【セッションNo．2】：Valence

Valenceにおける帳票（PDF）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

【アジェンダ】

- Valenceにおける従来の帳票出力について
- Valence6.0における新しいPDF出力
- pdfmakeを使用したPDFファイルの生成方法
- 画像やQR/バーコードを含むPDF作成
- ウィジェットデータを使用したPDF作成
- まとめ

Valenceにおける従来の帳票出力について

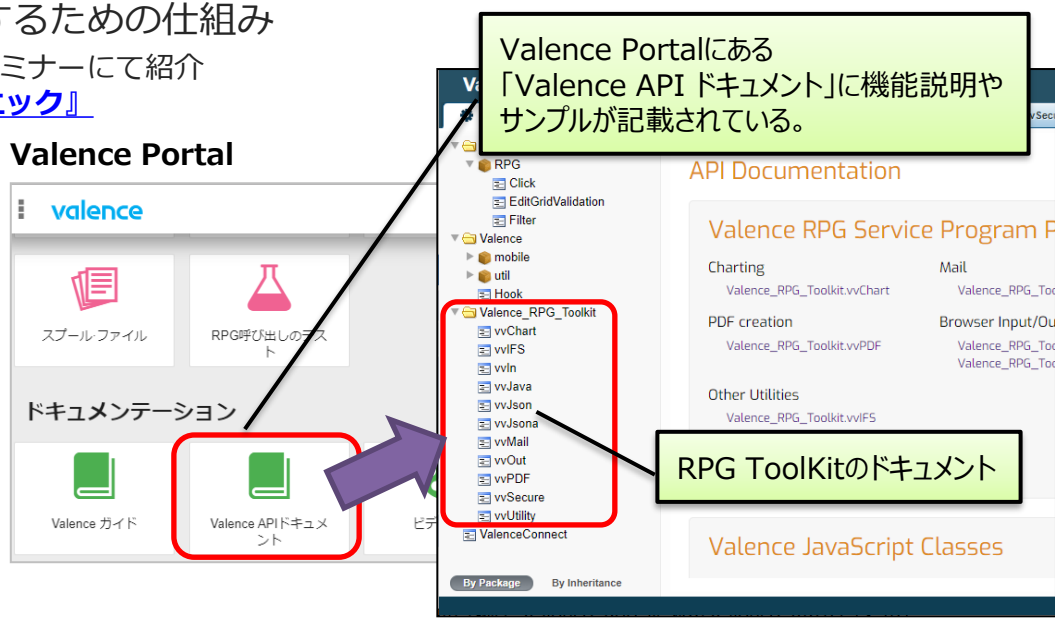
■ Valenceにおける従来の帳票出力について

- RPG ToolKit
  - RPGを使用して、Valenceを機能拡張するための仕組み
    - 具体的な使用方法は、第24回テクニカルセミナーにて紹介
      『Valence App Builder　RPG連携テクニック』

RPG ToolKit　の主な種類

分類	概要
vvIFS	IFS上のファイル操作
vvIn	ブラウザから情報取得
vvOut	ブラウザへレスポンス返却　および CSV/Excel等の出力
vvMail	メール送信
vvPDF	**PDFファイルの作成**
vvSecure	Valence認証情報取得
vvUtility	Valenceのユーティリティ機能

Valence Portal



Valenceの **“RPG ToolKit”** には、動的にPDFファイルを作成する為のAPIが以前より搭載されている！

Valenceにおける従来の帳票出力について

- vvPDF 使用例
  - サンプルプログラム１
    - カテゴリー別商品情報出力

IBM i

商品マスタ
MPRODP

DSPFMT　　レ　コ　ー　ド　設　計　書　　日付 2
時刻 1

物理ファイル	TECREP21/MPRODP	様式名	MPRODR	レコード長
様式記述	商品マスタ			

5= 詳細

選択	項目名	桁数	属性	キー順	開始	終了	テキスト記述／欄
_	ODPDCD	10	A	1 ANN	1	10	商品ＣＤ
_	ODPDNM	32	O		11	42	商品名
_	ODPDKN	20	A		43	62	商品カナ
_	ODTANK	9 0	S		63	71	単価
_	ODGEUN	9 0	S		72	80	原単価
_	ODUNIT	6	O		81	86	単位
_	ODCTCD	2	A		87	88	カテゴリＣＤ

行の位置指定 . . .
行 ....+....1....+....2....+....3....+....4....+.. | ..10...+...11....+...

商品ＣＤ	商品名		単位	カテゴリＣＤ
000001	C-00001	プリントＴシャツ フ	枚	06
000002	C-00002	アンダーシャツ ア	枚	06
000003	C-00003	インポートＴシャツ イ	枚	06
000004	D-00001	柑橘のど飴 カ	個	04
000005	D-00002	フルーツのど飴 フ	個	04
000006	J-10001	クリアーファイル ク	枚	03
000007	J-10002	フロッピーディスクケース フ	個	03
000008	J-10003	ワープロ用感熱紙 ワ	枚	03

App Builder画面
（Formウィジェット）

Formウィジェット
フィールド: CATEGO
カテゴリー一覧の選択リスト

PDF

画面で指示した
カテゴリCDに合致する
商品データを一覧形式でPDFに出力

■ Valenceにおける従来の帳票出力について

- vvPDF 使用例　（ボタンクリック: テンプレート『EXNABBTN』を使用）

```
0001.00      /copy qcpylesrc,vvHspec
0002.00     ** ------------------------------------------------------------
0003.00     **          TEC21PG10 : カテゴリ別商品情報出力
0004.00     ** ------------------------------------------------------------
0005.00     ** ------------------------------------------------------------
0006.00      /define includePDF
0007.00      /include qcpylesrc,vvNabBtn
0008.00     ** ------------------------------------------------------------
0009.00     ** program start
0010.00     ** ------------------------------------------------------------
0011.00      /free
0012.00        Initialize();
0013.00        Process();
0014.00        CleanUp();
0015.00        *inlr=*on;
0016.00      /end-free
0017.00     ** ------------------------------------------------------------
0018.00     p Process         b
0019.00     d                 pi
0020.00     d pdfDoc          s                   like(document)
0021.00     D TMPPATH         S             60A
0022.00     D VCATG           S              2A
0023.00     D SQLSTR          S            256A
0024.00      /free
0025.00        //フォーム上で入力された値を取得
0026.00        VCATG = GetFormChar('CATEGO');  //――カテゴリＣＤ
0027.00
```

1-①
vvPDFを使用する為の宣言を追加。

```
0028.00        //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)
0029.00        SQLSTR = 'SELECT * FROM MPRODP '
0030.00               + 'WHERE ODCTCD = ''' + VCATG + ''' '
0031.00               + 'ORDER BY ODCTCD, ODPDCD';
0032.00
0033.00        //PDFファイル保存先パスの指定
0034.00        TMPPATH = vvUtility_getValenceSetting('TEMP_PATH');
0035.00
0036.00        //データ取得結果を元に動的にPDFを作成
0037.00        vvPDF.path = %trim(TMPPATH) + 'TEMP_MPRODP.pdf';
0038.00        pdfDoc = vvPDF_newDocument(vvPDF);
0039.00        vvPdf_addTablefromSQL(vvPDF:pdfDoc:SQLSTR);
0040.00        vvPDF_closeDocument(pdfDoc);
0041.00
0042.00        //動的に作成されたPDFをダウンロード
0043.00        vvOut.download = '1';
0044.00        vvOut.file     = 'CATEGORY_' + VCATG + '.PDF';
0045.00        vvOut_file(vvPDF.path:vvOut);
0046.00
0047.00        //動的に作成されたPDFをIFS上から削除
0048.00        vvIfs_deleteFile(vvPDF.path);
0049.00      /end-free
0050.00     p                 e
```

