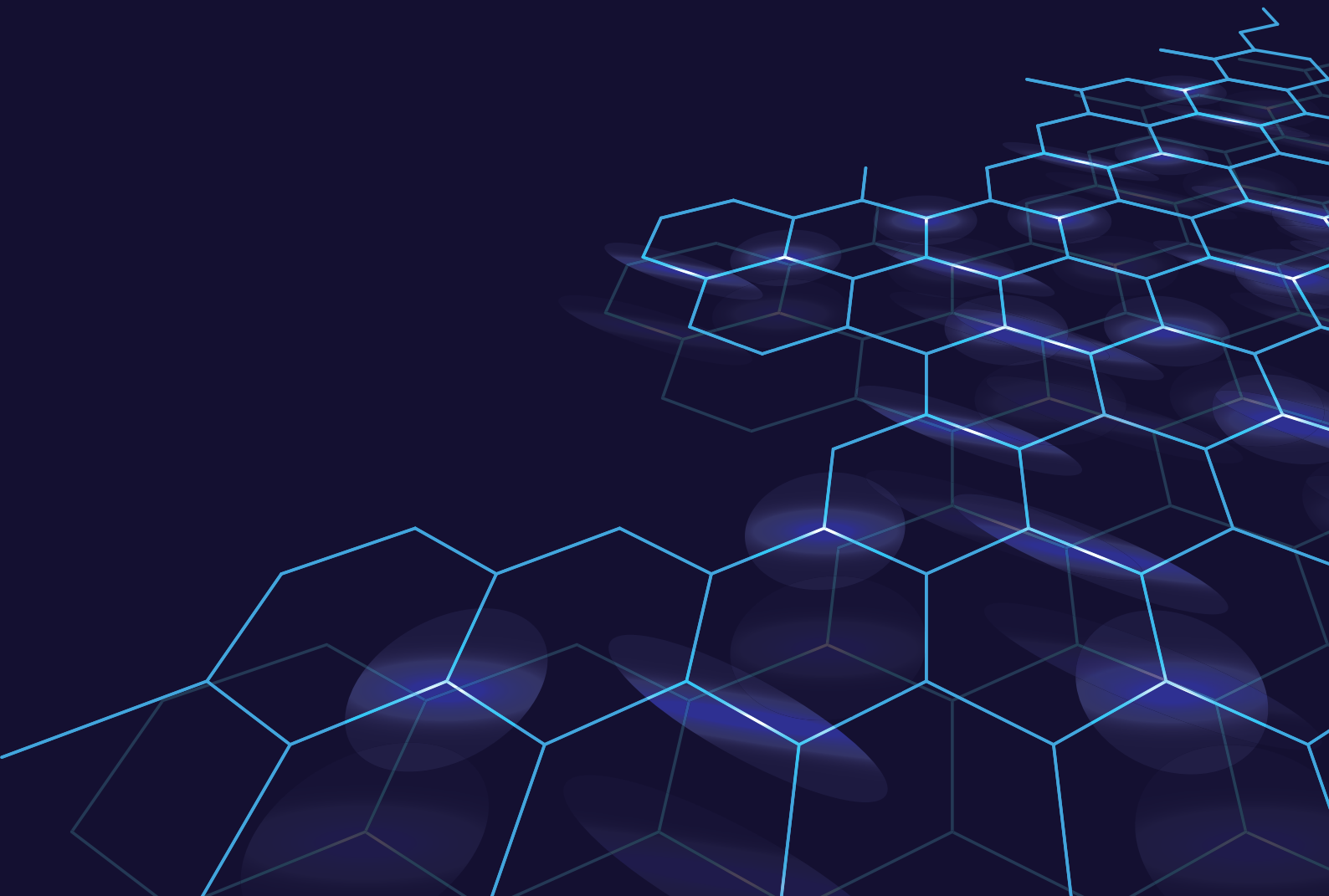
# MIGARO. TECHNICAL REPORT 2021

ミガロ. テクニカルレポート 2021年

No.14

株式会社ミガロ.



# MIGARO.
# Technical Report 2021

ミガロ. テクニカルレポート 2021年

No.14

## CONTENTS

目次

# ごあいさつ

いつもミガロ. 製品をご愛用いただき誠にありがとうございます。

「ミガロ.テクニカルレポート」は、Delphi/400、Valence、SP4iなどのミガロ.製品を使った開発に関する技術情報をお届けする論文集で、このたび第14号を発刊する運びとなりました。

さて、昨年の年頭に発生した新型コロナウイルスは今年も流行を続け、初秋を迎えるまでのほとんどの期間において、緊急事態宣言または蔓延防止等重点措置が発令される事態となりました。こうした中で、今回のテクニカルレポートでは、お客様にご負担をおかけすることのないよう、昨年に引き続き論文募集（ミガロ.テクニカルアワード）を行わず、ミガロ.SE論文だけを掲載することといたしました。ご理解とご了承を賜りますよう、お願い申し上げます。

予測が困難だったコロナ後の社会も、在宅勤務の定着や家庭内の消費活動の充実など、少しずつ変化の方向が明らかになってきました。このような社会変化に対応して、業務計画やシステム投資など経営戦略の見直しに取り組まれる企業も多いかと思います。弊社では、ローコード開発、オープン開発、RPG/COBOL開発の3つの異なるタイプのIBM i開発ツールを揃えることにより、システム戦略の見直しを検討されているIBM iユーザー企業様に、将来のシステム開発・保守に向けた幅広い選択肢をご提案いたします。

今回のSE論文はDelphi/400、SP4i、Valenceのすべての製品をテーマとして含んでいます。Delphi/400の2つの論文では、Delphiの大きな特徴であるコンポーネント開発をテーマとして、FireDACおよびVCLの活用方法を1論文ずつ取り上げました。もう一つのDelphi/400の論文ではIntraWebを使ったWeb化手法をご紹介しています。SP4iの論文は、洗練されたUIデザインを実現するためのHTML作成テクニックをご紹介します。またValenceの論文では、PDF帳票を自由に作成する新しい開発手法を詳しく解説しています。さまざまなテクニックを開発に活用いただくために、各論文は詳細な実装方法までご紹介しています。本レポートが少しでも皆様の開発・保守のお役に立てば幸いです。

最後に、ミガロ.テクニカルレポート第14号を発刊するにあたり、多くのお客様・パートナー様にご支援、ご協力をいただきましたことを、この場をお借りして厚く御礼申し上げます。

2021年12月

株式会社ミガロ.
代表取締役社長
上甲 將隆

# MIGARO.
# Technical Report 2021

ミガロ. テクニカルレポート 2021年

**No.14**

# Delphi/400

# 新規VCLコントロールを利用したユーザーインターフェース改善テクニック

株式会社ミガロ.
システム事業部 システム2課
**畑中　侑**

**略　歴**

生年月日:1983年7月6日
最終学歴:2006年　京都産業大学 法学部卒業
ミガロ入社年月:2006年4月　株式会社ミガロ. 入社
社内経歴:2006年4月　システム事業部配属

**現在の仕事内容:**

システムの受託開発を担当しており、
要件確認から納品・フォロー、
保守作業に至るまで、
システム開発全般に携わっている。



## 1.はじめに

Windows7のサポート終了に伴い、Windows10対応を検討し、既存アプリケーションのバージョンアップを実施する事が非常に多い。アプリケーションをバージョンアップする際にソース変更作業を伴う場合は、バージョンアップ前後でアプリケーションの動作に差異がないかチェックする。Delphi/400アプリケーションのバージョンアップ作業を経験された方は既にご存知のことと思うがWindows10への正式対応版Delphi/400 10Seattle以前のアプリケーションをバージョンアップする際には、ソース変更作業が伴うため新環境での動作が以前の環境と同じ挙動かどうか確認されたことと思う。バージョンアップ作業はシステム担当者から見ると、アプリケーションをシステム基盤の最新OSに対応するという重要なミッションであるが、ユーザー目線からするとどうだろうか。使い慣れたアプリケーションではあるが、以前と同じ見た目、動作ではバージョンアップの恩恵を感じにくいのではないだろうか。

そこで本稿では、既存アプリケーションのコンポーネントを、Delphi/400 10Seattleから追加されている新規VCLコントロールに置き換えることで、ユーザーにバージョンアップの効果を実感してもらえるよう、簡単なUI(ユーザーインターフェース)改善テクニックをご紹介する。

## 2. UI(ユーザーインターフェース)の改善

UIの改善といっても、大きく分けてビジュアル(視覚的)な改善と使い勝手に起因する操作性の改善が考えられる。本稿ではビジュアル的な改善テクニック2つと操作性に関連したレイアウト関する改善テクニックを説明する。

なお、ここではDelphi/400 10.2Tokyoを用いて操作方法をご紹介することとする。

### 2-1:アプリケーションのスタイル変更

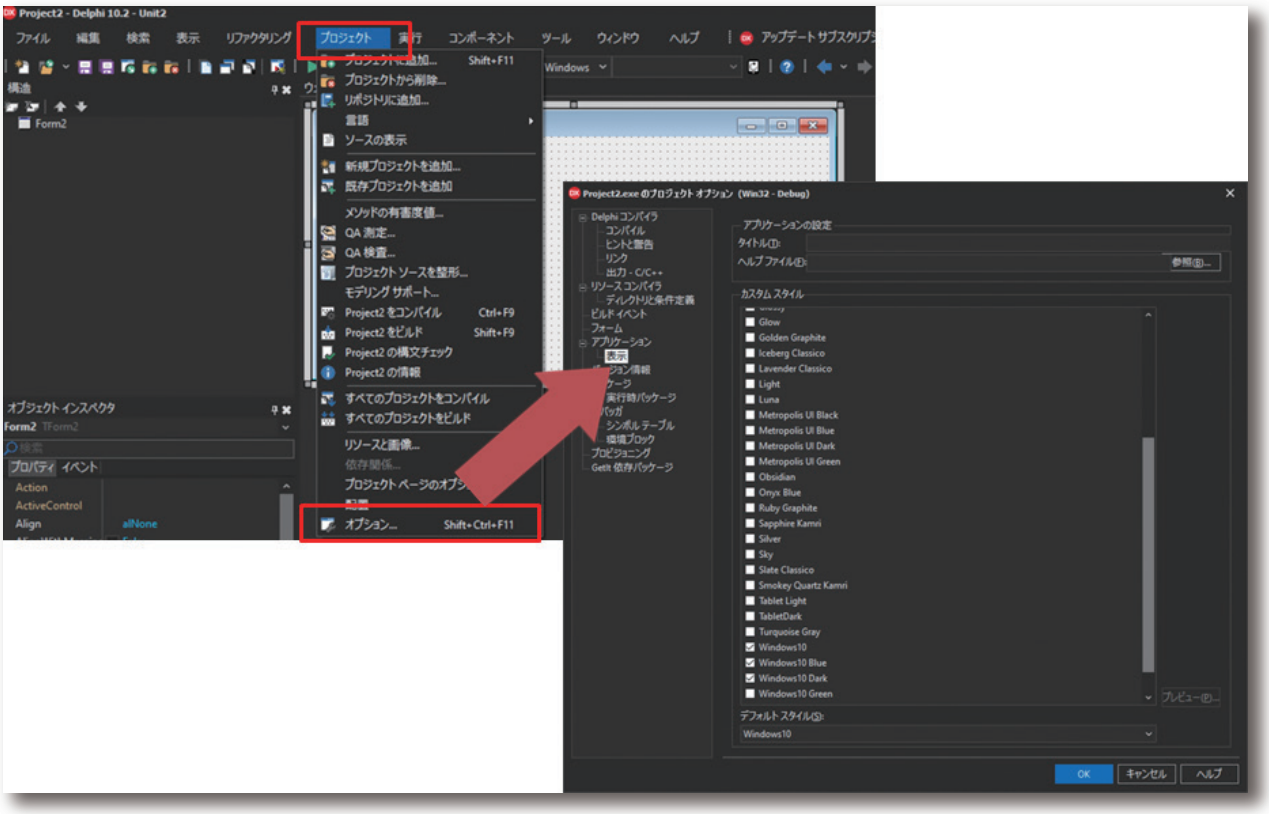
まずは簡単なUI改善の一つとして手軽に行える視覚的な改善方法をご紹介する。

Windows環境で実行されるアプリケーションにはスタイルという概念がある。バージョンアップにて単にアプリケーションをコンバージョンした場合はWindows10環境で実行しても以前と同じ旧スタイルが使用され、ユーザー側からすると、きれいなビジュアル内に古いアプリケーションが起動され違和感を抱く一因になる。Delphi/400 XE3以降のバージョンからアプリケーションに適用されるスタイルを変更することができるため、これを用いることでWindows10ベースのスタイルやその他のスタイルに既存アプリケーションを変更しユーザーのもつイメージを変えることができる。

変更方法は次の手順となる。

開発環境より|プロジェクト|オプション|と順に操作し表示されるプロジェクトオプション画面にて【アプリケーション】配下にある【表示】にある「カスタム スタイル」の該当項目にチェックを入れることで「デフォルトスタイル」の選択肢が増える。これを変更することでアプリケーション全体のスタイルを変更できる。**【図1】**

**図 1　カスタムスタイルの設定**



スタイル変更時の適用イメージは「カスタム スタイル」の該当項目を選択し「プレビュー」ボタンを押下すると確認できる。既存アプリケーションについても、このプロジェクトオプションより「カスタム　スタイル」を変更しコンパイルすると適用でき視覚的なイメージを一新した効果をユーザーに提供することが可能となる。

また、このスタイル変更はTStyleManager.SetStyleメソッドを用いることでソースでの記述も可能である。例えば既存アプリケーションのログイン画面や初期起動画面に、このスタイルをユーザーが選択できるような仕組みを組み込むことで、手軽にユーザーの好みにあったスタイルを提供することができる。実装方法は次の通りである。ここの実装例ではスタイルの選択方法はTComboBoxを用いたリストから行うことを想定している。あらかじめ、スタイルを選択し変更処理を搭載する画面のuses節に「Vcl.Themes」を追加する。これはスタイルを管理するクラスを用いるためである。

**【ソース1】**

**ソース 1**

**uses節への追加**

```
unit Before2Frm;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes,
  Vcl.Graphics,  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,
  Vcl.WinXPanels, Data.DB, Datasnap.DBClient, Vcl.Grids, Vcl.DBGrids, Vcl.Menus,
  Vcl.CategoryButtons, Vcl.Buttons, Vcl.Themes;
```

次にTComboBoxのリストに選択肢を追加しておく。(実装例ではフォーム生成時のイベントOnFormCreateイベントで実装している。)スタイルの候補値はTStyleManager.StyleNamesにアクセスすることで得られるため、TComboBoxのItemsプロパティに加える。また現在適用されているスタイルはTStyleManager.ActiveStyle.Nameで取り出すことができるため、リストの初期選択値として利用する。**【ソース2】**

**ソース 2**

**カスタムスタイルのリスト化**

```
procedure TForm1.FormCreate(Sender: TObject);
var
  StyleName: string;
begin
  // 画面スタイルのリスト作成
  for StyleName in TStyleManager.StyleNames do
    cbxVclStyles.Items.Add(StyleName);

  // 現在スタイルを初期選択とする
  cbxVclStyles.ItemIndex := cbxVclStyles.Items.IndexOf(TStyleManager.ActiveStyle.Name);

end;
```

最後にこのTComboBoxでリストを選択した際に実行されるOnChangeイベントに、選択値を指定したTStyleManager.SetStyleメソッドを実行すれば完了である。**【ソース3】**

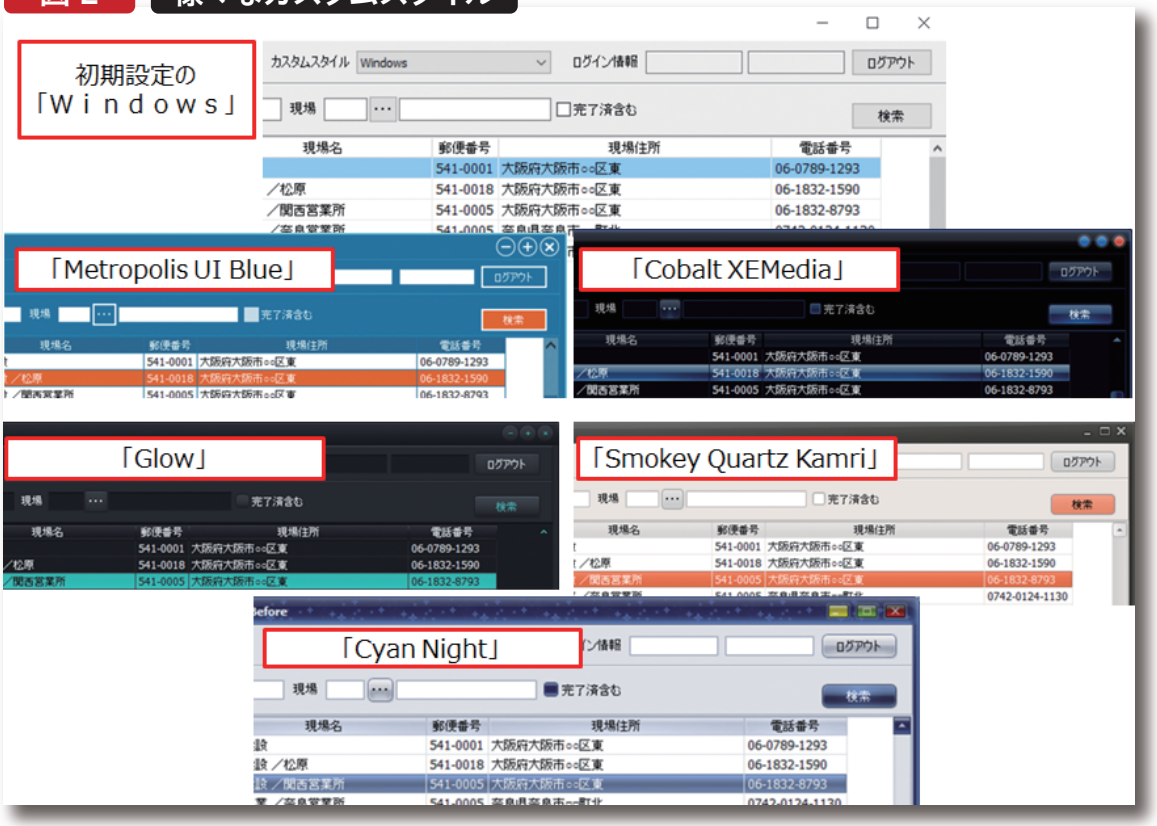
**ソース 3**

**カスタムスタイルの適用**

```
procedure TForm1.cbxVclStylesChange(Sender: TObject);
Begin
  // 選択スタイルの適用
  TStyleManager.SetStyle(cbxVclStyles.Text);
end;
```

上記の実装したものを実行すると、リストにてスタイルを変更すると画面全体がその選択されたスタイルに変更されることが確認できる。**【図2】**

**図 2　様々なカスタムスタイル**




スタイルを変更することで、画面の色味が変わり、それに合わせてコントロールのフォーカス時の反転カラーやタイトルバー上のボタン形状も変わる。例えば、初期設定のカスタムスタイル「Windows」では、表形式の選択行のカラーは青に対して、「MetropolisUIBlue」や「SmokeyQuartzKamri」では橙色となり、ボタンフォーカス時の反転も橙色となる。タイトルバーの「最小化」「最大化」「閉じる」の各機能ボタンの形状も「Cobalt XEMedia」や「Glow」では丸みを帯びた形となっている。「CyanNight」スタイルでは、タイトルバー部にきらめく大小の星が見て取れる。適用できるカスタムスタイルは40種あり、各々特長が違うため好みのスタイルをぜひ探して頂きたい。

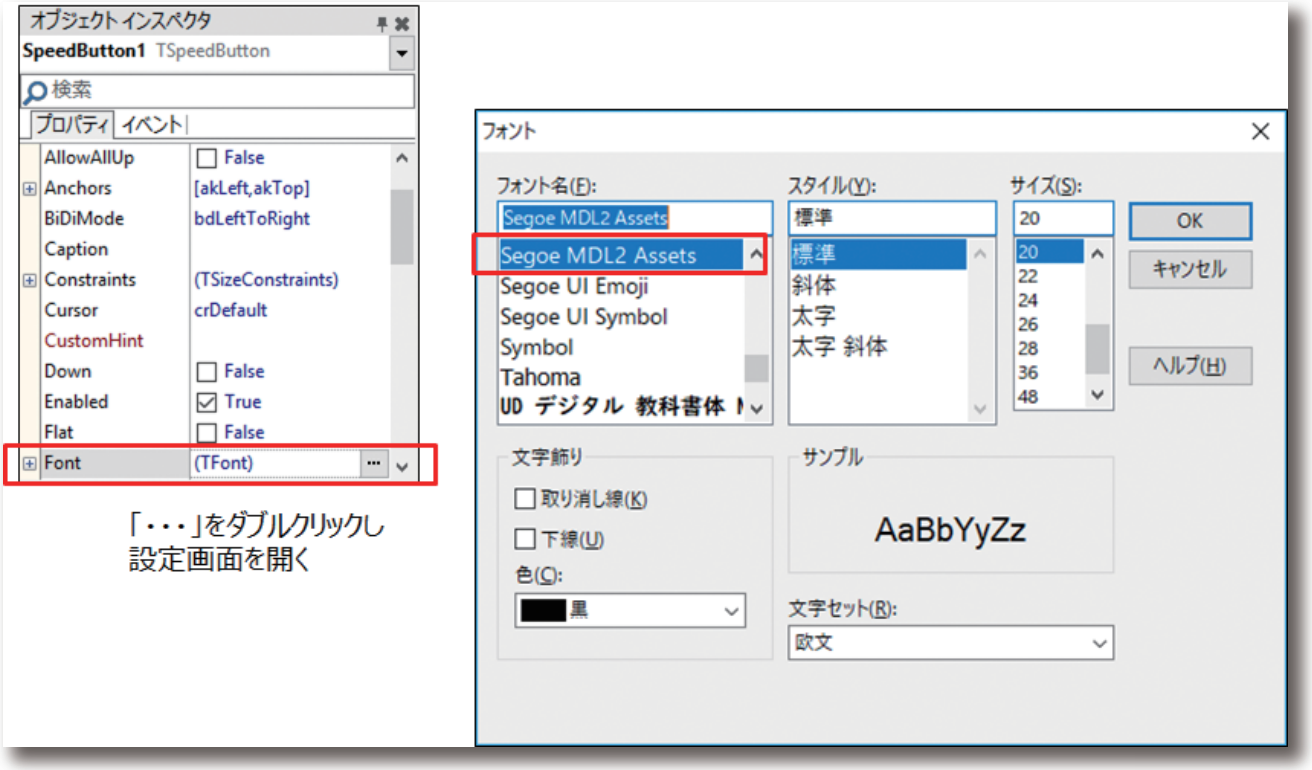
## 2-2：フォントを利用したアイコン変更

「カスタムスタイル」についてはアプリケーション全体の設定となるが、次は各画面に適用できる、手軽で視覚的な改善方法をご紹介する。

Windows10からは「Segoe MDL2 Assets」という記号群のフォントが追加されている。これを用いることで、今までアイコンイメージで補っていたボタンの機能内容を直感的に提供することができたり、単にアイコンイメージ＋機能名の幅に合わせていた項目幅を節約したりできる。具体的な利用方法は、TSpeedButtonのCaptionを例にとって次の手順となる。

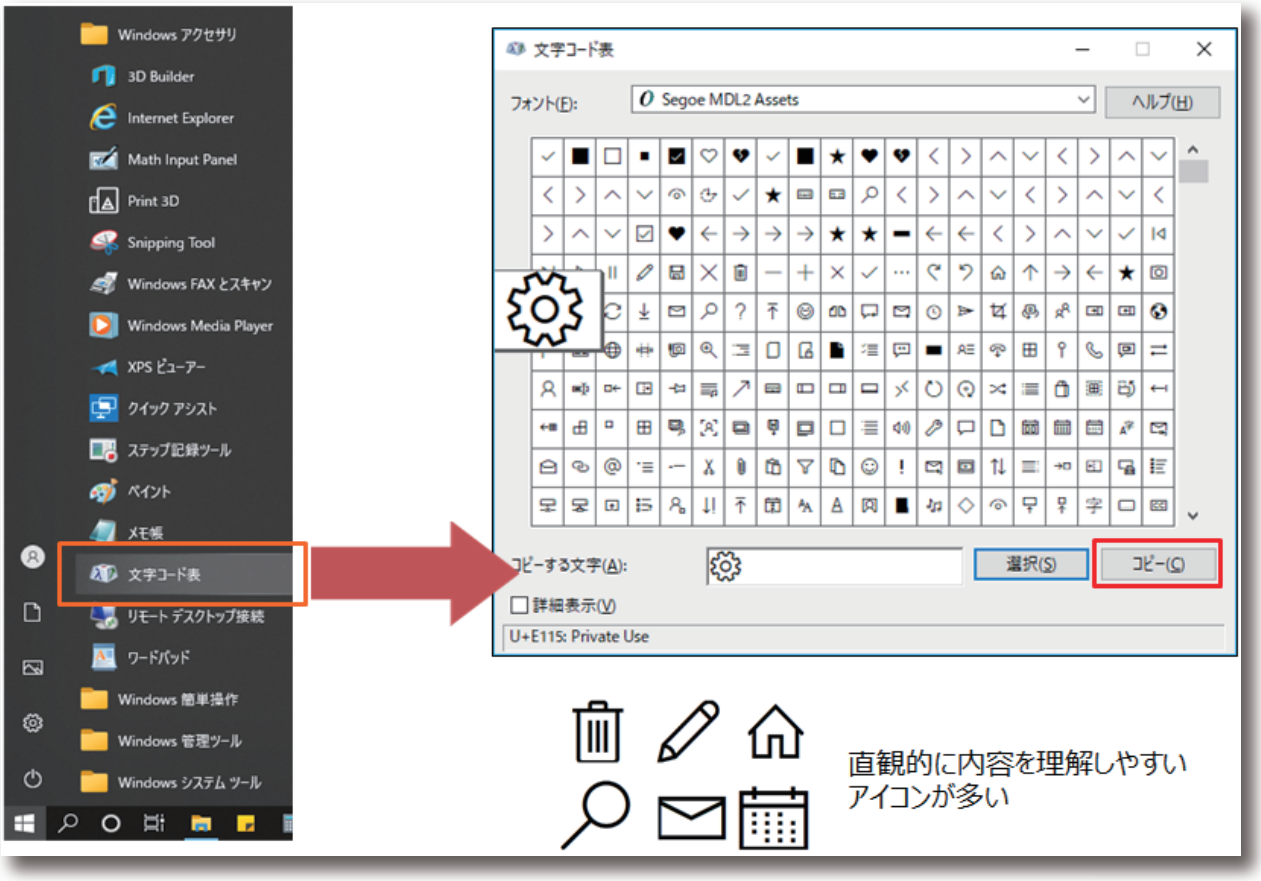
まずはTSpeedButtonのFontプロパティの設定画面にて、「フォント名」に「Segoe MDL2 Assets」を選択する。**【図3】**

**図 3　TSpeed Buttonプロパティの設定**



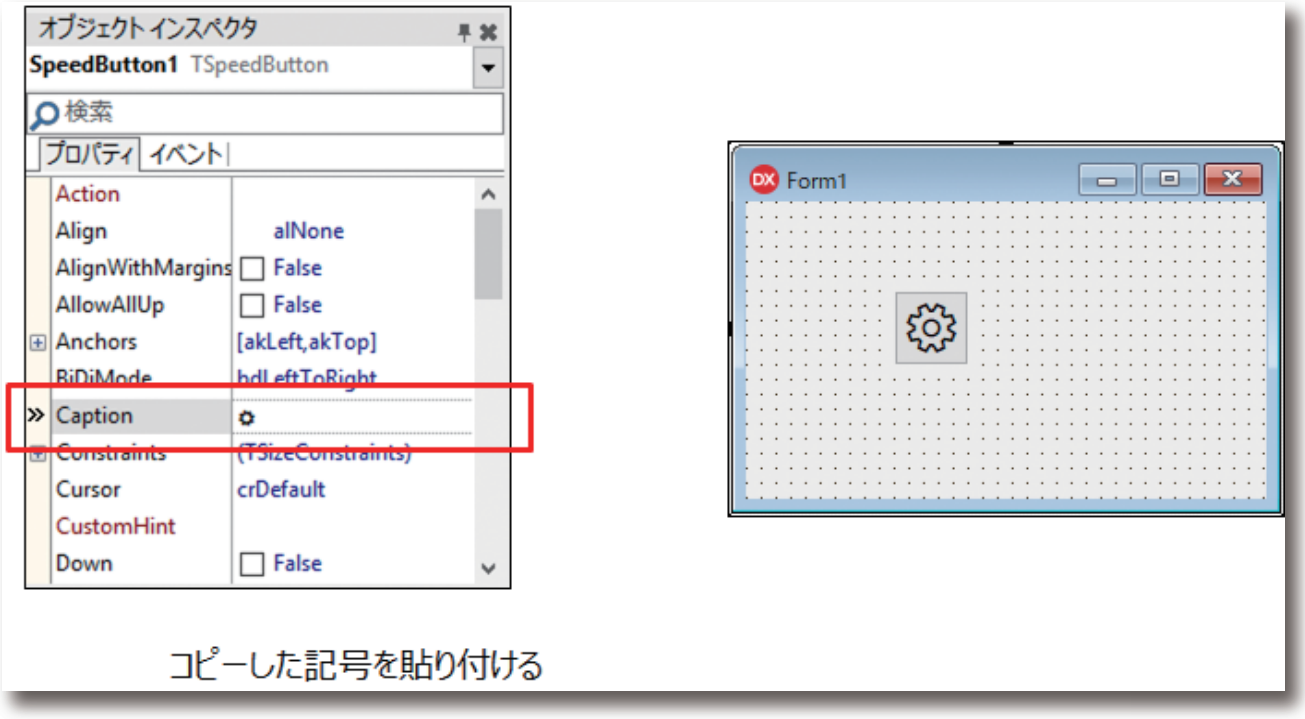
次にWindowsの文字コード表を起動する。方法はWindowsボタンより|Windowsアクセサリ|文字コード表|を選択するとダイアログが表示される。利用したい記号を選択し、ダイアログ上のコピーボタンを押下しクリップボードに保存する。**【図4】**

**図 4　文字コード表の起動**



最後にTSpeedButtonのCaptionプロパティにコピーした記号をペーストする。(記号によっては"・"と表示されるものもある)
以上で操作は完了である。【図5】

