Valence

Valenceにおける帳票出力について

株式会社ミガロ.
RAD事業部技術支援課
尾崎　浩司

略　歴

生年月日:1973年8月16日
最終学歴:1996年　三重大学 工学部卒業
ミガロ入社年月:1999年10月　株式会社ミガロ. 入社
社内経歴:1999年10月　システム事業部配属
2013年04月　RAD事業部配属

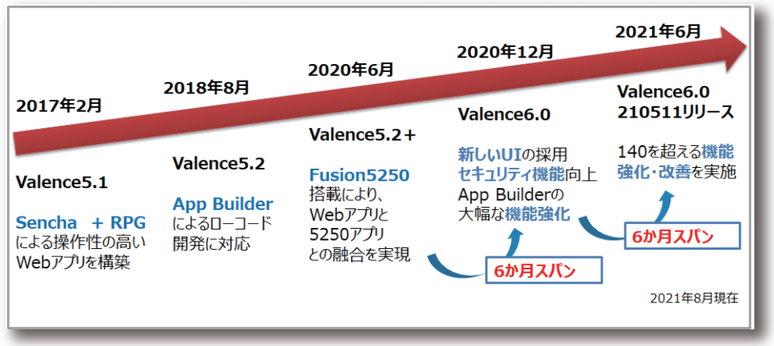
現在の仕事内容:

Delphi/400を中心としたテクニカルサポート対応や製品セミナーの講師などを担当している。

1.はじめに

Valenceは、IBM iに「最高のユーザーエクスペリエンスをもたらす」ことをコンセプトに、米CNX社が開発し、2008年より販売している製品である。2017年より日本国内での製品販売と技術サポートを当社ミガロ. が担当している。取り扱いを開始した2017年当時からIBM iを使用してモダンなWebアプリが構築できる環境として好評であったが、2018年に登場したValence5.2でローコード開発機能App Builderが追加された事により、飛躍的にアプリの開発生産性が向上した。さらに2020年以降はユーザーの要望を取り入れながら、短いスパンでバージョンアップを行っており、機能が強化されている。【図1】

図 1　Valence バージョンアップの変遷



近年Valenceは順調なバージョンアップにより、IBM i Web化開発ツールとして、特にUI(画面)の最適化が進んでいるが、最近サポートへの問合せで多いのが、Valenceにおける帳票出力についてである。一般的にはValenceは画面を最新化するもので、帳票出力の機能が無いと思われているかもしれないが、最新のValence6.0ではPDF帳票を出力する事が可能である。本稿では、ValenceからPDF帳票を出力する手法について紹介する。

2. Valenceから帳票を出力する従来の方法

Valenceには、従来よりRPGを使用して独自ロジックを追加できるRPG ToolKitというAPIが用意されている。このToolKitの活用例については、2019年度テクニカルレポート『「Valence App Builder」RPG連携テクニック』で詳しく紹介しているので、そちらを参照してほしい。このToolKitの中には、動的にPDFを作成する為のAPIも用意されている。このAPIを使用した例がサンプルプログラム1である。これは、Formウィジェット上で商品カテゴリーを選択し、選択したカテゴリーに合致する商品マスターのデータを一覧形式でPDFファイルに出力する処理である。【図2】

図 2　サンプルプログラム1:RPG ToolKitの例

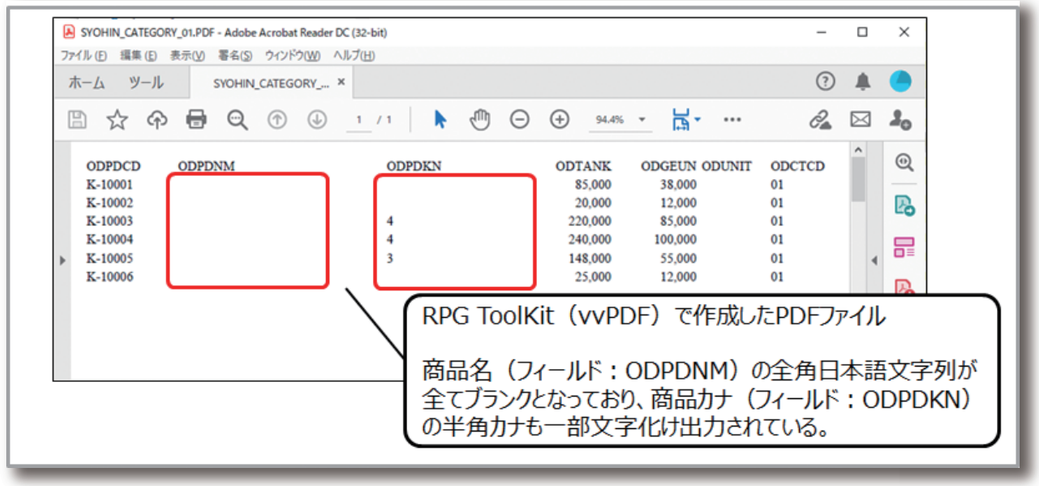
RPGコーディング例は、**【ソース1】**となる。

ソース1　RPG ToolKitを使用したPDF作成（TEC21PG10）

```
0001.00      /copy qcpylesrc,vvHspec
0002.00     ** ------------------------------------------------------------
0003.00     **          テクニカルレポート2021
0004.00     **          TEC21PG10：カテゴリ別商品情報出力
0005.00     ** ------------------------------------------------------------
0006.00     ** ------------------------------------------------------------
0007.00      /define includePDF
0008.00      /include qcpylesrc,vvNabBtn                    1-①
0009.00     ** ------------------------------------------------------------
0010.00     ** program start
0011.00     ** ------------------------------------------------------------
0012.00      /free
0013.00        Initialize();
0014.00        Process();
0015.00        CleanUp();
0016.00        *inlr=*on;
0017.00      /end-free
0018.00     ** ------------------------------------------------------------
0019.00     p Process         b
0020.00     d                 pi
0021.00     d pdfDoc          s                   like(document)
0022.00     D TMPPATH         S             60A
0023.00     D VCATG           S              2A
0024.00     D SQLSTR          S            256A
0025.00      /free
0026.00        //フォーム上で入力された値を取得
0027.00        VCATG = GetFormChar('CATEGO');  //----カテゴリＣＤ
0028.00
0029.00        //データの取得(商品マスタから指定されたカテゴリを抽出)
0030.00        SQLSTR = 'SELECT * FROM MPRODP '                1-②
0031.00               + 'WHERE ODCTCD = ''' + VCATG + ''' '
0032.00               + 'ORDER BY ODCTCD, ODPDCD';
0033.00
0034.00        //PDFファイル保存先パスの指定
0035.00        TMPPATH = vvUtility_getValenceSetting('TEMP_PATH');
0036.00
0037.00        //データ取得結果を元に動的にPDFを作成
0038.00        vvPDF.path = %trim(TMPPATH) + 'TEMP_MPRODP.pdf';   1-③
0039.00        pdfDoc = vvPDF_newDocument(vvPDF);
0040.00        vvPdf_addTablefromSQL(vvPDF:pdfDoc:SQLSTR);
0041.00        vvPDF_closeDocument(pdfDoc);
0042.00
0043.00        //動的に作成されたPDFをダウンロード
0044.00        vvOut.download = '1';
0045.00        vvOut.file     = 'CATEGORY_' + VCATG + '.PDF';   1-④
0046.00        vvOut_file(vvPDF.path:vvOut);
0047.00
0048.00        //動的に作成されたPDFをIFS上から削除
0049.00        vvIfs_deleteFile(vvPDF.path);
0050.00      /end-free
0051.00     p                 e
0052.00      /include qcpylesrc,vvNabBtn
```