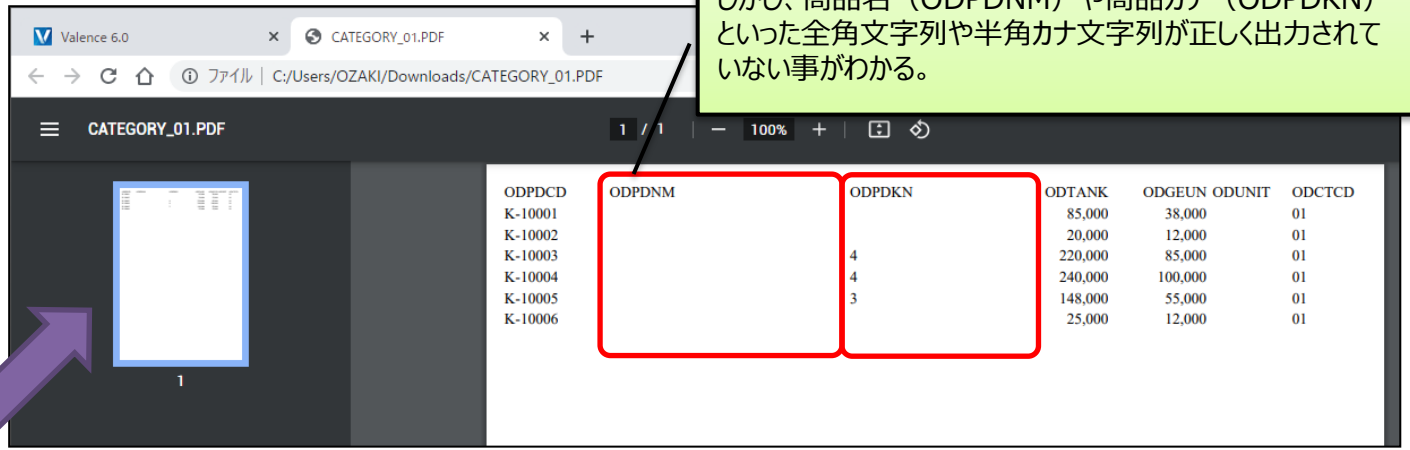
1-②
データを抽出する為のSQL文字列を作成。

1-③
vvPDFを使用してPDFを生成。

1-④
PDFをダウンロード。

■ Valenceにおける従来の帳票出力について

- vvPDF 使用例
  - サンプルプログラム１：実行例



ダウンロードされたPDFファイルを表示

条件に合致する商品マスターの情報が、一覧リスト形式でPDFに出力。

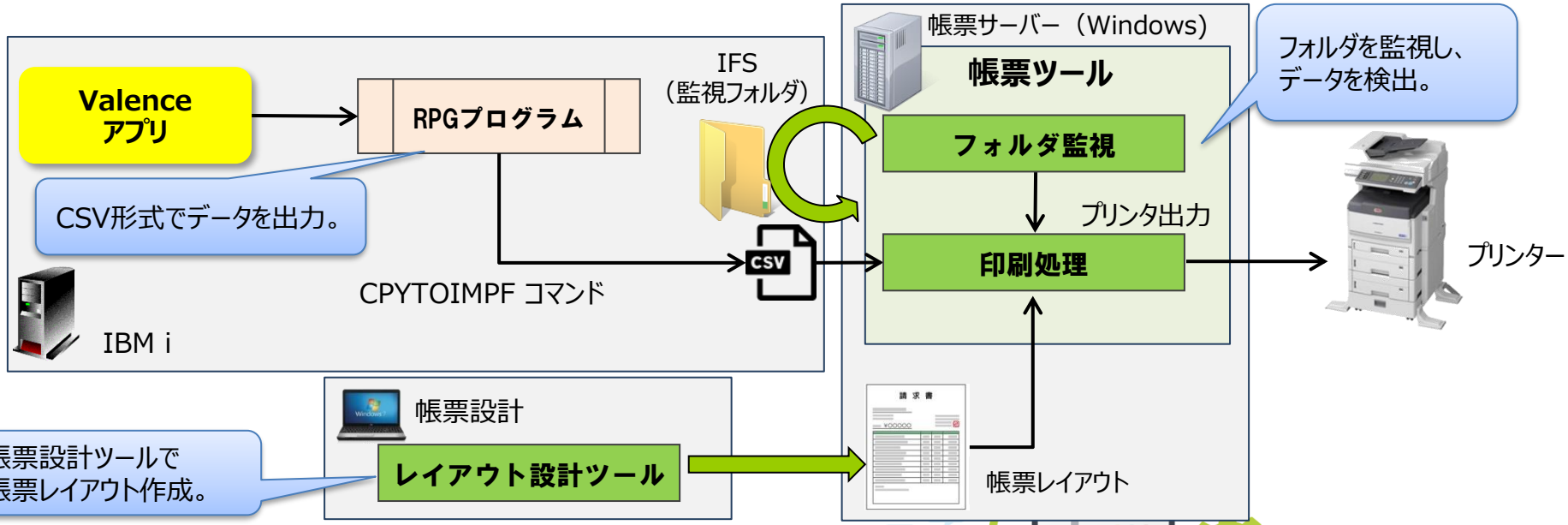
しかし、商品名（ODPDNM）や商品カナ（ODPDKN）といった全角文字列や半角カナ文字列が正しく出力されていない事がわかる。

IBM iに搭載されたPDF作成エンジンを使用しているが、日本のIBM iにも日本語フォントが含まれない為、漢字等が出力不可。

■ Valenceにおける従来の帳票出力について

- Valenceから帳票を出力する一般的な方法①

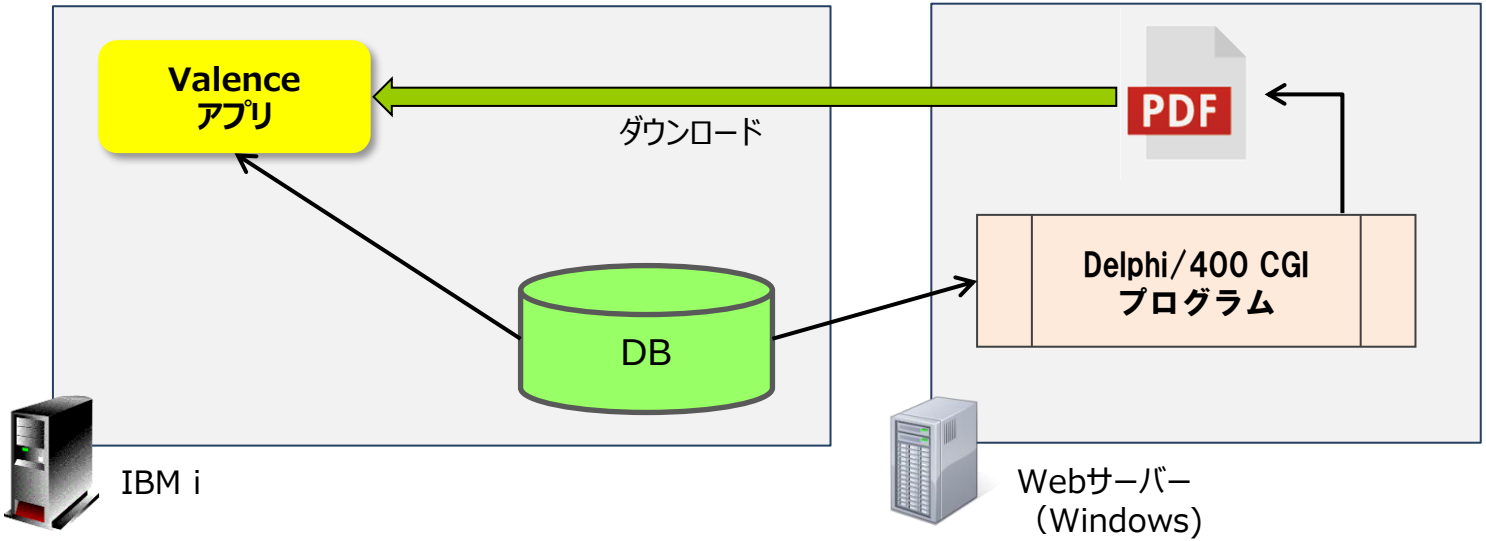
事前に定義した帳票レイアウトに、IBM i から出力されたCSVファイルを組み合わせて印刷を行う帳票ツール例



■ Valenceにおける従来の帳票出力について

- Valenceから帳票を出力する一般的な方法②

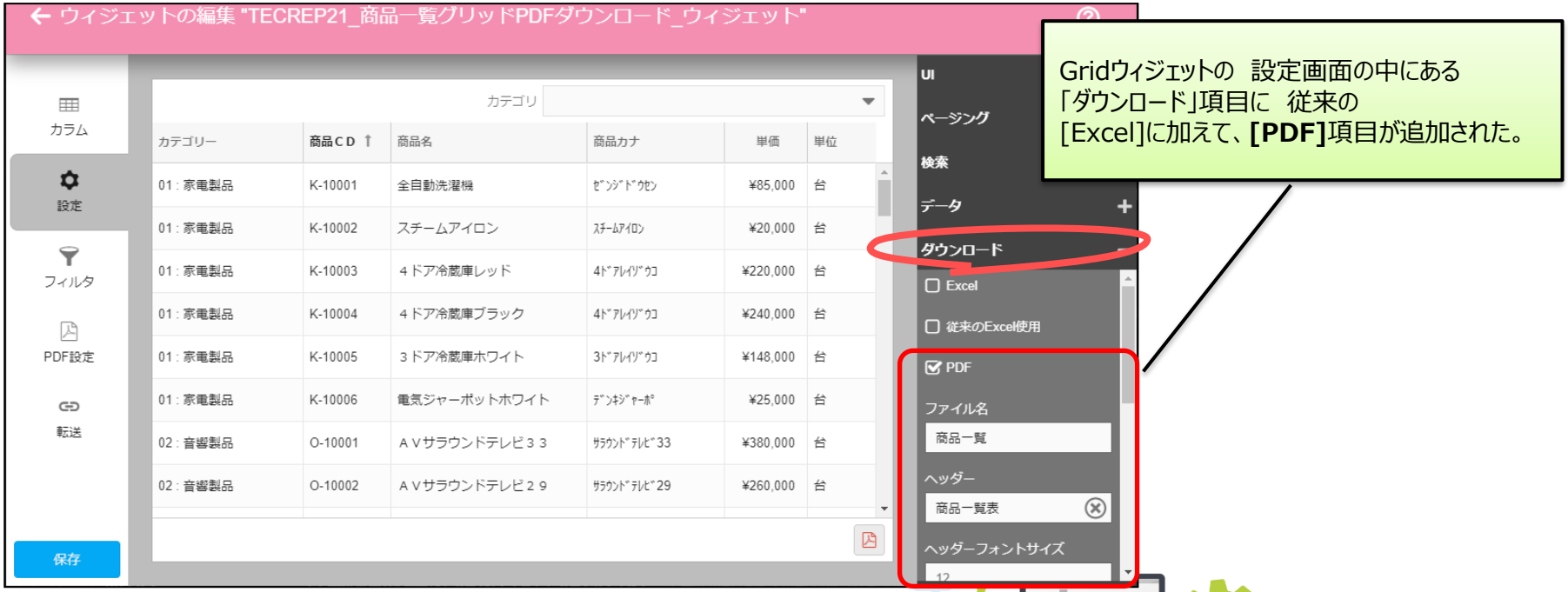
Delphi/400を組み合わせ： WebBroker（Webアプリ機能） + FastReport（帳票コンポーネント）



Valence6.0における新しいPDF出力

■ Valence6.0における新しいPDF出力

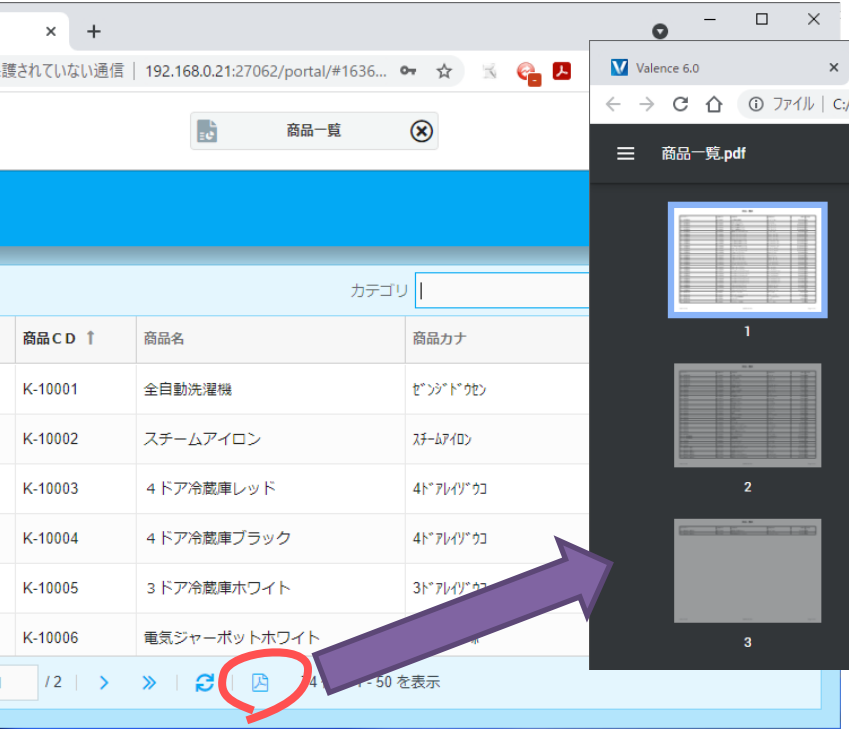
- Grid/Edit Gridウィジェットのダウンロード機能
  - ウィジェット設定画面：ダウンロード　にPDFが追加