**図 5　コピーした記号の貼り付け**



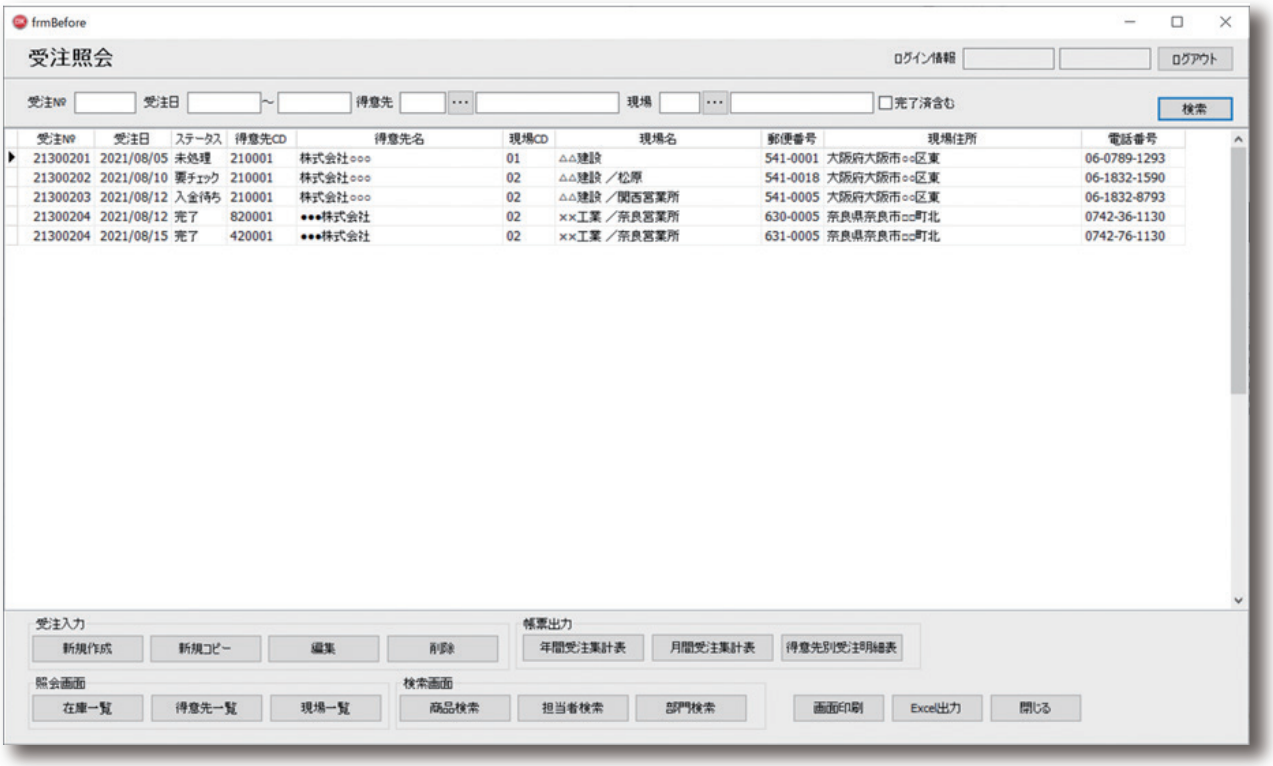

上記の操作で表示される文字コード表を確認すると、多数収録されていることが分かる。従来、削除イメージに用いられてきた「ゴミ箱」アイコンや編集イメージの「鉛筆」アイコン、メイン機能を表す「ホーム」アイコンがいわゆる今風のデザインで用意されている。これらを元に既存アプリケーションで共通的に用いていたアイコンイメージを差し替えたり、画面項目幅によりアイコンイメージの割り当てを避けていた項目に新たに割り当てたりすることでユーザーにはスタイリッシュなイメージを持ってもらえるのではないだろうか。

## 2-3：TSplitViewの活用で省スペース化

最後に操作性に関する改善テクニックをご紹介する。
既存アプリケーションを長年ユーザー要望に応えながら拡張していくにあたって、特定機能の画面が増え、それに伴い画面展開するためのボタン配置も増え画面レイアウトがボタン群で覆われている、またボタンに機能名を表すための表記も、場合によっては多くの文字数を要し項目幅をボタン群で統一するとスペースに難があるといったレイアウトに関する悩みはないだろうか。そのように雑多なボタン群をTSplitViewを使うことでスマートなレイアウトにすることができる。

TSplitViewは開閉可能なコンテナ領域を提供でき、普段は閉じておきスペースを省略、必要に応じて開いてそのスペースを活用できるようにする。また今回、TStackPanelを用いてボタン群の管理を容易にできる工夫を行うことにする。TStackPanelは直接レイアウトに作用するものではないが、複数のコントロールを配下に管理でき設計時の配置変更に役立つ。具体的な利用方法を説明するにあたり、変更前の画面を紹介する。**【図6】**

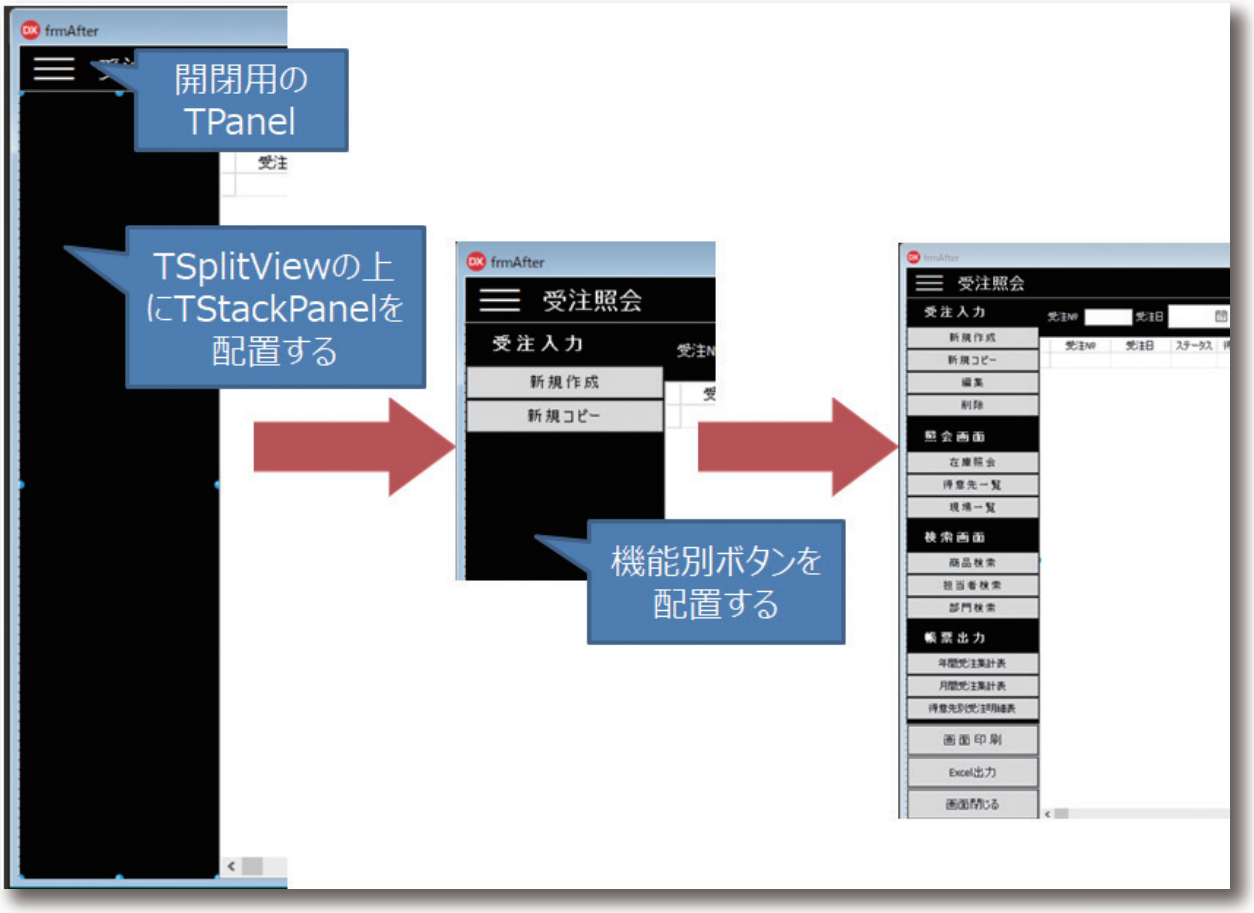
**図 6　変更前の画面イメージ**



変更前の画面は明細形式の照会機能となり、上部に検索条件、中央部には照会データエリア、下部にデータ編集指示や帳票出力、その他機能画面への遷移を行うボタン群を配置した画面構成となっている。このボタン群をTSplitViewを用いて開閉可能なサブメニューとする。

まず、開閉を指示するために画面上部にTPanelを配置する。例では背景色を持たせるためにTPanelを用いたがボタンでもよい。Captionには「2-2」でご紹介した文字コード表より三本ラインを指定している。次に該当画面にTSplitViewを配置する。今回は画面左側に開閉スペースを設けるため、Placementプロパティを「svpLeft」に設定する。**【図7】**

**図 7　コントロールの配置**



TSplitViewの開閉はOpenedプロパティのTrue(開く)/False(閉じる)で操作するため、先ほど配置したTPanelのOnClickイベントにOpenedプロパティの切り替えを実装する。**【ソース4】**

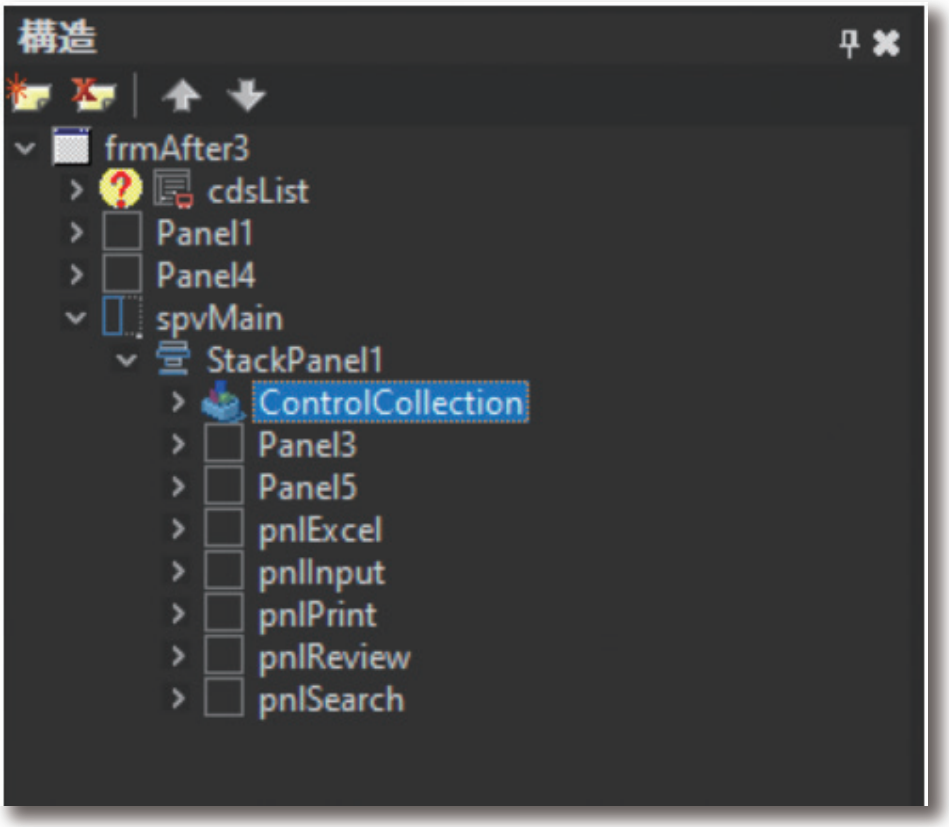
**ソース 4**

**TSplitViewの開閉**

```
procedure TForm1.pnlMenuClick(Sender: TObject);
begin
  // 横からスライド表示するアニメーションの適用(True:適用する False:適用しない)
  spvMain.UseAnimation := False;    // ちらつき防止のため本サンプルでは適用しない

  if spvMain.Opened then
    spvMain.Opened := False
  else
    spvMain.Opened := True;
end;
```

配置したTSplitView上にTStackPanelを配置し、Alignプロパティを「alClient」に設定する。さらに配置したTStackPanel上に機能カテゴリを表すパネルを配置、その下にボタン群を配置する。この際、各Alignプロパティを「alTop」として上詰めで配置していく。TStackPanel上にパネル、ボタンを配置すると、開発画面の「構造」エリアのTStackPanel配下にあるControlCollectionのさらに配下にTStackPanelControlItemが増えていくのが分かる。**【図8】**

**図 8　TStack Panel Control Item**



TStackPanelControlItemのControlプロパティを確認すると、配置したパネルやボタンが紐づいていることが分かる。この「構造」エリアのTStackPanelControlItemをドラッグアンドドロップし上位（または下位）に移動することで画面上の配置も合わせて入れ替わることができるため、今後、例えばボタンの配置を変更したい際にはメンテナンスが楽になる。最後にTSplitViewの下記プロパティを好みに応じて設定する。

・DisplayMode：TSplitViewのOpenedプロパティのTrue（開く）設定時のモーション指定

-svmDocked：隣接コントロールも同時に右に移動する

-svmOverlay：隣接コントロールに被さるようにTSplitViewが表示される

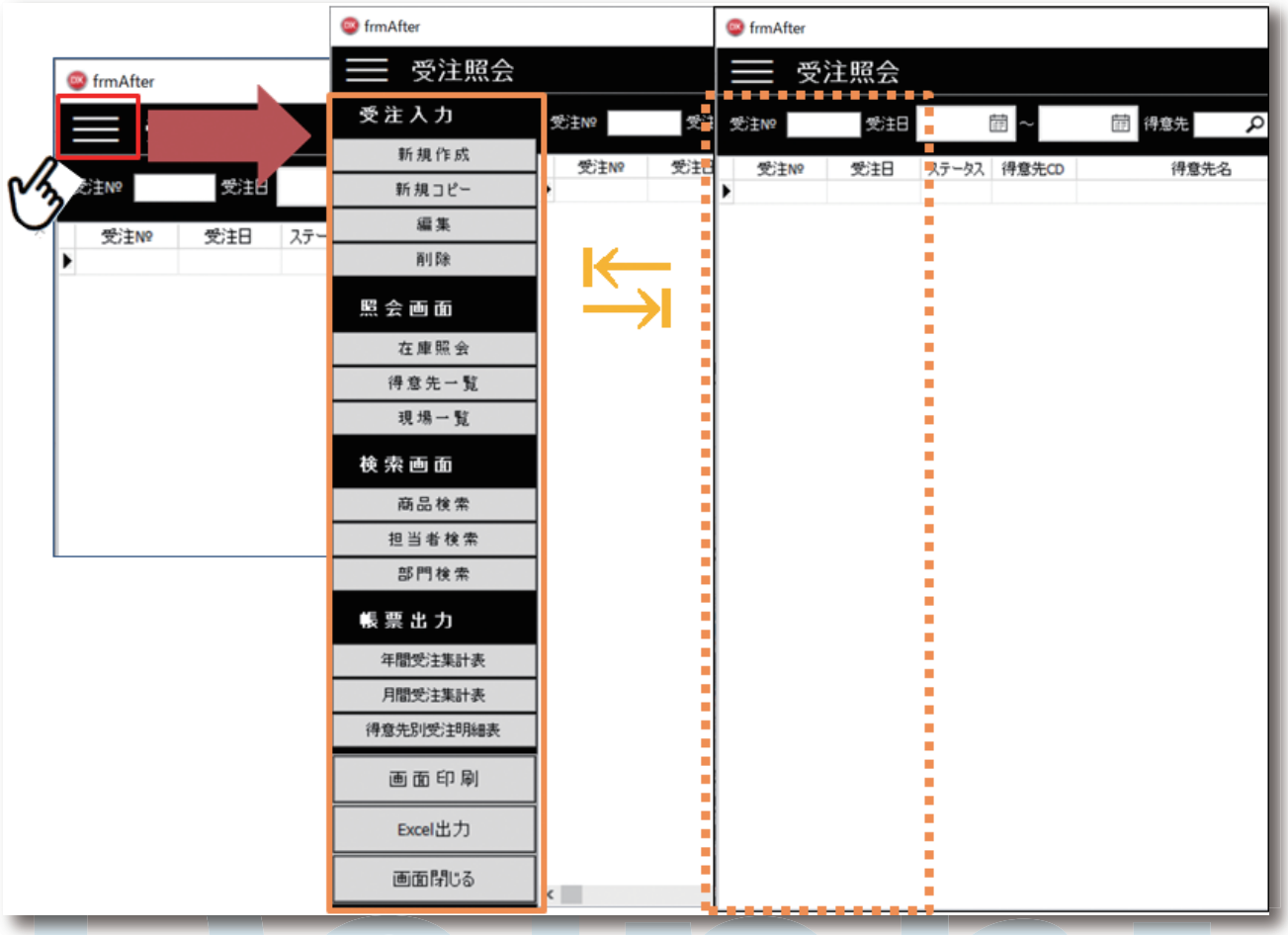
・CloseStyle：TSplitViewのOpenedプロパティのFalse（閉じる）設定時のスタイル指定。

-svcCollapse：完全に閉じた状態となる（非表示）

-svcCompact：CompactWidthの設定値に応じたスペースが表示される。以上の実装でアプリケーションを実行すると、開閉パネルをクリックするたびに画面左のボタンエリアが表示/非表示されることが確認できる。これで画面レイアウトの下部を占めていたボタン群を、任意で開閉できるエリアに移動しメイン部のスペースを広げることができた。

**【図9】**

**図 9　TSplit Viewの開閉イメージ**

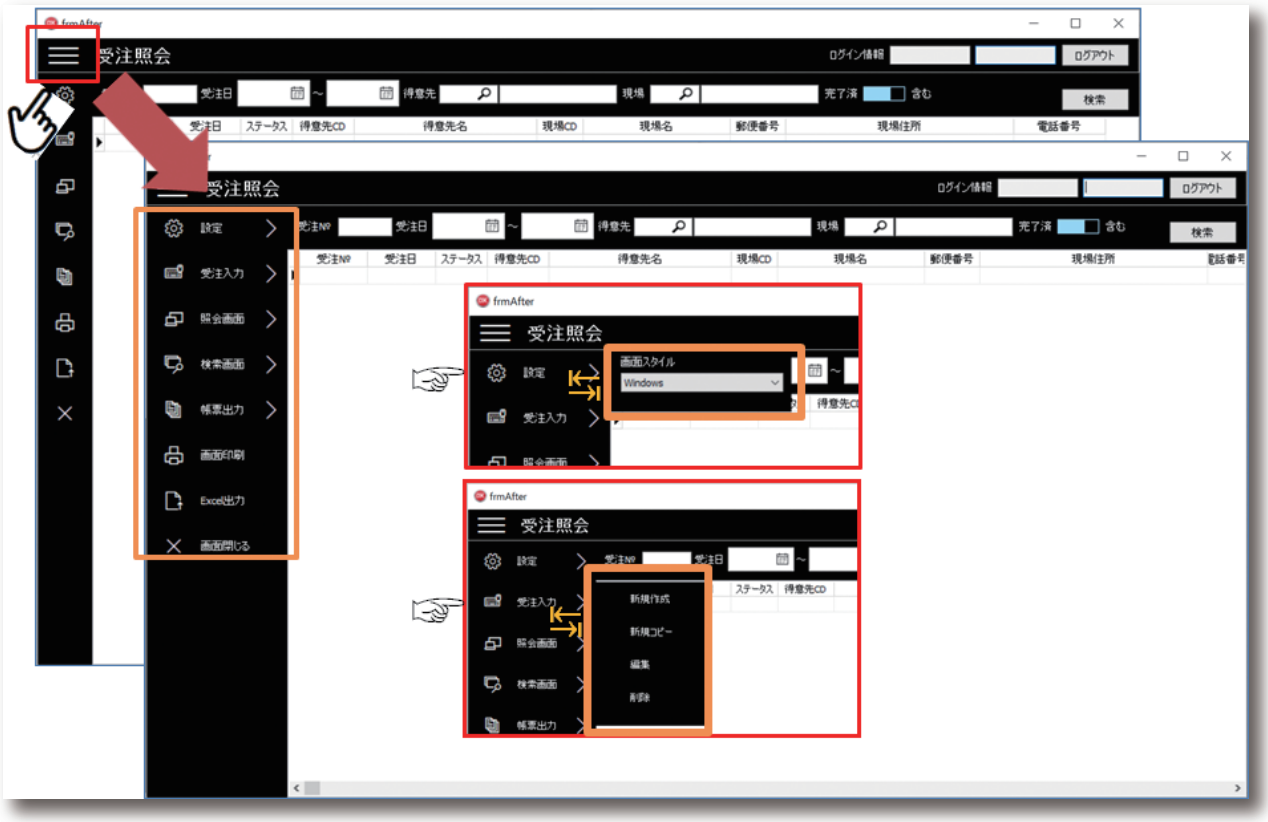


## 3.さいごに

本稿では、既存アプリケーションの一部のコントロールを、Delphi/400に新しく追加されたコントロールへ置き換えることでユーザーにバージョンアップの効果を実感してもらえる簡単なUI改善テクニックをご紹介した。

一つの参考として図「変更前の画面イメージ」を用いた、適用後のイメージを見て頂きたい。**【図10】**

**図 10　UI改善テクニック適用例**




画面左側にカスタムスタイルの変更を指示するための「設定」をはじめ、「受注入力」や「帳票出力」のような画面遷移や出力用の指示項目をカテゴリごとに開閉可能なTSplitViewに格納している。「設定」や「受注入力」については、「Segoe MDL2 Assets」フォントを用いた記号を配し、クリックするとさらに選択項目部を展開するようにしている。また指示項目についてはColorプロパティが適用可能なTPanelを用い画面上部と同じ色味にすることで、ビジュアルを画面内で統一している。

もしバージョンアップを単にコンバージョンで対応されたままの場合は、このテクニックを用い、UIの面でもバージョンアップの効果をユーザーに伝える役に立てて頂ければ幸いである。

# Delphi/400

# IntraWebを使用したWeb開発のTips紹介

株式会社ミガロ.
システム事業部
システム1課
**福井　和彦**

**略　歴**

生年月日:1972年3月20日
最終学歴:1994年　大阪電気通信大学 工学部卒業
ミガロ入社年月:2001年04月　株式会社ミガロ.入社
社内経歴:2001年04月　システム事業部配属

**現在の仕事内容:**

主にDelphi/400を使用したシステムの受託開発を担当しており、
要件確認から納品・フォローに至るまで、システム開発全般に
携わっている。

株式会社ミガロ.
システム事業部
システム1課
**石山　智也**

**略　歴**

生年月日:1988年4月5日
最終学歴:2012年　近畿大学 経済学部卒業
ミガロ入社年月:2019年6月　株式会社ミガロ.入社
社内経歴:2019年6月　システム事業部配属

**現在の仕事内容:**

主にDelphi/400を利用したシステムの受託開発をメインに担当している。
開発スキルの向上を目指し、日々精進している。



## 1.はじめに

近年、テレワークの増加や業務効率化の観点から業務システムのWeb化は注目の存在となっている。
Webシステムはクライアントサーバーシステムとは違い、クライアントPCにアプリケーションをインストールや再配布を行う手間が要らず、ブラウザとインターネット環境があればどこからでもシステムを使用できることが利点である。
Delphi/400で Web サーバー アプリケーションを構築するツールとしてIntraWebがある。IntraWebを使用すると、従来のGUIアプリケーションと同様にWebサーバーアプリケーションを構築することができる。本稿ではIntraWeb 15を使用したWebアプリケーション開発のTipsを紹介する。また実行結果は、MicrosoftのEdgeを使用して確認する。

## 2.IntraWebでの新規プロジェクトの作成

### 2-1.IntraWeb 15のインストール

本題に入る前に、IntraWeb 15を使用できる開発環境を整える必要がある。Delphi/400 10.2Tokyoを標準インストールすると、IntraWebは古いバージョンのバンドル版がインストールされている。IntraWeb 15を使用するには、古いバンドル版をアンインストールした後、IntraWeb 15をインストールする必要がある。その手順を簡単にまとめておく。

<手順>

① IntraWeb 15を次のサイトより取得する。
https://www.atozed.com/intraweb/download/v15/

② バンドル版をアンインストールするツールを次のサイトより取得する。
https://www.atozed.com/intraweb/bundled/removal-tool/

③ ②で取得したツール「IWBundledRemovalTool.exe」を実行してバンドル版をアンインストールする。この時「Delete IntraWeb package･･･」のチェックボックスはチェックしておく。

④ ①で取得したIntraWeb 15をインストールする。

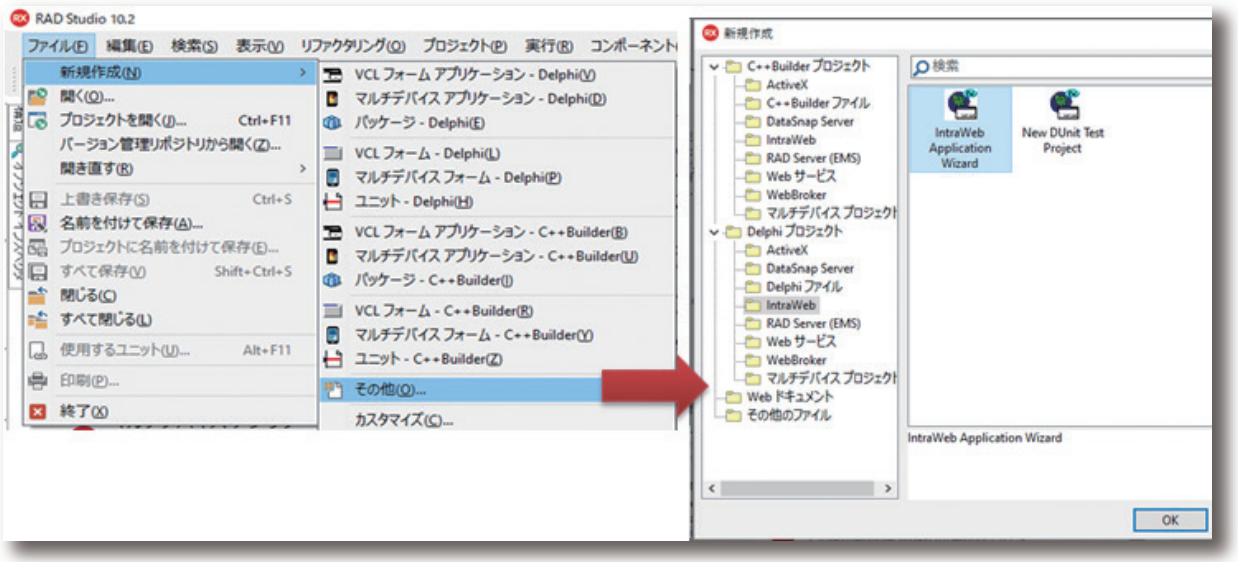
IntraWeb 15のライセンスキーを持っていない場合でも、評価版として使用することができる。ライセンスキーが必要な方は、ミガロ.営業担当まで問合せをしてほしい。

### 2-2.新規プロジェクトの作成

過去のレポートでもIntraWebでの新規プロジェクトの作成方法については掲載しているが、本稿では執筆時点での最新バージョンであるIntraWeb 15を例に解説するため改めて記載する。

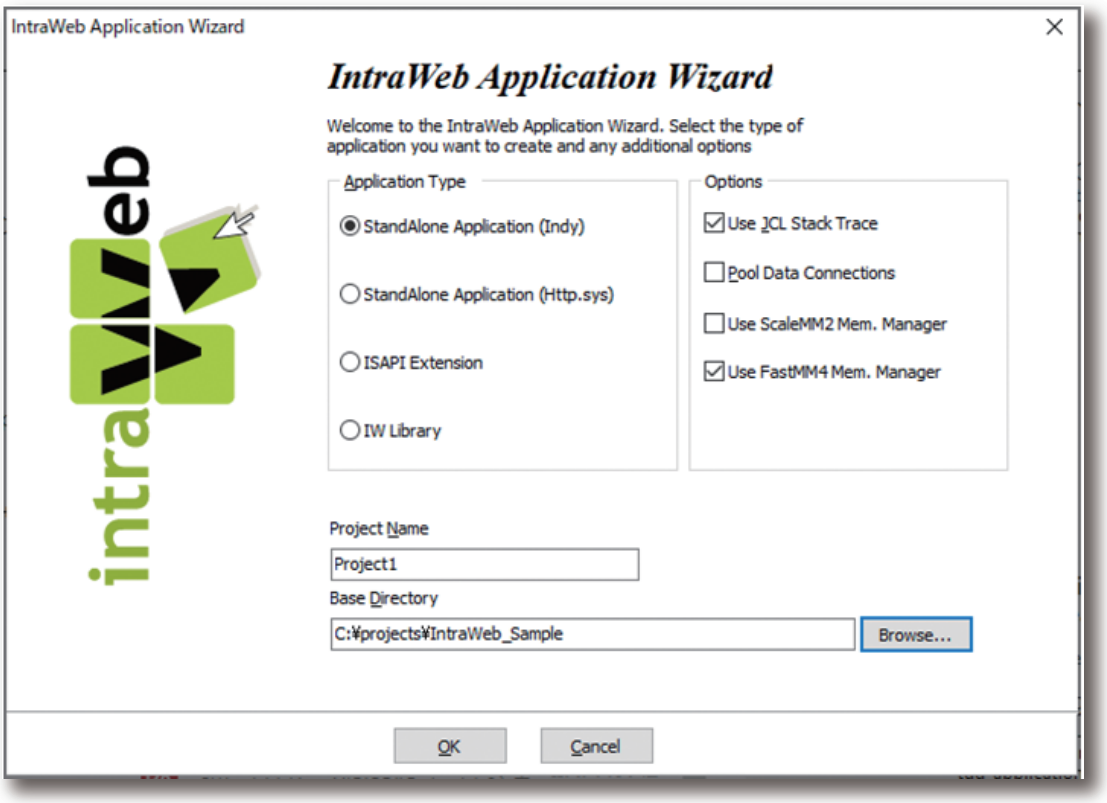
新規プロジェクトの作成は「ファイル」→「新規作成」→「その他」の順に選択し、表示された新規作成画面で「Delphi プロジェクト」→「IntraWeb」→「IntraWeb Application Wizard」をクリックする。**【図1】**

**図 1　IntraWeb新規プロジェクトの作成**



表示された「IntraWeb Application Wizard」でWebアプリケーションの種類、プロジェクト名、ベースディレクトリを設定する。**【図2】**

**図 2　IntraWeb Application Wizard**



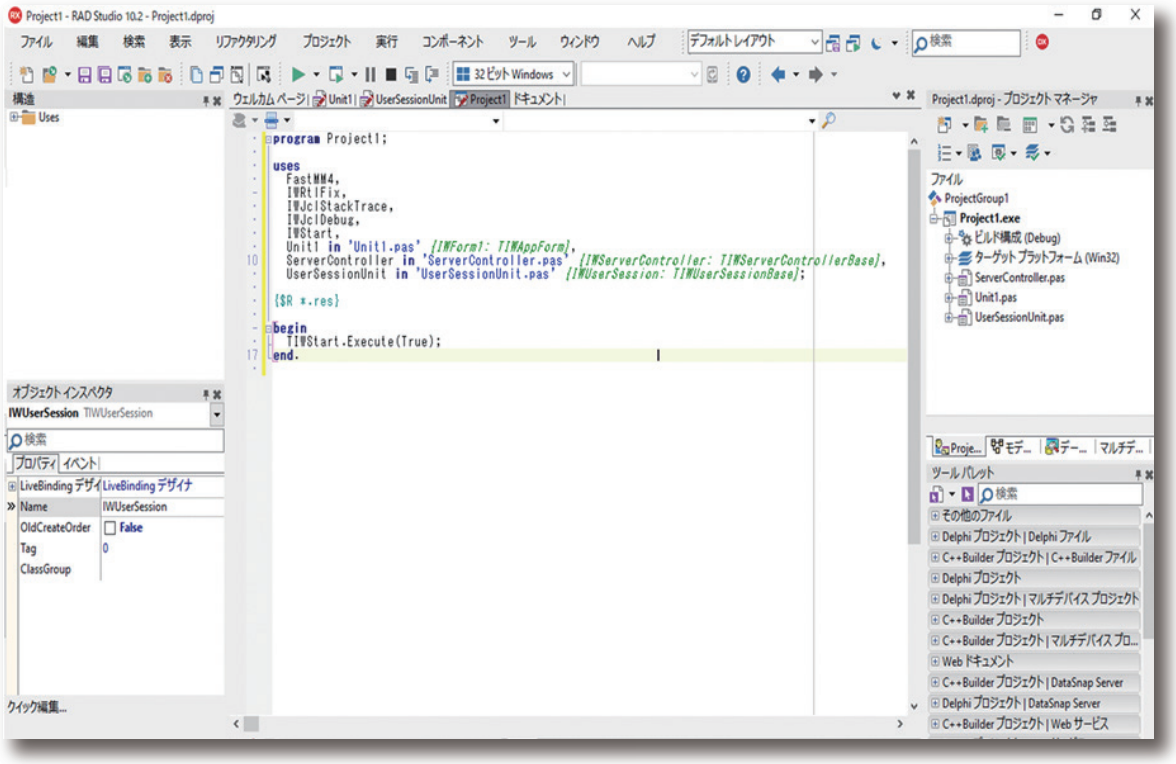

**【図2】**にあるWebアプリケーションの種類とオプションについて簡単に記載する。

<Webアプリケーションの種類>
① StandAlone Application（Indy）、
StandAlone Application（Http.sys）（スタンドアローン）
WebサーバーもIntraWebが提供するモード。
② ISAPI Extension（アプリケーション）
WebサーバーはIISを用いて提供するモード。
③ IW Library（ASPX配置）
②と同様にWebサーバーはIISを用いるが、ASP.NETアプリケーションとして配置可能なモード。