PDF出力には、vvPDFというAPIを使用するが、このAPIを使用する場合、ソースの宣言部に1-①のような宣言を追加すればよい。1-②で商品マスターからカテゴリーCDが合致するデータを抽出する為のSQL文を作成している。そして、1-③がPDFを作成しIFS上に保存する処理である。[vvPdf_addTablefromSQL]というAPIを使用すれば、SQL実行結果より一覧表PDFが作成できる。最後にIFS上のPDFをブラウザにダウンロードさせる処理が1-④である。このように、従来からValenceでは、RPG ToolKitを使用する事で、動的なPDF帳票が作成できた。ただ、この方法には、課題があり、残念ながら日本語を含む文字列は、ブランクで出力されてしまうのである。**【図3】**

図3　RPG ToolKitから出力されたPDF



Valence

このAPIは、開発元CNX社によるとIBM i側のPDF作成エンジンを使用しているとの事なのだが、このエンジンには残念ながら、日本のIBM iマシンであっても日本語フォントが標準搭載されておらず、日本語文字が欠落してしまうとの事である。折角ValenceにはPDF作成機能が用意されているにも関わらず、これまでこの機能を積極的にアピールしてこなかったのは、この為である。

もちろん、ValenceはIBM iを使用している為、従来通りスプールを使用した帳票は活用できるわけだが、PDFを作成したり、表現力のある帳票を作成したい場合、【図4】のように外部の帳票ツールと連携する、或いは【図5】のようにDelphi/400等を組み合わせて独自の帳票処理を作成するのが、これまでは一般的であった。

図4　Valenceと帳票ツールとの連携イメージ

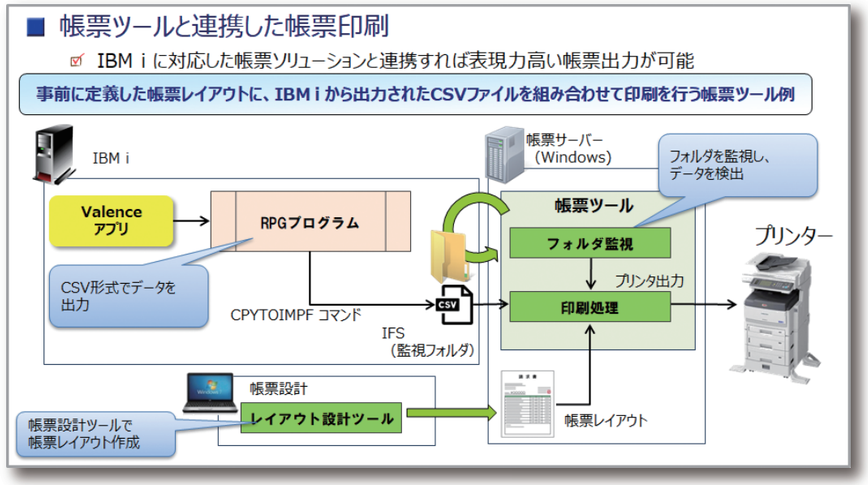
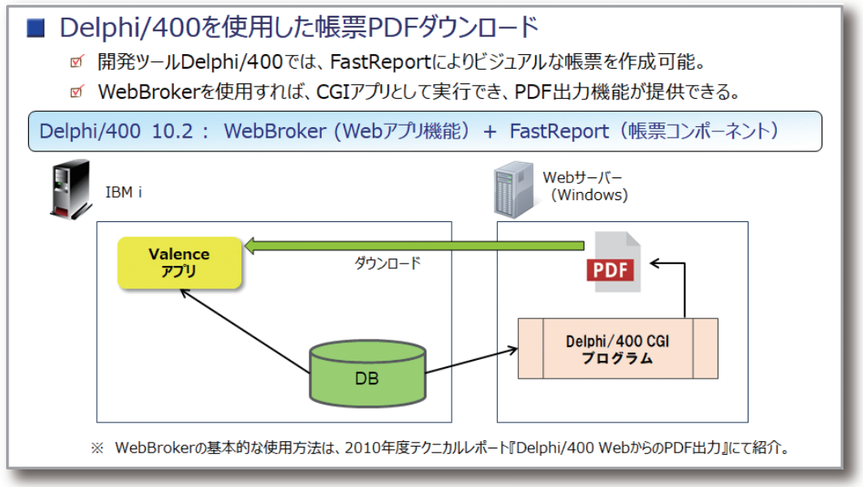


図5　ValenceとDelphi/400との連携イメージ



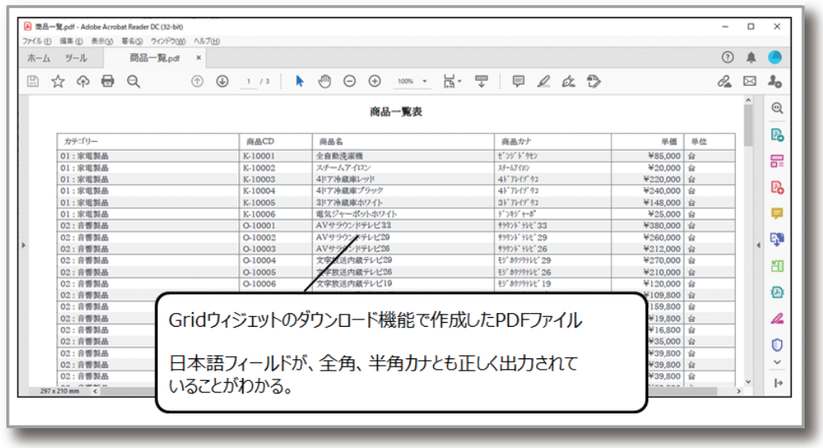
3. Valence6.0における新しいPDF帳票出力

Valenceは、2020年12月にバージョン6.0となったが、このバージョンからGridウィジェットのダウンロード機能において従来のExcelに加えてPDF形式が選択できるようになった。【図6】

図6　App Builder Gridウィジェット 設定画面

このPDF形式を選択したGridウィジェットを使用してアプリケーションを作成し、実行すると一覧データがPDF形式でダウンロードできる事が分かる。しかも、生成されたPDFを確認すると、日本語文字列を含めて正しく出力されているのである。【図7】

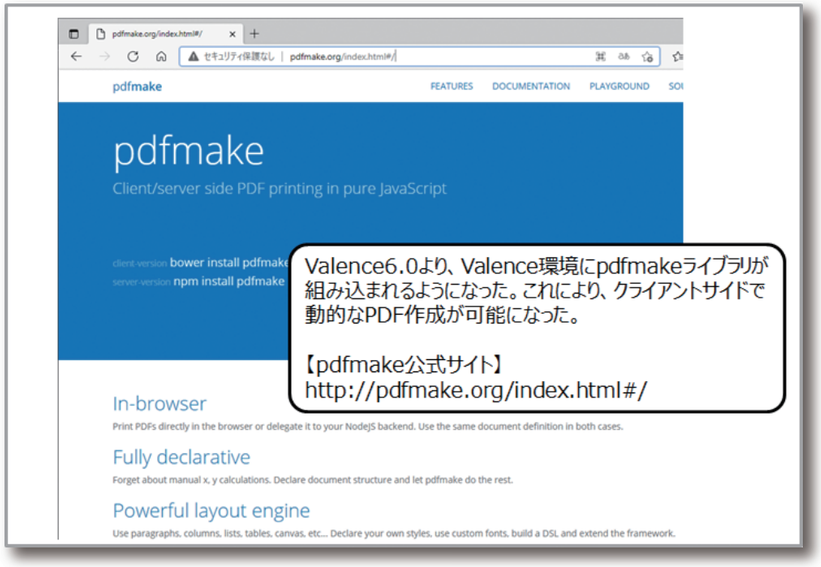
図7　Gridウィジェット ダウンロードで生成されたPDF



なぜ、日本語を含む文字列が正しく出力できるようになったのか再度開発元に確認したところ、バージョン6.0では従来のIBM iエンジンを使用する方法とは別にクライアント（ブラウザ）の機能を使用してPDFを作成できる方法を採用したことがわかった。

詳細を確認してみたところ、新たにpdfmakeと呼ばれるライブラリがValence環境に追加されたのである。pdfmakeは、JavaScriptを使用してクライアントサイドで動的なPDFを作成できるオープンソースライブラリである。**【図8】**