■ Valence6.0における新しいPDF出力

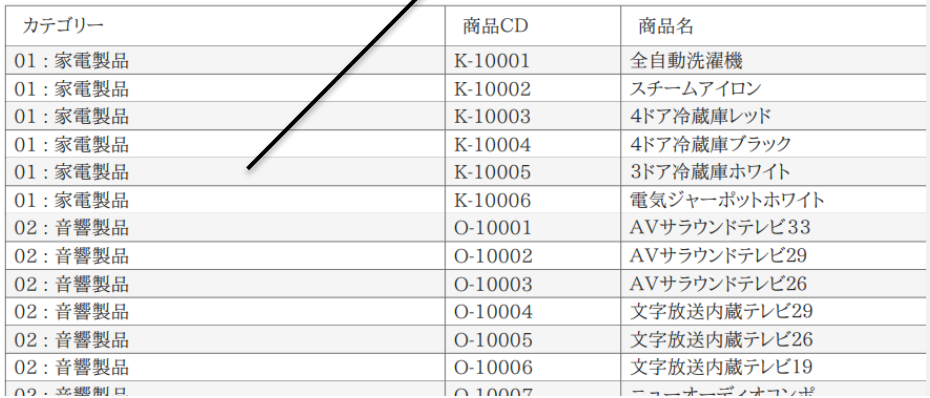
- Grid/Edit Gridウィジェットのダウンロード機能

ダウンロードされたPDFファイルを表示



Gridウィジェット　ダウンロードで出力されたPDF

日本語フィールドが正しく出力されていることがわかる。

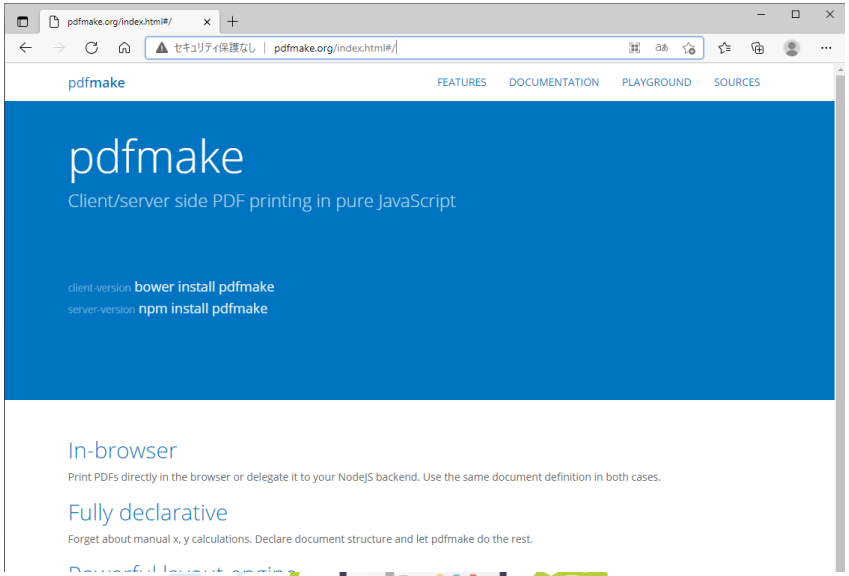


カテゴリー	商品CD	商品名
01：家電製品	K-10001	全自動洗濯機
01：家電製品	K-10002	スチームアイロン
01：家電製品	K-10003	4ドア冷蔵庫レッド
01：家電製品	K-10004	4ドア冷蔵庫ブラック
01：家電製品	K-10005	3ドア冷蔵庫ホワイト
01：家電製品	K-10006	電気ジャーポットホワイト
02：音響製品	O-10001	AVサラウンドテレビ33
02：音響製品	O-10002	AVサラウンドテレビ29
02：音響製品	O-10003	AVサラウンドテレビ26
02：音響製品	O-10004	文字放送内蔵テレビ29
02：音響製品	O-10005	文字放送内蔵テレビ26
02：音響製品	O-10006	文字放送内蔵テレビ19

実行例

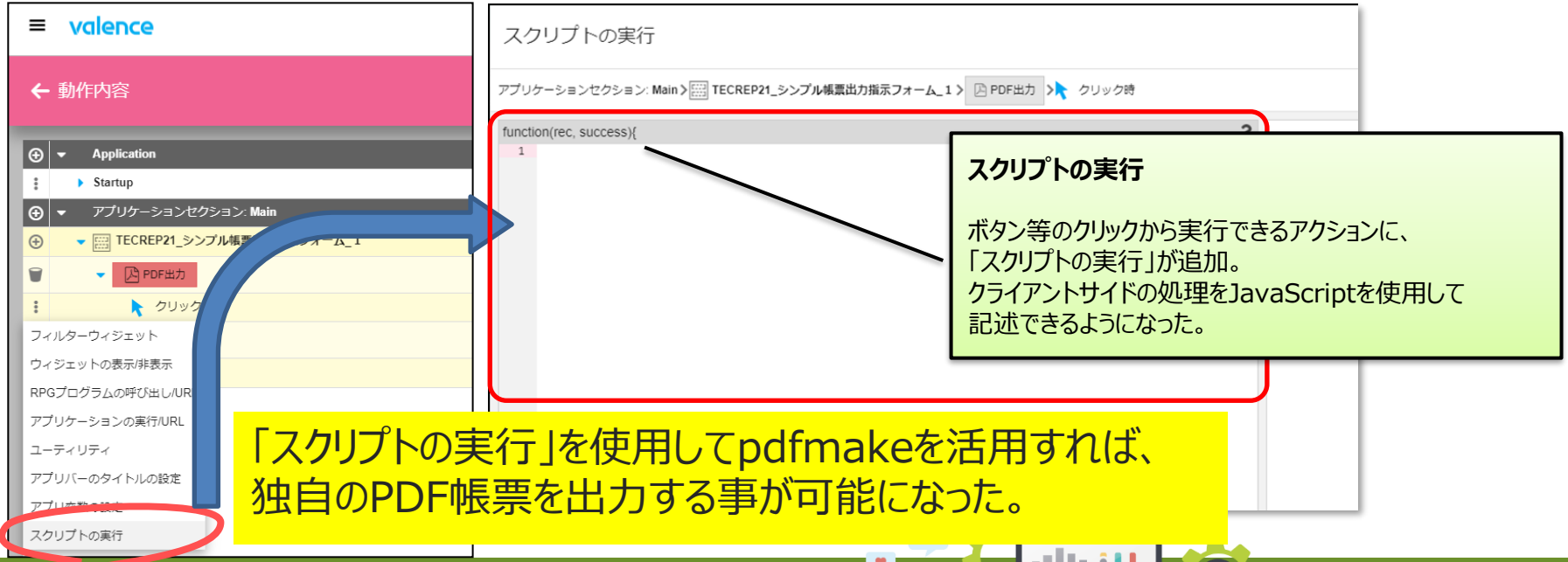
Valence6.0における新しいPDF出力

- なぜ、日本語を含むPDFが作成できるようになったか？
  - 開発元CNX社に確認したところ、従来のIBM i エンジンを使用した方法ではなく、クライアント側でPDFを作成するJavaScriptライブラリを使用している事が判明。
  - オープンソースライブラリである **“pdfmake”** が、Valence環境に追加された。
    - http://pdfmake.org/#/
  - pdfmakeも通常は、日本語フォントには非対応。
  - Valence環境上にあるpdfmakeには「**あおぞら明朝フォント**」が追加で組込み済の為、日本語出力も可能。



Valence6.0における新しいPDF出力

- Valence6.0に追加された「スクリプトの実行」
  - Valenceの動作内容（アクション）にて、従来のRPG呼び出しに加え、
    クライアントサイドで実行可能な処理（JavaScript）が定義可能になった。