<オプションの種類>
① Use JCL Stack Trace
JCL（JEDI Code Library）を使用してスタックトレースが可能になる。
② Pool Data Connections
データベースへアクセスの都度、接続（コネクション）を確立するのではなく、あらかじめ事前に一定数のコネクションを確立しておき、それを使い回すことが出来るようになる。
③ Use ScaleMM2 Mem.Manager、
Use FastMM4 Mem.Manager
メモリリークを検出するためにScaleMM2、FastMM4を使用することが可能になる。

【図2】の状態でOKボタンを押下すると新規プロジェクトが作成される。【図3】
開発画面には、ServerController、Unit、UserSessionUnitの3つが自動で生成される。
ServerControllerは、サーバー側の処理を扱うファイルで、例えばブラウザの戻るボタン押下時やアプリケーション終了時の制御を行うといったことが出来る。
Unitは実際に開発を行う画面で、このファイルに対してコンポーネントを貼り付けて開発を行う。
UserSessionUnitはクライアントサーバー開発を行う時のDataModuleと同じ扱いになる。
次の章では明細形式のWeb画面を作成するテクニックを紹介する。

**図 3　IntraWeb 新規プロジェクト**



## 3.明細形式のWeb画面を作成するテクニック

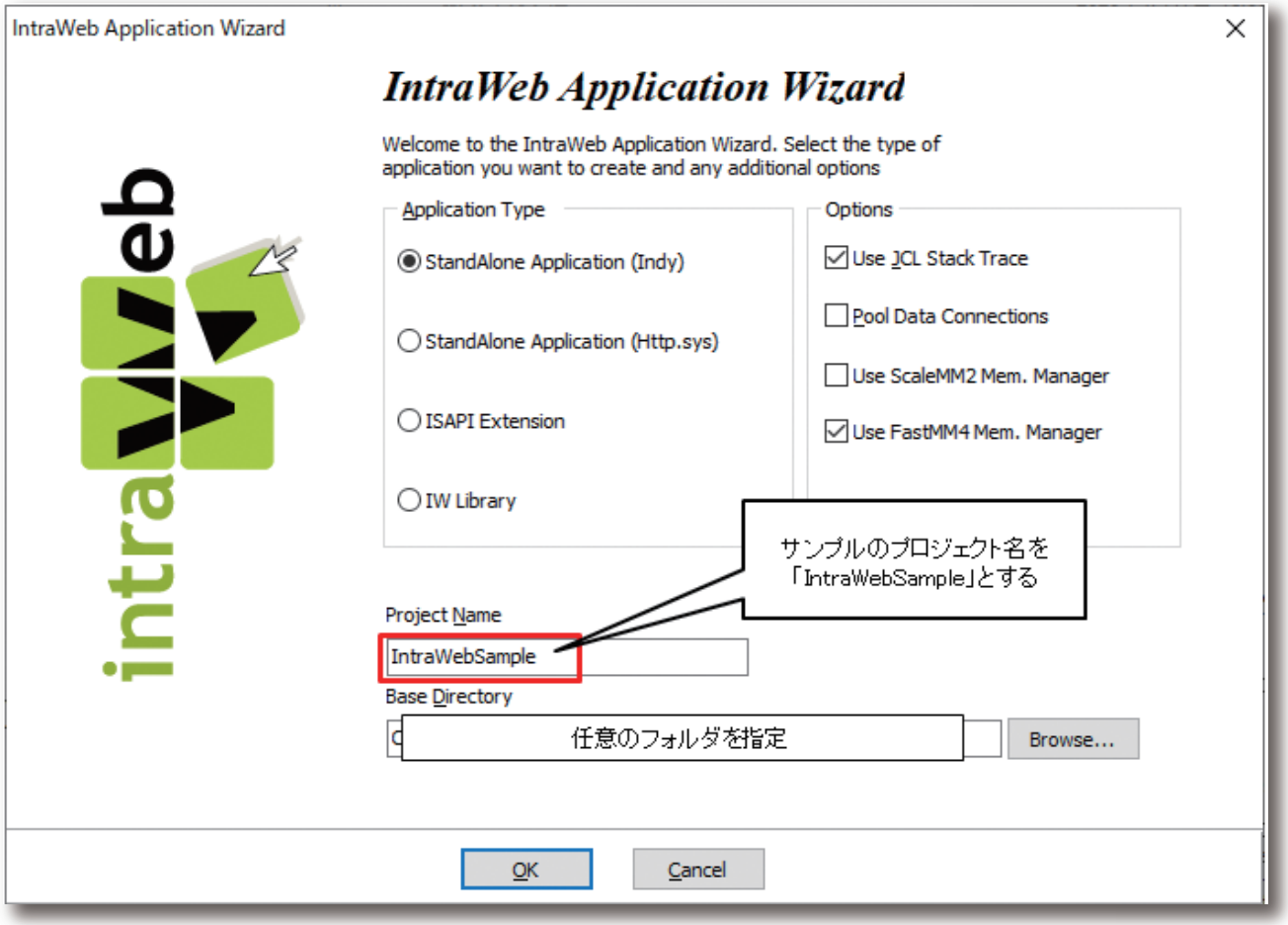
この章では明細形式のWeb画面でよく見られる「件数が一定数を超えた際に頁を分ける機能」や、「明細のタイトルをクリックしたら並び替えが行える機能」の実装方法について、次の順番に従って説明を行う。

① 「TIWGrid」を使用した明細表示
② 明細表示のカスタマイズテクニック
- 件数が一定数を超えた際に頁を分ける機能を実装
- 明細のタイトルをクリックしたら並び替えが行える機能を実装

### 3-1.「TIWGrid」を使用した明細表示

まず始めに本章以降で使用するプロジェクトを、プロジェクト名「IntraWebSample」として新規作成する。**【図4】**

**図 4　本稿で使用する新規プロジェクトの作成**

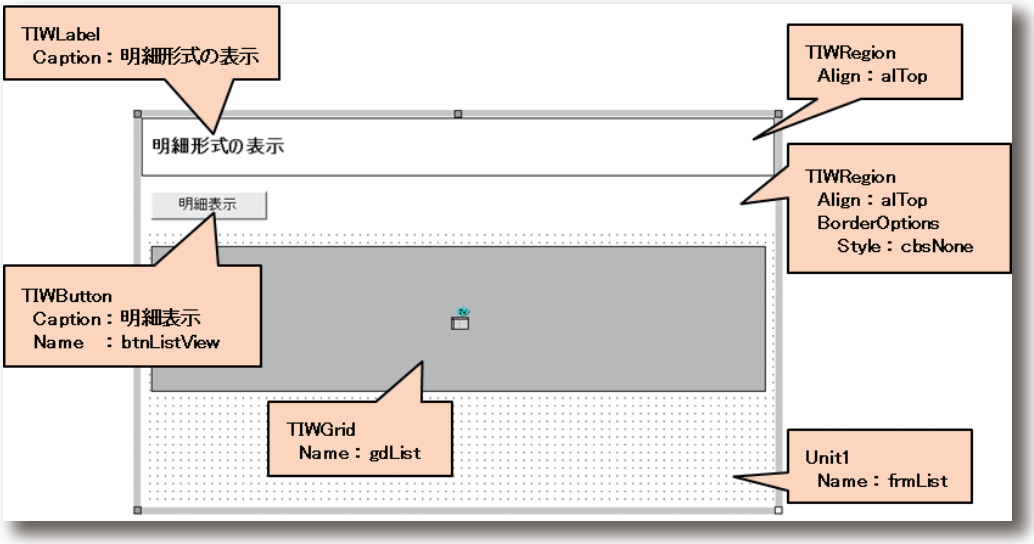


作成されたプロジェクトのUnit1を開き、表示されたフォームのNameプロパティに「frmList」と設定する。そして、【図5】に従ってコンポーネントを配置し各プロパティを設定する。本稿では明細表示のコンポーネントとしてTIWGridを使用する。配置と設定が終わったら、Unit1をファイル名「ListFrm.pas」として名前を付けて保存を行う。

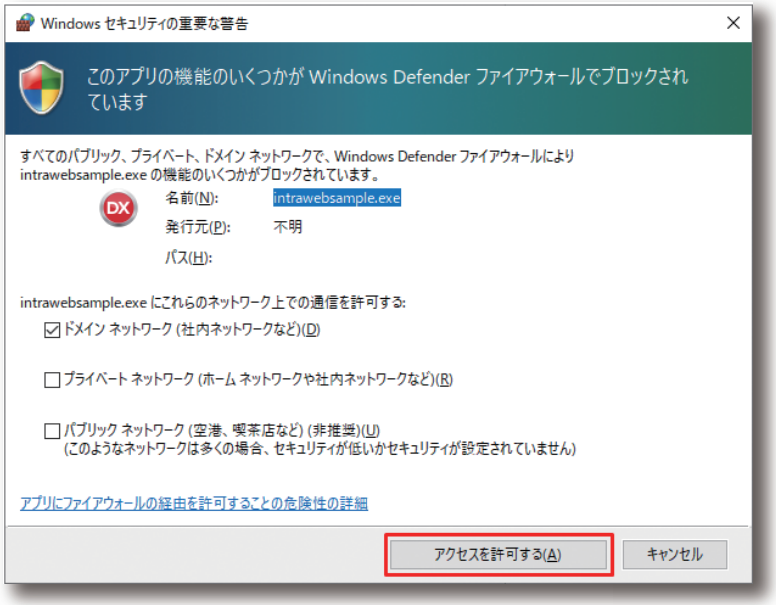
それでは、実行してブラウザにどのように表示されるか確認をしてみよう。実行方法は、メニューバーより「実行」→「実行」と選択するか、キーボードの「F9」を押下する。初回実行時、Windowsの警告が出る場合がある。

【図6】 警告が出たら「アクセスを許可する」をクリックして先に進める。

## 図 5 「明細形式の表示」の画面設計

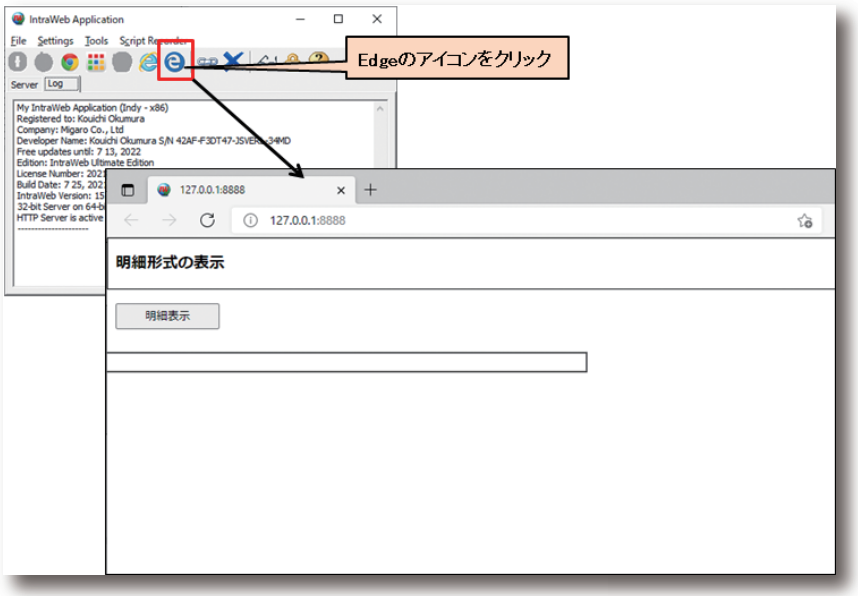



## 図 6 Windowsの警告画面



実行すると始めに「IntraWeb Application」の画面が開く。画面には各ブラウザのアイコンが並んでいる。その中からEdgeのアイコンをクリックする。そうするとEdgeが開きWeb画面が表示される。**【図7】**

**図 7　実行結果イメージ**

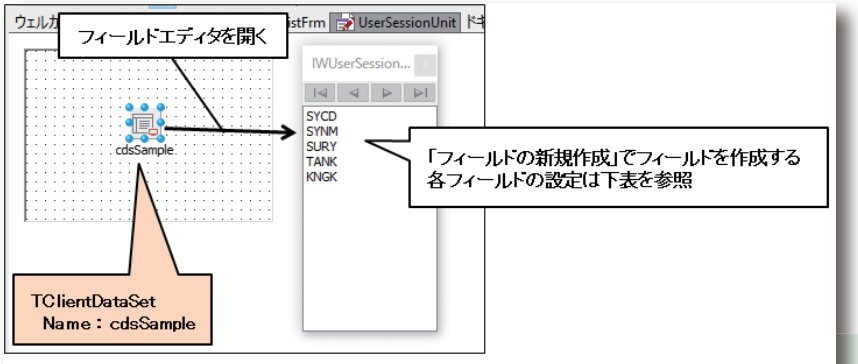



この時点では明細にはまだ何も表示はされない。実行結果が確認できたら、Edgeの「×」ボタンをクリックしてブラウザを閉じ、IntraWeb Application画面も同様に「×」ボタンをクリックして終了し開発画面へ戻る。

続いて、明細に表示するデータの準備を行う。データはプログラムでTClientDatasetへ作成して保持する。

UserSessionUnitを開き、画面にTClientDatasetを配置する。Nameプロパティは「cdsSample」と設定する。続いて**【図8】**に従ってTClientDatasetの各設定を行う。

**図 8　TClientDatasetの設定**




| FieldName | Name | DisplayLabel | データ型 | Size | FieldKind |
|---|---|---|---|---|---|
| SYCD | cdsSampleSYCD | 商品コード | String | 6 | fkData |
| SYNM | cdsSampleSYNM | 商品名 | String | 42 | fkData |
| SURY | cdsSampleSURY | 数量 | Integer | | fkData |
| TANK | cdsSampleTANK | 単価 | Integer | | fkData |
| KNGK | cdsSampleKNGK | 金額 | Currency | | fkData |

次に、UserSessionUnitのOnCreateイベントでcdsSampleのCreateDataSetメソッドを呼び出し、メモリ上にデータセットを作成する。**【ソース1】**

## ソース 1

**UserSessionUnitのOnCreateイベント**

```
procedure TIWUserSession.IWUserSessionBaseCreate(Sender: TObject);
begin
  // データセットを作成
  cdsSample.CreateDataSet;
end;
```

そして、データ作成の処理としてUserSessionUnitのPublicのプロシージャ「SampleDataCreate」を作成する。実装部は**【ソース2】**に従って記述する。この処理で100件のデータを作成する。UserSessionUnitの開発は以上となる。

## ソース 2

**UserSessionUnitのSampleDataCreateプロシージャ**

```
<宣言部>
 public
    { Public declarations }
    procedure SampleDataCreate;  // サンプルデータ作成
  end;
<実装部>
procedure TIWUserSession.SampleDataCreate;
var
  i: Integer;
begin
  // cdsSampleをクリア
  cdsSample.EmptyDataSet;

  for i := 1 to 100 do
  begin
    cdsSample.Append;
    cdsSampleSYCD.AsString   := FormatFloat('000000', i * 10);
    cdsSampleSYNM.AsString   := 'サンプル商品（' + FormatFloat('000000', i * 10) + '）';
    cdsSampleSURY.AsInteger  := i;
    cdsSampleTANK.AsInteger  := i * 1000;
    cdsSampleKNGK.AsCurrency := i * (i * 1000);
    cdsSample.Post;
  end;
end;
```

ここからは、frmListの明細へcdsSampleのデータを表示する処理を実装していく。frmListを開き、メニューバーより「ファイル」→「使用するユニット」を選び、ServerControllerを選択する。次にfrmListのPrivateプロシージャとして「GridClear」「GridLayout」「ListDataSet」の3つを宣言する。**【ソース3】**各プロシージャの実装部は**【ソース4～7】**に従って記述する。

## ソース 3

**frmListのPrivateプロシージャ宣言**

```
private
   { Private 宣言 }
   procedure GridClear;        // 明細初期化
   procedure GridLayout;       // 明細の項目幅調整
   procedure ListDataSet;      // 明細データセット
```

## ソース 4

**frmListのGridClearプロシージャ**

```
procedure TfrmList.GridClear;
begin
  with gdList do
  begin
    Clear;

    // 行数初期化
    RowCount := 1;

    // タイトル設定
    Cell[0, 0].Text := '商品コード';
    Cell[0, 1].Text := '商品名';
    Cell[0, 2].Text := '数量';
    Cell[0, 3].Text := '単価';
    Cell[0, 4].Text := '金額';

    // 列表題Alignment
    Cell[0, 0].Alignment := taCenter;
    Cell[0, 1].Alignment := taCenter;
    Cell[0, 2].Alignment := taCenter;
    Cell[0, 3].Alignment := taCenter;
    Cell[0, 4].Alignment := taCenter;
  end;
end;
```

## ソース 5

**frmListのGridLayoutプロシージャ**

```
procedure TfrmList.GridLayout;
begin
  with gdList do
  begin
    // 列幅の初期設定
    Cell[0, 0].Width := '100';  // [00] 商品コード
    Cell[0, 1].Width := '300';  // [01] 商品名
    Cell[0, 2].Width := '100';  // [02] 数量
    Cell[0, 3].Width := '170';  // [03] 単価
    Cell[0, 4].Width := '180';  // [04] 金額

    // 高さの初期設定
    Cell[0, 0].Height := '35';
  end;
end;
```

## ソース 6

**frmListのListDataSetプロシージャ**

```
procedure TfrmList.ListDataSet;
var
  i: Integer;
begin
  // 明細
  with gdList do
  begin
    // サンプルデータを参照
    UserSession.cdsSample.First;

    while not UserSession.cdsSample.Eof do
    begin
      RowCount := RowCount + 1;
      i := RowCount - 1;

      // Alignmentの設定
      Cell[i, 0].Alignment := taCenter;
      Cell[i, 1].Alignment := taLeftJustify;
      Cell[i, 2].Alignment := taRightJustify;
      Cell[i, 3].Alignment := taRightJustify;
      Cell[i, 4].Alignment := taRightJustify;

      【ソース7】へ続く
```

ServerControllerで宣言されている
UserSessionメソッドを使用して
UserSessionUnitのcdsSampleを参照する

## ソース 7

**frmListのListDataSetプロシージャ（続き）**

```
      // サンプルデータ
      Cell[i, 0].Text := UserSession.cdsSampleSYCD.AsString;                    // 商品コード
      Cell[i, 1].Text := UserSession.cdsSampleSYNM.AsString;                    // 商品名
      Cell[i, 2].Text := FormatFloat('#,0', UserSession.cdsSampleSURY.AsInteger);  // 数量
      Cell[i, 3].Text := FormatFloat('#,0', UserSession.cdsSampleTANK.AsInteger);  // 単価
      Cell[i, 4].Text := FormatFloat('#,0', UserSession.cdsSampleKNGK.AsCurrency); // 金額

      UserSession.cdsSample.Next;
    end;
  end;

  // 明細の項目幅調整
  GridLayout;
end;
```

続いてfrmListのOnCreateイベントに、**【ソース8】**に従って明細の初期表示の処理を実装する。そして「明細表示」ボタン（btnListView）のOnClickイベントには、**【ソース9】**のようにここまで設定してきた各プロシージャの実行を記述する。

## ソース 8

**frmListのOnCreateイベント**

```
procedure TfrmList.IWAppFormCreate(Sender: TObject);
begin
  // 明細初期設定
  gdList.RowCount    := 1;
  gdList.ColumnCount := 5;

  // 明細初期化
  GridClear;

  // 明細の項目幅調整
  GridLayout;
end;
```

【ソース4】参照

【ソース5】参照

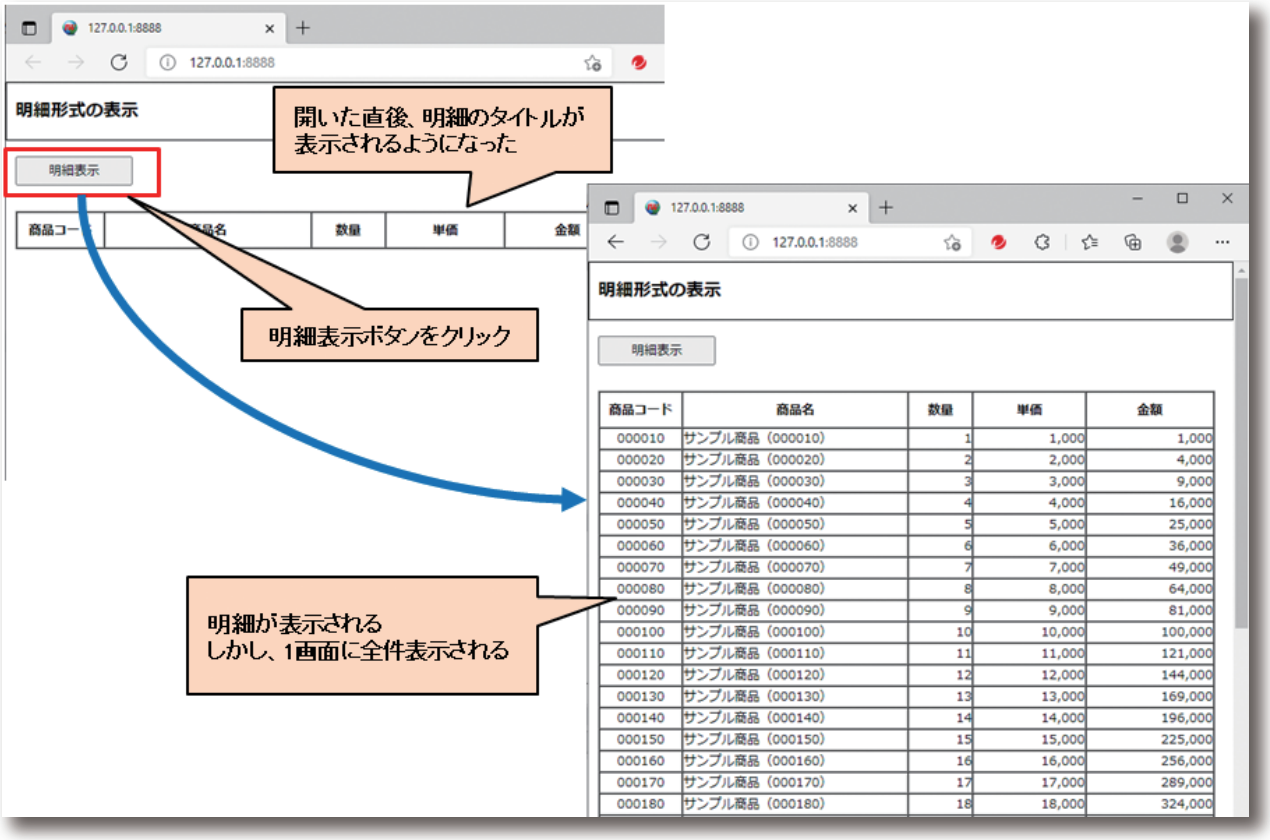
## ソース 9

**frmList.btnListViewのOnClickイベント**

```
procedure TfrmList.btnListViewClick(Sender: TObject);
begin
  // 明細初期化
  GridClear;
  // サンプルデータ作成
  UserSession.SampleDataCreate;
  // 明細データセット
  ListDataSet;
end;
```

それでは実行して確認してみよう。今回は【図7】の時とは違い明細のタイトルが表示されている。続いて画面の「明細表示」ボタンをクリックすると、明細にデータが表示される。しかし現時点では、1画面に全件を明細表示しただけとなる。**【図9】**

**図 9　実行結果イメージ（明細表示）**



### 3-2.明細表示のカスタマイズテクニック

この節では前節で作成した明細表示の画面に対して、機能追加の実装を進めていく。実装には外部ライブラリを使用する。IntraWebにおいても、画面をカスタマイズするために外部ライブラリを使用することが可能である。前節で画面に配置したTIWGridは、HTMLにはtableタグとして出力される。そのtableタグの表示内容をカスタマイズできる、「DataTables」という外部のjQueryプラグインライブラリを使用して機能追加の実装を行う。

（公式サイト：https://www.datatables.net/）

まず始めにfrmListのOnCreateイベントへ**【ソース10】**に従って追記を行う。ここではCDN（Content Delivery Network）にて公開されている、jquery DataTablesのスタイルシートとJavascriptの参照設定を追記する。参照先はURLで記述しており、その中にバージョン情報も含まれている。最新のバージョンについては、https://cdn.datatables.netを参照いただきたい。

続いてfrmListのOnRenderイベントへスクリプトの追加を記述する。**【ソース11】**

記述した中に有る”#TBLGDLIST”の”GDLIST”は、画面のTIWGridのNameプロパティの設定値を大文字にしたものとなる。もし**【図5】**の設定時にTIWGridのNameプロパティを異なる設定にしている場合は、その設定に合わせていただきたい。

**ソース 10**

**frmListのOnCreateイベント（追記）**

```
procedure TfrmList.IWAppFormCreate(Sender: TObject);
begin
  // jQuery DataTables の参照設定
  ContentFiles.Add('https://cdn.datatables.net/1.11.0/css/jquery.dataTables.min.css');
  ContentFiles.Add('https://cdn.datatables.net/1.11.0/js/jquery.dataTables.min.js');

  // 明細初期設定
  gdList.RowCount    := 1;
  gdList.ColumnCount := 5;

  // 明細初期化
  GridClear;

  // 明細の項目幅調整
  GridLayout;
end;
```

追記する

最新のバージョンは以下のサイトで確認
https://cdn.datatables.net

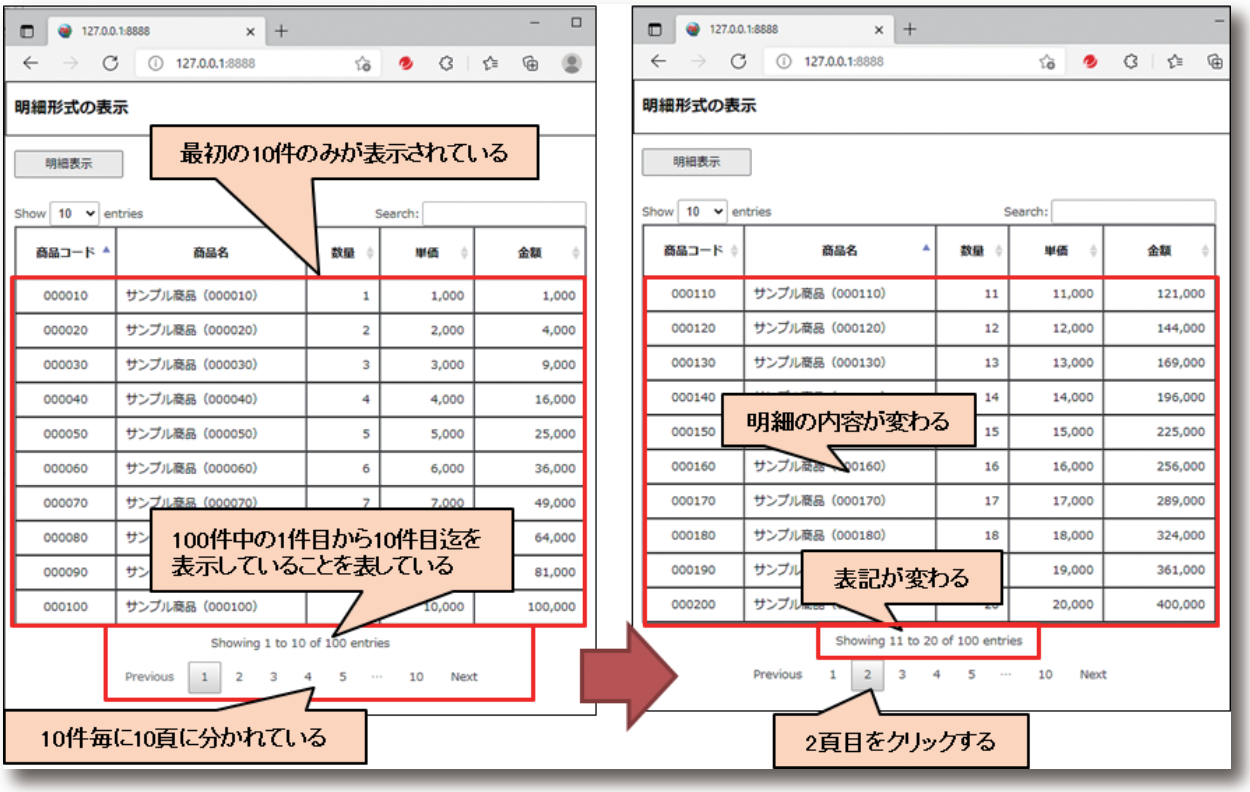
**ソース 11**

**frmListのOnRenderイベント**

```
procedure TfrmList.IWAppFormRender(Sender: TObject);
begin
  // HTMLスクリプトを追加
  AddToInitProc('$("#TBLGDLIST").DataTable();');
end;
```

それでは実行して確認してみよう。Web画面が開くと【図9】の時点から更に、いくつか項目や表記が追加されており、明細のタイトル部にも▲▼が表示されている。ただ全て英語表記となっている。明細を表示して追加された各機能を見ていこう。明細を表示すると最初の10件のみが明細に表示される。明細の下には「Showing 1 to 10 of 100 entries」と表記されており、100件中の1件目から10件目迄を表示していることを表している。更にその下には、「Previous」「1・・・10」「Next」が表示されている。これは、10件毎に10頁に分かれている各頁間を移動する機能になっている。それぞれクリックすると明細の内容が変わっているのと、「Showing（X）to（Y）of 100 entries」の（X）と（Y）の値が変わっているのが分かる。これで件数が一定数を超えた際に頁を分ける機能が実装されたことになる。【図10】

**図 10　件数が一定数を超えた際に頁を分ける機能**



次に明細のタイトルをクリックしてみると、クリックした項目の昇順で明細が並び変わる。しかも頁内ではなく、100件全体での並び替えとなっている。更に同じ項目をクリックすると今度は降順に切り替わる。これで明細のタイトルをクリックしたら並び替えが行える機能も実装されたことになる。**【図11】**

明細の上に追加された項目についても見てみる。左上のコンボBOX「Show」は1頁当りの件数を示している。初期値は「10」でコンボBOXリストには既定の値がセットされている。コンボBOXの値を変える毎に、選択した値の件数が1頁当りの件数となって、明細の表示内容が変わる。

また右上の「Search」は検索項目になる。Search項目に”12”と入力してみると、”1”を入力した時点で検索が行われ、”2”を入力すると更に検索が行われるのが分かる。しかも、この検索は、全件（100件）かつ全項目を対象に曖昧検索（入力した文字列を含む検索）を行っている。**【図12】**