図8　JavaScriptベースのPDF作成ライブラリ



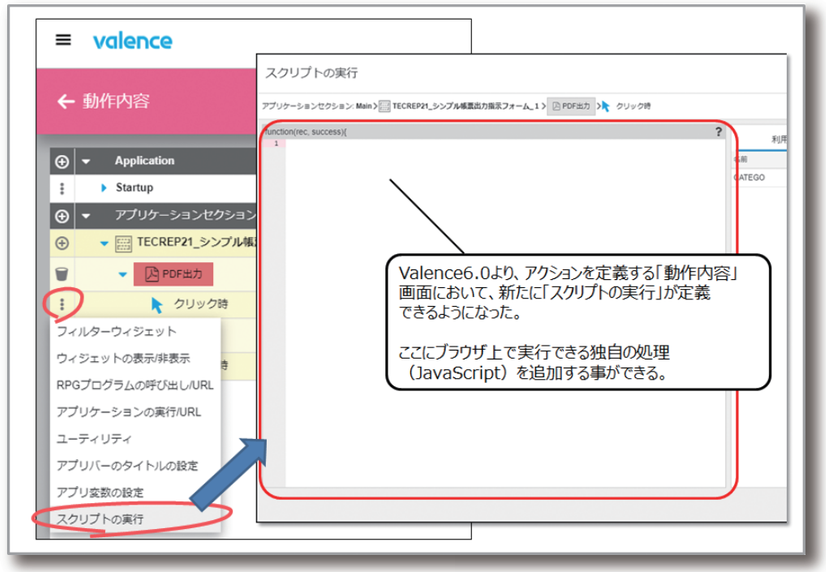
実はpdfmakeも標準では日本語の出力には対応していないのだが、独自のフォントを追加できるようになっている。日本版のValence環境にはデフォルトで「あおぞら明朝フォント」と呼ばれる日本語フォントが組み込まれている為、日本語を含む文字列のPDFが出力できるのである。

この新しいPDF出力機能をApp　Builderの標準機能として使用できるのは、2021年8月現在Grid/Edit Gridウィジェットに追加されたPDFダウンロード機能のみであるが、このライブラリはJavaScriptを使用する事で簡単にPDFを作成できる為、独自のPDF出力処理をApp Builderアプリケーションに追加する事ができる。具体的な作成方法を次節で紹介する。

4. pdfmakeを使用したPDF帳票作成方法

App Builderにおいて、ユーザーの操作に応答するアクション（処理）は「動作内容」画面で設定できる。従来からこの機能を設定する事で、例えばRPGプログラムの連携等が実装できたわけだが、Valence6.0からは新たに「スクリプトの実行」が定義できるようになり、クライアントブラウザ上で実行可能な処理（JavaScript）が定義できるようになった。**【図9】**

図9　App Builder「動作内容」に追加された「スクリプトの実行」

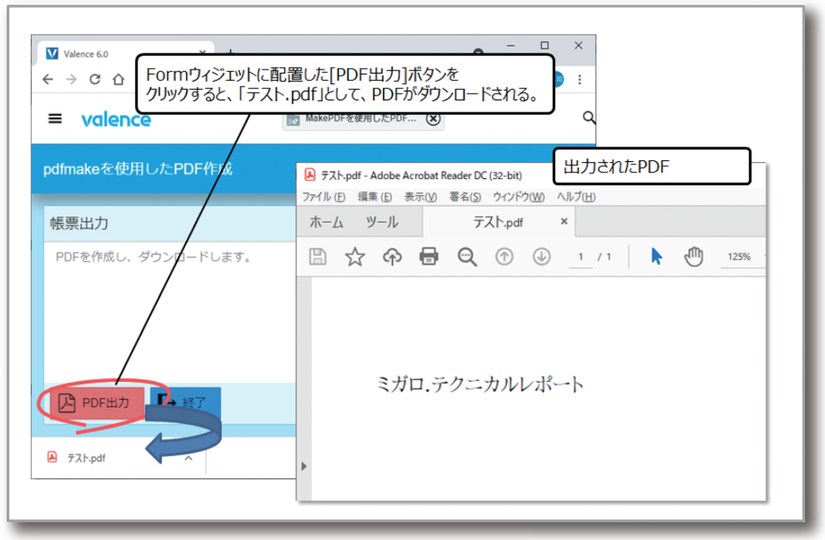


この機能を使用すれば、pdfmakeを使用した独自の帳票が作成できる。ここでは具体的な作成方法の紹介として、まず始めにシンプルなPDF出力アプリケーションを作成する。

4-1. シンプルなPDF出力処理の作成

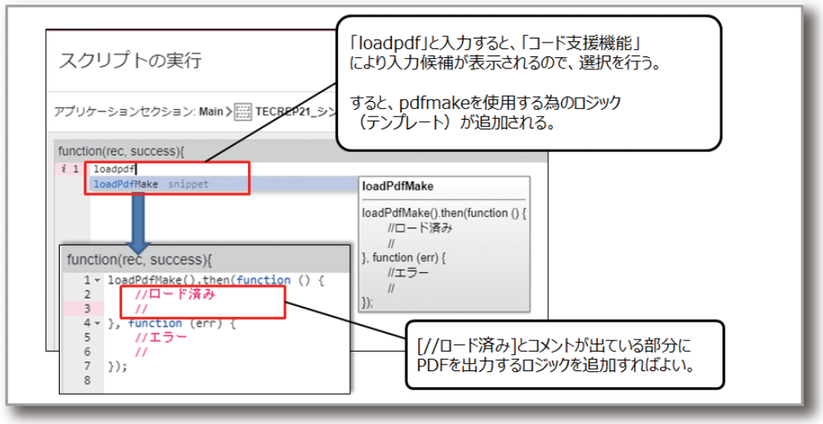
サンプルプログラム２は、Formウィジェットに配置した［PDF出力］ボタンをクリックすると、「ミガロ. テクニカルレポート」という日本語文字列を持つPDFファイルをダウンロードするというシンプルなPDF出力処理である。**【図10】**

図10　サンプルプログラム2：シンプルなPDF出力処理



App Builderの「アプリケーション作成」ステップにおける「動作内容」定義画面で、［PDF出力］ボタンに対し、アクション追加より「スクリプトの実行」を選択すると、スクリプトを記述する為のスクリプトエディタ画面が開く。エディタ上で"loadpdf"と入力すると、エディタに搭載された「入力コード支援機能」により、入力候補として、"loadPdfMake"が表示されるので、それを選択する。するとライブラリpdfmakeを読み込む為のテンプレートソースが自動的に追加される。**【図11】**

図11　pdfmake 読込処理をスクリプトに追加



"//ロード済み"と書かれたコメント部分がpdfmakeの機能が記述できる部分となる。ここに独自のPDFを生成し、出力する処理を記述すればよい。実際に記述したソース例が**【ソース2】**である。