pdfmakeを使用したPDFファイルの生成方法

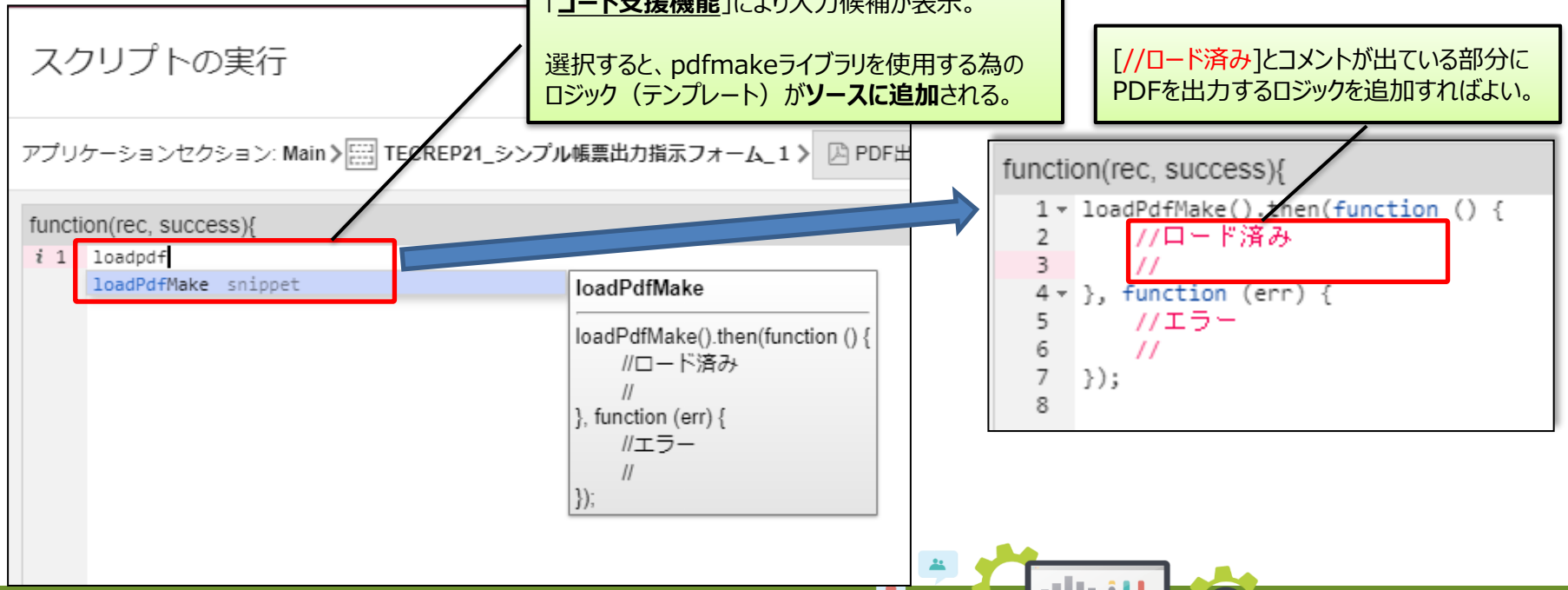
pdfmakeを使用したPDFファイルの生成方法

- pdfmakeをスクリプトに追加
  - loadPdfMakeメソッド

スクリプトエディタ上で、「loadpdf」と入力すると、「**コード支援機能**」により入力候補が表示。

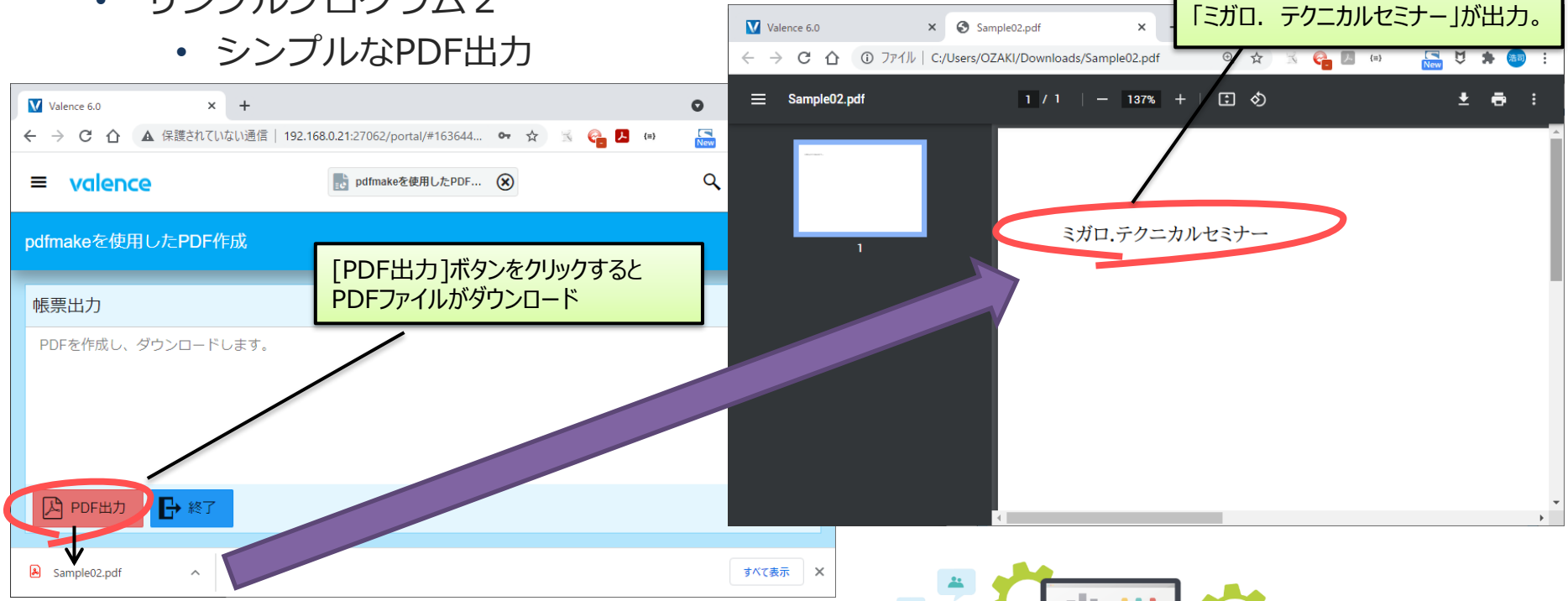
選択すると、pdfmakeライブラリを使用する為のロジック（テンプレート）が**ソースに追加**される。

[//ロード済み]とコメントが出ている部分にPDFを出力するロジックを追加すればよい。



■ pdfmakeを使用したPDFファイルの生成方法

- pdfmakeを使用したPDF出力
  - サンプルプログラム２
    - シンプルなPDF出力



pdfmakeを使用したPDFファイルの生成方法

- サンプルプログラム２
  - シンプルなPDF出力


```
//---- PDFMakeを起動
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'
        }
    };

    //---- PDFレイアウト作成
    var docDefinition = {
        //用紙指定
        pageSize: 'A4',                          // 用紙サイズ
        pageOrientation: 'landscape',            // 向き(横)
        pageMargins: [ 40, 60, 40, 60 ],         // 余白

        //出力する内容
        content:
        [
            {text: 'ミガロ. テクニカルセミナー'}    // 文字を1行出力
        ],

        //スタイル(書式)指定
        defaultStyle: {
            font: 'Aozora'                       // フォント指定
        }
    };

    //---- PDF生成
    var pdfDocGenerator = pdfMake.createPdf(docDefinition);
    pdfDocGenerator.download('Sample02.pdf');    //PDFダウンロード
}, function (err) {
    //エラー
});
```

2-①

2-②

2-③

2-①
Aozoraというフォント名でValenceに組み込まれた「あおぞら明朝フォント」を使用する事を宣言。

2-②
PDFのレイアウトを **JSON形式**で記述。
出力内容を **content要素**に定義。

2-③
PDFを生成して、作成されたPDFをファイルとしてダウンロード。

pdfmakeを使用したPDFファイルの生成方法

- loadPdfMakeメソッドの使用方法
  - PDFの生成

```
var pdfDocGenerator = pdfMake.createPdf(docDefinition);
```

  - 生成されたPDFの出力

```
pdfDocGenerator.download('Sample02.pdf');      //pdfをダウンロード
pdfDocGenerator.open();                        //pdfファイルを開く
pdfDocGenerator.print();                       //pdfを印刷プレビューで開く
```

  - レイアウト作成用変数　(例：docDefinition)　**各要素をJSON形式で記述**

要素	概要	記述例
pageSize	用紙サイズを指定	pageSize: 'A4'　(A4サイズ)　　'A4','B5', 'A3' ... 等と指定
pageOrientation	用紙の向きを指定	pageOrientation: 'landscape'　(横向き) / 'portrait'（縦向き）
pageMargins	余白を指定	pageMargins: [40, 60, 40, 60]　左、上、右、下の順
content	印刷内容を指定	content: 'This is a sample PDF'
defaultStyle	全体に適用する初期スタイル	defaultStyle: { font: 'Aozora' fontSize: 15 }

■ pdfmakeを使用したPDFファイルの生成方法

- ウィジェット上のフィールド値やアプリ変数値を出力
  - ウィジェット上のフィールド値
    - スクリプトでは、**rec.get([フィールドID])** で取得できる。
  - アプリ変数の値
    - スクリプトでは、**getAppVar([変数名])** で取得できる。

Formウィジェットの設計画面　（下図の真ん中部分が、Formのプレビュー）

ウィジェット上のフィールド値（例）

```
商品ＣＤ：rec.get('F1_ODPDCD')
商品名：　rec.get('F1_ODPDNM')
```

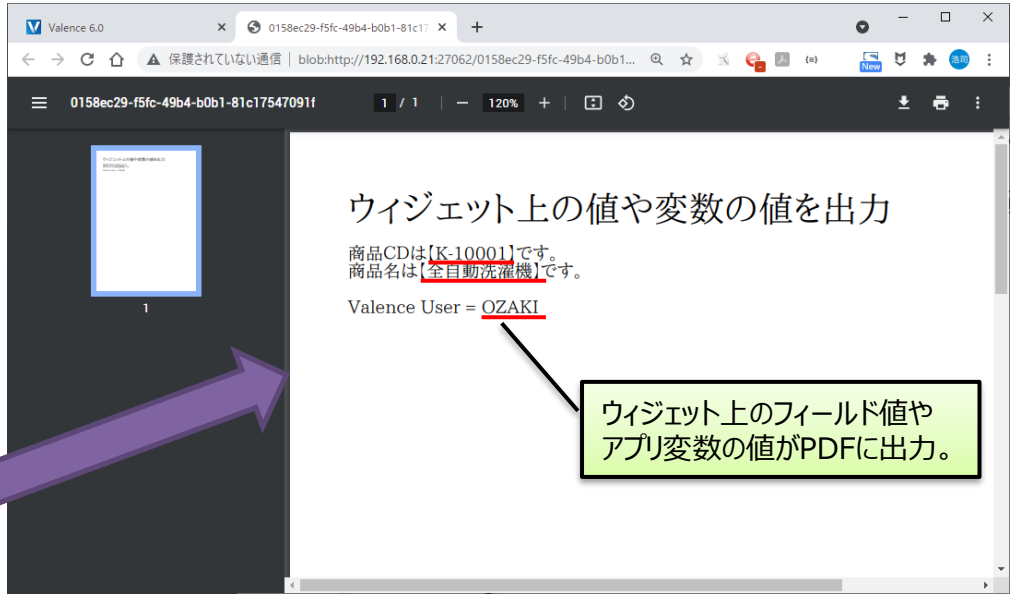
アプリ変数の値（例）

```
getAppVar('nabUser')
```