**図 11　明細の並び替え機能**

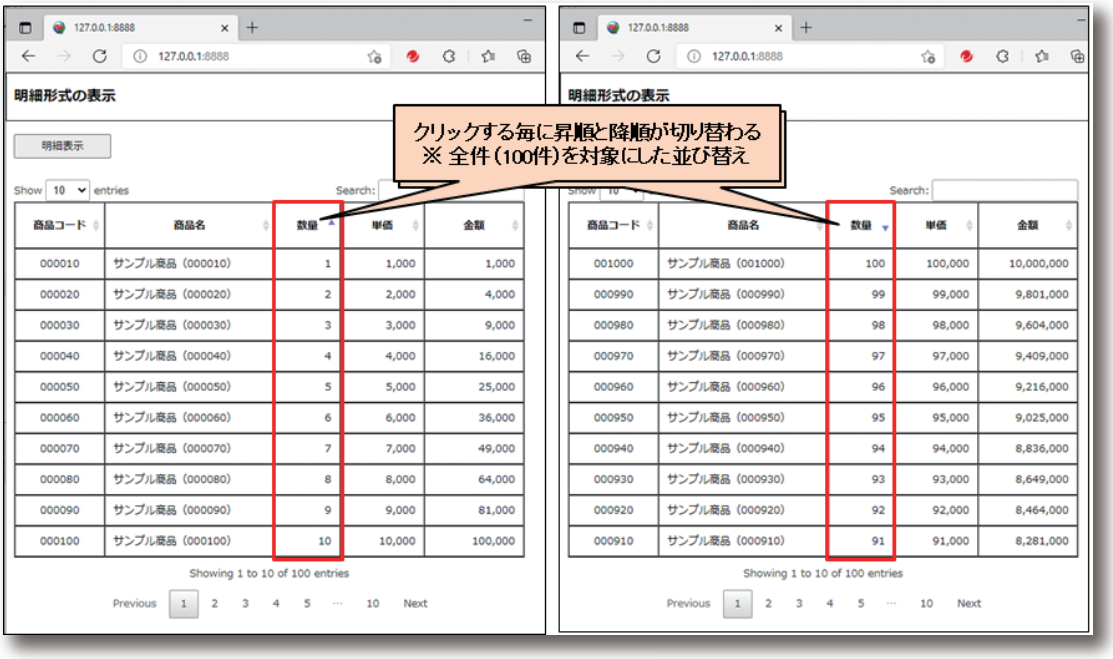
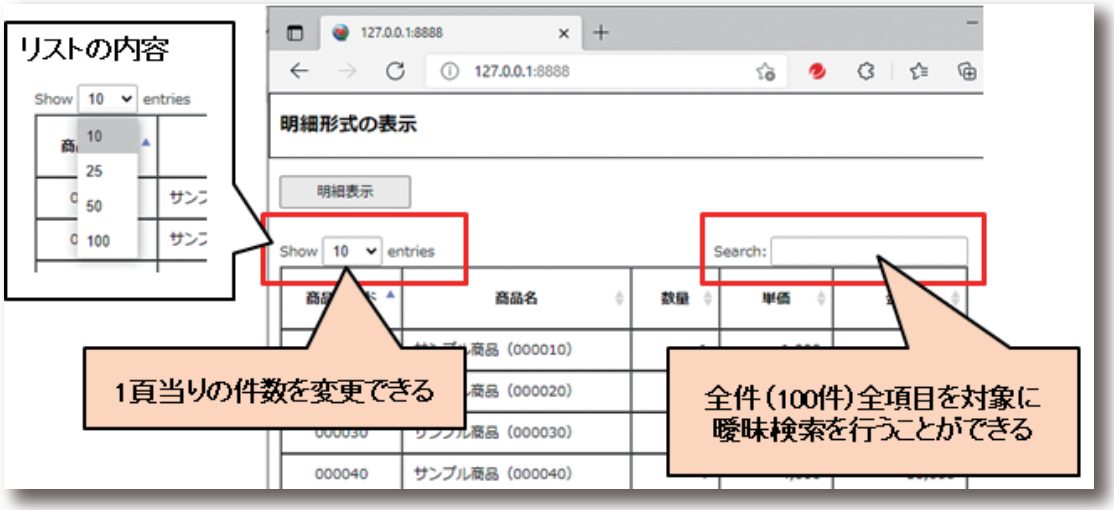



**図 12　追加機能「Show」「Search」**



ここまでの機能を僅か3行のソースコードを追加するだけで実現することができる。また、ShowコンボBOXやSearch項目の非表示や、1頁当りの件数の初期値を変更することも可能である。frmListのOnRenderイベントのソースコードを変更することで実装できる。詳しくは**【ソース12】**を参照し、実装して結果を確認してもらいたい。

この節の最後に、追加された表記が英語になっているものを日本語に変更する方法を説明する。変更箇所はfrmListのOnRenderイベントで、ソースコードを**【ソース13】**に従って変更することで日本語表記に変わる。ここでもCDNにて公開されているJSON（JavaScript Object Notation）を参照する。

## ソース 12

**frmListのOnRenderイベント（変更）**

```
procedure TfrmList.IWAppFormRender(Sender: TObject);
var
  sScriptStr: string;
begin
  // HTMLスクリプトを追加
//  AddToInitProc('$("#TBLGDLIST").DataTable();');

  sScriptStr := '$("#TBLGDLIST").DataTable({';
  sScriptStr := sScriptStr + 'displayLength: 20';
  sScriptStr := sScriptStr + ', lengthChange: false ';
  sScriptStr := sScriptStr + ', searching: false ';
  sScriptStr := sScriptStr + '});';

  //HTMLスクリプトを追加
  AddToInitProc(sScriptStr);
end;
```

「displayLength: 20」
1頁当りの件数の初期値を設定

「lengthChange: false」
ShowコンボBOXを非表示にする

「searching: false」
Search項目を非表示にする

## ソース 13

**frmListのOnRenderイベント（日本語対応）**

```
procedure TfrmList.IWAppFormRender(Sender: TObject);
var
  sScriptStr: string;
begin
  // HTMLスクリプトを追加
//  AddToInitProc('$("#TBLGDLIST").DataTable();');

  sScriptStr := '$.extend( $.fn.dataTable.defaults, {';
  sScriptStr := sScriptStr + 'language: {';
  sScriptStr := sScriptStr
                + 'url: "http://cdn.datatables.net/plug-ins/9dcbecd42ad/i18n/Japanese.json"';
  sScriptStr := sScriptStr + '}';
  sScriptStr := sScriptStr + '});';

  sScriptStr := sScriptStr + '$("#TBLGDLIST").DataTable();';

  //HTMLスクリプトを追加
  AddToInitProc(sScriptStr);
end;
```

実行して確認してみよう。【図10】では英語表記になっていた箇所が日本語表記に変わっている。【図13】

以上で「明細形式のWeb画面を作成するテクニック」の説明は終了となる。次の章ではCSVファイルを取扱うテクニックを紹介する。

**図 13　日本語表記対応**


## 4. CSVファイルのアップロード／取込を行うテクニック

この章では「CSVファイルをアップロード」し「アップロードしたCSVファイルのデータを取り込み画面明細へ表示する」方法について説明を行う。

### 4-1.CSVファイルのアップロード

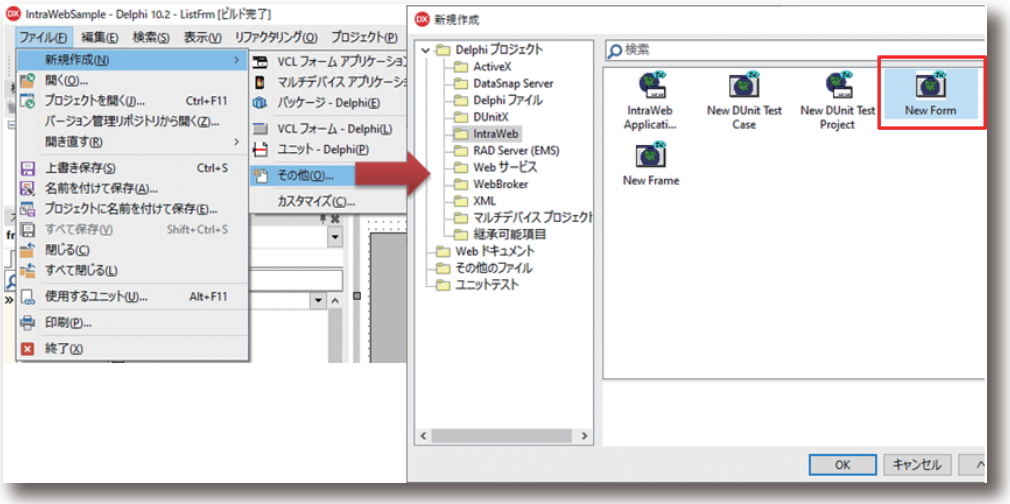
最初はCSVファイルをアップロードする機能を実装していく。新しくフォームを追加して実装するため、メニューバーより「ファイル」→「新規作成」→「その他」の順に選択し、表示された新規作成画面で「Delphiプロジェクト」→「IntraWeb」→「New Form」をクリックする。【図14】

新しく追加された「Unit1」のNameプロパティを「frmCSVUpload」と設定する。【図15】に従ってコンポーネントを配置し各プロパティを設定する。この時「TIWFileUploader」のTextStringsプロパティの設定も合わせて行う。【図16】

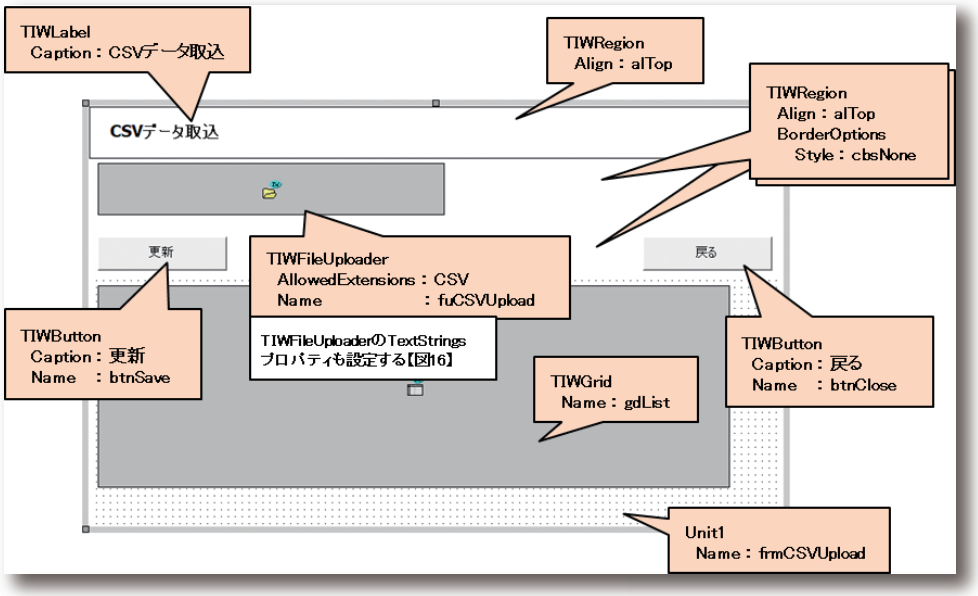
この画面は「TIWFileUploader」を使用してアップロードしたCSVファイルを読み込み、そのデータを「TIWGrid」に表示をする。そして「更新」ボタンをクリックしたら、前章で設定したUserSessionUnitのcdsSampleへ更新を行い、当画面を閉じてfrmListの明細へ反映させる動きとなる。

配置と設定が終わったら、Unit1をファイル名「CSVUploadFrm.pas」として「名前を付けて保存」を行う。

## 図 14 「CSVアップロード用画面」の追加




## 図 15 「CSVアップロード用画面」の画面設計




## 図 16 TIWFileUploaderのTextStringsプロパティの設定

| TextStrings | (TIWFileUploaderTextStrings) |
|---|---|
| CancelButtonText | キャンセル |
| DragText | ここにファイルをドロップしてください。 |
| EmptyErrorText | {file} は空です。ファイルを再度選択してください。 |
| MinSizeErrorText | {file} is too small, minimum file size is {minSizeLimit}. |
| MultipleFileDropNotAllowedText | ドロップできるのは1つのファイルだけです。 |
| NoFilesErrorText | アップロードするファイルがありません。 |
| OfTotalText | of |
| OnLeaveWarningText | ファイルがアップロードされていますが、今離れるとアップロードがキャンセルされてしまいます。 |
| RemoveButtonText | 除去 |
| SizeErrorText | {file} is too large, maximum file size is {sizeLimit}. |
| TypeErrorText | {file} の拡張子が正しくありません。{extensions}ファイルのみが許可されます。 |
| UploadButtonText | ファイルアップロード |
| UploadErrorText | アップロードエラー |

始めに、「TIWFileUploader」を使用してアップロードしたCSVファイルを読み込み、そのデータを「TIWGrid」に表示するところまでを実装していく。前章でfrmListに実装したのと同様に、frmCSVUploadのPrivateプロシージャとして「GridClear」「GridLayout」の2つを宣言する。合わせてプライベート変数「FUploadFile」もString型で宣言しておく。

**【ソース14】** 各プロシージャの実装部については、GridClearは**【ソース4】**、GridLayoutは**【ソース5】**と同様の処理を実装する。

## ソース 14

**frmCSVUploadのPrivateプロシージャと変数の宣言**

```
private
    { Private 宣言 }
    FUploadFile: String;        // CSVファイル保管場所

    procedure GridClear;        // 明細初期化
    procedure GridLayout;       // 明細の項目幅調整
```

そしてfrmCSVUploadのOnCreateイベントにも、前章でfrmListに実装したのと同様に明細の初期表示の処理を実装する。**【ソース8】**

続いて、TIWFileUploaderの各イベントに処理を実装していく。まずはOnAsyncUploadCompletedイベントに**【ソース15】**に従って処理を実装する。ここでは、CSVファイルのアップロードが終わった時点で、アップロードしたCSVファイルのフルパスをFUploadFileに退避する。

次にOnAsyncUploadSuccessイベントに**【ソース16〜17】**に従って処理を実装する。ここでは、OnAsyncUploadCompletedイベントで取得したCSVファイルのフルパス：FUploadFileを使用して、アップロードしたCSVファイルを開き、画面明細へ展開する処理となっている。

## ソース 15

**frmCSVUpload.fuCSVUploadのOnAsyncUploadCompletedイベント**

```
procedure TfrmCSVUpload.fuCSVUploadAsyncUploadCompleted(Sender: TObject;
  var DestPath, FileName: string; var SaveFile, Overwrite: Boolean);
begin
  //アップロードしたファイルのパスとファイル名を取得
  FUploadFile := DestPath + FileName;
end;
```

## ソース 16

### frmCSVUpload.fuCSVUploadのOnAsyncUploadSuccessイベント

```
procedure TfrmCSVUpload.fuCSVUploadAsyncUploadSuccess(Sender: TObject;
  EventParams: TStringList);
var
  i: Integer;
  sStrList: TStringList;
  sRowList: TStringList;
begin
  // 明細初期化
  GridClear;

  //CSVデータを保持するリストの作成
  sStrList := TStringList.Create;
  try
    //行データを保持するリストの作成
    sRowList := TStringList.Create;
    try
      sStrList.LoadFromFile(FUploadFile);
      //先頭行は、タイトルの為無視して、2行目から取得
      for i := 1 to sStrList.Count - 1 do
      begin
        //行のテキスト（カンマ区切り文字列）をsRowListにセット
        sRowList.Clear;
        sRowList.CommaText := sStrList[i];
        //sRowListの要素より明細へ値をセット
        gdList.RowCount := gdList.RowCount + 1;
```

【ソース17】へ続く

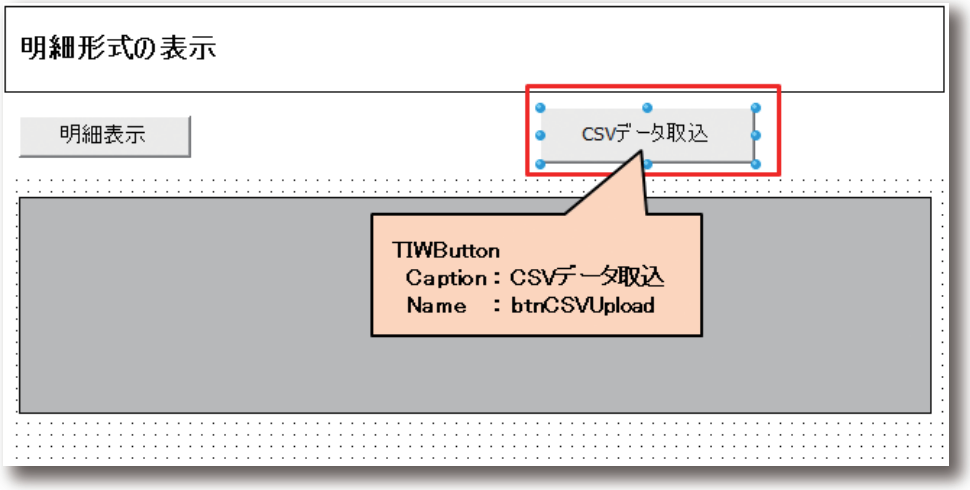
## ソース 17

### frmCSVUpload.fuCSVUploadのOnAsyncUploadSuccessイベント（続き）

```
        // Alignmentの設定
        gdList.Cell[gdList.RowCount - 1, 0].Alignment := taCenter;
        gdList.Cell[gdList.RowCount - 1, 1].Alignment := taLeftJustify;
        gdList.Cell[gdList.RowCount - 1, 2].Alignment := taRightJustify;
        gdList.Cell[gdList.RowCount - 1, 3].Alignment := taRightJustify;
        gdList.Cell[gdList.RowCount - 1, 4].Alignment := taRightJustify;

        // サンプルデータ
        gdList.Cell[gdList.RowCount - 1, 0].Text := sRowList[0];
        gdList.Cell[gdList.RowCount - 1, 1].Text := sRowList[1];
        gdList.Cell[gdList.RowCount - 1, 2].Text := FormatFloat('#,0', StrToIntDef(sRowList[2], 0));
        gdList.Cell[gdList.RowCount - 1, 3].Text := FormatFloat('#,0', StrToIntDef(sRowList[3], 0));
        gdList.Cell[gdList.RowCount - 1, 4].Text := FormatFloat('#,0', StrToCurrDef(sRowList[4], 0));

      end;
    finally
      sRowList.Free;
    end;
  finally
    sStrList.Free;
  end;

  // 明細の項目幅調整
  GridLayout;
end;
```

ここまで実装できたら、frmListからfrmCSVUploadを呼び出すように、frmListへ処理を追加する。frmListを開き【図17】に従ってTIWButtonを追加する。そして追加した「CSVデータ取込」ボタン（btnCSVUpload）のOnClickイベントには【ソース18】に従って処理を実装する。合わせてfrmListのuses節に「CSVUploadFrm」を追加し、プライベート変数「frmCSVUpload」の宣言を行う。

**図 17　frmListに「CSVデータ取込ボタン」を追加**




**ソース 18**

**frmListのbtnCSVUploadのOnClickイベント**

＜usesに「CSVUploadFrm」を追加＞

```
uses
  Classes, SysUtils, IWAppForm, IWApplication, IWColor, IWTypes,
  IWVCLBaseControl, IWBaseControl, IWBaseHTMLControl, IWControl, IWCompLabel,
  Vcl.Controls, Vcl.Forms, IWVCLBaseContainer, IWContainer, IWHTMLContainer,
  IWHTML40Container, IWRegion, IWCompGrids, IWCompButton, CSVUploadFrm;
```

＜private変数を宣言＞

```
 private
    { Private 宣言 }
    frmCSVUpload: TfrmCSVUpload;

    procedure GridClear;        // 明細初期化
    procedure GridLayout;       // 明細の項目幅調整
    procedure ListDataSet;      // 明細データセット
```

＜OnClickイベント＞

```
procedure TfrmList.btnCSVUploadClick(Sender: TObject);
begin
  frmCSVUpload := TfrmCSVUpload.Create(WebApplication);
  frmCSVUpload.Show;
end;
```

実行する前にアップロード用のCSVファイルを準備しておく。CSVファイルのデータイメージは**【図18】**を参照いただきたい。それでは実行して確認してみよう。今節で追加したfrmListの「CSVデータ取込」ボタンをクリックすると、frmCSVUploadへ遷移する。画面上部には背景色が赤色の「ファイルアップロード」ボタンが表示されている。CSVファイルをアップロードする方法は以下の2通りがある。**【図19】**

①「ファイルアップロード」ボタンをクリックして、ファイル選択。ダイアログを開き、ダイアログよりCSVファイルを選択する。

②CSVファイルを直接「ファイルアップロード」ボタンへドラッグ＆ドロップする。

どちらかの方法でCSVファイルのアップロードを行う。そうすると画面の明細にデータが表示され、アップロードしたCSVファイルのデータが確認できる。**【図20】**

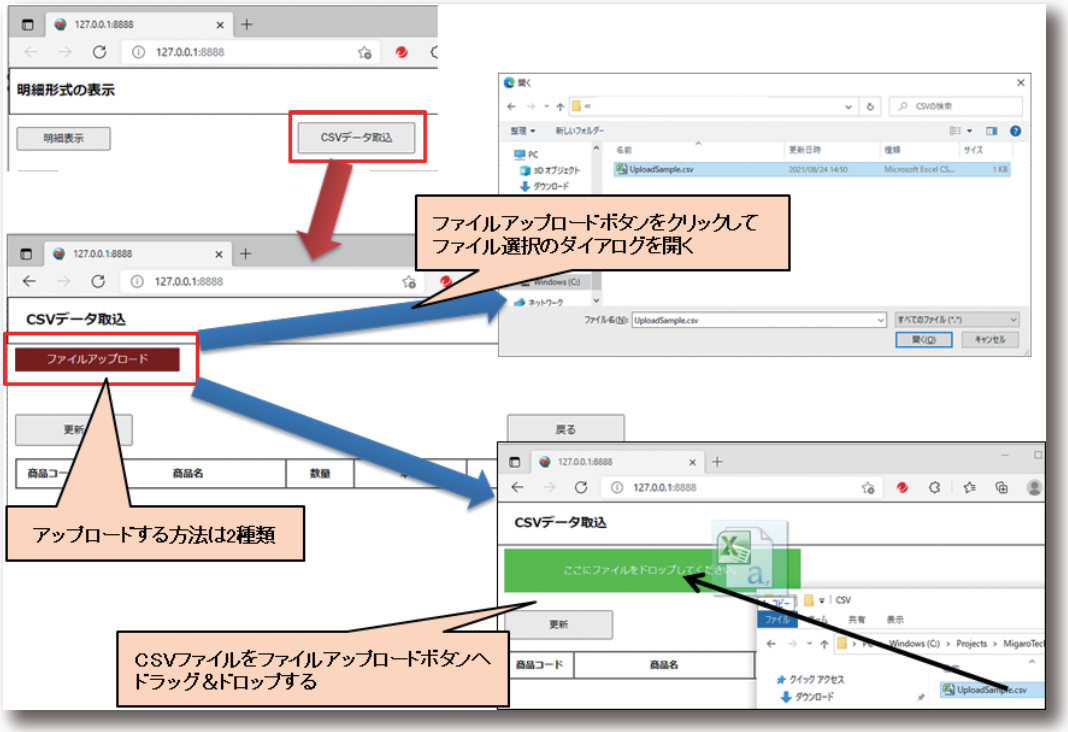
では、CSVファイル以外をアップロードしたらどうなるのか確認してみよう。PDFファイルを用意しアップロードを行うと、**【図21】**のようにエラーが発生する。これは、**【図15】**でTIWFileUploaderのAllowedExtensionsプロパティに”CSV”を設定したことで、拡張子を制限しているためである。

TIWFileUploaderは先程の2通りのアップロードの方法を既に実装しており、またWebサーバーで実行した場合には、アップロードしたファイルをWebサーバー内のフォルダに保管する機能も実装している。そのためTIWFileUploaderを使用することで、Webアプリケーションでのファイルアップロード機能を簡単に実装することが可能となる。

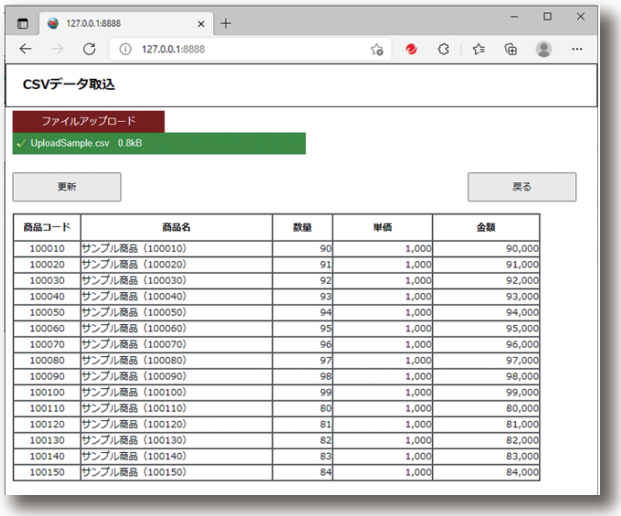
**図 18　CSVファイルのデータイメージ**

```
1  商品コード,商品名,数量,単価,金額
2  "100010","サンプル商品（100010）",90,1000,90000
3  "100020","サンプル商品（100020）",91,1000,91000
4  "100030","サンプル商品（100030）",92,1000,92000
5  "100040","サンプル商品（100040）",93,1000,93000
6  "100050","サンプル商品（100050）",94,1000,94000
7  "100060","サンプル商品（100060）",95,1000,95000
8  "100070","サンプル商品（100070）",96,1000,96000
9  "100080","サンプル商品（100080）",97,1000,97000
10 "100090","サンプル商品（100090）",98,1000,98000
11 "100100","サンプル商品（100100）",99,1000,99000
12 "100110","サンプル商品（100110）",80,1000,80000
13 "100120","サンプル商品（100120）",81,1000,81000
14 "100130","サンプル商品（100130）",82,1000,82000
15 "100140","サンプル商品（100140）",83,1000,83000
16 "100150","サンプル商品（100150）",84,1000,84000
```

## 図 19　実行結果イメージ（CSVアップロード）

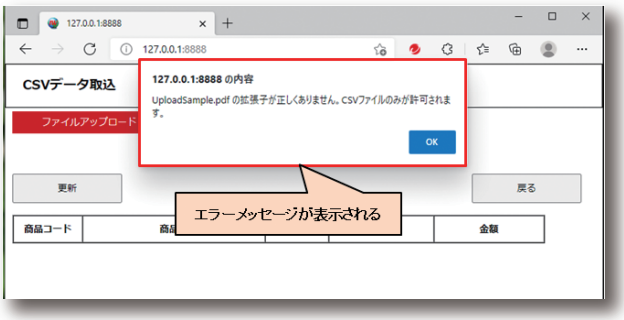



## 図 20　実行結果イメージ（CSVアップロード）




| 商品コード | 商品名 | 数量 | 単価 | 金額 |
|---|---|---|---|---|
| 100010 | サンプル商品（100010） | 90 | 1,000 | 90,000 |
| 100020 | サンプル商品（100020） | 91 | 1,000 | 91,000 |
| 100030 | サンプル商品（100030） | 92 | 1,000 | 92,000 |
| 100040 | サンプル商品（100040） | 93 | 1,000 | 93,000 |
| 100050 | サンプル商品（100050） | 94 | 1,000 | 94,000 |
| 100060 | サンプル商品（100060） | 95 | 1,000 | 95,000 |
| 100070 | サンプル商品（100070） | 96 | 1,000 | 96,000 |
| 100080 | サンプル商品（100080） | 97 | 1,000 | 97,000 |
| 100090 | サンプル商品（100090） | 98 | 1,000 | 98,000 |
| 100100 | サンプル商品（100100） | 99 | 1,000 | 99,000 |
| 100110 | サンプル商品（100110） | 80 | 1,000 | 80,000 |
| 100120 | サンプル商品（100120） | 81 | 1,000 | 81,000 |
| 100130 | サンプル商品（100130） | 82 | 1,000 | 82,000 |
| 100140 | サンプル商品（100140） | 83 | 1,000 | 83,000 |
| 100150 | サンプル商品（100150） | 84 | 1,000 | 84,000 |

## 図 21　CSVファイル以外をアップロードした場合



### 4-2.CSVファイルの取込と明細への反映方法

前節では、CSVファイルをアップロードしてデータを画面明細に取り込むところを実装してきた。この節ではアップロードしたデータを、「更新」ボタンをクリックすることでUserSessionUnitのcdsSampleへ更新し、当画面を閉じてfrmListの明細へ反映させる機能を実装していく。

始めにUserSessionUnitにプライベート変数「FDataDspMode」と、Property「DataDspMode」を設定する。**【ソース19】**

**ソース 19**

**UserSessionUnitのプライベート変数とProperty**

```
private
  { Private declarations }
  FDataDspMode: Boolean;

public
  { Public declarations }
  procedure SampleDataCreate;   // サンプルデータ作成

  //  「明細形式の表示」のデータ表示モード（True：データ表示）
  property DataDspMode: Boolean read FDataDspMode write FDataDspMode;
end;
```

次にfrmCSVUploadを開き、メニューバーより「ファイル」→「使用するユニット」を選び、ServerControllerを選択する。そして、「更新」ボタン（btnSave）のOnClickイベントに**【ソース20～21】**に従って処理を記述する。ここでは、最初にcdsSampleのデータを全て削除した後で、CSVファイルのデータをcdsSampleへ追加している。その後でUserSessionUnitのDataDspModeプロパティを「True」にしている。DataDspModeプロパティに設定した値は、frmListに戻った際に使用する。最後に「Self.Release」で画面を閉じている。

また、「戻る」ボタン（btnClose）のOnClickイベントにも**【ソース22】**に従って処理を記述する。frmCSVUploadに対する機能の実装は以上となる。

**ソース 20**

**frmCSVUpload.btnSaveのOnClickイベント**

```
procedure TfrmCSVUpload.btnSaveClick(Sender: TObject);
var
  i: Integer;
  sStrList: TStringList;
  sRowList: TStringList;
begin
  // UserSessionのcdsSampleをクリア
  UserSession.cdsSample.EmptyDataSet;

  //CSVデータを保持するリストの作成
  sStrList := TStringList.Create;
  try
    //行データを保持するリストの作成
    sRowList := TStringList.Create;
    try
      sStrList.LoadFromFile(FUploadFile);
      //先頭行は、タイトルの為無視して、2行目から取得
      for i := 1 to sStrList.Count - 1 do
      begin
        //行のテキスト（カンマ区切り文字列）をsRowListにセット
        sRowList.Clear;
        sRowList.CommaText := sStrList[i];
```

【ソース21】へ続く

## ソース 21

**frmCSVUpload.btnSaveのOnClickイベント（続き）**

```
        // サンプルデータを追加
        UserSession.cdsSample.Append;
        UserSession.cdsSampleSYCD.AsString    := sRowList[0];                  // 商品コード
        UserSession.cdsSampleSYNM.AsString    := sRowList[1];                  // 商品名
        UserSession.cdsSampleSURY.AsInteger   := StrToIntDef(sRowList[2], 0);  // 数量
        UserSession.cdsSampleTANK.AsInteger   := StrToIntDef(sRowList[3], 0);  // 単価
        UserSession.cdsSampleKNGK.AsCurrency  := StrToCurrDef(sRowList[4], 0); // 金額
        UserSession.cdsSample.Post;
      end;
    finally
      sRowList.Free;
    end;
  finally
    sStrList.Free;
  end;

  // データ表示モードセット
  UserSession.DataDspMode := True;

  // 閉じる
  Self.Release;
end;
```

## ソース 22

**frmCSVUpload.btnCloseのOnClickイベント**

```
procedure TfrmCSVUpload.btnCloseClick(Sender: TObject);
begin
  // 閉じる
  Self.Release;
end;
```

続いてfrmListを開き、OnRenderイベントを【ソース23】に従って変更する。ここではUserSessionUnitのDataDspModeプロパティが「True」の場合に、明細をクリアしてからcdsSampleのデータを明細に表示している。またOnRenderイベントはfrmCSVUploadから戻ってきた際にも発生するため、frmCSVUploadの更新ボタンをクリックすれば、DataDspModeプロパティが「True」になっているため、CSVファイルのデータで洗い替えられたcdsSampleのデータがfrmListの明細に表示されることになる。