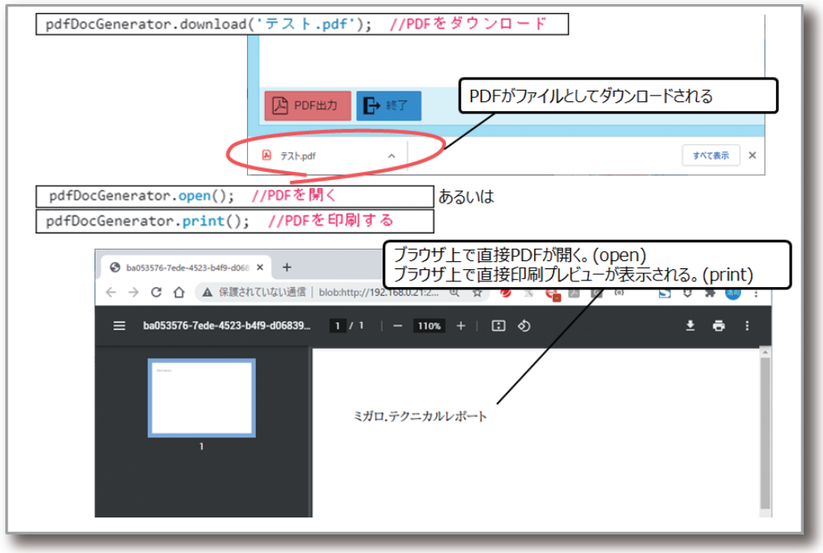
ソース2　サンプルプログラム2：シンプルなPDF出力処理

```
function(rec, success){
1  //---- PDFMakeを起動
2  loadPdfMake().then(function () {
3      //---- Valenceに導入済みの日本語フォントを指定
4      pdfMake.fonts = {
5          Aozora: {
6            normal: 'AozoraMinchoRegular.ttf',
7            bold:   'AozoraMincho-bold.ttf'
8          }
9      };
10
11     //---- PDFレイアウト作成
12     var docDefinition = {
13         //用紙指定
14         pageSize: 'A4',                          // 用紙サイズ
15         pageOrientation: 'landscape',            // 向き（横の場合のみ指定）
16         pageMargins: [ 40, 60, 40, 60 ],         // 余白
17
18         //出力する内容
19         content:
20         [
21             {text: 'ミガロ. テクニカルレポート'}    // 文字を1行出力
22         ],
23
24         //スタイル（書式）指定
25         defaultStyle: {
26             font: 'Aozora'                       // フォント指定
27         }
28     };
29
30     //---- PDF生成
31     var pdfDocGenerator = pdfMake.createPdf(docDefinition);
32     //---- 生成したPDFを出力（下記のいずれかを実行）
33     pdfDocGenerator.download('テスト.pdf');  //PDFをダウンロード
34
35 }, function (err) {
36     //エラー
37 });
```

2-①　2-②　2-③

2-①の部分が、pdfmakeに組み込まれた日本語フォントを指定する部分である。これを書かないと日本語文字列が出力できないので、このまま埋め込むようにしてほしい。
2-②が帳票を作成する部分で、この例のようにpdfmakeでは、帳票のレイアウトの作成をdocDefinitionという変数に対し、JSON形式で値をセットする形で行う。一見複雑そうだが、実際のPDFレイアウトは、この中の"content"という要素である。この例では一行だけ「ミガロ. テクニカルレポート」という文字列をテキスト出力項目(text)として定義しているのである。このソース例では追加情報として、別途用紙サイズや向き、余白を設定しているが、これは無くても良い。また最後の部分で"defaultStyle"を定義しているが、これは2-①で定義したフォントをこのレイアウトで使用するという設定なので、日本語を出力する際にはこの指定は必須となる。
2-③がPDFを生成し、結果をダウンロードさせる部分である。"pdfMake.createPdf(docDefinition)"というメソッドを呼び出す事で、2-②で定義したレイアウトを使ってPDFを生成でき、生成された結果をpdfDocGeneratorという変数に代入している。あとは生成されたPDFをどのように処理するかを記述すればよく、この例ではdownload()というメソッドを使用してブラウザ上にダウンロードさせるようにしている。この部分はdownload()以外にopen()あるいはprint()といった指定もできる。それぞれの実行結果は**【図12】**のようになる。

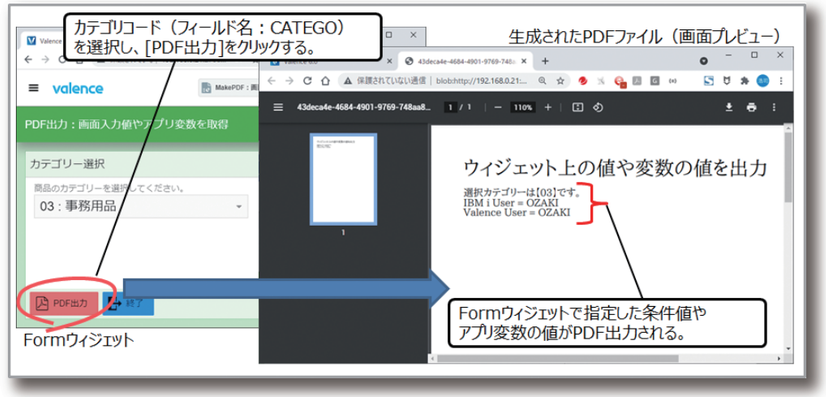
図12　pdfmakeを使用したPDF出力方法



4-2. 画面入力値や変数値を出力するPDF作成

次にウィジェット上で入力した値やValenceで定義できるアプリ変数の値等をPDF帳票に埋め込む方法を紹介する。サンプルプログラム3は、Formウィジェット上で選択した商品カテゴリーの値(フィールド名:CATEGO)とValenceにデフォルトで定義されたアプリ変数であるIBMiユーザー/Valenceユーザーの値をPDFに出力するものである。**【図13】**

図13　サンプルプログラム3:画面入力値／変数の値をPDF出力



このPDF出力処理のソース例は、**【ソース3】**である。

ソース3　サンプルプログラム3:画面入力値／変数の値をPDF出力

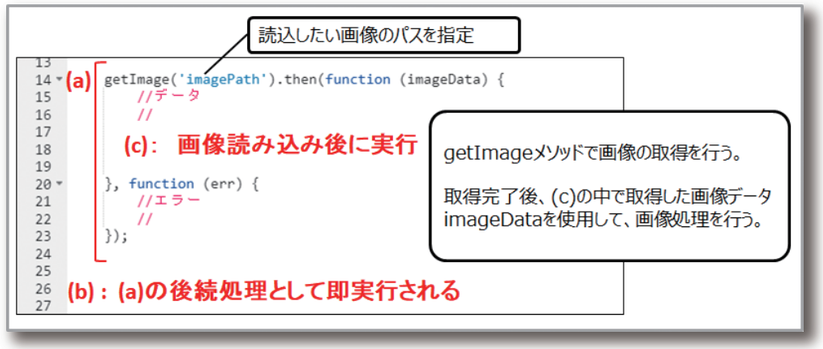
```
function(rec, success){
  1  //---- PDFMakeを起動
  2  loadPdfMake().then(function () {
  3      //---- 日本語フォント定義
  4      pdfMake.fonts = {
  5          Aozora: {
  6            normal: 'AozoraMinchoRegular.ttf',
  7            bold:   'AozoraMincho-bold.ttf'
  8          }
  9      };
 10      //---- PDFレイアウト作成                                          3-①
 11      var docDefinition = {
 12          //出力する内容
 13          content: [
 14              {text: 'ウィジェット上の値や変数の値を出力', fontSize: 24},
 15              {text: ' '},
 16              {text: '選択カテゴリーは【' + rec.get('CATEGO') + '】です。'},  //ウィジェット上のフィールド
 17              {text: 'IBM i User = ' + getAppVar('nabIbmI') },              //アプリ変数（IBMiユーザー）
 18              {text: 'Valence User = ' + getAppVar('nabUser') }             //アプリ変数（Valenceユーザー）
 19          ],
 20          //スタイル（書式）指定
 21          defaultStyle: {
 22              font: 'Aozora'                    // 日本語フォント
 23          }
 24      };
 25      //---- PDF生成
 26      var pdfDocGenerator = pdfMake.createPdf(docDefinition);
 27      //---- 生成したPDFを出力
 28      pdfDocGenerator.open();   // PDFを開く
 29  }, function (err) {
 30      //エラー
 31  });
```

先程と同様"content"要素内でPDFレイアウトを作成しているのだが、ここでは、recというオブジェクト変数やgetAppVerというメソッドを使用していることがわかる。rec変数がウィジェット上に定義されたフィールドにアクセスする為に用意された変数で、getメソッドを使用すればウィジェット上のフィールド値を取得する事ができる。
また、getAppVerメソッドがApp Builderのアプリケーションに定義された「アプリ変数」値を取得するものである。このようにアプリケーション実行時にユーザーがウィジェットに入力した値や、内部的に保持しているアプリ変数の値を使用してPDF帳票を作成するのも容易であることがわかる。