※[**nabUser**]は、デフォルトで用意されている変数名で、
Valenceにログインしているユーザー名が取得できる。

pdfmakeを使用したPDFファイルの生成方法

- ウィジェット上のフィールド値やアプリ変数値を出力
  - サンプルプログラム３
    - 画面値や変数値の出力



pdfmakeを使用したPDFファイルの生成方法

コード例

```
//---- 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'
        }
    };
    //---- PDFレイアウト作成
    var docDefinition = {
        //出力する内容
        content: [
            {text: 'ウィジェット上の値や変数の値を出力', fontSize: 24},
            {text: ' '},
            {text: '商品CDは【' + rec.get('F1_ODPDCD') + '】です。'},        //商品CD
            {text: '商品名は【' + rec.get('F1_ODPDNM') + '】です。'},        //商品名
            {text: ' '},
            {text: 'Valence User = ' + getAppVar('nabUser') }              //アプリ変数(Valenceユーザー)
        ],
        //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);
    pdfDocGenerator.open();                                                // PDFを開く
}, function (err) {
    //エラー
});
```

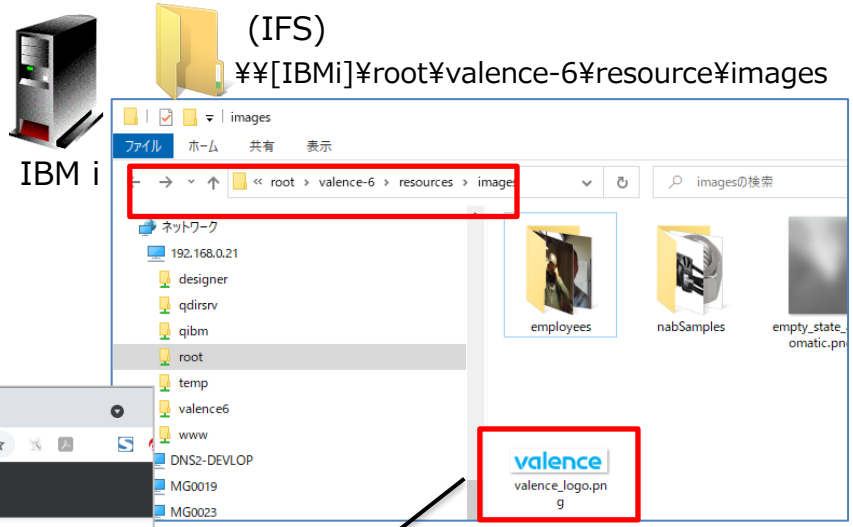
3-①
フィールド値の取得には、
reg.get([フィールドID])
アプリ変数値の取得には、
getAppVar([変数名])
を使用。

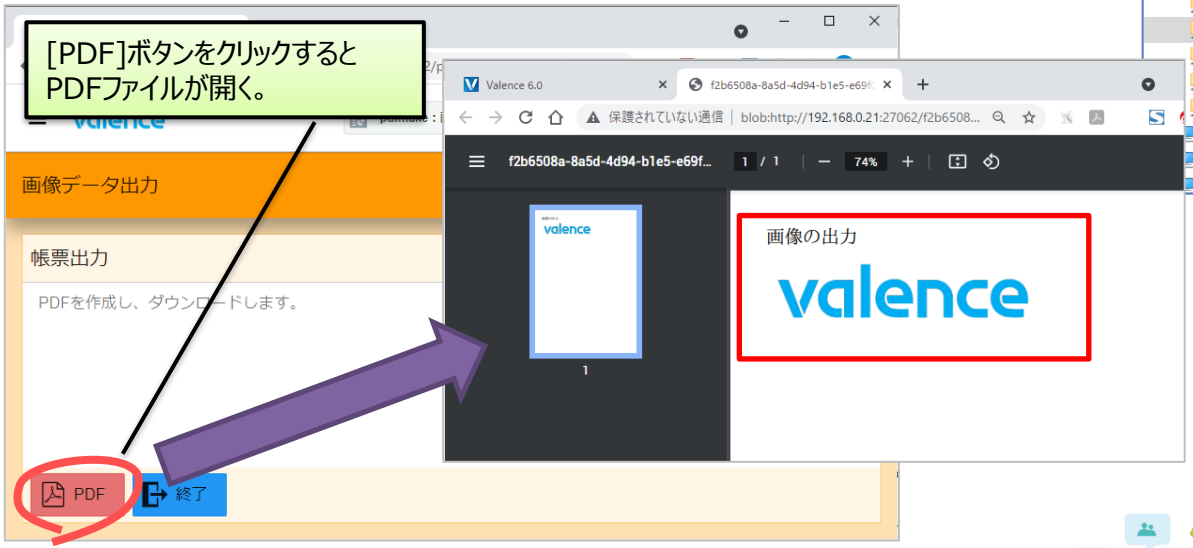
3-①

画像やQR/バーコードを含むPDF作成

■ 画像やQR/バーコードを含むPDF作成

- 画像を含むPDF作成
  - サンプルプログラム４
    - IFS上にあるValenceロゴをPDFに出力

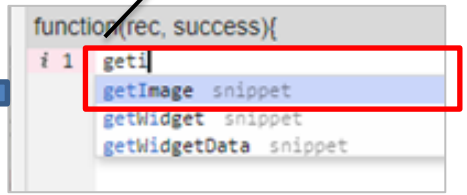




■ 画像やQR/バーコードを含むPDF作成

- Valence スクリプト上で画像を取得する方法
  - getImageメソッド
    - 画像の読込は、非同期で行われる点に注意。

スクリプトエディタ上で
getImageと入力すると
画像取得処理のテンプレートが挿入できる。



読込したい画像のPATHを指定

```
13
14  (a) getImage('imagePath').then(function (imageData) {
15          //データ
16          //
17
18          (c)： 画像読み込み後に実行
19
20      }, function (err) {
21          //エラー
22          //
23      });
24
25
26  (b)： (a)の後続処理として、(c)より前に実行される
27
```

getImageメソッドで画像が取得された後に
(c)の部分が実行される。
この中で、取得した画像データ変数
imageDataを使用して、画像処理を行う。

■ 画像やQR/バーコードを含むPDF作成

- 画像を含むPDF作成
  - サンプルプログラム４
    - ソース例

```
//----- 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'
        }
    };
    //---- 画像データ取得
    getImage('/resources/images/valence_logo.png').then(function (imageData) {
        //---- 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
            ],
            //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);
         pdfDocGenerator.open();     // ファイルを開く
    }, function (err) {
        //---- 画像取得エラー
});
}, function (err) {
    //エラー
});
```

4-①
IFS上の画像ファイルを取得し、**imageData変数**へセット。

4-①

4-②
getImageメソッドの取得後に、帳票レイアウトを定義して、PDFを生成。生成されたPDFを開く。

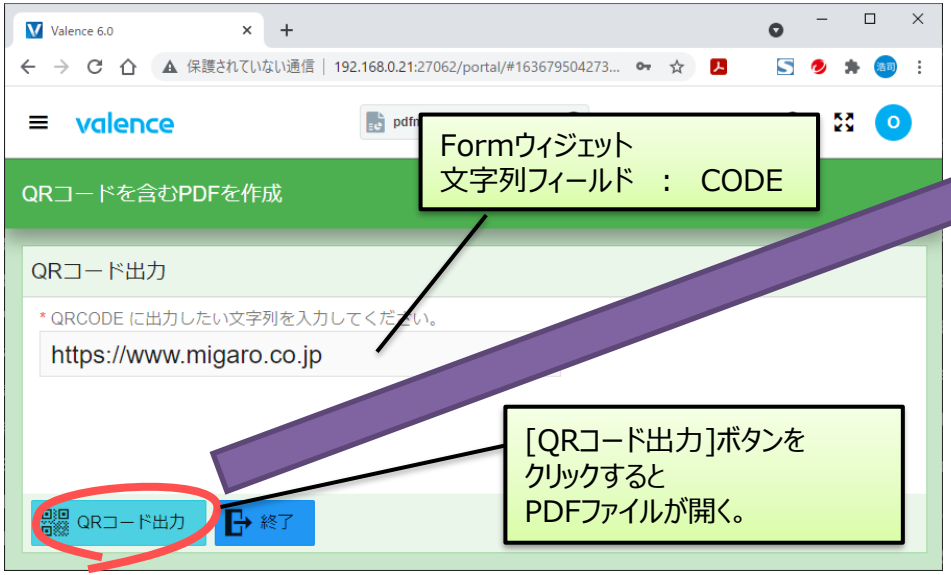
4-②

4-③

4-③
画像のPDF出力は、pdfmakeの**image要素**に取得した**imageData変数**をセットすればよい。

■ 画像やQR/バーコードを含むPDF作成

- QRコードを含むPDF作成
  - サンプルプログラム５
    - 入力した文字列をQRコードに変換して出力



■ 画像やQR/バーコードを含むPDF作成

- QRコードを含むPDF作成
  - サンプルプログラム５
    - ソース例

```
//---- PDFMakeを起動
loadPdfMake().then(function () {

    //----- フォーム上の入力値を取得
    const CodeVal = rec.get('CODE');  // 文字列フィールド[CODE]

    //---- 日本語フォント定義
    pdfMake.fonts = {
        Aozora: {
          normal: 'AozoraMinchoRegular.ttf',
          bold:   'AozoraMincho-bold.ttf'
        }
    };

    //---- PDFレイアウト作成
    var docDefinition = {
        //出力する内容
        content: [
            {text: 'QRコードの出力', fontSize: 24},
            {text: '入力値'},
            {text: CodeVal, fontSize: 16},
            {text: ' '},
            {qr: CodeVal}                          // QRコード
        ],
        //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);
    pdfDocGenerator.open();
}, function (err) {

});
```

5-①

5-①
rec.get('CODE')
[CODE]フィールドを値を取得

5-②
QRコードを出力
qr：[QRコード化したい文字列]

5-②

pdfmakeにはQRコード（2次元バーコード）を出力できる要素が用意されている。
しかし、JANコード等の1次元バーコードを出力する要素はない。**方法はないか？**