## ソース 23

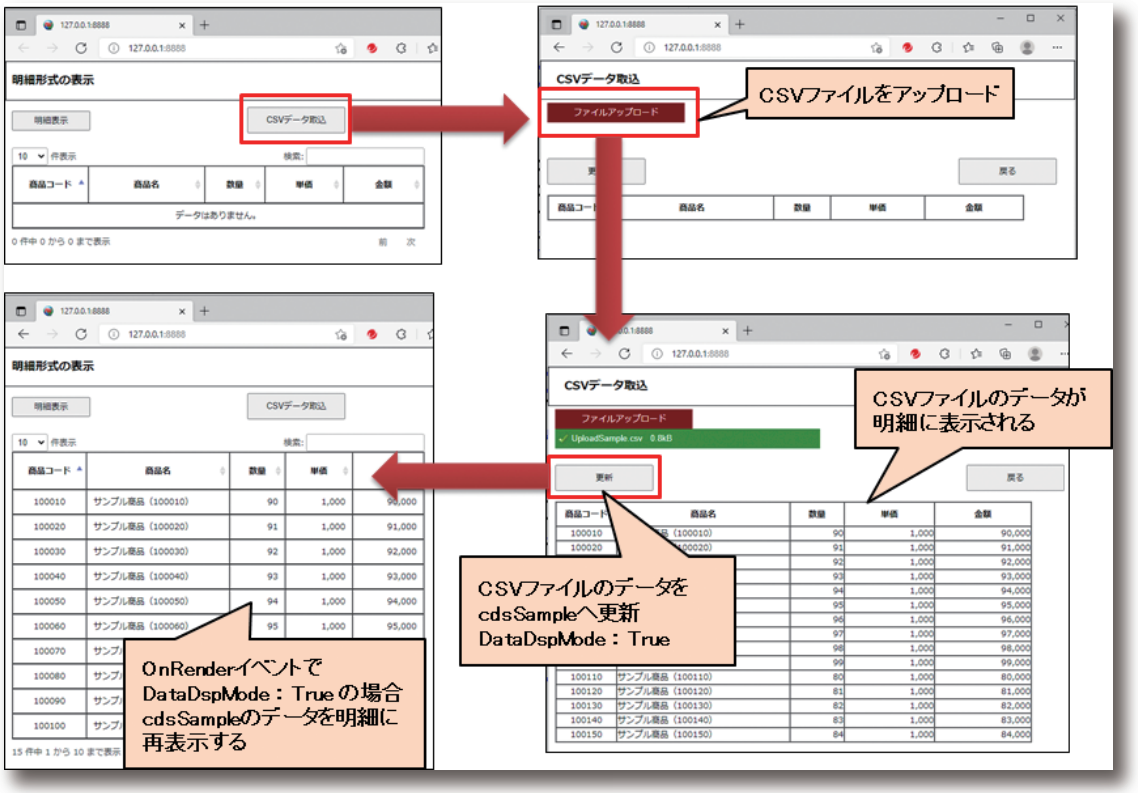
**frmListのOnRenderイベント（変更）**

```
procedure TfrmList.IWAppFormRender(Sender: TObject);
var
  sScriptStr: string;
begin
  sScriptStr := '$.extend( $.fn.dataTable.defaults, {';
  sScriptStr := sScriptStr + 'language: {';
  sScriptStr := sScriptStr
        + 'url: "http://cdn.datatables.net/plug-ins/9dcbecd42ad/i18n/Japanese.json"';
  sScriptStr := sScriptStr + '}';
  sScriptStr := sScriptStr + '});';

  sScriptStr := sScriptStr + '$("#TBLGDLIST").DataTable();';

  //HTMLスクリプトを追加
  AddToInitProc(sScriptStr);

  // CSVデータ取込画面から戻ってきたときに明細を再表示する
  if (UserSession.DataDspMode) then
  begin
    UserSession.DataDspMode := False;

    // 明細初期化
    GridClear;
    // 明細データセット
    ListDataSet;
  end;
end;
```

それでは実行して確認してみよう。frmCSVUploadでCSVファイルをアップロードした後「更新」ボタンをクリックすると、frmListの明細にアップロードしたCSVファイルのデータが表示されることが確認できる。【図22】

以上で「CSVファイルのアップロード／取込を行うテクニック」の説明は終了となる。

**図 22　CSVファイルのアップロード／取込／明細表示**




## 5.さいごに

本稿ではIntraWebを使用したWeb開発のTipsとして、Web画面での明細形式の表示を行う際のテクニックと、CSVファイルをアップロードして取り込み画面に表示する方法について紹介した。これらの機能は、コンポーネントの設定と、CDNで公開されているライブラリの利用、そして少しのソースコードを実装することで実現することができた。また、本稿を執筆するにあたり参考にした、IntraWebの開発元であるAtozed社が公開しているデモプログラムには、IntraWebの活用方法が他にも沢山紹介されている。こちらも是非参考にしていただきたい。
（URL：https://github.com/Atozed/IntraWeb）
本稿がIntraWebを使用してWeb開発を行う際の参考となれば幸いである。

# Delphi/400

# FireDACを活用した Delphi/400ロジック最新化テクニック

株式会社ミガロ.
RAD事業部技術支援課
**佐田　雄一**

**略　歴**

生年月日:1985年12月6日
最終学歴:2009年　甲南大学 経営学部卒業
ミガロ入社年月:2009年04月　株式会社ミガロ. 入社
社内経歴:
2009年04月　システム事業部配属
2019年04月　RAD事業部配属

**現在の仕事内容:**

Delphi/400を利用した
システム開発や保守作業の経験を経て、
現在はDelphi/400のサポート業務を担当。



## 1.はじめに

Delphi/400 Version 5が2000年に日本で販売を開始してから早くも20年が経過したが、実は当時から存在するロジックの大半が現在の最新バージョン(Delphi/400 10.2 Tokyo)でも利用することができる。しかし当時はWindows 95や98がまだ現役だった頃で、その後のWindows OS側の技術革新で、初期のコンポーネントやロジックの中には動作にあたって制約が生じているものも存在する。

本稿ではSQL等のIBM i側の資産や、最新のFireDAC接続を用いて、最新バージョンでさらに使いやすくなったコンポーネントに関するトピックと、先述の動作に制約が発生しているソースをリフレッシュするためのトピックの2つを紹介する。

なお、本稿ではDelphi/400 10.2 TokyoおよびV7R1以上のIBM i環境を使用している。

## 2.TClientDatasetに代わるデータセットTFDMemTableの活用術

### ① TFDMemTableとは

IBMiのデータベースファイルからデータを参照して表形式で表示する際に、もっとも一般的なコンポーネントの配置順は「データセットコンポーネント（TFDQuery・TSQLQuery等）⇒TDataSetProvider⇒TClientDataSet⇒TDataSource⇒TDBGrid」となるだろう。

ここで使用するTClientDataSetコンポーネントは取得したデータを内部メモリでキャッシュするための汎用インメモリデータセットで、単独使用やdbExpress接続におけるキャッシュ機能の実現の為に用いられる。最新のDelphi/400においてもTClientDataSetやTDataSetProviderといったコンポーネントをこれまで通り使用することは可能であるが、古いBDE接続やdbExpress接続のアーキテクチャにあわせて作られており、Version 5の頃から基本的な仕様はそのままとなっている。また、Delphi開発元のエンバカデロ・テクノロジーズ社は今後の更新が行われる可能性は限りなく低いとしており、FireDAC接続でのデータキャッシュにおいては後継となるTFDMemTableコンポーネントの使用を推奨している。

dbExpress接続では単方向での通信だったものがFireDAC接続においては双方向で通信可能なため、FireDAC接続ではTClientDataSetを経由させずに「TFDQuery⇒TDataSource⇒TDBGrid」といったコンポーネントの配置が可能である。一方、旧バージョンにおいてBDE接続やdbExpress接続で作成されたアプリケーションをFireDAC接続に移行する場合には、元々TClientDataSetで使用されていた内部計算項目の使用や、古いParadoxに代わるローカルデータベースとしての活用がTFDMemTableコンポーネントによって可能になっている。

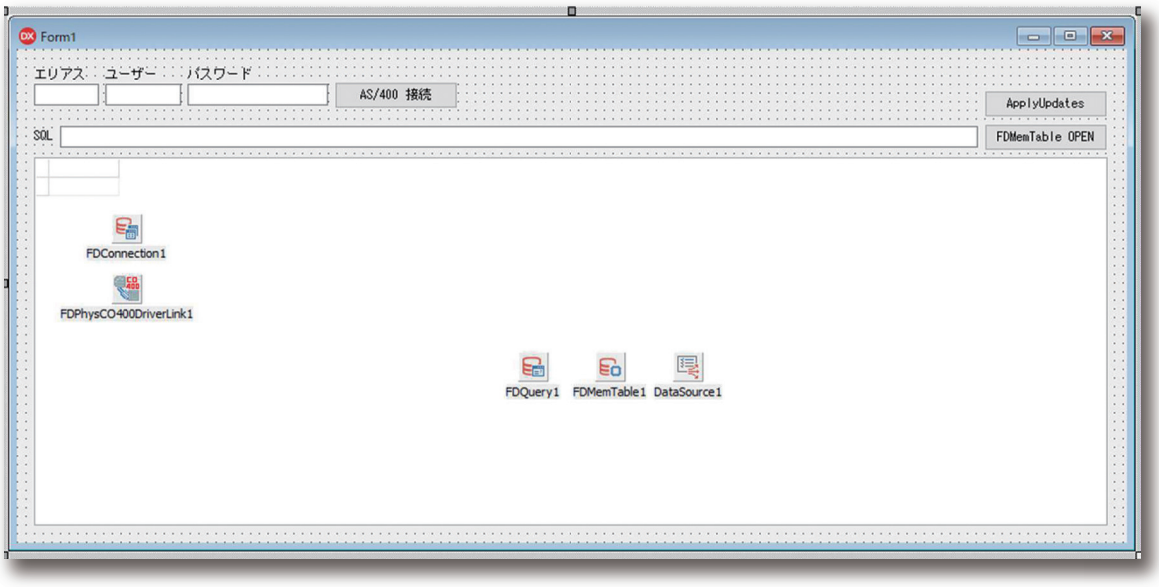
次項より、FireDAC接続のために最適化された新しいデータセット、TFDMemTableの使用方法について紹介する。

### ② TFDMemTableを使ったデータ照会方法

本項ではTClientDataSet・TDataSetProviderの代替として、TFDMemTableを使ってデータを取得する手順を紹介する。

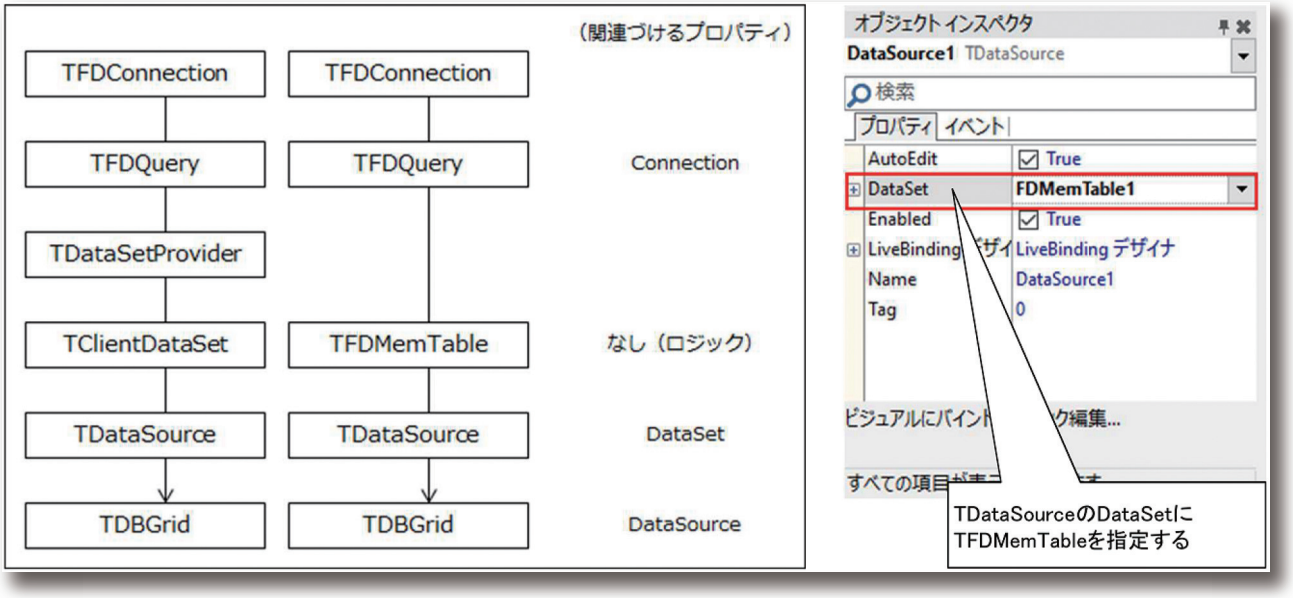
Delphiを起動してプロジェクトを新規作成したら、まずフォーム上にFireDAC接続を行うために必要なTFDConnection・TFDPhysCO400DriverLinkと、TFDQuery・TFDMemTable・TDataSource・TDBGridの各データベースコンポーネントおよび、画面操作に使用するEditやButton等を配置する【図1】。TFDConnectionを使ったFireDAC接続の設定自体は過去のテクニカルレポートでも取り上げているため、本稿では割愛する。（2018年のテクニカルレポートに掲載の拙著「FireDAC実践プログラミングテクニック」を参照されたい。）

**図 1　サンプルの画面設計**



TDBGridのDataSourceプロパティにTDataSourceを設定するのは従来と同様だが、TDataSourceのDataSetプロパティにはTClientDataSetの代わりにTFDMemTableを設定する【図2】。また、オブジェクトインスペクタ上のプロパティではTFDMemTableとTFDQueryを接続できる箇所がないため、代替としてロジックを使って、TFDQueryで取得したデータをTFDMemTableへ連携する。連携する方法はいくつか存在する【ソース1】【ソース2】。

**図 2　データセット指定**




**ソース 1**

CloneCursorを使ったTFDMemTableでのデータ参照

```
FDQuery1.Close;
FDMemTable1.Close;
FDQuery1.SQL.Text := 'SELECT * FROM TESTTBL21 ';
FDQuery1.Open;
FDMemTable1.CloneCursor(FDQuery1, False, False);
```

**ソース 2**

Dataの代入を使ったTFDMemTableでのデータ参照

```
FDQuery1.Close;
FDMemTable1.Close;
FDQuery1.SQL.Text := 'SELECT * FROM TESTTBL21 ';
FDQuery1.Open;
FDMemTable1.Data  := FDQuery1.Data;
```

**【ソース1】**はCloneCursorメソッドを使用する方法で、**【ソース2】**はDataプロパティを代入する方法である。本項ではこの2パターンを例示したが、それぞれの特徴や利点を紹介する。

いずれの例においても事前にTFDQueryをオープンしておく点は同じであるが、**【ソース1】**でCloneCursorメソッドを行う場合は、パラメータにデータ取得元となるTFDQueryを指定することで、そのTFDQueryにあるデータのポインタのみを複製し、内部的なデータは同じものを共有するイメージとなる。

対して**【ソース2】**でDataを代入する方法ではその時点で取得済みのデータをそのままTFDMemTableにコピーするイメージとなる。そのため、TFDQuery側でFetchOptionsプロパティの設定によって最初の50件までしかレコードを取得していない場合、TFDMemTableにコピーされるレコードもその件数のみとなる。たとえば元のデータの件数が非常に多く時間がかかる場合にTFDQuery側でレコードの取得件数を制限する際には、CloneCursorメソッドを使用する場合よりも処理の高速化が期待できる。

照会したデータはTClientDataSetと同じようにIndexDefs・IndexNameプロパティによるローカルでの並び替えや、後述するJSON形式でのデータ出力などTFDQuery（またはTFDTable、以下同じ）では実現できない機能が存在する。なお、TClientDataSetにも同様にコピーが可能な「Data」プロパティが存在しているが、TFDMemTableのDataとは内部の型が異なるため互換性はなく、直接代入するようなロジックを記述するとコンパイルエラーになる。TClientDataSetからTFDMemTableへのデータ同期が必要な場合はCopyDataSetメソッドを使うと、異なる接続方式のデータセットからTFDMemTableにデータを転送することができる**【ソース3】**。パラメータに同じFireDAC接続のTFDQueryを指定することも可能で、この場合はDataを代入する際と同じ挙動になる。

## ソース 3

**他種のデータセットからTFDMemTableへのデータコピー**

```
// [例] TClientDataSetでローカルデータを読み込み
ClientDataSet1.LoadFromFile('C:¥～～～¥DATA.xml');
// CopyDataSetでデータコピーするとTFDMemTableでも扱える
FDMemTable1.CopyDataSet(ClientDataSet1, [coStructure, coRestart, coAppend]);
```

### ③ TFDMemTableを使ったデータ更新方法

BDE接続・dbExpress接続においてはTClientDataSetのApplyUpdatesメソッドを使うことで、接続されたQueryコンポーネント経由でインメモリデータセットの変更内容をIBM iに更新できたが、TFDMemTableにおいてもロジックが少々異なるものの、ApplyUpdatesメソッドで同様の更新が可能である。

FireDAC接続においてはTFDQueryでも紐付くTDBGridを直接編集することや、Edit～Postのメソッドを行うことで内部的に更新SQLを発行し、接続先のDBに更新を行うことができる。(更新自体を可能にするためのUpdateOptionsプロパティの設定については、前章でも紹介した2018年のテクニカルレポートに掲載の拙著「FireDAC　実践プログラミングテクニック」を参照されたい。)

このとき、更新に使用するTFDQueryのCachedUpdatesプロパティがFalseならPostやDeleteの時点で即時更新が行われ、Trueなら更新情報を内部でキャッシュし、ApplyUpdatesメソッドを行った時点で一括更新が行われる。

BDE接続・dbExpress接続との違いとしては更新先ファイルへのジャーナル開始やトランザクション制御が不要な点が挙げられるが、複数件のデータを更新している途中でエラーになった場合のリスクを考慮すれば、従来通りジャーナルやトランザクション制御を使用しておいた方が安全である。

TFDMemTableを使用してApplyUpdatesメソッドを実行する際の前提条件としては、以下の3点があげられる。

・前項で紹介した2種類の手順のうち、**【ソース1】**のCloneCursorメソッドを使ってデータを取得している。

・ApplyUpdatesメソッドを実際に行うのは紐づいているTFDQueryとなるため、ApplyUpdates実行時までTFDQueryが開いたままになっている。(TFDMemTable側でApplyUpdatesメソッドを実行しても更新は反映されない)

・キャッシュ更新を有効にするため、先述のTFDQueryのCachedUpdatesプロパティがTrueになっている。

これらの条件が揃った状態でTFDMemTableのデータを変更し、**【ソース4】**のようなロジックでApplyUpdatesメソッドを実行すると、Open時からキャッシュされた変更がIBM　iに反映される。

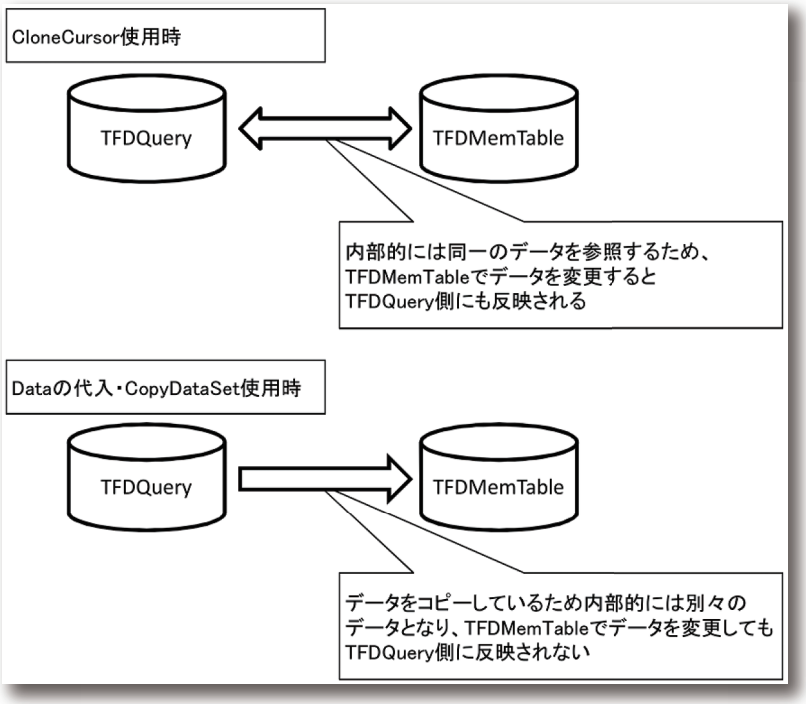
**ソース 4**

**TFDMemTableのデータ更新**

```
// 紐づくTFDQuery側でApplyUpdatesする
FDQuery1.ApplyUpdates(-1);
```

前項で紹介したデータ取得手順のうちCloneCursorメソッドを使う手順の場合はTFDQueryと同じデータを内部で共有しているためApplyUpdatesメソッドによって更新が反映されるが、Dataをコピーする手順やCopyDataSetメソッドを使用する手順の場合は別々のデータとなる【図3】ため、ApplyUpdatesメソッドを行ってもIBM iに更新は反映されない。

**図 3　TFDMemTableへのデータ転送方法の違い**




### ④TFDMemTableを使ったローカルDB活用

TFDMemTableはTClientDataSetと同じように、LoadFromFileメソッドやSaveToFileメソッドで外部データを読み込んだり書き出したりすることができる。例えば処理速度向上のためローカルPC上に値が長期間変わることのないマスタファイルのデータをコピーしておきたい場合に有効である。

保存先と形式をパラメータに指定してSaveToFileメソッドを実行すれば、指定した場所にファイルが出力される【ソース5】。

**ソース 5**

TFDMemTableのローカルデータ保存と読み込み

```
// ローカルデータにJSON形式で保存する例
FDMemTable2.SaveToFile('FDMem2.json', sfJSON);

// ローカルデータから読み込む例
FDMemTable2. LoadFromFile('FDMem2.json');
```

```
sfAuto   : 自動（拡張子で判断）
sfBinary : 専用バイナリ形式
sfXML    : XML形式
sfJSON   : JSON形式
```

TClientDataSetとの違いとしては、保存形式によって出力に使用するコンポーネントの配置が必要になること【図4】と、出力形式の違いである。前項で紹介したDataプロパティと同様にTFDMemTableで書き出すデータとTClientDataSetで書き出すデータの内容では大きく異なるため互換性は無いが、TFDMemTableではXML形式に加えてJSON形式でも出力することができる【図5-1～3】。TClientDataSetでXMLデータを出力した場合と同様に、これらを他のプログラムやデータベースで流用することが可能である。

**図 4　TFDMemTable保存に必要なコンポーネント**

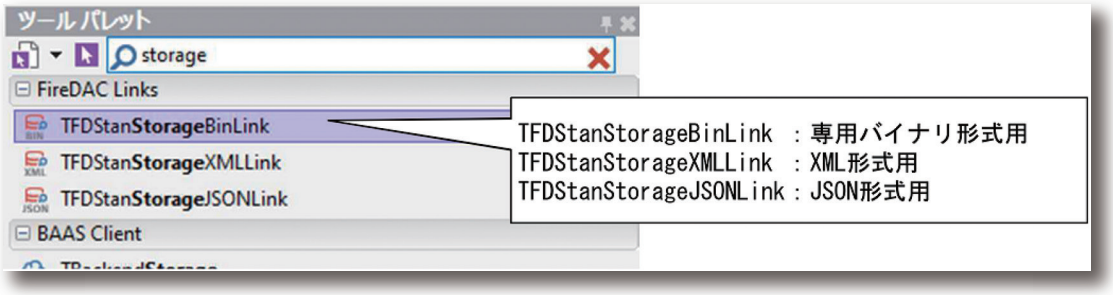
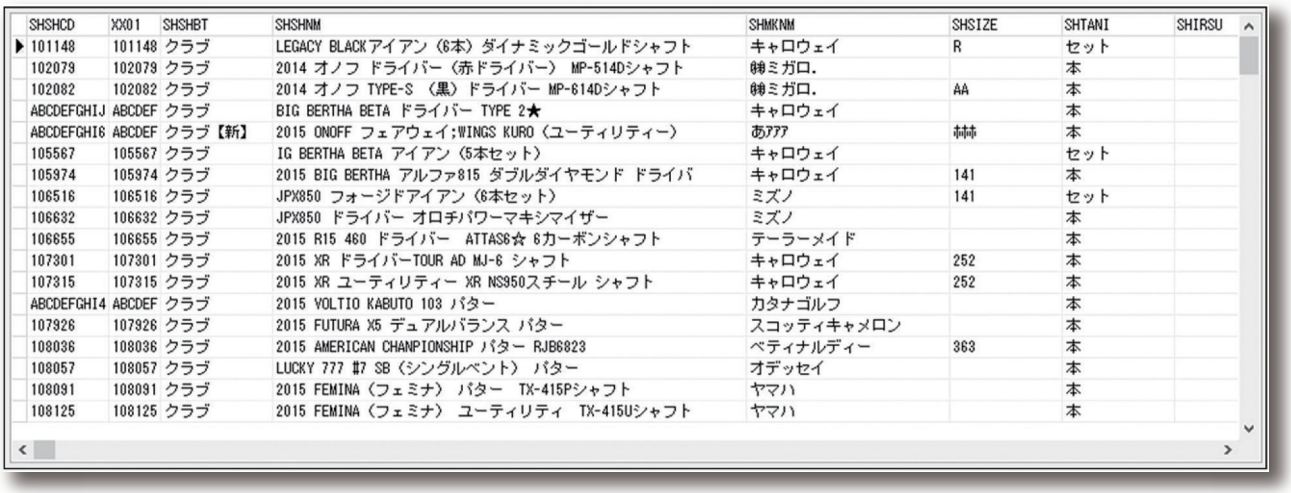
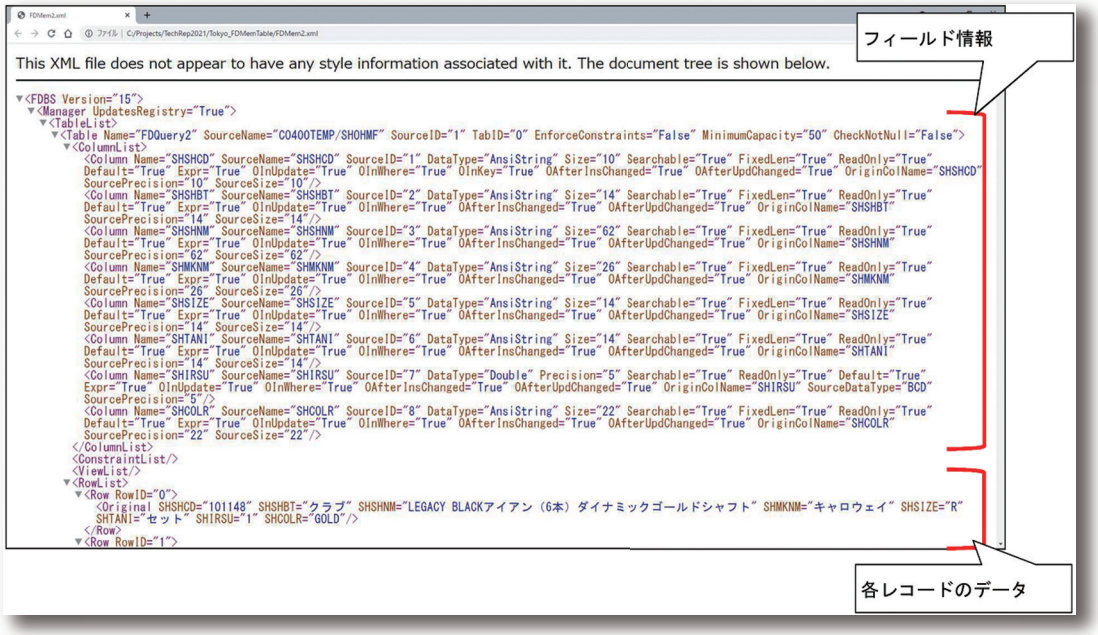



**図 5-1　TFDMemTableの出力前データ**

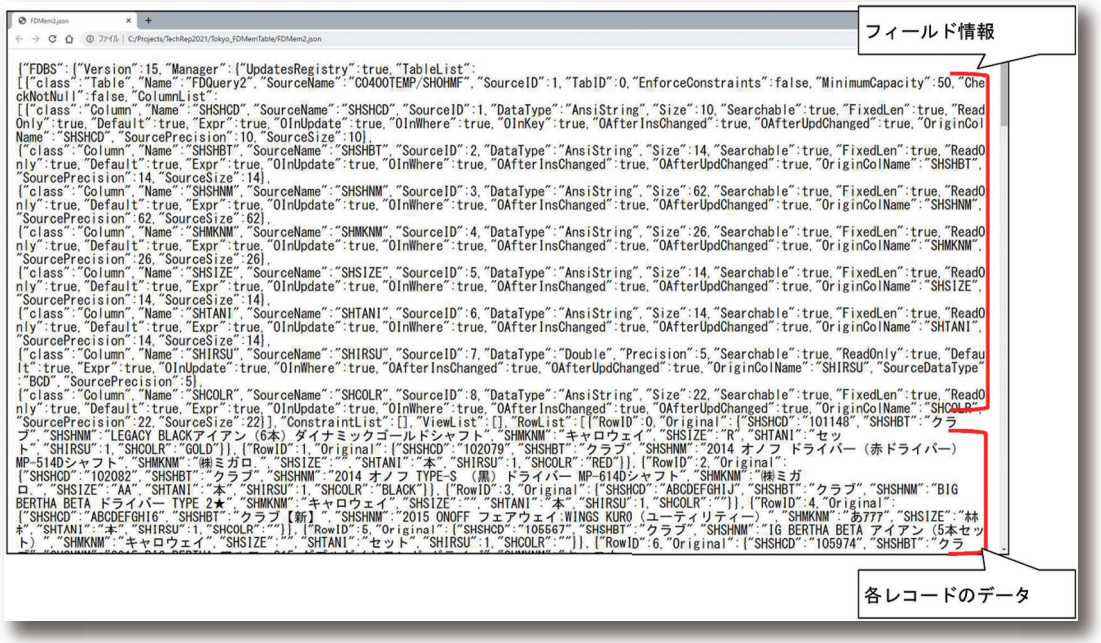


| SHSHCD | XX01 | SHSHBT | SHSHNM | SHMKNM | SHSIZE | SHTANI | SHIRSU |
|---|---|---|---|---|---|---|---|
| 101148 | 101148 | クラブ | LEGACY BLACKアイアン（6本）ダイナミックゴールドシャフト | キャロウェイ | R | セット | |
| 102079 | 102079 | クラブ | 2014 オノフ ドライバー（赤ドライバー） MP-514Dシャフト | ﾀﾏﾐｶﾞﾛ. | | 本 | |
| 102082 | 102082 | クラブ | 2014 オノフ TYPE-S （黒）ドライバー MP-614Dシャフト | ﾀﾏﾐｶﾞﾛ. | AA | 本 | |
| ABCDEFGHIJ | ABCDEF | クラブ | BIG BERTHA BETA ドライバー TYPE 2★ | キャロウェイ | | 本 | |
| ABCDEFGHI6 | ABCDEF | クラブ【新】 | 2015 ONOFF フェアウェイ;WINGS KURO（ユーティリティー） | ｱｱｱ | ﾀﾀﾀ | 本 | |
| 105567 | 105567 | クラブ | IG BERTHA BETA アイアン（5本セット） | キャロウェイ | | セット | |
| 105974 | 105974 | クラブ | 2015 BIG BERTHA アルファ815 ダブルダイヤモンド ドライバ | キャロウェイ | 141 | 本 | |
| 106516 | 106516 | クラブ | JPX850 フォージドアイアン（6本セット） | ミズノ | 141 | セット | |
| 106632 | 106632 | クラブ | JPX850 ドライバー オロチパワーマキシマイザー | ミズノ | | 本 | |
| 106655 | 106655 | クラブ | 2015 R15 460 ドライバー ATTAS6☆ 6カーボンシャフト | テーラーメイド | | 本 | |
| 107301 | 107301 | クラブ | 2015 XR ドライバーTOUR AD MJ-6 シャフト | キャロウェイ | 252 | 本 | |
| 107315 | 107315 | クラブ | 2015 XR ユーティリティー XR NS950スチール シャフト | キャロウェイ | 252 | 本 | |
| ABCDEFGHI4 | ABCDEF | クラブ | 2015 VOLTIO KABUTO 103 パター | カタナゴルフ | | 本 | |
| 107926 | 107926 | クラブ | 2015 FUTURA X5 デュアルバランス パター | スコッティキャメロン | | 本 | |
| 108036 | 108036 | クラブ | 2015 AMERICAN CHANPIONSHIP パター RJB6823 | ベティナルディー | 363 | 本 | |
| 108057 | 108057 | クラブ | LUCKY 777 #7 SB（シングルベント） パター | オデッセイ | | 本 | |
| 108091 | 108091 | クラブ | 2015 FEMINA（フェミナ） パター TX-415Pシャフト | ヤマハ | | 本 | |
| 108125 | 108125 | クラブ | 2015 FEMINA（フェミナ） ユーティリティ TX-415Uシャフト | ヤマハ | | 本 | |