4-3. 画像を含むPDF作成

次は、画像をPDF出力する方法を紹介する。Valenceを使用して画像情報を取得する為には、Valence専用のメソッドを使用する。スクリプトエディタ上で"getImage"と入力すると、「入力コード支援機能」により画像取得処理のテンプレートが追加できる。**【図14】**

図14　画像読み込み処理（getImage）



【図14】の処理だが、(a)の部分が画像を読み込むメソッドとなっている。画像の読込が完了すると、取得した画像情報がimageDataにセットされるので、取得後の画像処理を(c)の部分に記述すればよい。但し、このメソッドには注意点がある。画像の読込処理自体は非同期で行われる為、(a)の後続処理部分である(b)の部分は、(a)の実行後に即実行されてしまう点である。(c)の部分は、画像情報が取得された後に実行される為、(c)より(b)の部分の方が先に実行されてしまうのである。この処理順序に対する対処例は、本稿の後半（サンプルプログラム８）で紹介する。

ここでは、getImageメソッドを使用した具体的なサンプルを紹介する。サンプルプログラム４は、Formウィジェット上の[PDF出力]ボタンをクリックすると、IFS上に保存されたValenceのロゴ画像をPDF出力するものである。**【図15】**

図15　サンプルプログラム4：画像データをPDF出力

サンプルプログラム４のソース記述例は**【ソース4】**である。

ソース4　サンプルプログラム4：取得した画像データからPDF出力

```
function(rec, success){
//----- PDFMake を読込
loadPdfMake().then(function () {
    //---- Valenceに導入済みの日本語フォントを指定
    pdfMake.fonts = {
        Aozora: {
          normal: 'AozoraMinchoRegular.ttf',
          bold:   'AozoraMincho-bold.ttf'
        }
    };

    //---- 画像データ取得
    getImage('/resources/images/valence_logo.png').then(function (imageData) {    4-①
      //---- 画像取得成功
      //---- PDFレイアウト作成
      var docDefinition = {
          //出力する内容
          content: [
              {text: '画像の出力', fontSize: 20},
              {text: ' '},
              {image: imageData, width: 300}      // 画像      4-③
          ],
          //スタイル（書式）指定
          defaultStyle: {
              font: 'Aozora'                      // 日本語フォント
          }
      };

      //---- PDF生成
      var pdfDocGenerator = pdfMake.createPdf(docDefinition);
      //---- 生成したPDFを出力
      pdfDocGenerator.open();    // ファイルを開く
    }, function (err) {
        //---- 画像取得エラー
});
}, function (err) {
    //エラー
});
```

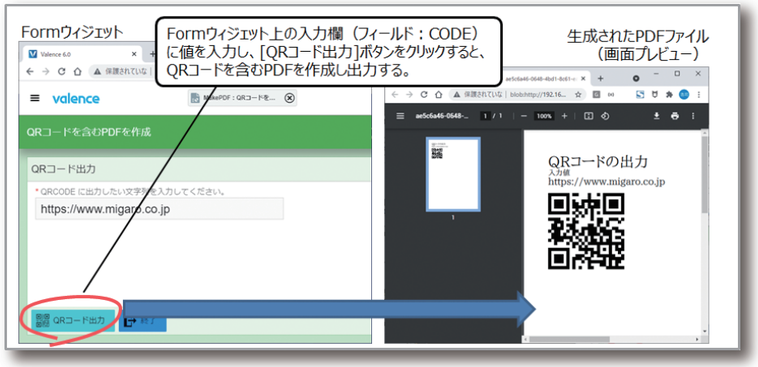
4-②

4-①が画像を読み込む部分となる。IFS上にあるValenceのロゴ画像(/resources/images/valence_logo.png)を取得している。画像が取得できれば、4-②の部分が実行される為、この部分でPDFファイルのレイアウト作成およびPDF生成を行っている。pdfmakeでは4-③のように画像項目(image)に取得した画像データ(imageData)をセットするだけで画像がPDFに出力できる。

4-4. QRコードやバーコードを含むPDF作成

帳票出力においてニーズが高いのが、QRコード/バーコードの出力である。pdfmakeはQRコードも容易に作成できる。サンプルプログラム５は、Formウィジェット上に入力した値を使用して動的にQRコードを作成し、それをPDF出力するものである。【図16】

図16　サンプルプログラム5：QRコードをPDF出力



サンプルプログラム５のソース記述例は【ソース5】である。

ソース5　サンプルプログラム5：QRコードをPDF出力

```
function(rec, success){
1  //---- PDFMakeを起動
2  loadPdfMake().then(function () {
3      //---- 日本語フォント定義
4      pdfMake.fonts = {
5          Aozora: {
6              normal: 'AozoraMinchoRegular.ttf',
7              bold:   'AozoraMincho-bold.ttf'
8          }
9      };
10
11     //----- フォーム上の入力値を取得
12     const CodeVal = rec.get('CODE');  // 画面入力値      5-①
13
14     //---- PDFレイアウト作成
15     var docDefinition = {
16         //出力する内容
17         content: [
18             {text: 'QRコードの出力', fontSize: 24},
19             {text: '入力値'},
20             {text: CodeVal, fontSize: 16},
21             {text: ' '},
22             {qr: CodeVal}                          // QRコード    5-②
23         ],
24         //スタイル（書式）指定
25         defaultStyle: {
26             font: 'Aozora'                         // 日本語フォント
27         }
28     };
29     //---- PDF生成
30     var pdfDocGenerator = pdfMake.createPdf(docDefinition);
31     //---- 生成したPDFを出力
32     pdfDocGenerator.open();   // PDFを開く
33 }, function (err) {
34  //エラー
35 });
```

5-①でFormウィジェット上のフィールド入力値を取得しており、取得した変数を使用して5-②のようにQRコード項目（qr）に文字列をセットするだけなので、とても簡単である。次にバーコードだが、pdfmakeには残念ながらバーコードを出力する機能は含まれていない。従ってバーコードを出力するには一工夫必要となる。今回は、JsBarcodeと呼ばれるJavaScriptでバーコードを作成できるライブラリを活用する。【図17】

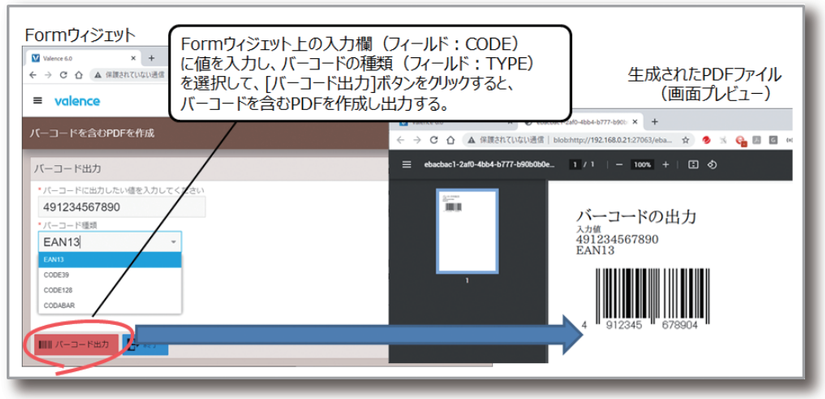
図17　JavaScriptベースのバーコード作成ライブラリ

このライブラリを使用すれば、任意の文字列から動的にバーコード画像を作成する事ができる。こういった外部のライブラリを組み込む場合、一般的にはHTMLの中にJavaScriptが参照するライブラリのCDNを定義するのだが、App Builderの場合、「ポータル管理」の「設定」ページの中で外部ライブラリを追加できるようになっている。今回は、JsBarcodeのCDNを設定した。【図18】