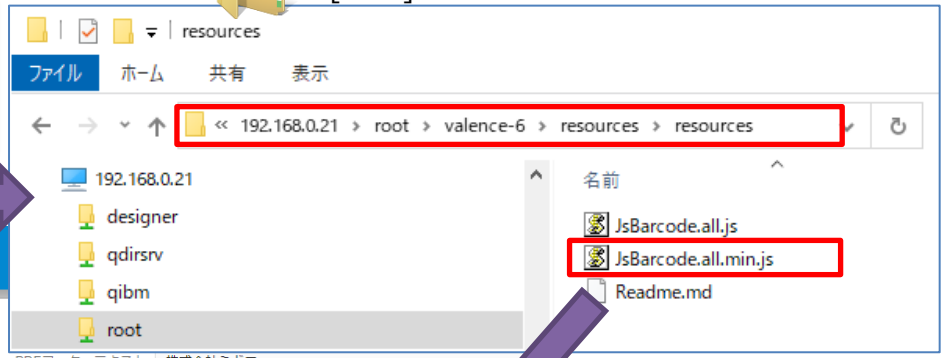
■ 画像やQR/バーコードを含むPDF作成

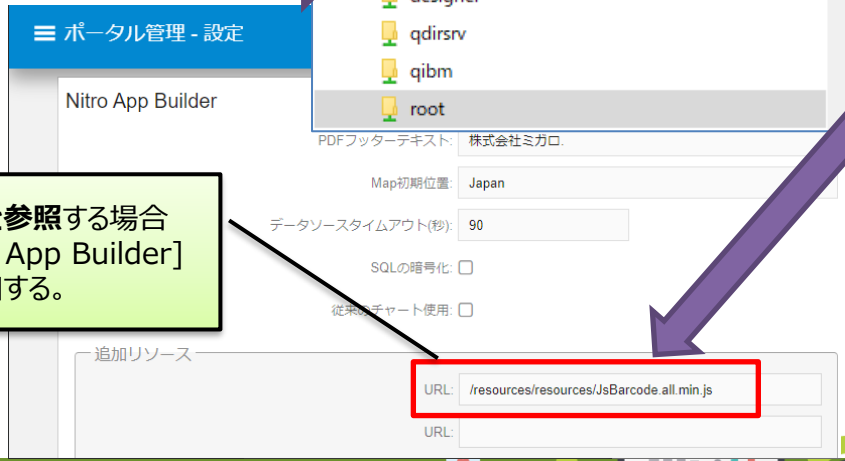
- バーコード作成ライブラリを活用
  - JsBarcode (https://Lindell.me/JsBarcode/)

ダウンロードしたライブラリ(**JsBarcode.all.min.js**)をValenceの[resources]フォルダにコピー。

(IFS)
¥¥[IBMi]¥root¥valence-6¥resources¥resources

JsBarcode
JavaScriptを使用して、バーコードを生成するライブラリ。 **バーコード画像**が生成できる。

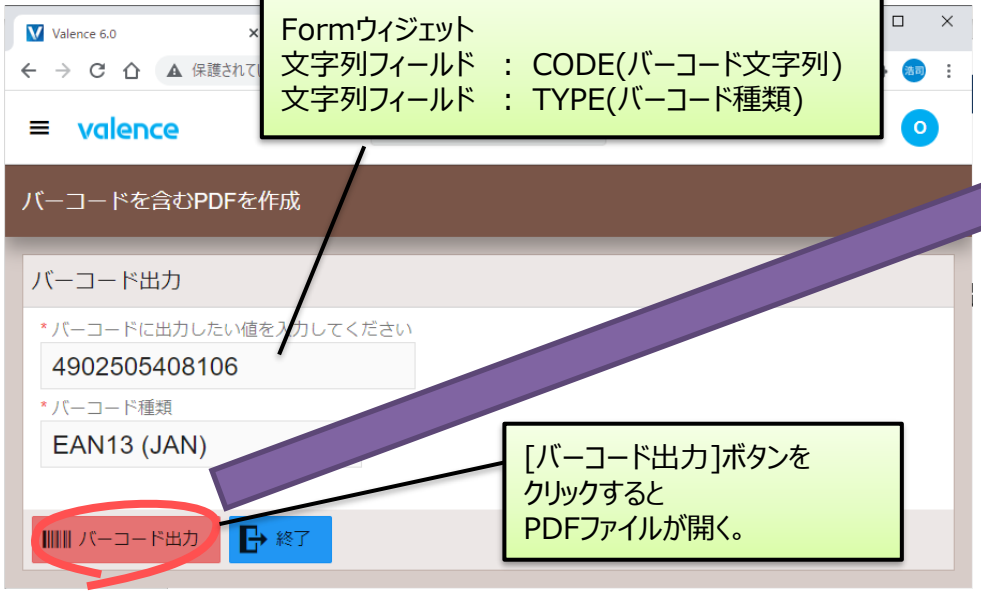




App Builderから**外部ライブラリを参照**する場合[ポータル管理]→[設定]→[Nitro App Builder]にある**"追加リソース"にパスを追加**する。

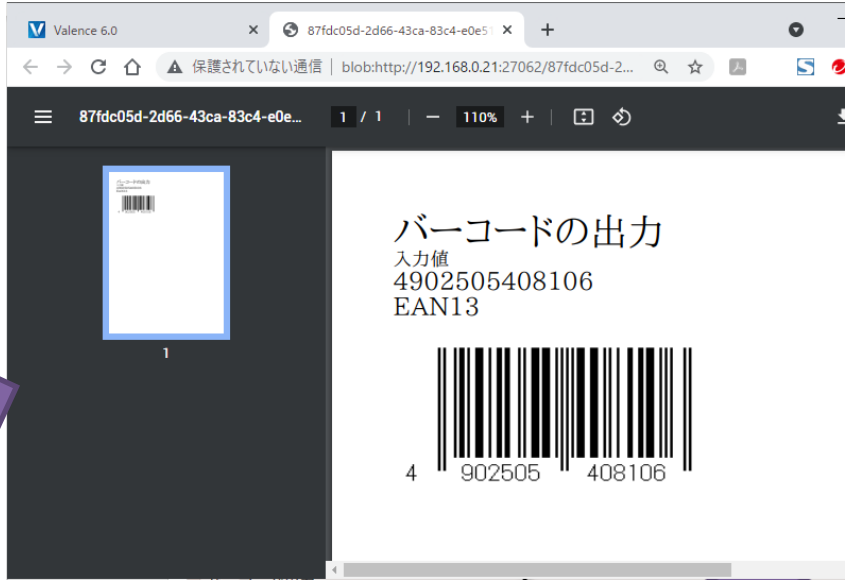
■ 画像やQR/バーコードを含むPDF作成

- バーコードを含むPDF作成
  - サンプルプログラム６
    - 入力した文字列をバーコードに変換して出力



Formウィジェット
文字列フィールド　：　CODE(バーコード文字列)
文字列フィールド　：　TYPE(バーコード種類)

[バーコード出力]ボタンを
クリックすると
PDFファイルが開く。



■ 画像やQR/バーコードを含むPDF作成

- バーコードを含むPDF作成
  - サンプルプログラム６
    - ソース例

6-②
canvas(画像)要素を生成し
JsBarcode を使用してバーコード画像を作成。

JsBarcode([画像要素], 変換文字列, [書式])

6-①
[CODE]/[TYPE]フィールド値を取得。
バーコード種類：下記が指定可能。
EAN13　（＝JAN）
CODE39
CODE128
CODABAR　（＝NW-7）

6-③
pdfmakeの画像(image)として出力。

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

    //----- 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
```

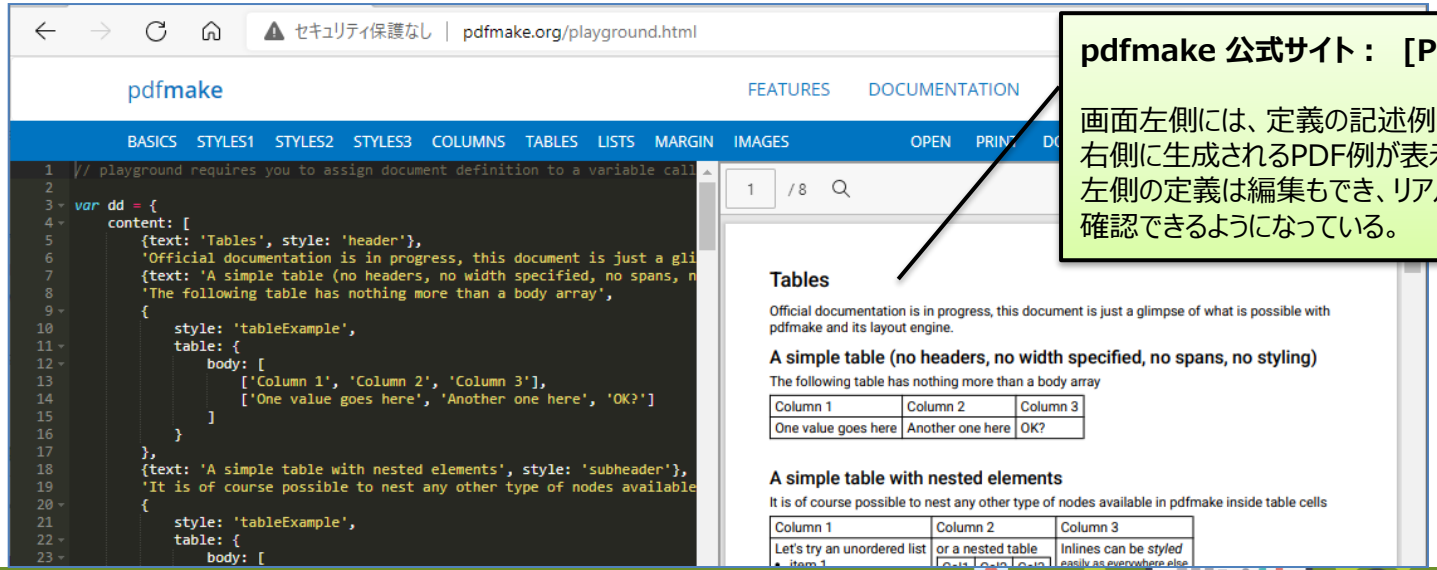
```
    //----- キャンバスを作成しバーコードを発行
    var canvas = document.createElement('canvas');
    JsBarcode(canvas, CodeVal,  {format: TypeVal});

    //---- 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}
        ],
        //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);
    pdfDocGenerator.open();
}, function(err) {