## 図 5-2 TFDMemTableのXML出力サンプル




## 図 5-3 TFDMemTableのJSON出力サンプル




これ以外にTFDMemTableで実現可能な機能については、docwiki(エンバカデロ社の公式オンラインヘルプ)にも多数掲載されているので、そちらも参照されたい。

従来のTClientDataSetでもプログラムを作成することは可能だが、今後の新規開発においてはTFDMemTableコンポーネントの使用も検討いただきたい。

## 3. TSpool400に代わるSQLメインのスプール活用術

Delphi/400では、IBMiのスプールファイルを参照するためのTListSpool400・TSpool400コンポーネントが存在する。しかしこれらはBDEベースの古い技術で作成されており、最新のDelphi/400 10.2 Tokyoでも動作にはBDEが必要となっている。(Windows10ではBDEが互換での動作となるため、アプリケーションを実行するには管理者権限が必要で、かつ64bitアプリケーションには対応していない)

本章ではこのTListSpool400・TSpool400コンポーネントに代わり、FireDAC接続のSQLでスプール情報を参照する方法について紹介する。Delphi/400 10.2 TokyoにおいてはFireDAC接続も64bitアプリケーションには対応していないが、本章で記載の各SQLはdbExpress接続でも実行が可能である。64bitアプリケーションで本章のスプール活用を実施される場合は、dbExpress接続を利用して頂き、本文中の「TFDQuery」を「TSQLQuery」に読み替えていただきたい。

### ①スプールのリスト表示

従来はTListSpool400コンポーネントを使って指定したOUTQ内にあるスプールの一覧を表示し、その中にある個別のスプールの内容をTSpool400コンポーネントで表示していたが、この一連の操作をFireDAC接続のSQLで代替して行う手順を紹介する。

まず、TListSpool400コンポーネントでは「LibraryName」「OutQName」プロパティでOUTQのライブラリと名前を指定しているが、どのライブラリに目的のOUTQが存在するかわからない場合があっただろう。IBM i(V7R1以上)ではシステムライブラリ「QSYS2」内にOUTQの一覧情報を保持しており、『SELECT OUTQ, OUTQLIB, FILES, TEXT FROM QSYS2/OUTPUT_QUEUE_INFO』というSQLを実行することでその一覧を表示することができる。

(「OUTPUT_QUEUE_INFO」はビューの名前となっており、代わりに実ファイル名の「OUTQ_DTL」も指定可能である。またこのファイルにはDelphi/400が対応していない型のフィールドが存在するため、上記のフィールドのみを指定するのが望ましい。)

取得した4つのフィールドはそれぞれ

・OUTQ＝対象OUTQの名前
・OUTQLIB＝対象OUTQが格納されたライブラリ
・FILES＝現在そのOUTQに格納されているスプールの数
・TEXT＝テキスト

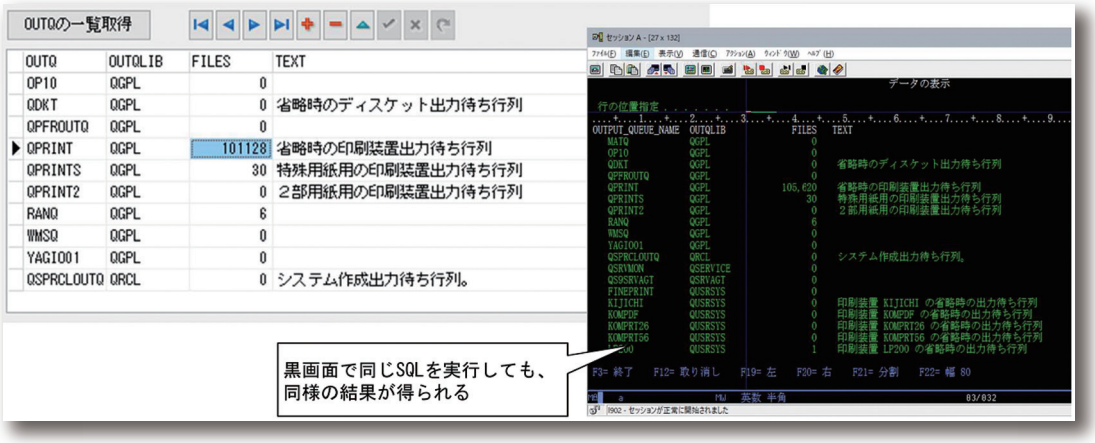
を示しているので、目的のOUTQの名前とライブラリを確認する。**【ソース6】【図6】**

**ソース 6**

IBM i 内のOUTQ一覧をSQLで取得する例(V7R1以上)

```
FDQuery1.Close;
FDQuery1.SQL.Text := 'SELECT OUTQ, OUTQLIB, FILES, TEXT FROM QSYS2/OUTPUT_QUEUE_INFO';
FDQuery1.FetchOptions.Mode := fmAll; // 一度に全件取得
FDQuery1.Open;
```

**図 6　OUTQの一覧表示**



次に指定したOUTQから個別スプールの一覧を取得する。こちらも先ほどと同様にIBM i(V7R1以上)でシステムライブラリ「QSYS2」内にスプールファイルの一覧情報を保持しており、
『SELECT SPOOLED_FILE_NAME, FILE_NUMBER, JOB_NAME,CREATE_TIMESTAMP FROM QSYS2/OUTPUT_QUEUE_ENTRIES WHERE OUTQ = (OUTQ名) AND OUTQLIB = (ライブラリ名) AND CREATE_TIMESTAMP = (作成日時)』
というSQLを実行することで、その一覧を表示することができる。**【ソース7】【図7】**

**ソース 7**

**OUTQ内のスプール一覧をSQLで取得する例(V7R1以上)**

```
FDQuery2.Close;
FDQuery2.SQL.Text := ' SELECT SPOOLED_FILE_NAME, FILE_NUMBER, JOB_NAME, CREATE_TIMESTAMP ' +
                     ' FROM QSYS2/OUTPUT_QUEUE_ENTRIES ' +
                     ' WHERE OUTQ = :OUTQ AND OUTQLIB = :OUTQLIB ' +
                     ' AND CREATE_TIMESTAMP >= :CRTDATE ';
FDQuery2.ParamByName('OUTQ').AsString      := 'QPRINT';  // OUTQ名
FDQuery2.ParamByName('OUTQLIB').AsString   := 'QGPL';    // ライブラリー名
FDQuery2.ParamByName('CRTDATE').AsDateTime := StrToDateTime('2021/4/1'); // 作成日の最小値
FDQuery2.FetchOptions.Mode := fmAll; // 一度に全件取得
FDQuery2.Open;
```

日付時刻型で渡すため、文字列にして
StrToDateTimeを行う

**図 7　スプールの一覧表示**

（「OUTPUT_QUEUE_ENTRIES」はビューの名前となっており、代わりに実ファイル名の「OUTQ_INFO」も指定可能である。またすべてのスプールの情報を取得しようとすると処理時間および処理結果の件数が膨大になるため、取得フィールドを絞るとともにWHERE句を使用することで処理時間の短縮を図っている。このほかにも、WHERE句に「JOB_NAME LIKE '%〜〜〜%'」を付加してスプールを発行したユーザー名で絞り込んだり、SQL文の末尾に「FETCH FIRST 〜〜 ROWS ONLY」を付加して取得件数を絞り込んだりするのも有効である。）

### ②個別スプールの取得

前項で取得したスプールの情報をパラメータとしてIBM iに対してCPYSPLFコマンドを発行することで、個別スプールの内容を物理ファイルに出力することができる。BDE接続を使用していた従来のTSpool400コンポーネントでも、内部的には同様のコマンドを発行してQTEMPに出力される一時ファイルを通して処理が行われていた。

CPYSPLFコマンドにはいくつかのパラメータが必要だが、それぞれ前項で取得したスプールファイル名、ジョブ名、タイムスタンプ等の各フィールド値を使用する。具体的には**【ソース8】**のように記述する。タイムスタンプは日付と時刻に分割する必要があるため、**【ソース8】**のロジックでうまくいかない場合はご利用のIBM i環境の設定値をご確認いただきたい。

また、出力先となる一時ファイルを事前に作成しておくのがポイントで、このファイルには対象スプールの桁数以上の長さを持つOタイプフィールド1つのみを定義しておく。作成方法はDDSの作成以外にも、**【図8】**のようなコマンド発行や、CREATE TABLEのSQL発行でも問題ない。この一時ファイルは本稿では便宜上通常のライブラリに作成しているが、QTEMPに作成しても問題はない。

**ソース 8**

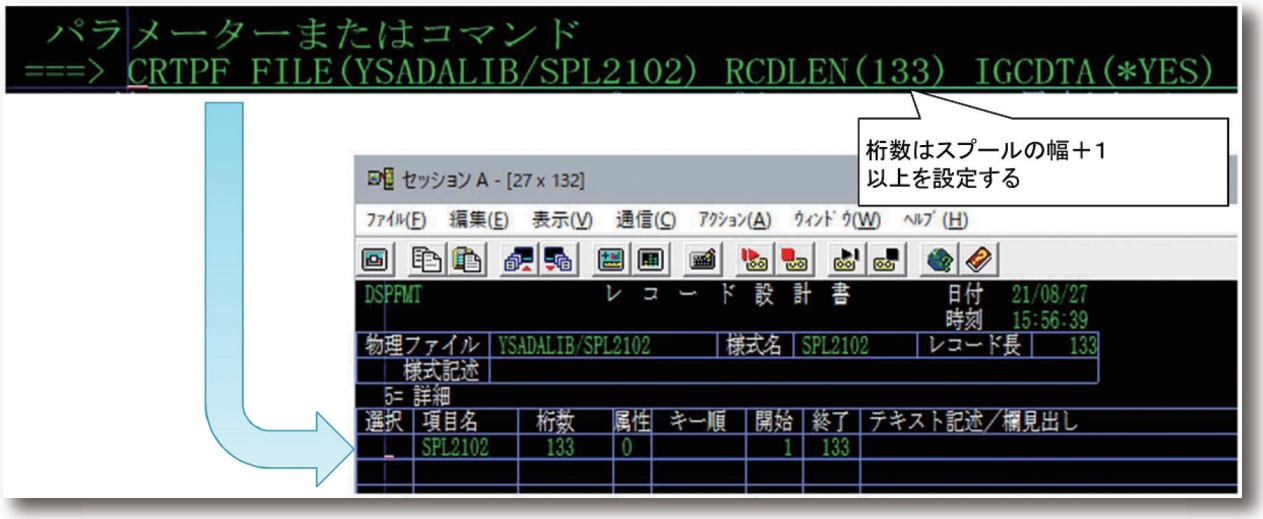
**CPYSPLFコマンドを発行して一時ファイルにスプールを出力**

```
var
  sFIL: String; // 出力先ファイル名
  sCMD: String; // コマンド文の保管
  sDAT: String; // タイムスタンプの保管
begin

  // 出力先ファイルの前回出力値をクリア（ファイルは既に存在する前提）
  sFIL := 'YSADALIB/SPL2102';
  AS400.RemoteCmd('CLRPFM FILE(' + sFIL + ')');

  // タイムスタンプの作成（書式例『'2020/02/03' '17:34:10'』）
  sDAT := Trim(FDQuery2.FieldByName('CREATE_TIMESTAMP').AsString);
  sDAT := QuotedStr(Copy(sDAT, 1, 10)) + ' ' +
          QuotedStr(Copy(sDAT, 12, 8));

  // コマンド文の作成
  sCMD := 'CPYSPLF FILE(' + Trim(FDQuery2.FieldByName('SPOOLED_FILE_NAME').AsString) + // スプールファイル名
          ') TOFILE('     + sFIL +                                                     // コピー先
          ') SPLNBR('     + Trim(FDQuery2.FieldByName('FILE_NUMBER').AsString) +       // スプールファイル番号
          ') JOB('        + Trim(FDQuery2.FieldByName('JOB_NAME').AsString) +          // ジョブ番号/ユーザー/ジョブ名
          ') CRTDATE('    + sDAT +                                                     // 作成日＋作成時刻
          ') TOMBR(*FIRST)' +  // 最初のメンバーに出力
          '  CTLCHAR(*FCFC) '; // 制御文字（改行・改ページ設定時に使用）

  // TAS400コンポーネントで作成したコマンドを発行し、一時ファイルに書き出し
  AS400.RemoteCmd(sCMD);
end;
```

【図8】で作成した一時ファイル

コマンドにはこの書式で
日付と時刻を分割して渡す必要がある

**図 8　一時ファイルを事前に作成しておく**




### ③ 取得したスプールの操作

前項のCPYSPLFコマンドで取得した個別スプールを参照するには、通常のファイルを参照する際と同様にTFDQueryで一時ファイルをSELECTする。すると、**【図9】**のように各レコードの1桁目に内部識別用の文字（ANSI用紙制御コード）を保持していることを確認できる。IBM iのスプールファイルをWRKSPLFコマンドで参照した場合や従来のTSpool400コンポーネントで参照した場合は、この制御コードによって空行や改ページの挿入を内部的に判断している。

主な制御コードはそれぞれ以下の意味を持っている。

- 「 」（スペース）：何もしない。
- 「0」：上に1行あけて出力する。
- 「-」：上に2行あけて出力する。
- 「+」：改行せずに前の行の内容を上書きする。（ただしスペース文字の場合は上書きしない）
- 「1」：改ページを行う。

これらを実装するためには**【ソース9】**のように記述する。

## ソース 9

### 一時ファイル内のスプールをTMemoに出力

```
const
  cFILE = 'YSADALIB/SPL2102'; // 一時ファイル名
var
  sANS: String; // ANSI用紙制御コードの保管用
  sTXT: String; // 制御コードを除いた文字列

  i: Integer;       // For文用
  sBEF: AnsiString; // 結合先（上側）の行の文字列
  sAFT: AnsiString; // 結合元（下側）の行の文字列
begin

  // 内容をメモに表示
  FDQuery3.Close;
  FDQuery3.SQL.Text := ' SELECT * FROM ' + cFILE;
  FDQuery3.FetchOptions.Mode := fmAll; // 一度に全件取得
  try
    FDQuery3.Open;
    Memo1.Lines.Clear;

    while not FDQuery3.Eof do
    begin
      sTXT := FDQuery3.Fields[0].AsString;

      // 整形しないでそのまま出力する場合
      if not(chkSEIKEI.Checked) then
      begin
        Memo1.Lines.Add(sTXT);
      end

      // 整形する場合
      else
      begin
        // 1桁目＝ANSI用紙制御コード
        sANS := Copy(FDQuery3.Fields[0].AsString, 1, 1);
        // 2桁目以降＝制御コードを除いた文字列
        sTXT := Copy(sTXT, 2, Length(sTXT));

        // 0 ＝上に1行あけて出力する
        if (sANS = '0') then
        begin
          Memo1.Lines.Add('');
          Memo1.Lines.Add(sTXT);
        end

        // - ＝上に2行あけて出力する
        else
        if (sANS = '-') then
        begin
          Memo1.Lines.Add('');
          Memo1.Lines.Add('');
          Memo1.Lines.Add(sTXT);
        end

        // + ＝改行せずに前の行の内容を上書きする
        else
        if (sANS = '+') then
        begin
          // 結合先（上側）の行の文字列
          sBEF := AnsiString(Memo1.Lines[Memo1.Lines.Count - 1]);
          sBEF := AddSISO(sBEF); // シフト文字代替スペースの付加【ソース10】
          // 結合元（下側）の行の文字列
          sAFT := AnsiString(sTXT);
          sAFT := AddSISO(sAFT); // シフト文字代替スペースの付加【ソース10】
          // 結合先（上側）が短い場合は結合元（下側）と長さを合わせる
          if (Length(sBEF) < Length(sAFT)) then
          begin
            sBEF := sBEF + StringOfChar(' ', Length(sAFT) - Length(sBEF));
          end;
          // バイト単位で文字を上書きする
          for i := 1 to (Length(sAFT)) do
          begin
            if (sAFT[i] <> ' ') then
            begin
              sBEF[i] := sAFT[i];
            end;
          end;

          // 結合後の文字列をメモの最下行に再セット
          sBEF := RmvSISO(sBEF); // シフト文字代替スペースの除去【ソース11】
          Memo1.Lines[Memo1.Lines.Count - 1] := sBEF;
        end

        // 1 ＝改ページを行う
        else
        if (sANS = '1') then
        begin
          // 改ページ時の処理は要件によって異なるため割愛
          Memo1.Lines.Add('★★ここで改ページ★★');
          Memo1.Lines.Add(sTXT);
        end

        // スペース＝何もしない
        else
        begin
          Memo1.Lines.Add(sTXT);
        end;
      end;

      FDQuery3.Next; // 一時ファイルの次の行へ
    end;

  finally
    FDQuery3.Close;
  end;
```

ファイルの内容を順次Memo1に書き出す

1桁目と2桁目以降を分割する

1桁目＝「0」の場合の処理

1桁目＝「-」の場合の処理

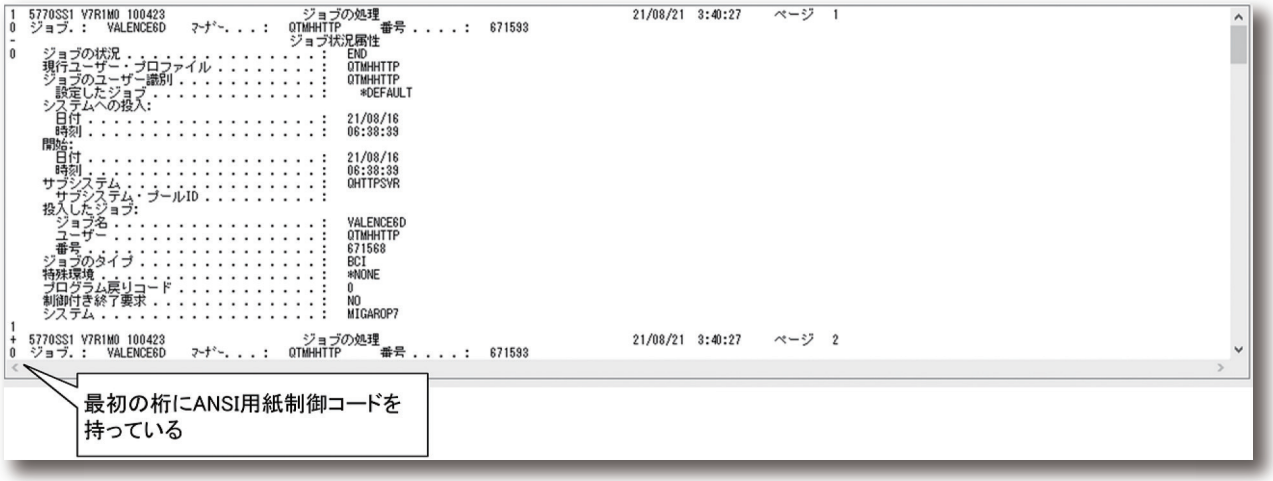
1桁目＝「+」の場合の処理

1桁目＝「1」の場合の処理

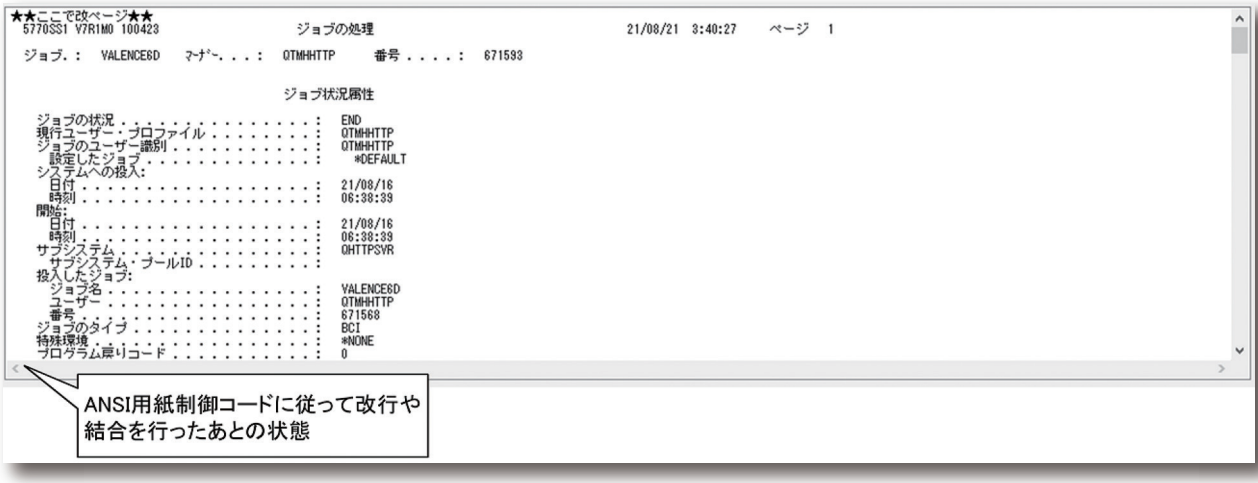
1桁目＝それ以外の場合の処理

すると、【図9】のように取得されていたデータが【図10】のように整形される。

## 図 9 整形前のスプール内容イメージ




## 図 10 整形済みのスプール内容イメージ



「+」の制御時のみDelphi側で実装する手順が少々複雑で、IBM i側のシフト文字を考慮して文字列の結合を行う必要がある。具体的には、結合元と結合先の文字列の中の全角文字と半角文字の境界に疑似的にシフト文字の代替となるスペースを付加【ソース10】し、結合完了後には逆に全角文字と半角文字の境界にセットされた疑似的なスペースを除去する【ソース11】。また、【ソース9】のロジックで結合を行っている箇所はバイト単位で結合する必要があるため、AnsiStringにキャストしている。この結果、2バイト(全角)文字の1バイト目または2バイト目だけが結合対象となった場合は文字化けを起こすことがあるが、IBM i側でWRKSPLFコマンド等を使ってスプールの内容を参照しても同様の結果になる。

**ソース 10**

**シフト文字代替スペースの付加を行う関数の例**

```
{*******************************************************************************
 目的: 文字列に対して、SI/SOの位置にダミーの半角スペースをセット
       (※uses節にSystem.AnsiStringsが必要)
 引数: 元の文字列     戻値: 整形された文字列
*******************************************************************************}
function AddSISO(AStr: AnsiString): String;
var
  i: Integer;
  S: AnsiString;
begin
  // (初期値) 1バイト目が全角の場合
  if (System.AnsiStrings.ByteType(AStr, 1) = mbLeadByte) then
    S := ' '
  // (初期値) 1バイト目が半角の場合
  else
    S := '';

  // 文字列を確認し、全角と半角の切替ポイントにダミーの半角スペースをセット
  for i := 1 to Length(AStr) do
  begin
    S := S + AStr[i];

    if (System.AnsiStrings.ByteType(AStr, i) = mbSingleByte) and
       (System.AnsiStrings.ByteType(AStr, i + 1) = mbLeadByte) then
    begin
        S := S + ' ';
    end;

    if (System.AnsiStrings.ByteType(AStr, i) = mbTrailByte) and
       (System.AnsiStrings.ByteType(AStr, i + 1) = mbSingleByte) then
    begin
        S := S + ' ';
    end;
  end;

  Result := String(S); // 結果を返却
end;
```

**ソース 11**

**シフト文字代替スペースの除去を行う関数の例**

```
{*****************************************************************************
 目的: 文字列に対して、SI/SOの位置にセットしたダミーの半角スペースをカット
 引数: 元の文字列(※SI/SO考慮済みの文字列が入る前提)　戻値: 整形された文字列
*****************************************************************************}
function RmvSISO(AStr: WideString): String;
var
  i: Integer;
  bSO: Boolean;
begin
  Result := '';
  bSO    := False;
  if AStr <> '' then
  begin
    // 文字単位で後ろからカウント
    for i := Length(AStr) downto 2 do
    begin
      // 1つ後の文字でフラグセットされた半角スペース(SO)のとき、セットしない
      if (bSO) then
        bSO := False
      else
      // 半角スペース(SI)かつ1つ前の文字が全角のとき、セットしない
      if (AStr[i] = ' ') and (Length(AnsiString(AStr[i - 1])) = 2) then
      begin  end  // 何もしない
      else
      begin
        // 文字をセット
        Result := AStr[i] + Result;
        // 全角かつ1つ前の文字が半角スペース(SO)のとき、フラグセット
        if (Length(AnsiString(AStr[i])) = 2) and (AStr[i - 1] = ' ') then
          bSO := True;
      end;
    end;

    // 全角始まりでない場合は1文字目(SIでない)を最後に足す
    if (not bSO) then
      Result := AStr[1] + Result;
  end;
end;
```

なお、制御コードを全く使用せずにただスプールの内容を順次参照するだけでよい場合は、**【ソース8】**のCPYSPLFコマンド発行時に『CTLCHAR(*FCFC)』パラメータを除外すれば、ANSI用紙制御コードが無い状態の文字列を取得できる。状況に合わせて使い分けていただきたい。

## 4.まとめ

Delphi/400では過去バージョンでコーディングされたプログラムや古くから存在するIBM iの資産を活用できることが大きな利点ではあるが、Windowsや周囲の環境の技術革新に伴ってDelphi/400も進歩を続けており、特にアプリケーションを新規開発する場合は最新のFireDAC接続を活用した方が得策であると言える。

本稿の記述を参考に、最新バージョンのDelphiならびにDelphi/400に関心をもっていただければ幸いである。

# SmartPad4i

# 洗練されたUIデザインを簡単に実現！ HTML作成テクニック

株式会社ミガロ.
RAD事業部技術支援課
**國元 祐二**

**略　歴**

生年月日：1979年3月27日
最終学歴：2002年　追手門学院大学 文学部アジア文化学科卒業
ミガロ入社年月：2010年10月　株式会社ミガロ.入社
社内経歴：2010年10月　RAD事業部配属

**現在の仕事内容：**

SmartPad4i（JC/400）、Business4Mobile、
Valenceの製品試験やサポート業務、
導入支援などを行っている。



## 1.はじめに

システム開発に携わっていれば、頻繁にUIという単語を聞くのではないだろうか？UIはユーザーインターフェース（User Interface）の略称で、利用者（ユーザー）と機械（アプリケーション）の間でやりとりするための接点（インターフェース）を指す。

UIは、アプリケーションの印象や使い勝手の良さをユーザーが判断する大きな要素になるため、非常に重要だ。UIについては様々な考えがあるが、直感的に操作できるUIが良いUIだと考える。

筆者は最近、スマートフォンを買い替えた。昔は分厚い取扱い説明書が付いてきたものだが、そのスマートフォンには取扱説明書がついてなかった。しかし、取扱説明書がなくても操作や扱いに困ることはない。直感的に操作ができるようなUIなので、スマートフォンには取扱説明書を含めていないのだと思う。

機器だけでなく、一般的に利用される、アプリケーションも同様で、普段日常的に利用しているメール、SNSや動画配信サービス等のサービスも直感的に利用できるようにUIが設計されていることが多い。

弊社では、SmartPad4iというIBMiや5250アプリケーションのGUI化が行える開発ツールを販売している。

SmartPad4iで作成するアプリケーションのUIも直感的に操作できることが理想だと思う。

## 2. SmartPad4iでの開発

IBM iプログラム開発者がWebアプリケーションを作成するための開発ツールとして、SmartPad4iは非常に優れている。SmartPad4iの開発では、IBM iプログラム開発者は5250アプリケーション開発と同様に、IBM i側のファイルや業務ロジックをIBM iプログラム（RPG/COBOL/CLプログラム）で作成するが、5250アプリケーション開発でインターフェースとなる画面ファイル（DDS）の代わりにHTMLでインターフェースを作成する。
HTMLは簡単な言語のため、シンプルな画面を作成する場合、開発者は簡単に画面を作成できる。また、SmartPad4iで使用するHTMLはCSSやJavaScriptなどでカスタマイズできるため、自由度が高く様々な機能を追加することもできる。

最近の開発では、PCブラウザだけでなく、スマートデバイスの表示にも対応するため、デバイスのサイズに依存しない、レスポンシブデザインが必要になる場合も多い。レスポンシブデザインは一つのHTMLで、PCブラウザやスマートデバイス（スマートフォン、タブレット）のような画面サイズの異なる表示に対応するHTML作成技術だ。レスポンシブデザインに対応するには、HTMLとCSSの知識がかなり必要となる。
一から全てCSSを作成するような事をしなくても、レスポンシブデザインで、使いやすい洗練されたUIのHTMLを簡単、効率的に開発する方法を本稿で紹介する。

## 3. CSSフレームワーク

### 3.1 HTMLとCSS

HTMLは「HyperText Markup Language」の略称である。「HyperText」とは、「Textを超える」という意味で、テキスト文章だけでなく、画像やリンクも含まれる情報が、文章構造「Markup」として表現された言語「Language」という意味だ。HTMLでも多少は文章の見た目や装飾を行うことができる。しかし、一般的には、視覚的・感覚的効果を定義する手段としてCSS（Cascading Style Sheets）を利用する。

CSSはHTMLやXMLで記述された文章の体裁、表示をカスタマイズするために利用されるスタイルシート言語である。1994年10月10日にホーコン・ウィウム・リー（Håkon Wium Lie）氏によって提案された。CSSはバージョンアップごとに機能が追加されて表現の幅が広がってきている。**【図1】**

**図 1　CSSの歴史**

| CSS制定年数とバージョン | 内容 |
|---|---|
| 1996年　CSS Level1 | HTMLへ視覚的に装飾する事を目的に策定された |
| 1998年　CSS Level2 | CSS1を含む新しいプロパティが追加<br>CSS1からプロパティ数が約2倍になった |
| 2011年　CSS Level2 Revision1 | CSS level2の改訂や修正 |
| 2018年　CSS Level3 | CSS2.1を中核に新しい機能の追加改良が実施された |


CSS3では、ボーダーの線を角丸
アニメーション表示の拡張
ボックスのシャドー表示（影表示）
等様々な描画設定が追加された

CSSのプロパティ設定が増えている分、HTML作成ではレスポンシブデザインの対応も加わりCSSの記述量が多くなってきた。少ないCSSの記述でHTMLを作成する手段として、CSSフレームワークを利用する方法がある。

### 3.2 CSSフレームワークとは？

CSSフレームワークはボタンやフォームの入力要素、レイアウトなどの部品が集まったCSSパーツ集のようなものだ。実現するために、たくさんのCSS記述が必要なデザインでも、CSSフレームワークを利用すると、少ない記述で簡単、効率的にHTMLを作成することができる。

### 3.3 CSSフレームワークのメリット・デメリット

CSSフレームワークを利用するメリットには以下がある。
①デザイナーが作成するような見た目のUIを作成できる
②効率的にHTMLを作成できる
③HTMLの記述に統一感があり、チーム単位での開発に向いている
良い面ばかりのようであるが、CSSフレームワークにはもちろんデメリットもある。
①CSSフレームワークを利用するための学習コストが発生する
②CSSファイルに使われないものも含めて、フレームワーク全体のCSS設定が含まれるためファイルサイズが通常作成するCSSファイルより大きなサイズになる
③軽微なデザインの調整が難しい場合がある
（レイアウトの微調整やフレームワークの規定デザインを一部変更する場合等）

メリット、デメリットもあるCSSフレームワークであるが、デメリットを考えても、レスポンシブデザインの対応や、洗練されたデザインのUIが適用されることのメリットは大きい。洗練されたUIのHTMLを効率的に作成する手段として、CSSフレームワークは非常に有用だ。