図18　外部ライブラリの組み込み

サンプルプログラム6は、Formウィジェット上でコードの値とバーコードの種類を指定して、動的にバーコードを作成し、それをPDF出力するものである。【図19】

図19　サンプルプログラム6：バーコードをPDF出力



サンプルプログラム6のソース例は【ソース6】である。

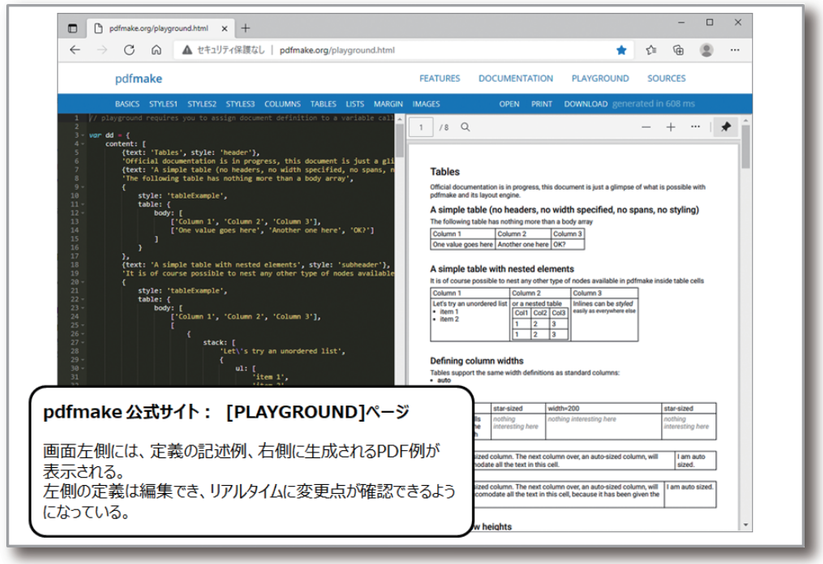
ソース6　サンプルプログラム6：バーコードをPDF出力

```
function(rec, success){
//---- PDFMakeを起動
loadPdfMake().then(function () {
    //---- 日本語フォント定義
    pdfMake.fonts = {
        Aozora: {
            normal: 'AozoraMinchoRegular.ttf',
            bold:   'AozoraMincho-bold.ttf'
        }
    };

    //----- フォーム上の入力値を取得
    const CodeVal = rec.get('CODE');  // 入力したコード
    const TypeVal = rec.get('TYPE');  // バーコード種類 (EAN13, CODE39, CODE128, CODABAR, ...)   6-①

    //----- キャンバスを作成しバーコードを発行
    var canvas = document.createElement('canvas');
    JsBarcode(canvas, CodeVal,  {format: TypeVal});   6-②

    //---- PDFレイアウト作成
    var docDefinition = {
        //出力する内容
        content: [
            {text: 'バーコードの出力', fontSize: 24},
            {text: '入力値'},
            {text: CodeVal, fontSize: 16},
            {text: TypeVal, fontSize: 16},
            {text: ' '},
            {image: canvas.toDataURL(), width: 200, height: 100}  // バーコード画像   6-③
        ],
        //スタイル（書式）指定
        defaultStyle: {
            font: 'Aozora'//日本語フォント
        }
    };
    //---- PDF生成
    var pdfDocGenerator = pdfMake.createPdf(docDefinition);
    //---- 生成したPDFを出力
    pdfDocGenerator.open();   // PDFを開く
}, function(err) {
    //エラー
});
```

6-①でFormウィジェット上で入力したコードとバーコードの種類を取得している。6-②は、動的に画像要素（canvas）を生成し、JsBarcodeメソッドを使用してバーコード画像を生成している。このメソッドではパラメータに画像要素、コード文字列、そしてバーコードの種類を指定する。すると画像要素に生成された画像データが格納されるので、後は6-③のようにpdfmakeの画像項目（image）にcanvas画像をセットすればよい。

本節ではpdfmakeを使用して動的にPDFを生成する例を紹介したが、pdfmakeは帳票レイアウト作成にJSONを使用する。定義の詳細は公式サイトのドキュメントに詳しく載っているので、そちらを参照してほしいのだが、サイトの中に[PLAYGROUND]というページがあり、ここでは色々な要素の実装例を確認する事ができる。【図20】

図20　pdfmake PLAYGROUNDページ



この[PLAYGROUND]では左側の定義部分を直接変更する事もでき、その結果はリアルタイムに右側の帳票レイアウトに反映される。実際に帳票レイアウト設計する際には、まずここで定義を試してみるとよいだろう。

5. Gridウィジェットデータを活用した一覧帳票作成

本節では、ウィジェット上に表示されているデータソースの内容から動的に一覧表形式のPDFを生成する方法を紹介する。さらに一覧表に画像情報を追加するカスタマイズ方法も紹介する。

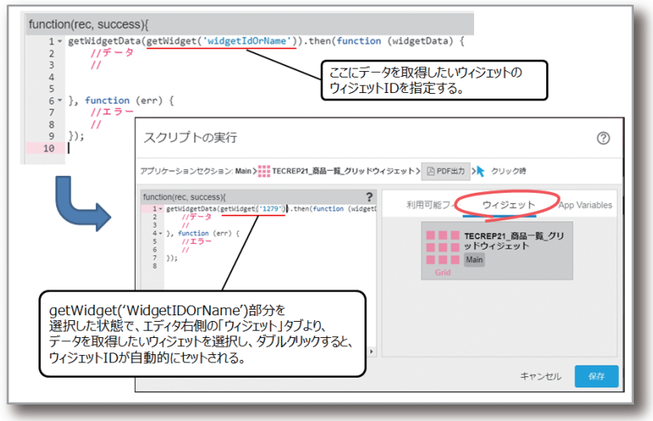
5-1. ウィジェットデータの取得方法

サンプルプログラム7は、Gridウィジェットに表示されている商品マスターの内容を一覧表の形式でPDFに出力するものである。**【図21】**

図21　サンプルプログラム7:ウィジェットデータのPDF出力

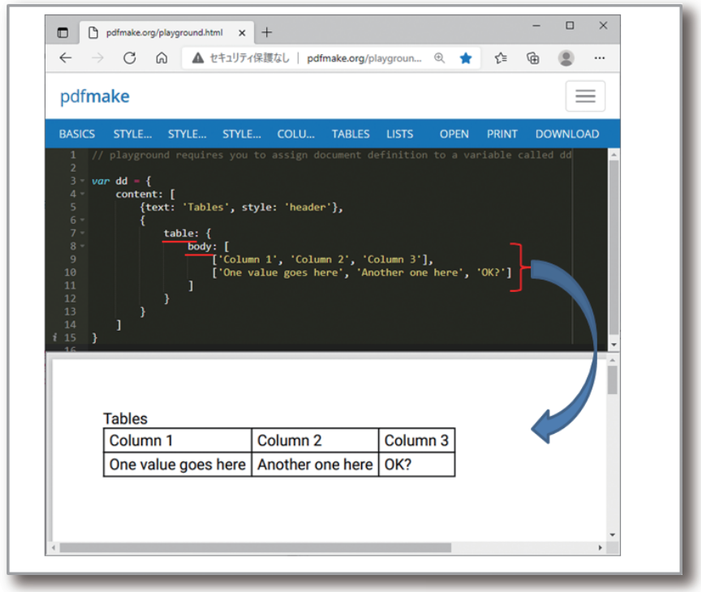
ウィジェット上のデータは次のようにすれば取得できる。スクリプトエディタ上で"getWidgetData"と入力すると表示される「入力コード支援機能」の候補を選択する事でテンプレートが自動的に追加される。追加されたメソッドの中にある"getWidget('WidgetIdOrName')"の部分に、データを取得したいウィジェットを指定すればよいのだが、これはスクリプトエディタ右側の「ウィジェット」タブに表示されるウィジェット一覧から目的のウィジェットを選択すればよい。選択したウィジェットが内部で保持しているウィジェットIDが自動的にセットされる。**【図22】**