});
```

■ 画像やQR/バーコードを含むPDF作成

- pdfmakeの主な要素
  - 公式サイトのドキュメントに定義方法が記載。
  - PLAYGROUND (http://pdfmake.org/playground.html)
    - pdfmakeの各要素の実装例をブラウザ上で直接確認可能。



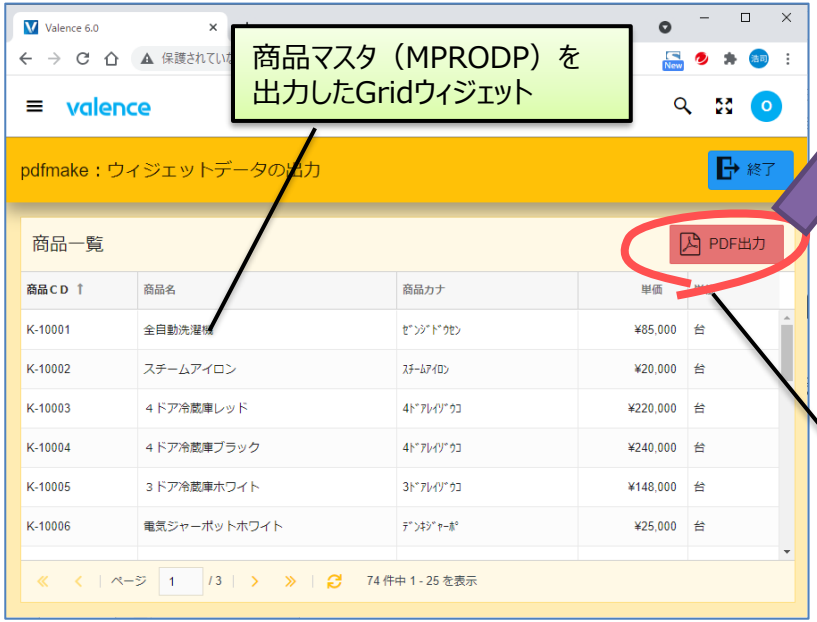
pdfmake 公式サイト： [PLAYGROUND]ページ

画面左側には、定義の記述例、
右側に生成されるPDF例が表示される。
左側の定義は編集もでき、リアルタイムに変更点が
確認できるようになっている。

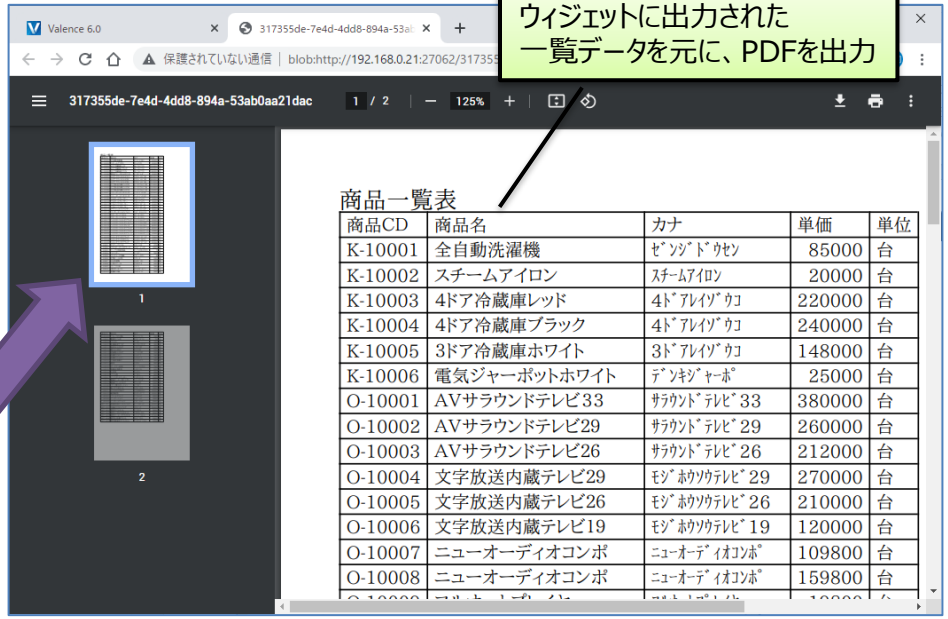
ウィジェットデータを使用したPDF作成

■ ウィジェットデータを使用したPDF作成

- ウィジェットデータのPDF作成
  - サンプルプログラム７
    - Gridウィジェットの一覧情報を出力



商品マスタ（MPRODP）を出力したGridウィジェット



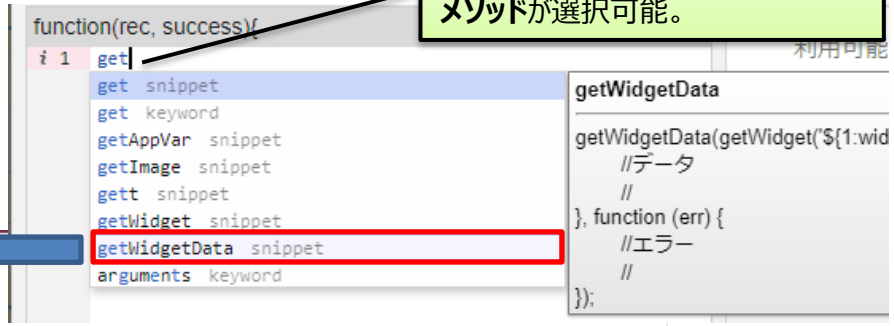
ウィジェットに出力された一覧データを元に、PDFを出力

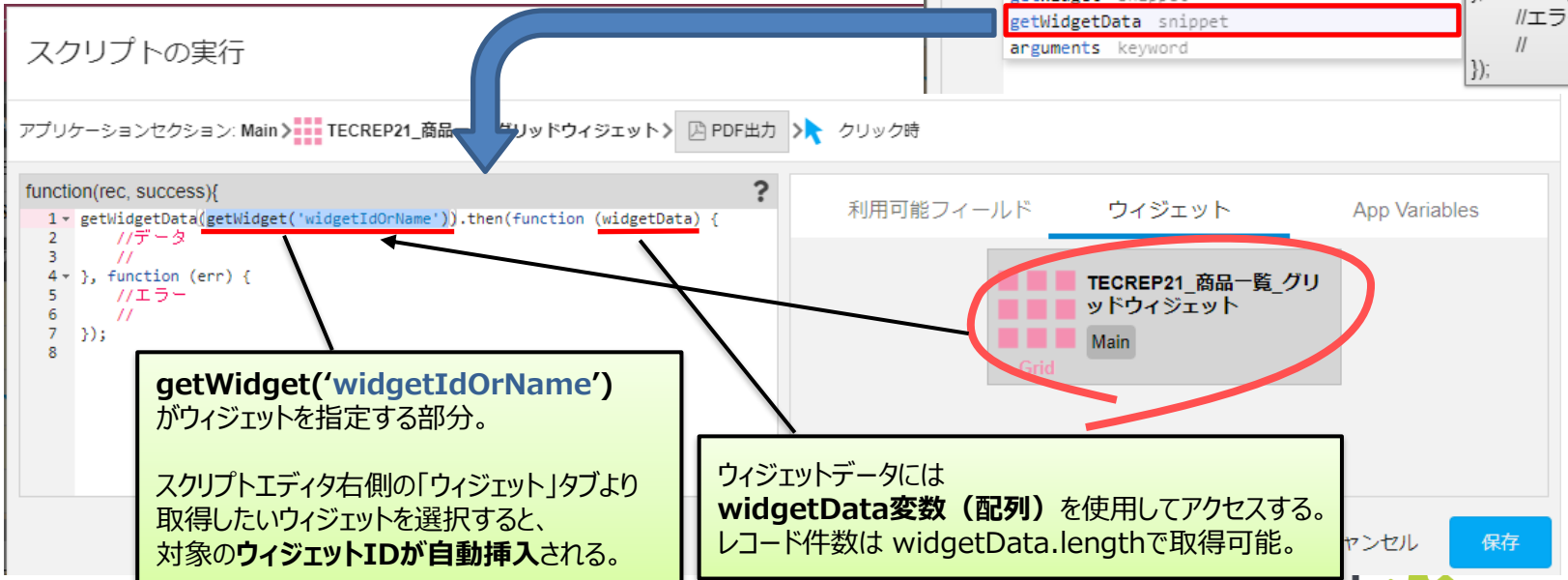
[PDF出力]ボタンをクリックするとPDFファイルが開く。

■ ウィジェットデータを使用したPDF作成

- ウィジェットデータの取得方法
  - getWidgetDataメソッド
    - 指定したウィジェットのデータを取得

スクリプトエディタ上で
getと入力すると**getWidgetData**
メソッドが選択可能。





getWidget(‘widgetIdOrName’)
がウィジェットを指定する部分。

スクリプトエディタ右側の「ウィジェット」タブより
取得したいウィジェットを選択すると、
対象の**ウィジェットIDが自動挿入**される。

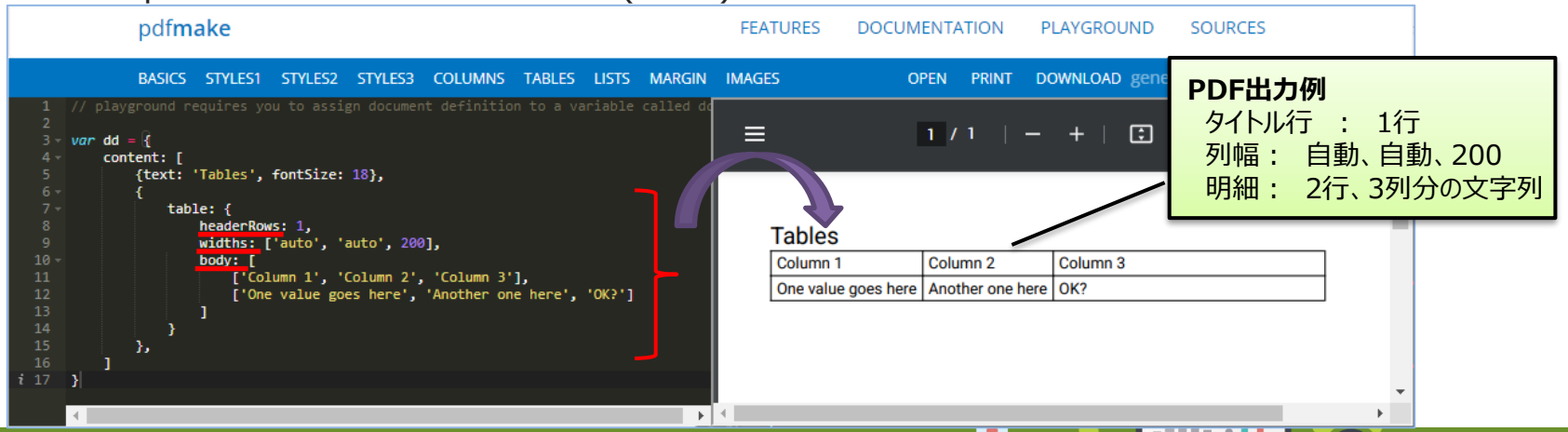
ウィジェットデータには
widgetData変数（配列）を使用してアクセスする。
レコード件数は widgetData.lengthで取得可能。

■ ウィジェットデータを使用したPDF作成

- pdfmakeにおける表の作成
  - table要素

要素	概要	備考
headerRows	タイトル行の行数	改ページの際に、先頭に出力されるタイトル部分の行数を指定
widths	各列の列幅	各列毎の幅を指定 例：100(固定幅) /'20%'(割合)/'auto'(列幅自動)/'*'(残りを埋める)
body	表の内容（明細）	1行分のデータを[ ]でまとめて、各セルをカンマで区切る　（配列として定義）

  - pdfmake PLAYGROUND で表(table)を確認した例



■ ウィジェットデータを使用したPDF作成

- サンプルプログラム7
  - ソース例