### 3.4 CSSフレームワークの種類

CSSフレームワークは数多く存在している。インターネット検索を行えば、数多くのフレームワークを見つける事ができるはずである。CSSフレームワークでは、Twitter社が開発したbootstrapやamazonやadobeなどのソフトウェア会社が採用しているFoundationなどが有名だ。数多くのCSSフレームワークがあり、それぞれ利用方法やメリット、デメリットも異なるため、どのCSSフレームワークを利用するか迷うことも多いと思う。
数多くあるCSSフレームワークの中から、本稿ではSmartPad4iに適したCSSフレームワークとしてBulma CSSフレームワーク（https://bulma.io/）を紹介したい。

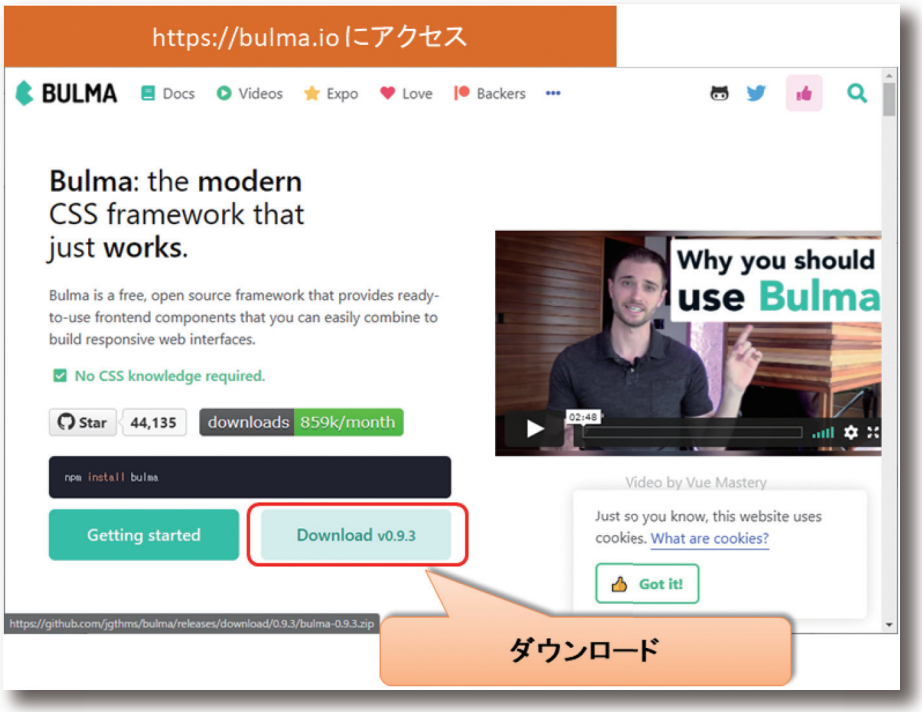
### 3.5Bulma CSSフレームワークとは？

Bulma CSSフレームワーク（Bulma）は2016年にMITライセンスの下で配布された新しいCSSフレームワークだ。Bulmaのメリットとしては、CSSの知識がなくてもデザインが可能であること、少しの学習コストで利用できる点がある。
数多くのCSSフレームワークの中でBulmaはSmartPad4iと相性が良い点が2つある。
ひとつは、CSSだけのシンプルなCSSフレームワークである点だ。通常、CSSフレームワークはフレームワークに付随するJavaScriptファイルとセットになっていることが多い。しかし、BulmaはCSSファイルだけのシンプルなCSSフレームワークのため、JavaScript部分は独自に記述することができる。
もうひとつは、通常のHTMLタグを使用している点だ。SmartPad4iではコンボボックス<select>タグやラジオボタン<input　type="radio">をHTMLタグとして利用する必要がある。CSSフレームワークによってはコンボボックスなどを（li）タグとして記述する場合もあるが、Bulmaではselectタグをそのまま利用するため、JavaScriptでタグを書き換える必要もなく、<select>タグを でラップするような、少しの追加記述だけで、SmartPad4iのコンボボックスやラジオボタンが使用できる。

### 3.6 Bulma CSSフレームワークの導入

Bulmaを利用するには、Bulma公式サイトからcssファイルをダウンロードする。**【図2】**

**図 2　Bulma.cssの入手**




ダウンロードしたzipファイルを展開後/bulma/cssフォルダにあるbulma.min.cssをWebサイトのディレクトリにコピーして読み込むだけだ。

例えば、/smartpad4i/SP4IREP21/css/bulma.min.cssにCSSを配置した場合には、HTMLへ**【ソース1】**1-①のように記述する。

**ソース 1**

**Bulma.cssの読み込み例【HTML】**

```
1  <!doctype html>
2  <html>
3  <head>
4    <meta charset="shift_jis">
5    <title>メニュー画面例</title>
6    <meta name="viewport" content="width=device-width, initial-scale=1">
7    <!-- css framework -->
8    <link rel="stylesheet" href="../../../smartpad4i/html/SP4IREP21/css/bulma.min.css">
9    <!-- user css -->
10   <link rel="stylesheet" href="../../../smartpad4i/html/SP4IREP21/css/style.css">
11   <meta name="format-detection" content="telephone=no">
12 </head>
13 <body>

～省略～
```

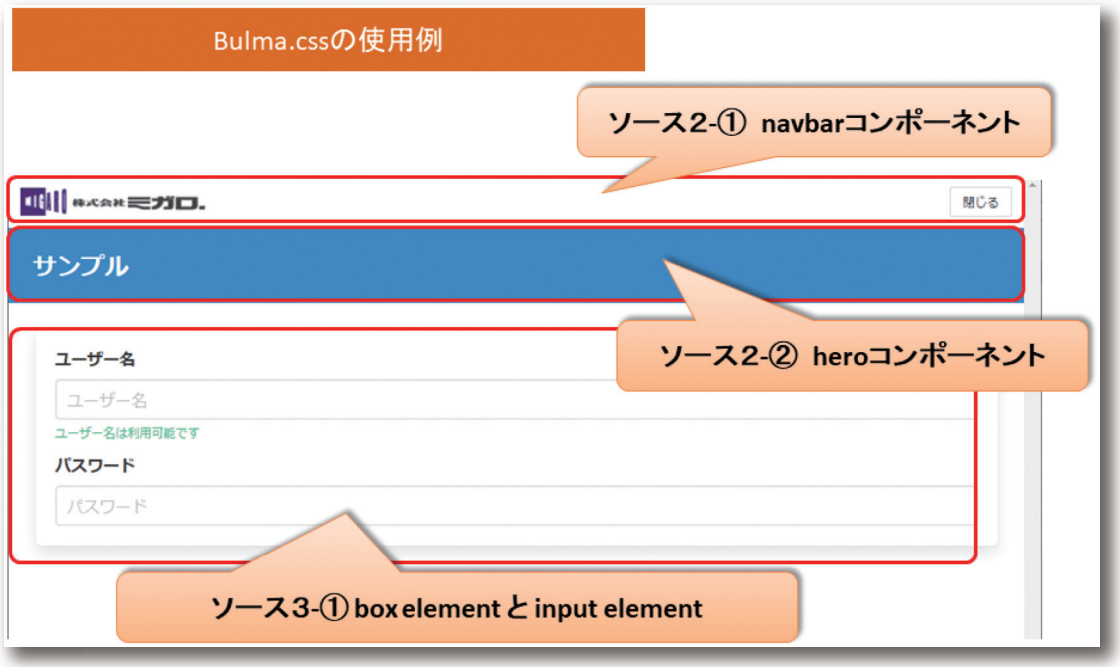
1-①

### 3.7 Bulma CSSフレームワークの使用

Bulmaはcssファイルを読込みこんだ後、HTMLの記述で要素にclass属性を設定することで使用できる。また公式サイトに様々な記述例が記載されているため、公式サイトを確認することで大抵の画面は作成可能だ。

また、Bulmaでは、部品を組み合わせるように画面を作成していくことができる。ナビゲーションメニューが作成できるnavbarコンポーネント、タイトル表示に使用できるheroコンポーネント、簡単な入力欄の要素を設定した画面例とHTMLの例が**【図3】【ソース2】【ソース3】**だ。

**図 3　画面例**



## ソース 2

**navbarコンポーネントの例 2-①【HTML】**

```
19 <!-- navbarコンポーネント -->
20 <nav class="navbar"> ----------(1)
21   <div class="navbar-brand"> ----------(2)
22     <a class="navbar-item"> ----------(3)
23       <img src="img/migaro_logo.png">
24     </a>
25   </div>
26   <div class="navbar-end"> ----------(4)
27     <a href="#" id="BTNF3" class="navbar-item"> ----------(3)
28       <button class="button is-small is-outlined"><span>閉じる</span></button>
29     </a>
30   </div>
31 </nav>
32 <!-- /navbarコンポーネント -->
```

**heroコンポーネントの例 2-②【HTML】**

```
33 <!-- heroコンポーネント -->
34 <section class="hero is-small is-info"> ----------(1)
35   <div class="hero-body"> ----------(2)
36     <h1 class="title is-4"> ----------(3)
37       <span>サンプル</span>
38     </h1>
39   </div>
40 </section>
41 <!-- /heroコンポーネント -->
```

## ソース 3

**box elementとinput elementの例 3-①【HTML】**

```
42    <!-- container -->
43    <main class="container">----------(1)
44      <div class="box mt-5">----------(2)
45        <div class="field">----------(3)
46          <label class="label">ユーザー名</label>
47          <div class="control">----------(4)
48            <input class="input" type="text" placeholder="ユーザー名">----------(5)
49          </div>
50          <p class="help is-success">ユーザー名は利用可能です</p>----------(6)
51        </div>
52        <div class="field">
53          <label class="label">パスワード</label>
54          <input class="input" type="password" placeholder="パスワード" >
55        </div>
56      </div>
57    </main>
58    <!-- /container -->
```

class属性に設定しているclass名はBulmaに定義されているclass名になる。

では、【図3】について補足する前に、【図3】を構成するために必要となる、HTMLで多用するタグについて簡単に説明する。

まず、 タグはdivisionの略で分割を意味するタグになる。 タグ単体では特別な意味を持たず、CSSを適用することで意味のあるタグに変化する。

次に、<a>タグはanchorの意味で、anchorは錨を意味する単語である。<a>タグの要素をクリックすると、指定したURLに遷移することができる。

最後に、<label>タグと<input>タグについて説明する。<label>タグは要素の項目名と入力欄やラジオボタン、チェックボックスなどの要素と関連付けを行うためのタグである。<input>タグはフォームの入力欄に使用されるタグだ。type属性を設定することで、入力欄やラジオボタン、チェックボックス、ボタン等様々な形に変化するタグである。

では、**【図3】**の説明に入る。**【図3】**の「navbarコンポーネント」は**【ソース2-①】**で構成されている。また、class属性に設定されているclass名は全て、Bulmaで定義されているclass名である。⑴では主要なナビゲーションを表すタグ<nav>タグにclass属性navbarを指定している。これにより、<nav>タグはBulmaのnavbarコンポーネントとして表示される。次行の⑵では、navbarコンポーネント内の要素を設定する。 タグにnavbar-brand class属性を設定している。navbar-brandはブランドロゴの画像を定義する場合に指定するclass名である。設定するとロゴは左側で常に表示状態となる。22行目、27行目のnavbar内のタグには⑶のようにnavbar-itemを指定する必要があり、設定するとnavbar内の要素として認識される。⑷のclass名navbar-endはnavbar内の要素を右寄せで表示するために設定している。

次に、図3の「heroコンポーネント」は**【ソース2-②】**で構成されている。heroコンポーネントはスクリーン全幅の印象的なバナーを表示することができるコンポーネントである。例えば、サイトタイトルなどを表示するのに使用できる。**【ソース2-②】**の⑴では、特定の<section>タグにclass属性heroとis-small,is-infoを指定している。class属性は半角スペース区切りでclass名を記述すると、複数のclass属性を設定することができる。<section>タグは意味や機能のひとまとまりを表す際に使用するタグである。heroと一緒に指定しているis-smallはheroコンポーネントの縦サイズとなる。例えば、is-largeと指定すれば、縦幅がより大きいバナーを表示することができる。is-infoは色の指定で、ブルーに表示するには、is-infoと指定する。例えば、is-successはグリーン、is-dangerはレッドのようにBulmaで指定された色を適用することができる。⑵のhero-bodyはheroコンポーネントの要素に指定する。⑶のtitleとis-4はタイトルの文字としてサイズ4で表示する意味になる。サイズはis-1～is-6まであり、大きい数値になると、小さい文字で表示できる。is-xを指定しない場合はデフォルトサイズのis-3が、文字サイズに適用される、少しフォントを小さくするためis-4を指定している。

最後に、**【図3】**の「box elementとinput element」は**【ソース3-①】**で構成されている。この要素の前に定義されている⑴のmainタグに設定されている、containerについて説明する。containerはレイアウトに使用される。設定されると画面サイズに合わせて横幅が自動的に切り替わるようになっている。画面サイズがFullHD（1408px）以上であれば、containerの横幅は、1344pxに、画面サイズが1216px～1407pxの場合には、containerの横幅は1152pxになり、画面サイズが1024px～1215pxの場合には横幅が960pxとなる。画面サイズが1023px以下の場合には、containerの横幅は画面サイズに合わせて100%の横幅に変化する。レスポンシブデザインに対応できるシンプルな要素である。

box elementは⑵でdivタグに指定している。boxを設定すると要素を配置できるシンプルなコンテナとなる。白い背景に影がついた見た目に変化する。フォームの入力要素を定義する場合には、⑶のようにclass属性にfieldを設定する。fieldはlabelやフォームの入力要素、help用のテキスト表示などを内包できるコンテナである。コンテナ内⑷のcontrolも単一のinput要素やselect要素、button要素、icon要素の表示を拡張するためのコンテナである。アイコンを設定する場合には、⑷のコンテナ内に追加する必要がある。⑸の<input>タグには、class属性inputを設定することで、入力欄が横幅全体に広がり、入力欄のエッジ部分が角丸に変化する。⑹はヘルプ用のテキストになる。<p>タグはParagraphの略で段落を意味するタグだ。class属性名helpを設定することで例えば、注意事項を記述、または、エラーメッセージを出力する際に利用できる。

Bulmaではclass属性名を設定するだけで簡単に要素をカスタマイズできることが分かったと思う。また、Bulmaは他の機能やリソースを組み込み、利用できる点にも魅力がある。Bulmaはフォントアイコンのライブラリにも依存しないため、自分の好みのフォントアイコンも使用可能だ。より洗練されたUIを作成するために、Bulmaと併用して利用できるWebアイコンフォントを次節で紹介する。

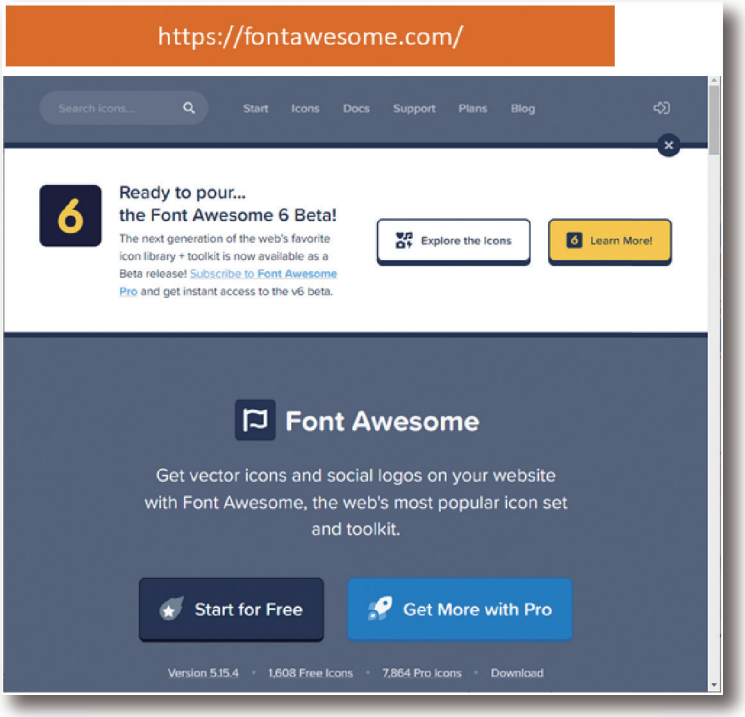
## 4. Webアイコンフォント

WebアイコンフォントはHTMLのタグで簡単にアイコンを表示できるサービスだ。画像のアイコンと異なり、拡大しても画像のように荒くなることもなく、色やサイズも簡単に変更できる。スマートデバイスのWebアプリケーションで使用されることも多くなってきていることもあり、アイコンで直感的に機能が理解できるWebアイコンフォントは、洗練されたWebアプリケーションに必須の要素だと思う。Webアイコンフォントの中で最も有名なFont Awesomeについて説明する。

### 4.1 Font Awesome

WebアイコンフォントもCSSフレームワークと同様に様々あるが、世界で最も使用されているのはFont Awesomeだ。無料プランでも1500個を超えるアイコンが利用できる。**【図4】**

**図 4　Font Awesome**



## 4.2 Font Awesomeの導入

**ソース 4**

**Font Awesomeの読み込み【HTML】**

```
1  <!doctype html>
2  <html>
3  <head>
4    <meta charset="shift_jis">
5    <title>メニュー画面例</title>
6    <meta name="viewport" content="width=device-width, initial-scale=1">
7    <link rel="apple-touch-icon" href="icon.png">
8    <!-- Font Awesome -->
9    <script src="https://use.fontawesome.com/releases/v5.3.1/js/all.js" defer ></script>
10   <!-- css framework -->
11   <link rel="stylesheet" href="../../../smartpad4i/html/SP4IREP21/css/bulma.min.css">
12   <!-- user css -->
13   <link rel="stylesheet" href="../../../smartpad4i/html/SP4IREP21/css/style.css">
14   <meta name="format-detection" content="telephone=no">
15 </head>
```

4-①

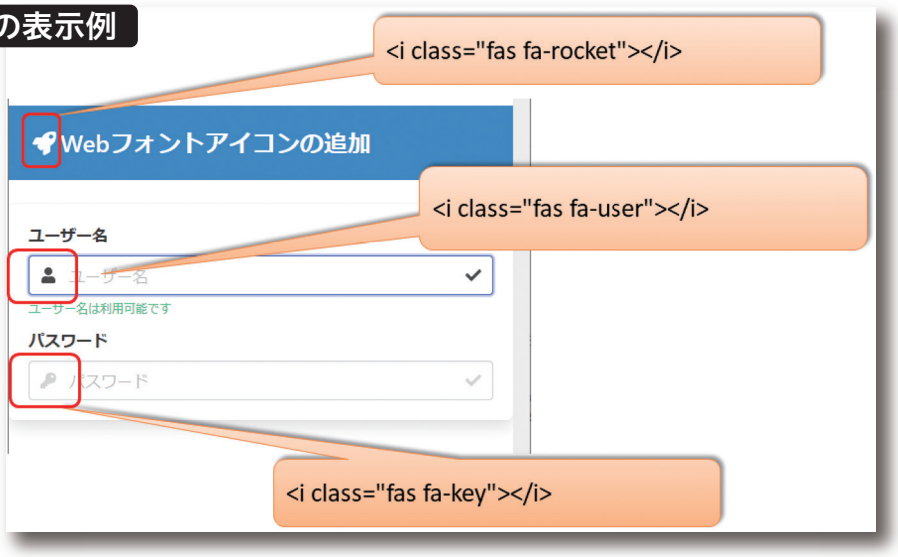
Font Awesomeを導入するのは簡単で、**【ソース4】**4-①のようにjsファイルを読み込むだけだ。

jsファイルはFont AwesomeのCDNからダウンロードできる。CDNは（コンテンツ・デリバリー・ネットワーク）の略称で、サーバのコンテンツを世界各地に分散する仕組みである。簡単に説明すると、世界中に分散されたサーバがあり、利用者は最も適切なサーバからデータをダウンロードできる。もちろん、開発者が管理するWebサーバ上にFont Awesomeのファイルを配置することもできるが、CDNを利用するとサーバの負荷を減らすことができるため、理由がなければCDNからダウンロードすることをお勧めする。

## 4.3 Font Awesomeの使用方法

Font Awesomeはタグを記述するだけで利用ができる。例えば、ロケットのフォントアイコンを表示する場合には、を追加するだけだ。**【図5】**

**図 5　Font Awesomeの表示例**



class属性に設定するfa-xxxxを変更することで様々なアイコンを表示することができる。利用可能なアイコンについては公式サイトの検索欄に入力して探すことができる。【図6】

**図 6　Font Awesome アイコン検索**

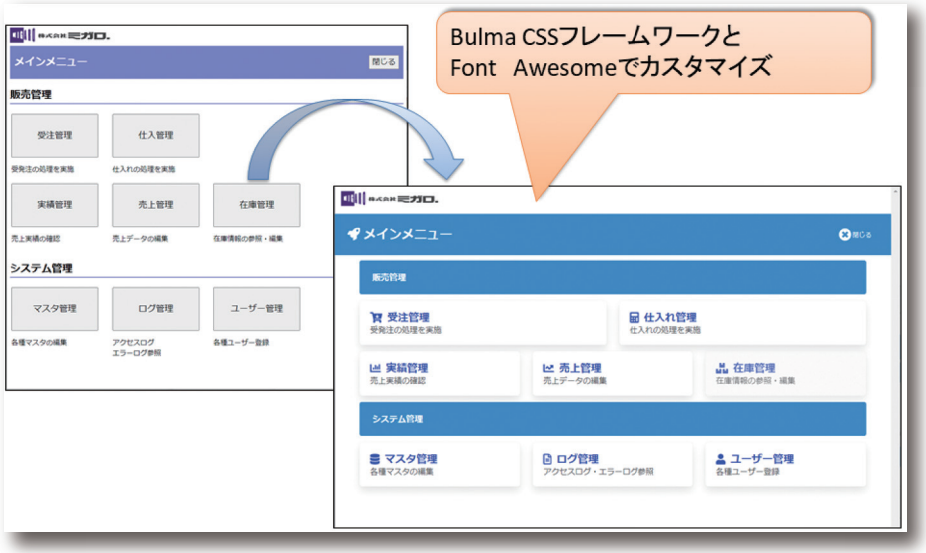

## 5.HTML作成例

BulmaとFont Awesomeを利用すると簡単に洗練されたUIのHTMLが作成可能だ。SmartPad4iの画面(HTML)を例にして説明する。

### 5.1メニュー画面の作成例

Bulmaを使用すると【図7】のように簡単に見栄えが良い画面が作成できる。また、フォントアイコン表示により、直感的にどのような機能であるかわかりやすい画面にもなる。

**図 7　メニュー画面例**



さらに、Bulmaを利用すると、HTML作成時に少し意識するだけで、レスポンシブデザインのHTMLを作成することができる。**【図8】**

**図 8　レスポンシブデザイン**


例で作成したメニュー画面は、「Tile element」で作成して、アイコンはFont Awesomeを利用した。**【図9】【ソース5】**

**図 9　Tile element**

## ソース 5

**Tile element を利用したメニュー【HTML】**

```
50    <div class="tile is-ancestor mb-0">----------------------------------------(1)
51      <div class="tile is-parent">-----------------------------------------------(2)
52        <a href="#" class="box-link" onclick="startApp(1); return false;">--------(3)
53          <article class="tile is-child box">-----------------------------------(4)
54            <p class="title has-text-link-dark">
55              <i class="fas fa-cart-arrow-down"></i>-----------------------------(5)
56              <span class="app-title">受注管理</span>
57            </p>
58            <p class="subtitle">受発注の処理を実施</p>
59          </article>
60        </a>
61      </div>
62      <div class="tile is-parent">
63        <a href="#" class="box-link" onclick="startApp(2); return false;">
64          <article class="tile is-child box">
65            <p class="title  has-text-link-dark" >
66              <i class="fas fa-calculator"></i>
67              <span class="app-title">仕入れ管理</span>
68            </p>
69            <p class="subtitle">仕入れの処理を実施</p>
70          </article>
71        </a>
72      </div>
73    </div>
```

**box-link【CSS】**

```
.box-link{
    display:flex;
    width:100%;
}
```

divタグにclass属性tile を設定するだけでTile Element を利用できる。(1),(2),(4)のclass属性tileはネストして利用することになる。補足をすると、**【ソース5】**(1)のクラス名is-ancestorは「Tile element」の祖先要素となる。ネストして定義するclass属性tileの要素をラップする要素だ。また、mb-0は要素下部の余白を0にするmargin-bottom:0pxの意味だ。mb-0からmb-6まで設定できる。詳細は公式サイトを確認してほしい。

(4)の箇所のarticleタグに設定したclass属性is-child, boxは設定したタグの子要素をboxの見た目で表示するために指定している。(5)の箇所はFont Awesomeのアイコンを表示するように記述した。

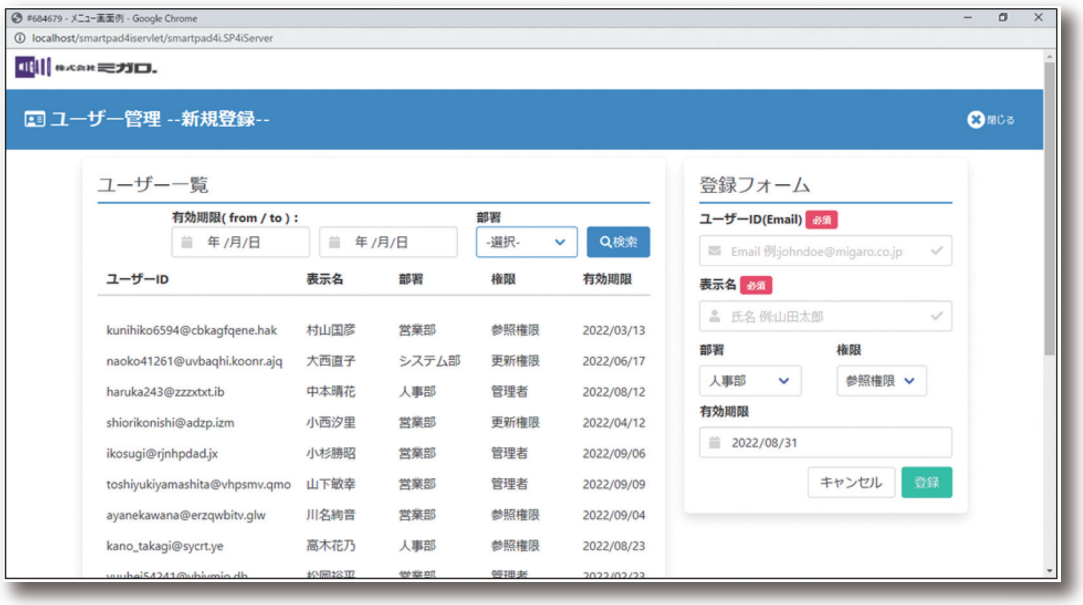
さらに、今回タイル要素をアプリケーションのリンク要素にするため、(3)の箇所にアンカータグ<a>を追加している。アンカータグのサイズ調整には、独自にbox-linkというクラスを定義した。独自に追加したCSSはこれだけで、後はBulmaの機能ですべて記述できた。Bulmaを利用することで、HTMLの見た目は洗練され、レスポンシブデザインにも簡単に対応できる。さらに、CSSを記述する時間を大幅に短縮もできた。

また、Bulmaは入力欄のカスタマイズも容易で、洗練された入力画面を作成可能だ。次節では入力画面の作成例を紹介する。

### 5.2 入力欄の作成例

入力フォームはBulmaの公式サイト例が豊富なため作成に困ることはないだろう。入力欄の例として作成したのが【図10】だ。左側に一覧、右側に入力フォームがある。

**図 10　入力欄の例**




入力欄にはclass属性名inputを設定するだけで、自動的に横幅が100%に設定され、エッジが角丸の入力欄に変化する。【図10】の入力欄はアイコン表示をカスタマイズしているため、記述の追加が必要となる。【図11】の赤枠の記述例が【ソース6】だ。

**図 11　入力欄の例**

## ソース 6

**図11の入力欄のHTML記述 【HTML】**

```
131       <div class="field">-----------------------------------------------------(1)
132         <label class="label">ユーザーID(Email)-----------------------------------(2)
133           <span class="tag is-danger">必須</span>-----------------------------(3)
134         </label>
135         <div class="control has-icons-left has-icons-right">-------------------(4)
136           <input id="INP01" class="input" type="email"
                placeholder="Email 例:johndoe@migaro.co.jp">
137           <span class="icon is-small is-left">-----------------------------------(5)
138             <i class="fas fa-envelope"></i>
139           </span>
140           <span class="icon is-small is-right">----------------------------------(6)
141             <i class="fas fa-check"></i>
142           </span>
143         </div>
144         <p class="help is-danger error" id="ERR01"></p>-----------------------(7)
145       </div>
```

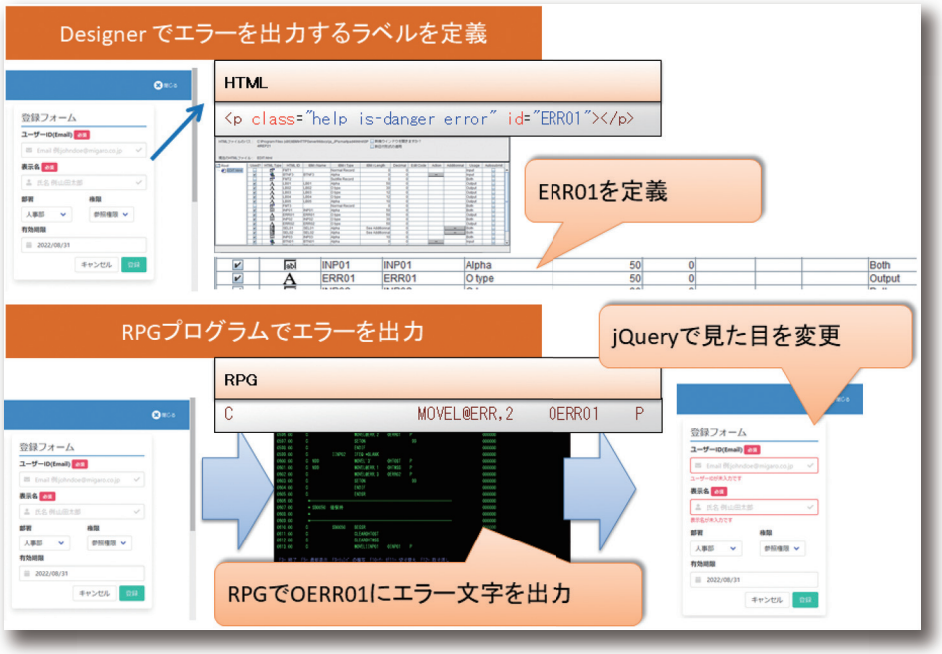
**【ソース6】**のclass属性名は全て、Bulmaで定義されている。まず、(1)のように入力欄をラップするように、 タグを記述してclass属性名にfieldを設定する。fieldは入力欄等、要素のシンプルなコンテナである。次に、(2)の箇所で、入力欄のラベルを表示するため<label>タグを追加している。<label>タグにはclass属性labelを設定する。(3)の行でラベルの横に、「必須」を表示しているが、これもclass属性名にtagを設定して、Bulmaの「tag element」の表示を使用している。一緒に設定している、is-dangerで色を赤色にできる。アイコンを表示するためには、 タグにて(4)のようにhas-icons-left has-icons-rightを追加する必要がある。もちろん、左側のみにアイコンを表示したい場合はhas-icons-leftだけで構わない。入力欄の定義の後、(5)(6)のようにspanタグでiconを定義する。is-small is-leftはis-smallでアイコンのサイズ、is-leftでアイコンの位置を指定できる。

また、(7)のid属性ERR01、<p>タグはRPG側からエラーを出力するために定義している。JavaScriptとの連携も必要となるが、未入力の場合には**【図12】**のように、わかりやすいエラーメッセージを表示することができる。