図22　ウィジェットデータの取得処理（getWidgetData）



これで目的のウィジェット上のデータがスクリプトの中で取得できるようになり、取得したGridウィジェットの一覧データがwidgetData変数にレコード件数分の配列として格納される。
取得したデータから一覧形式のPDFを作成する際に使用するのが、pdfmakeの表(table)という要素である。先程紹介したpdfmakeの[PLAYGROUND]で"table"要素の基本形を作成してみたのが、**【図23】**である。このように表を示す"table"要素の子要素としてbody部を定義し、その中に表のレイアウトを作成すればよい。

図23　pdfmakeによる表(table)の作成



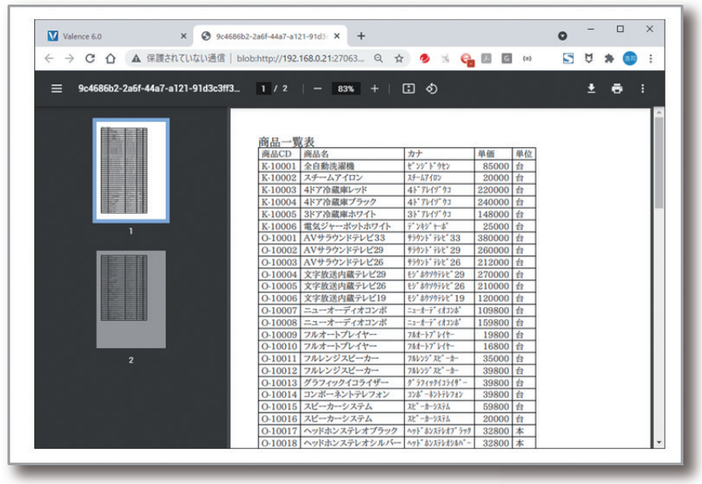
この仕組みを使用して作成したサンプルプログラム7のソース例が**【ソース7】**である。
7-①で宣言しているtableBody変数が、動的にbody部を作成する為に用意した配列である。7-②では、表のタイトル行を作成している。7-③では、forループを使用して、wigdetData変数に格納されたウィジェットデータの配列をレコード単位で順番に取得しながら、tableBody変数に値をセットしている。そして、7-④の部分でtableBody変数をpdfmakeの表項目(table)のbody部にセットしている。

ソース7　サンプルプログラム7:ウィジェット上のデータを元にPDFを作成

```
function(rec, success){
var tableBody = [];     // 表の要素を保持する変数   7-①

//----- PDFMakeを読込
loadPdfMake().then(function () {
    //----- Valenceに導入済みの日本語フォントを指定
    pdfMake.fonts = {
        Aozora: {
          normal: 'AozoraMinchoRegular.ttf',
          bold:   'AozoraMincho-bold.ttf'
        }
    };

    //----- ウィジェットデータの取得
    getWidgetData(getWidget('1279')).then(function (widgetData) {
        //----- 取得したデータをtableBody配列へ格納
        // 列タイトルを要素に追加
        tableBody.push(['商品ＣＤ','商品名','カナ','単価','単位']);  7-②
        // 取得したデータを要素に追加
        for (i = 0; i < widgetData.length; i++) {
            var rowData = [];
            rowData.push({text: widgetData[i].F1_ODPDCD});  // 商品ＣＤ
            rowData.push({text: widgetData[i].F1_ODPDNM});  // 商品名
            rowData.push({text: widgetData[i].F1_ODPDKN});  // 商品カナ
            rowData.push({text: widgetData[i].F1_ODTANK, alignment: 'right'});  // 単価
            rowData.push({text: widgetData[i].F1_ODUNIT});  // 単位
            //tableBodyに要素を追加
            tableBody.push(rowData);
        }                                                                          7-③

        //----PDFレイアウト作成
        var docDefinition = {
            //出力する内容
            content: [
                {text: '商品一覧表', fontSize: 16},
                {table: {
                    headerRows: 1,                          // 列タイトル行
                    width: ['auto','auto','auto','auto','auto'],    // 列幅指定
                    body: tableBody                         // tableBody(表の要素)
                    }
                }
            ],                                                                     7-④
            //スタイル（書式）指定
            defaultStyle: {
                font: 'Aozora'      // 日本語フォント
            }
        };
        //---- PDF生成
        var pdfDocGenerator = pdfMake.createPdf(docDefinition);
        //---- 生成したPDFを出力
        pdfDocGenerator.open();   // PDFを開く
    }, function (err){
        //エラー
    });
}, function (err) {
    //エラー
});
```

【ソース7】を組み込んだアプリケーションを実行すると、Gridウィジェット上に一覧表示された商品マスターの情報を元に、表形式の一覧PDFファイルを動的に作成する事ができる。【図24】

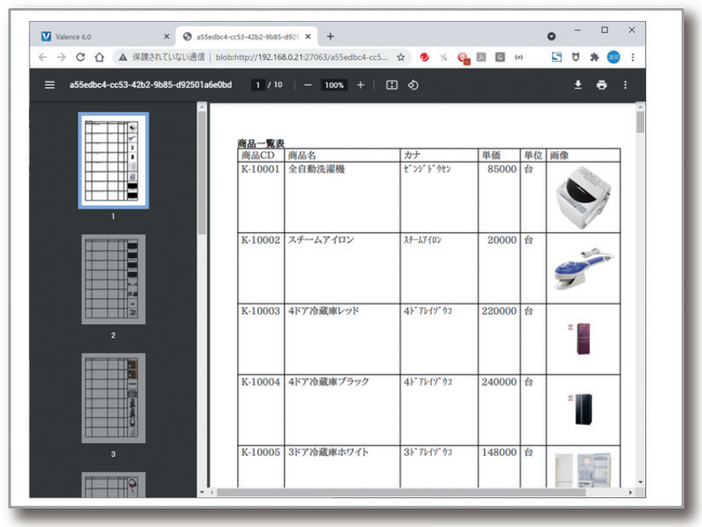
図24　サンプルプログラム7によって生成されたPDF



5-2. 画像を追加した一覧帳票のカスタマイズ

次に、この一覧表PDFをカスタマイズする例を紹介する。今回はIFSの中にある"/resources/images/products/"ディレクトリに"[商品CD].jpg"という形式で格納された商品画像を使用する。先程作成した商品マスターの一覧表に画像列を追加して、画像付きの一覧表が出力できるように変更する。完成したサンプルプログラム８からＰＤＦを出力すると次のような実行結果となる。【図25】

図25　サンプルプログラム8によって生成されたPDF



前節で紹介した画像の取得方法であるgetImageメソッドを【ソース7】に追加すると、【ソース8】のような形になると考えられるだろう。処理を追加した部分は次のとおりである。8-①でタイトル行に"画像"列を追加している。そして、8-②の部分でgetImageメソッドを使用してIFS上にある"[商品CD].jpg"というファイル名の画像を取得し、画像が取得できた場合に、画像フィールドの列情報をtableBody変数へ追加している。

