を 15mm に変更しています。

ページ余白の変更

```
1:     section {
2:         padding-bottom: 15mm;
3:     }
```

改ページしたページ数に基づいて、ページ番号を付与します。
ページ番号は、以下のようなCSSの定義がなされています。位置の変更などは、jsdoc.cssのロード後にCSSの定義を再設定してください。

ページ番号のCSS

```
1:.page_count {
2:     text-align: center;
3:     position: absolute;
4:     width: 100%;
5:     bottom: 5mm;
6:     left: 0;
7:}
```

(3)ヘッダ／フッタの追加

このドキュメントの上部と下部に太い線が表示され、上部には当社のドメインが表示されていると思います。
このような、ヘッダとフッタを各ページに表示する機能です。
ヘッダとフッタは、下記のように section の外に、com_yscjp_jsapp_jsdoc_header (ヘッダの場合)か com_yscjp_jsapp_jsdoc_footer (フッタの場合)をIDに指定し、記述します。
これらのIDが指定されている要素は、非表示となります。

ヘッダとフッタ

```
1:    <body>
2:    <div id="com_yscjp_jsapp_jsdoc_header">
3:        <img src="../../images/yscjp_com.png"
4:            alt="YSCJP.com" height="17" width="126" />
5:    </div>
6:    <div id="com_yscjp_jsapp_jsdoc_footer"></div>
7:    <section>
```

ヘッダやフッタは、上下に固定する様にスタイルシートで指定されています。position は必ず absolute のままとしてください。
このドキュメントのヘッダとフッタのスタイルは、以下のように指定されています。
横幅は、デフォルトで画面いっぱい(width: 100%)となっているので、コンテンツの幅に合わせる場合は、width に絶対値を指定する必要があります。

ヘッダとフッタのCSS

```
1:    .com_yscjp_jsapp_jsdoc_header {
2:        margin: 10mm 15mm 0 15mm;
3:        width: 180mm;
4:        border-bottom: 1mm solid black;
5:    }
6:    .com_yscjp_jsapp_jsdoc_footer {
7:        margin: 0 15mm 11mm 15mm;
8:        width: 180mm;
9:        border-top: 1mm solid black;
10:   }
```

(4)テーブル項番の自動採番

テーブルの先頭列に、自動的に項番を振る機能です。
table 要素のクラス属性に item_number を指定することで、項番を振ることができます。

項番	項目 1	項目 2	備考
1	01234	4567	
2	01234	4567	
3	01234	4567	

表1. テーブルに項番を振るサンプル

(5)図表番号の自動採番

caption 要素と figcaption 要素の内容に、それぞれ『表n.』や『図n.』を付与する機能です。n の部分は、1から始まり、ドキュメント全体でカウントアップされる数字です。

abcdef	01234	4567	
ghijkl	56789	****	

表2. キャプションに表番を付与するサンプル

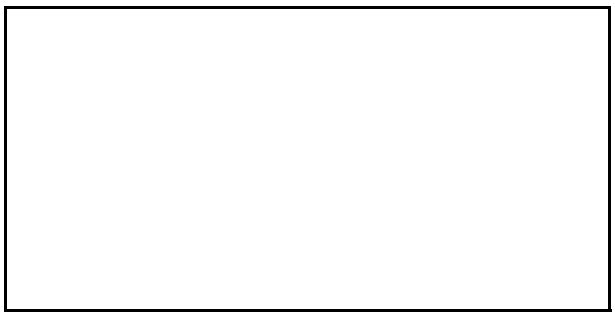

図1. キャプションに図番を付与するサンプル

上記『表2.』とか上記『図1.』のように、 table 要素や figure 要素のIDを指定して、図表番を文章中に埋め込むことも出来ます。
指定例は、以下のとおりです。

図表番を文章中に埋め込むHTML

```
1:    <span data-com_yscjp_targetId="table_sample_2"></span>
2:    <span data-com_yscjp_targetId="figure_sample_1"></span>
```

1-2. 注意点

(1)要素のposition

section 要素の子要素には、CSSの position に、absolute や fixed は指定できません。
指定した場合、正しく表示されなくなります。

(2)テーブルの幅

テーブルの幅に100%を指定すると、正しく表示されません。
横幅いっぱいにテーブルを表示する場合は、クラス属性に h2_table 又は h3_table を指定してください。

サンプルの表	備考
横幅いっぱいに表示する	備考

h2 要素と同じネストに表示するテーブルの場合は h2_table 、h3要素と同じネストに表示するテーブルの場合は h3_table となります。
上記テーブルは、 class="h3_table" が指定されています。

ページ単位のドキュメント用HTML

```
1:    <table class="h3_table">
2:        <tr><td>サンプルの表</td><td>備考</td></tr>
3:        <tr><td>横幅いっぱいに表示する</td><td>備考</td></tr>
4:    </table>
```

h2_table や h3_table の定義内容は、デフォルトの文字サイズやインデントサイズを変更しない前提のものです。デフォルトから変更した場合は、適宜定義し直してください。

(3)スクリプトの呼び出し順序

jsdoc.jsは、内部で window の load イベントにハンドラを設定して、処理を呼び出しています。
jsdoc.jsの実行が終了した後に、要素のサイズやDOMの構成が変更された場合、インデントや自動ページ替えの結果が正しく表示されない可能性があります。
要素のサイズやDOMの構成が変更されるような処理は、jsdoc.jsの実行前に処理が終了するようにしてください。

1-3. 将来の夢(^_^;

どう見ても、スマホ対応できそうですよね。
目次とかを自動で作成することも、それほど難しくはないのですが、あまり必要性を感じないというか...
更新履歴とかは、やっぱり欲しいかな。
テーブルの width: 100%; とかには、なんとか対応したいのですが、どうすればよいのか今の所不明。
ページ単位での表示と、通常のWeb形式での表示を簡単に切り替えるようなインターフェイスの追加。
ページ番号や、各タイトルの前に付与する項番の編集箇所を切り出して、簡単にオーバーライドできるようにする。