```
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('1281')).then(function (widgetData) { 7-②
        //----- 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']); 7-③
        // 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);
        }
        //----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
                    widths: ['auto','auto','auto','auto','auto'],   // 列幅指定
                    body: tableBody                                 // 表の要素
                    }
                }
            ],
            //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);
        pdfDocGenerator.open();
    }, function (err){
        //エラー
    });
}, function (err) {
    //エラー
});
```

7-①
表body部の配列情報を保持する変数（tableBody配列）を宣言/初期化。

7-②
Gridウィジェットのデータを取得。

7-③
表のタイトル行を配列に追加。(push)

7-④
行の各項目を保持する変数を使用して各フィールドの値をpushし、1行分のデータをtableBody配列に追加。

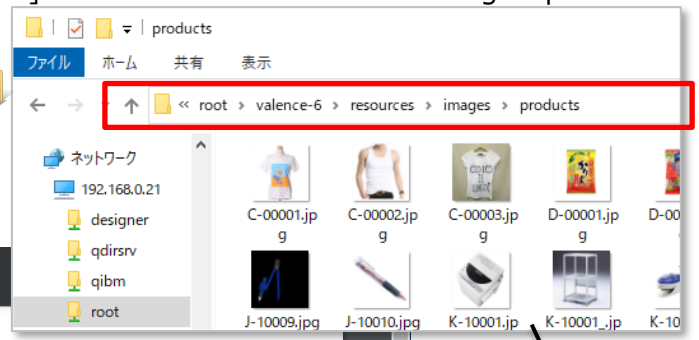
7-⑤
tableBody配列を使用して表を作成。

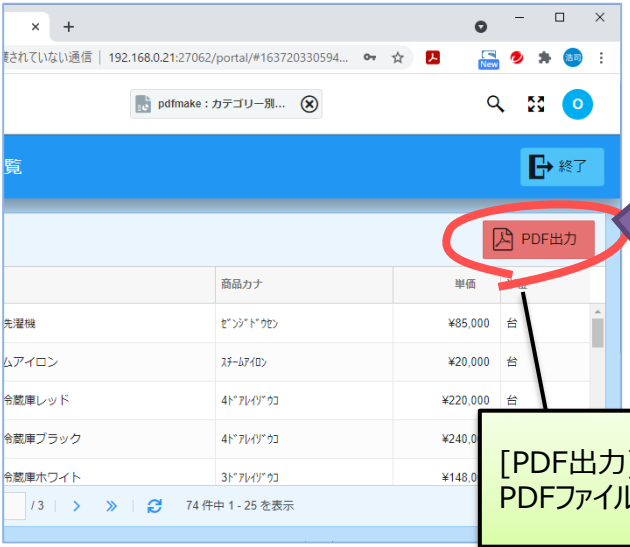
■ ウィジェットデータを使用したPDF作成

- 画像を追加した一覧帳票のカスタマイズ
  - サンプルプログラム８
    - IFS上にある商品画像を一覧に追加

¥¥[IBMi]¥root¥valence-6¥resources¥images¥products¥

(IFS)





IFS上に
“[商品CD].jpg”として
画像を用意。

[PDF出力]ボタンをクリックすると
PDFファイルが開く。

■ ウィジェットデータを使用したPDF作成

- サンプルプログラム７に画像処理を追加したソース（一部抜粋）
  - このソースは動作しない

```
//----- ウィジェットデータの取得
getWidgetData(getWidget('1281')).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','画像']);  8-①
    // 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

(a)     //----- 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）
        const imgPath = '/resources/images/products/' + widgetData[i].F1_ODPDCD + '.jpg';
        getImage(imgPath).then(function (imageData) {
            // 画像取得成功の場合
            rowData.push({image: imageData, width: 85});     // 画像を出力
        }, function (err) {
            // 画像取得エラーの場合
            rowData.push({text: "画像無し"});                // 画像無しメッセージ
        });                                                  (c)   8-②

(b)     //tableBodyに要素を追加
        tableBody.push(rowData);
    }
                                【以下続く】
```

8-①
列タイトルに「画像」を追加。

8-②
追加した画像取得処理
getImageメソッド

(a)getImageによる画像読み込みは、
非同期に実行される為、
(c)の[画像項目の追加]部分より、
(b)の[明細行の追加]の部分が
先に実行されてしまい、正常に動作しない。

■ ウィジェットデータを使用したPDF作成

- 画像読み込みを確実に処理するには…
  - getImageメソッドをネストする必要がある。

```
getImage(imgPath).then(function (imageData) {
    // 1つ目の画像取得処理

    getImage(imgPath).then(function (imageData) {
        // 2つ目の画像取得処理

        getImage(imgPath).then(function (imageData) {
            // 3つ目の画像取得処理

        }, function (err) {
            // エラー
        });
    }, function (err) {
        // エラー
    });
}, function (err) {
    // エラー
});
```

  - 処理件数が可変の場合、どうすればよいか？

getImage処理を含む明細作成処理を
「**再帰呼び出し処理**」とすれば、実装する事が可能。

■ ウィジェットデータを使用したPDF作成

- 再帰呼び出し処理に変更
  - ソース例

9-① makeDetail関数
ｉ行目のウィジェットデータより明細要素を作成。
（画像処理を含む。）

```
var tableBody = [];     // 表の要素を保持する変数

//----- 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('1281')).then(function (widgetData) {
        //----- 明細行作成サブルーチン
        function makeDetail(i, dataCount) {
            //------ 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
            if (i >= dataCount)  {
                makePDF();          // PDF作成処理を呼出
                return;
            }
            else
            {
                //----- 明細行の作成
                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, alienment: 'right'});   //単価
                rowData.push({text: widgetData[i].F1_ODUNIT});   //単位

                //----- 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)
                const imgPath = '/resources/images/products/' + widgetData[i].F1_ODPDCD + '.jpg';
                getImage(imgPath).then(function (imageData) {
                    // 画像取得成功の場合
                    rowData.push({image: imageData, width: 85});     //画像を出力
                    // 行明細書き込み
                    tableBody.push(rowData);     //行の要素をtableBodyへ追加
                    // 次データの取得(再帰呼び出し)
                    i++;                         //カウントアップ
                    makeDetail(i, dataCount);    //次行の明細を作成
                }, function (err) {
                    // 画像取得エラーの場合
                    rowData.push({text: "画像無し"});//画像無しメッセージ
                    // 行明細書き込み
                    tableBody.push(rowData);     //行の要素をtableBodyへ追加
                    // 次データの取得(再帰呼び出し)
                    i++;                         //カウントアップ
                    makeDetail(i, dataCount);    //次行の明細を作成
                });
            }
        }
```

9-①

全件明細を処理した場合
9-②makePDF関数を
呼び出す。

9-①

行番号 iをカウントアップし
9-①makeDetail関数を
再帰呼び出しする。

【次ページへ続く】

■ ウィジェットデータを使用したPDF作成

- 再帰呼び出し処理に変更
  - ソース（続き）

```
//----- PDF作成サブルーチン
function makePDF() {
    //----- PDFレイアウト作成
    var docDefinition = {
        // 出力する内容
        content: [
            {text:'カテゴリ別商品一覧', bold: true},
            {table: {
                layout: 'lightHorizontalLines',
                headerRows: 1,                                          //列タイトル行数
                widths: ['auto','auto','auto','auto','auto','auto'],    //列幅指定
                body: tableBody                                         //表要素
                }
            }
        ],
        // 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);
    pdfDocGenerator.open();
}
```

9-②

9-② makePDF関数
PDFを生成し、開く。

```
        //----- 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','画像']);
        //----- 変数初期化
        var i = 0;                                  // 処理カウンター
        const dataCount = widgetData.length;       // データ件数
        //----- 明細作成とPDF作成処理の実行
        makeDetail(i, dataCount);
    }, function (err){
        //エラー
    });
}, function (err) {
    //エラー
});
```

9-③

9-③ メインプログラム
タイトル行の明細要素を作成した後
ウィジェット行のカウント変数ｉの初期化、
ウィジェットデータのレコード数を取得し、
9-①のmakeDatail関数を呼び出す。

まとめ

■ まとめ

- **Valenceにおける従来の帳票出力について**
  - vvPDFを使用した帳票作成（日本語非対応）
  - 従来の一般的な帳票連携方法
- **Valence6.0における新しいPDF出力**
  - GridウィジェットのPDF出力（日本語対応）
  - pdfmakeがValence環境に追加
- **pdfmakeを使用したPDFファイルの生成方法**
  - pdfmakeを使用したPDF出力
  - 画面上の値やアプリ変数の使用
- **画像やQR/バーコードを含むPDF作成**
  - 画像を含むPDF出力
  - QRコード作成
  - バーコード作成（JsBarcode利用）
- **ウィジェットデータを使用したPDF作成**
  - getWidgetDataを使用した一覧データ出力
  - 画像を含む一覧データ出力

ご清聴ありがとうございました。