ソース8　サンプルプログラム8：画像項目を追加（※動作しない）

function(rec, success){

```
1  var tableBody = [];     // 表の要素を保持する変数
2
3  //----- PDFMakeを読込
4  loadPdfMake().then(function () {
5      //----- Valenceに導入済みの日本語フォントを指定
6      pdfMake.fonts = {
7          Aozora: {
8            normal: 'AozoraMinchoRegular.ttf',
9            bold:   'AozoraMincho-bold.ttf'
10         }
11     };
12
13     //----- ウィジェットデータの取得
14     getWidgetData(getWidget('1279')).then(function (widgetData) {
15         //----- 取得したデータをtableBody配列へ格納
16         // 列タイトルを要素に追加
17         tableBody.push(['商品ＣＤ','商品名','カナ','単価','単位','画像']);  8-①
18         // 取得したデータを要素に追加
19         for (i = 0; i < widgetData.length; i++) {
20             var rowData = [];
21             rowData.push({text: widgetData[i].F1_ODPDCD});  // 商品ＣＤ
22             rowData.push({text: widgetData[i].F1_ODPDNM});  // 商品名
23             rowData.push({text: widgetData[i].F1_ODPDKN});  // 商品カナ
24             rowData.push({text: widgetData[i].F1_ODTANK, alignment: 'right'});  // 単価
25             rowData.push({text: widgetData[i].F1_ODUNIT});  // 単位
26
27             //----- 商品画像の取得（商品ＣＤ.jpgというファイル名で画像を取得）  8-②
28             const imgPath = "/resources/images/products/" + widgetData[i].F1_ODPDCD + '.jpg';
29             getImage(imgPath).then(function (imageData) {
30                 // 画像取得成功の場合
31                 rowData.push({image: imageData, width: 85});    // 画像を出力
32             }, function (err) {
33                 // 画像取得エラーの場合
34                 rowData.push({text: "画像無し"});               // 画像無しメッセージ
35             });
36
37             //tableBodyに要素を追加
38             tableBody.push(rowData);
39         }
40
41         //----PDFレイアウト作成
42         var docDefinition = {
43             //出力する内容
44             content: [
45                 {text: '商品一覧表', fontSize: 16},
46                 {table: {
47                     headerRows: 1,                                  // 列タイトル行
48                     width: ['auto','auto','auto','auto','auto','auto'],   // 列幅指定
49                     body: tableBody                                 // tableBody(表の要素)
50                     }
51                 }
52             ],
53             //スタイル（書式）指定
54             defaultStyle: {
55                 font: 'Aozora'      // 日本語フォント
56             }
57         };
58         //---- PDF生成
59         var pdfDocGenerator = pdfMake.createPdf(docDefinition);
60         //---- 生成したPDFを出力
61         pdfDocGenerator.open();   // PDFを開く
62     }, function (err){
63         //エラー
64     });
65 }, function (err) {
66     //エラー
67 });
```

一見この【ソース8】は正しい処理に見えるかもしれないが、残念ながら、このプログラムは実行しても帳票は出力されない。なぜならば、getImageメソッドの項で紹介した通り、画像の取得処理自体は非同期で行われる為、この記述方法だと画像が取得されるより前にループが繰り返されてしまうのである。

この課題を解決する為に処理を書き直したのが、【ソース9-1&2】である。ウィジェットデータを取得するgetWidgetDataメソッドの処理結果部分を大きく変更した。

今回、1明細分のtableBodyを作成する為のmakeDetailサブルーチン(9-②)と、PDFを作成、出力する為のmakePDFサブルーチン(9-④)を新たに定義して処理を分割した。この処理では、まず始めに9-①が実行されて、タイトル行を設定した後に、9-②を呼び出している。9-②では画像取得処理であるgetImageメソッドを使用しているが、今回は画像取得が完了してから、次の明細行の処理に進むように、9-③のような形でmakeDetailサブルーチンを再帰呼び出しする実装としている。明細行数分だけ再帰呼び出しを行い、最終行の処理が終わったら、最後に9-④を呼び出して、PDFを生成しているのである。少し複雑な処理となってしまったが、今回のような場合に、再帰呼び出し処理は有用なので、是非このソースを参考にしてほしい。

ソース9-1　サンプルプログラム8:再帰呼び出し処理に修正

```
function(rec, success){
 1  var tableBody = [];     // 表の要素を保持する変数
 2
 3  //----- PDFMakeを読込
 4  loadPdfMake().then(function () {
 5      //----- Valenceに導入済みの日本語フォントを指定
 6      pdfMake.fonts = {
 7          Aozora: {
 8            normal: 'AozoraMinchoRegular.ttf',
 9            bold:   'AozoraMincho-bold.ttf'
10          }
11      };
12
13      //----- ウィジェットデータの取得
14      getWidgetData(getWidget('1279')).then(function (widgetData) {
15          //----- 明細行作成サブルーチン
16          function makeDetail(i, dataCount) {
17              //------ 最終レコードに到達した場合PDF作成処理を呼び出して本処理終了
18              if (i >= dataCount)  {
19                  makePDF();    // PDF作成処理を呼出
20                  return;
21              }
22              else
23              {
24                  //----- 明細行の作成
25                  var rowData = [];
26                  rowData.push({text: widgetData[i].F1_ODPDCD});  // 商品ＣＤ
27                  rowData.push({text: widgetData[i].F1_ODPDNM});  // 商品名
28                  rowData.push({text: widgetData[i].F1_ODPDKN});  // 商品カナ
29                  rowData.push({text: widgetData[i].F1_ODTANK, alignment: 'right'});  // 単価
30                  rowData.push({text: widgetData[i].F1_ODUNIT});  // 単位
31
32                  //----- 商品画像の取得（商品ＣＤ.jpgというファイル名で画像を取得）
33                  const imgPath = '/resources/images/products/' + widgetData[i].F1_ODPDCD + '.jpg';
34                  getImage(imgPath).then(function (imageData) {
35                      // 画像取得成功の場合
36                      rowData.push({image: imageData, width: 85});    // 画像を出力
37                      // 行明細書き込み
38                      tableBody.push(rowData);    //行の要素をtableBodyへ追加
39                      // 次データの取得（再帰呼び出し）
40                      i++;                        //カウントアップ
41                      makeDetail(i, dataCount);   //次行の明細を作成
42                  }, function (err) {
43                      // 画像取得エラーの場合
44                      rowData.push({text: "画像無し"});               // 画像無しメッセージ
45                      // 行明細書き込み
46                      tableBody.push(rowData);    //行の要素をtableBodyへ追加
47                      // 次データの取得（再帰呼び出し）
48                      i++;                        //カウントアップ
49                      makeDetail(i, dataCount);   //次行の明細を作成
50                  });
51              }
52          }
```

次レコードが無い場合、9-④へ進む

9-②

9-③

次レコードがある場合 再び9-②を呼び出す

ソース9-2　サンプルプログラム8:再帰呼び出し処理に修正(続き)

```
53      //----- PDF作成サブルーチン
54      function makePDF() {
55          //----- PDFレイアウト作成
56          var docDefinition = {
57              // 用紙指定
58              pageSize: 'A4',                         // 用紙サイズ
59              pageMargins: [35, 45, 35, 45],          // 余白
60              // 出力する内容
61              content: [
62                  {text:'商品一覧表', bold: true},
63                  {table: {
64                      headerRows: 1,                                          // 列タイトル行数
65                      width: ['auto','auto','auto','auto','auto','auto'],     // 列幅指定
66                      body: tableBody                                         // 表要素
67                      }
68                  }
69              ],
70              // スタイル（書式）指定
71              defaultStyle: {
72                  font:'Aozora'   // 日本語フォント
73                  }
74              };
75          //----- PDF生成
76          var pdfDocGenerator= pdfMake.createPdf(docDefinition);
77          //----- 生成したPDFを出力
78          pdfDocGenerator.open();
79      }
80          //----- 列タイトルを要素に追加
81          tableBody.push(['商品ＣＤ','商品名','カナ','単価','単位','画像']);
82          //----- 変数初期化
83          var i = 0;                              // 処理カウンター
84          const dataCount = widgetData.length;    // データ件数
85          //----- 明細作成とPDF作成処理の実行
86          makeDetail(i, dataCount);
87      }, function (err){
88          //エラー
89      });
90  }, function (err) {
91      //エラー
92  });
```

9-④

9-①

9-② を呼び出す

6. さいごに

本稿では、Valenceで実現可能なPDF帳票作成方法について紹介してきた。従来バージョンのValenceでは、日本語を含むPDFを出力する事ができず、帳票ツール等との組み合わせが現実的な手法であったが、Valence6.0ではValence単体で日本語を含むPDFが動的に生成できるようになった。

クライアントサイドでのPDF作成となる為、バッチ処理の実行結果を帳票印刷するような用途にはそぐわないが、画面からのリスト出力や各種定型帳票の出力等には充分活用できる。

JavaScriptを使用したコーディングによる帳票作成となる為、若干のJavaScriptスキルが必要とはなるが、pdfmakeによる帳票レイアウト作成の手順さえ習得できれば、今回のサンプルソースを参考に十分活用できるので、皆様もValenceからのPDF帳票出力にチャレンジして頂ければ幸いである。

Valence