## 図 12 エラー表示例

エラー表示の仕組みは、HTML側であらかじめエラーメッセージを表示するラベル(id属性ERR01)を定義しておき、Designerで属性を設定、RPGプログラムでエラーチェックを行い、RPG側からラベルのフィールドのアウトプットフィールド(OERR01)へ文字列を設定している。**【図13】**

**図 13　エラー表示の仕組み**




エラーの文字がRPG側で出力された場合に、JavaScriptを利用して画面の見た目を変更した。

実装例のJavaScriptでは、jQueryを利用している。jQueryについては2019年度のテクニカルレポート、[SmartPad4iの インターフェース 機能拡張]に取り上げているため、ここでは割愛する。

**【図12】**のように表示するために、**【ソース6】**(7)の<p>タグにclass属性errorを追加している。エラーメッセージがRPG側で設定された場合に、JavaScriptでエラーメッセージを表示状態に変化させ、入力欄の枠を赤に変更する。JavaScriptの例は**【ソース7】**になる。

**ソース 7**

**エラー表示を実行するJavaScript例【JavaScript】**

```
<script>
  //HTMLが読み込まれた後に実行
  $(function(){----------------------------------------------------------------(1)
   //class属性 error(エラーメッセージ)の要素でループ
    $('.error').each(function(){-------------------------------------------(2)
      //エラーメッセージが出力されている場合
      if($(this).text()){-----------------------------------------------(3)
        //エラーメッセージをフェードイン表示後、入力欄に赤枠を設定
        $(this).fadeIn().prev('div').find('select, input')
                .css('border-color','#FF0000'); ----------------------(4)
      }
    });
  });
</script>
```

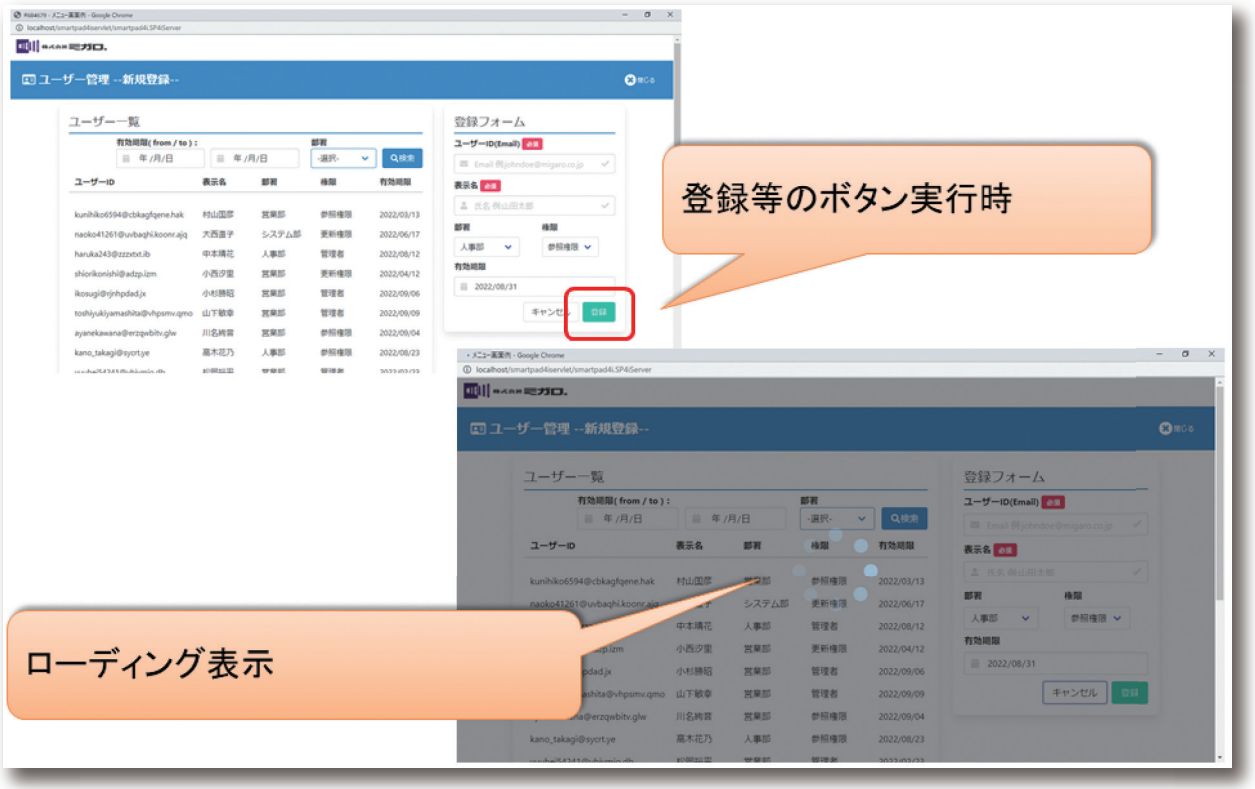
jQueryでは、HTMLが読み込まれた後に処理を実行するために、(1)のような記述を行う。(2)はclass属性名errorの要素をjQueryのセレクタ、$()で取得後、eachメソッドで取得した要素をループしている。$()はjQueryで要素を取得するための関数でセレクタと呼ばれる。

$('.error')と記述すると、class属性にerrorが設定された要素を全て取得してくれる。
(3)や(4)で使用されている、thisはeachメソッドでループしている対象の要素だ。つまり、error属性が設定された要素にエラー内容の文字列がRPGプログラムから出力されている場合には、fadeInメソッドでフェードイン表示させ、一つ前の タグ要素内にある<select>タグ、<input>タグのcss属性border-colorプロパティを赤色(#FF0000)に設定している。
SmartPad4iでのHTML画面作成にBulmaを利用すると、短時間で洗練されたUIが作成できることが分かったと思う。その他、SmartPad4iアプリケーションに簡単な実装で取り入れられる機能をご紹介したい。

## 6.お勧めのUIカスタマイズ(CSSローディング表示)

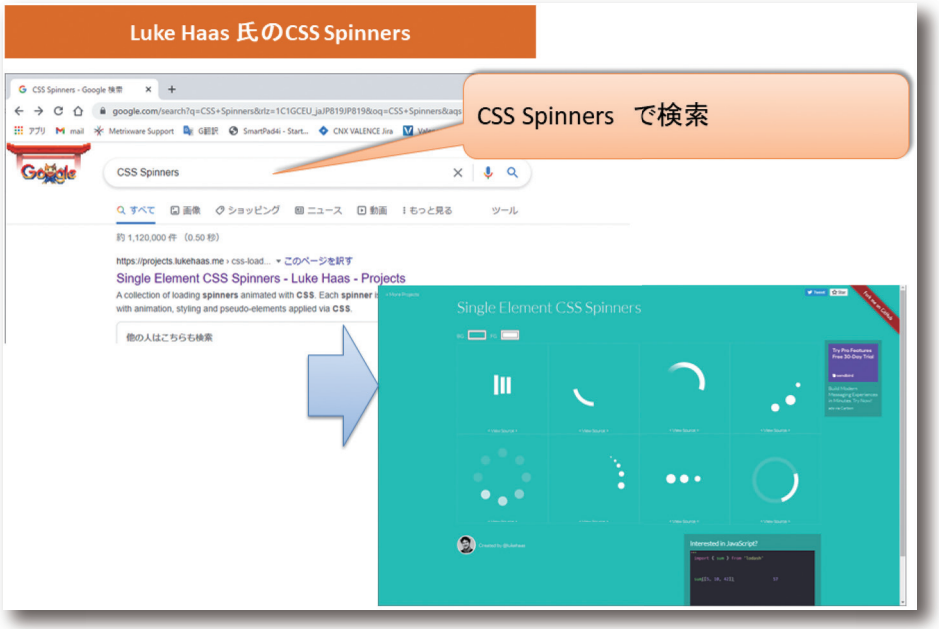
SmartPad4iをはじめとするWebアプリケーションは、「Webサーバへの要求(リクエスト)」→「Webサーバからの応答(レスポンス)」の流れで処理され、要求後に、応答を待つ必要がある。応答が返却される間の画面を工夫することで、ユーザーの体感的な待ち時間軽減を図ることができる。例えば、多用するのが、ローディング画面の表示である。【図14】次に、SmartPad4iにおいてローディング画面を表示する方法を紹介する。

図 14 ローディング表示



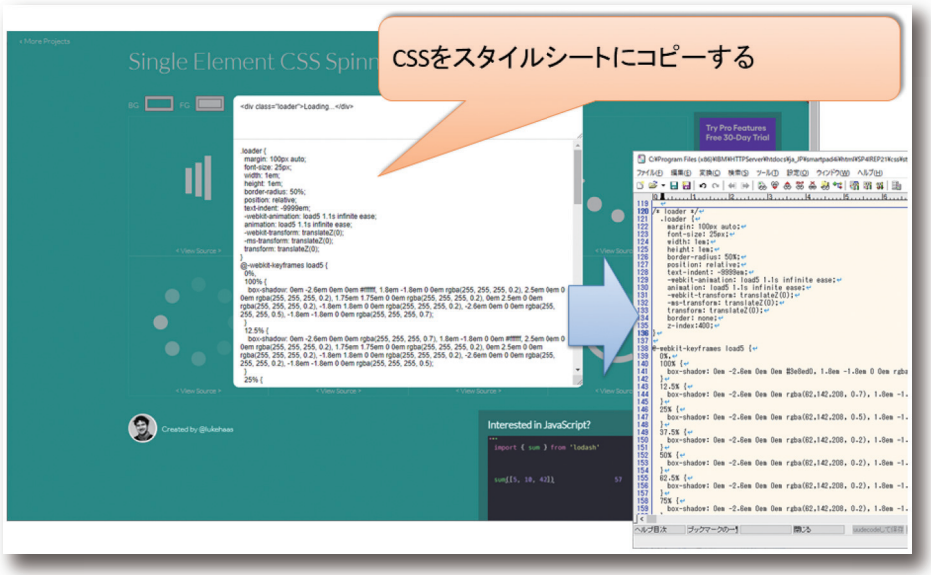
【図14】のローディング画面のアニメーションはCSSで実現している。CSSのアニメーションはLuke Haas氏のCSS Spinnersを使用した。【図15】

## 図 15 ローディングCSS Spinners




Luke Haas 氏のサイトでは動的にローディング画面のアニメーションが作成可能で、好きなデザインのローディングアニメーションを取得できる。ローディングアニメーションはCSSのコードとして所得できるため、スタイルシートに張り付けて、HTML内にタグ Loading... を追加するだけだ。【図16】

## 図 16 ローディングCSS Spinners



しかし、アニメーションはHTMLのタグを追加した箇所に表示されるため、アニメーションをラップする要素を作成して、画面中央に表示した。**【ソース8】**

**ソース 8**

**HTML追加例【HTML】**

```
16  <body>
17    <div class="loader-wrap" >
18      <div class="loader">Loading...</div>
19    </div>
```

divタグにclass名、loader-wrapを設定して
class名 loaderのタグを囲う

**CSS追加例【CSS】**

```
105  /* loaderを中央に表示*/
106  .loader-wrap {
107     position: fixed;
108     display: flex;
109     align-items: center;
110     justify-content: center;
111     width: 100%;
112     height: 100%;
113     top: 0px;
114     background-color:#000000;
115     opacity:0.5;
116     z-index:100;
117  }
```

wrapした要素のCSSを設定

**【ソース8】**の設定を行うことで、ローディング画面が画面中央に表示されるようになる。最後に、ローディング画面の表示、非表示をJavaScriptで制御する。**【ソース9】**

**ソース 9**

**ローディング画面を表示するJavaScript例【JavaScript】**

```
275  <script>
276    //HTMLが読み込まれた後に実行
277    $(function(){
         //エラーメッセージ
         ～【ソース6の箇所省略】 ～
286      //ローディングを非表示
287      $('.loader-wrap').fadeOut();
288    });
289
290    //IBMi送信時
291    function cansubmit(){
292      $('.loader-wrap').fadeIn();
293    }
294  </script>
```

HTML読み込み後、
wrapした要素をフェードアウトで非表示

WEBサーバ送信前に実行される、
SmartPad4iのcansubmit関数を利用
フェードインでローディング画面を表示

以上で、ローディング画面の実装が完了する。ローディング画面は簡単なCSS設定とJavaScriptで記述できるため、是非実装してほしい。

## 7.さいごに

本稿では、Bulma CSSフレームワークのよるUI作成とレスポンシブ対応、WEBアイコンフォント、メニュー画面や入力画面の例、ローディング表示の画面カスタマイズを紹介した。数あるCSSフレームワークの中で、Bulmaは SmartPad4i開発との相性がよいため、是非利用して洗練された、直感的に使用できるUIのアプリを作成してもらいたい。

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:**

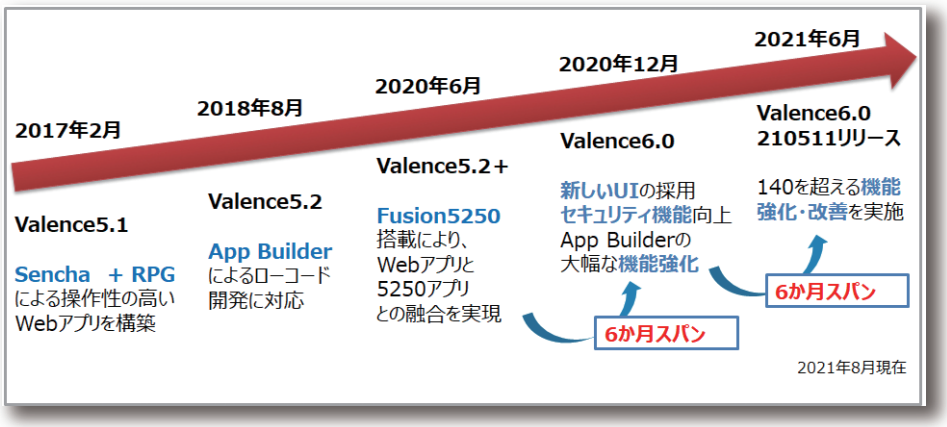
Delphi/400を中心としたテクニカルサポート対応や製品セミナーの講師などを担当している。



## 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**

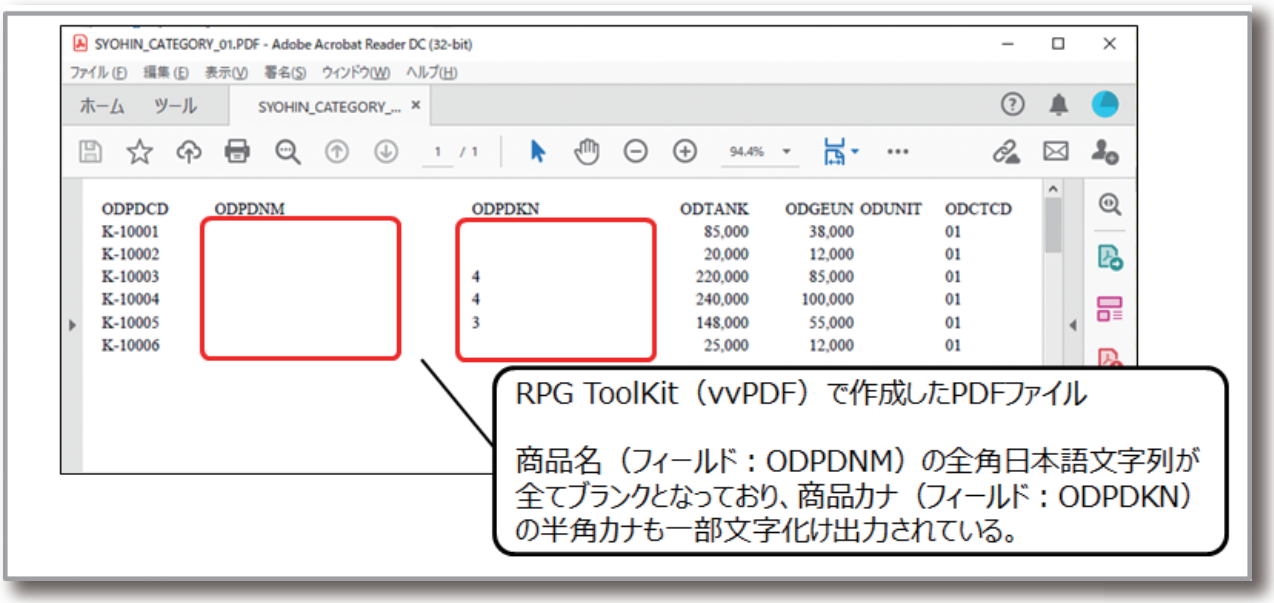
RPGコーディング例は、**【ソース1】**となる。

## 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                    1-①
0008.00      /include qcpylesrc,vvNabBtn
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';                               1-④
0045.00        vvOut.file     = 'CATEGORY_' + VCATG + '.PDF';
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**

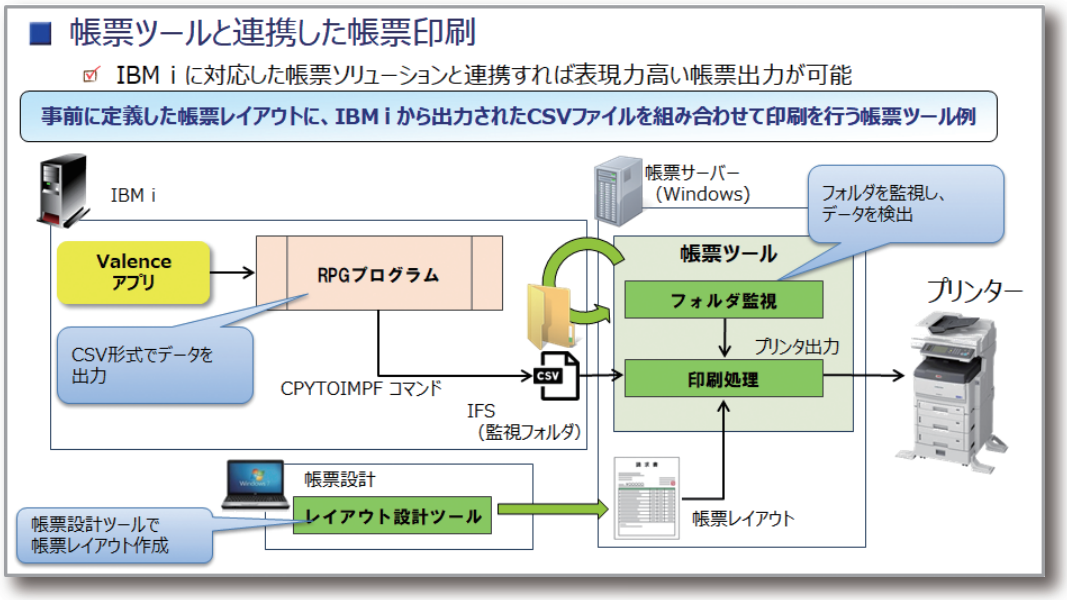
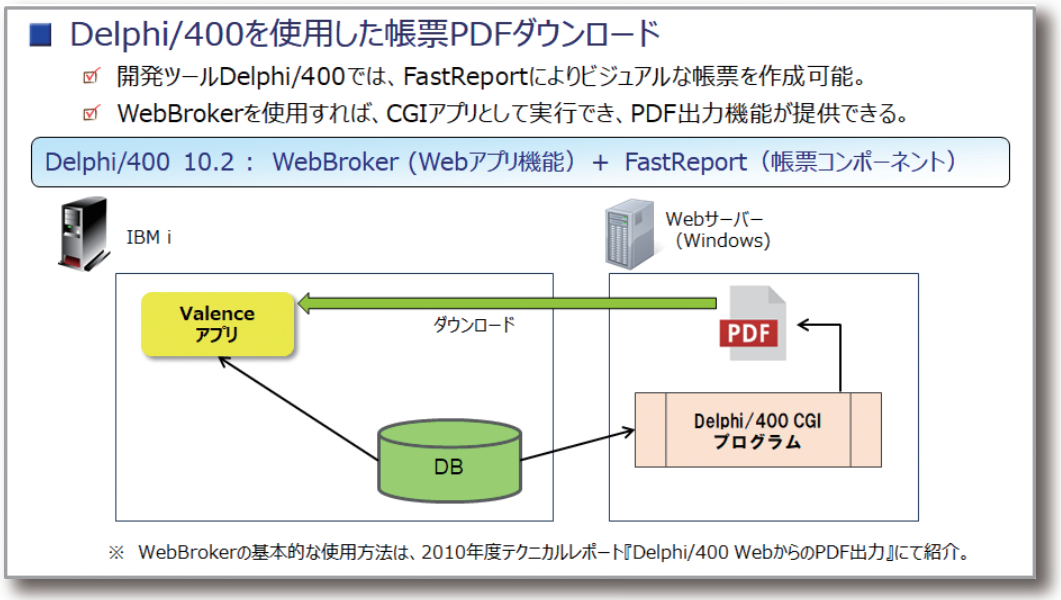



**図5　ValenceとDelphi/400との連携イメージ**



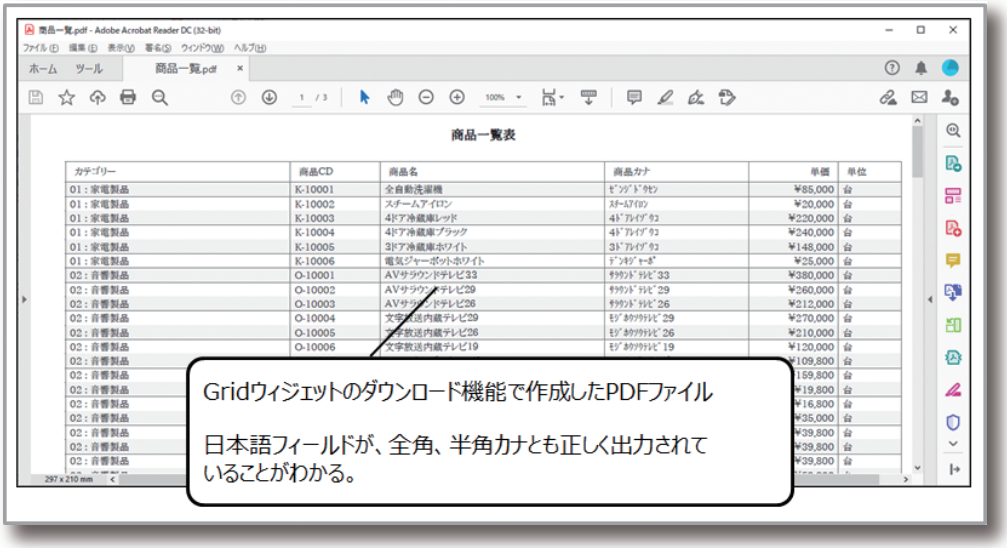
## 3. Valence6.0における新しいPDF帳票出力

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】

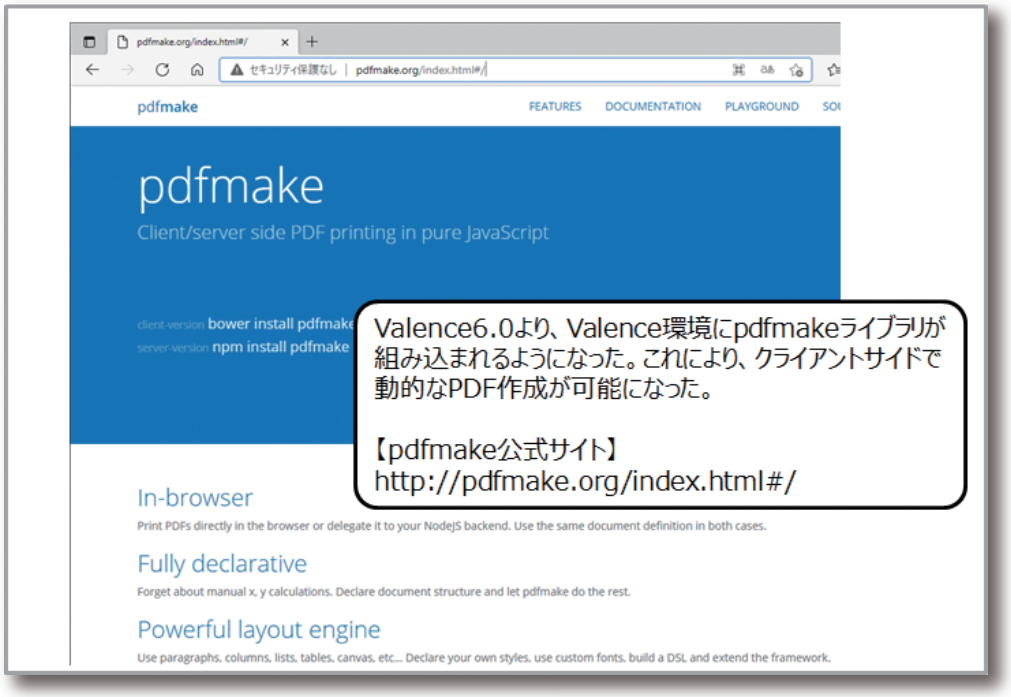
**図7　Gridウィジェット ダウンロードで生成されたPDF**



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。

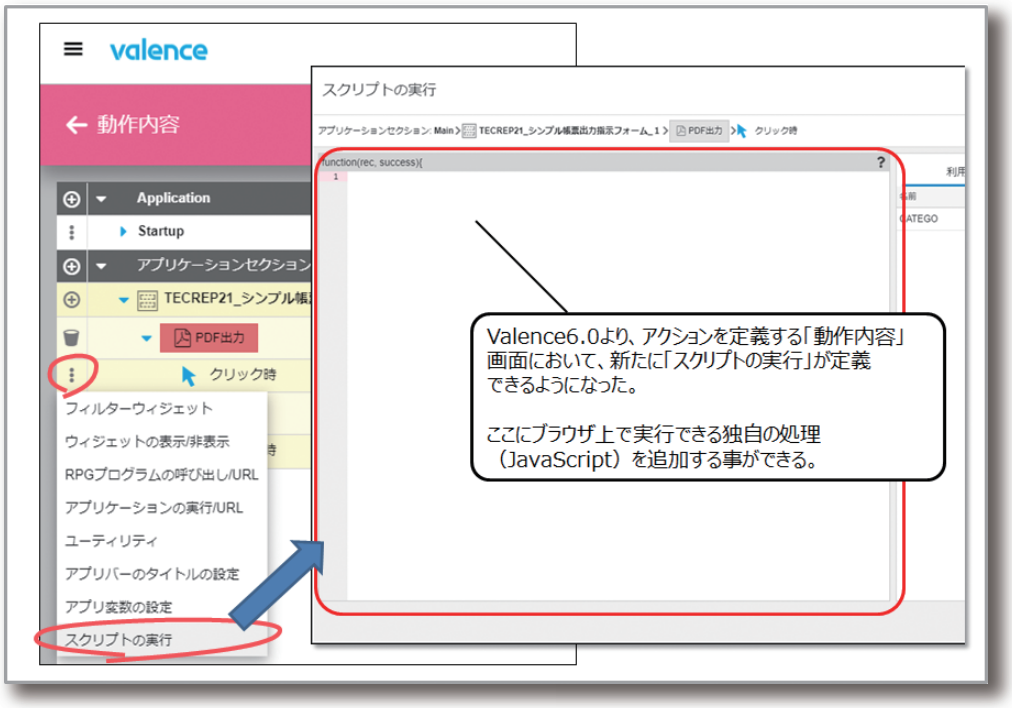
この新しいPDF出力機能をApp　Builderの標準機能として使用できるのは、2021年8月現在Grid/Edit Gridウィジェットに追加されたPDFダウンロード機能のみであるが、このライブラリはJavaScriptを使用する事で簡単にPDFを作成できる為、独自のPDF出力処理をApp Builderアプリケーションに追加する事ができる。具体的な作成方法を次節で紹介する。

Valence

## 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」**

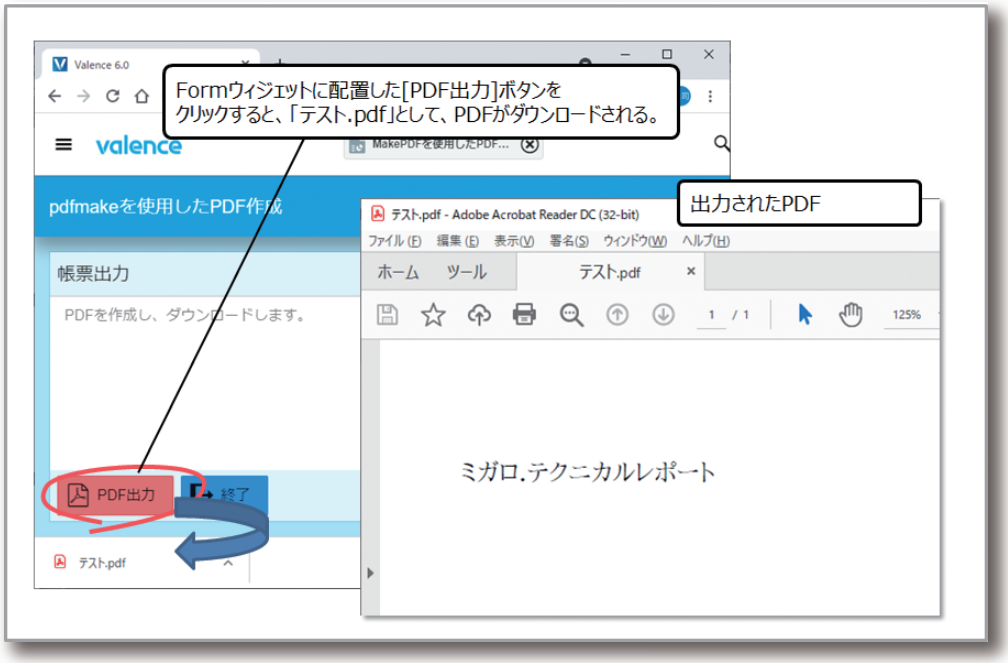



この機能を使用すれば、pdfmakeを使用した独自の帳票が作成できる。ここでは具体的な作成方法の紹介として、まず始めにシンプルなPDF出力アプリケーションを作成する。

### 4-1. シンプルなPDF出力処理の作成

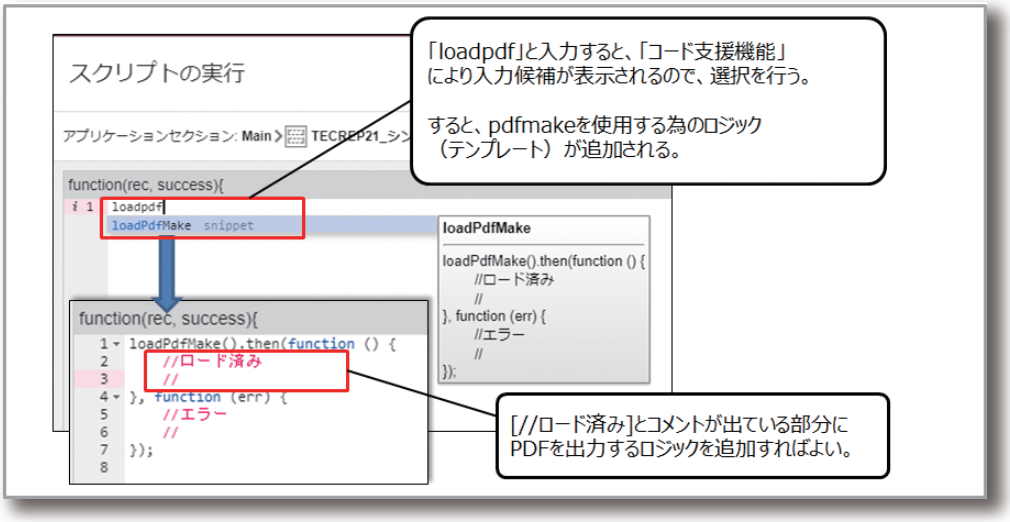
サンプルプログラム２は、Formウィジェットに配置した［PDF出力］ボタンをクリックすると、「ミガロ. テクニカルレポート」という日本語文字列を持つPDFファイルをダウンロードするというシンプルなPDF出力処理である。**【図10】**

**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。

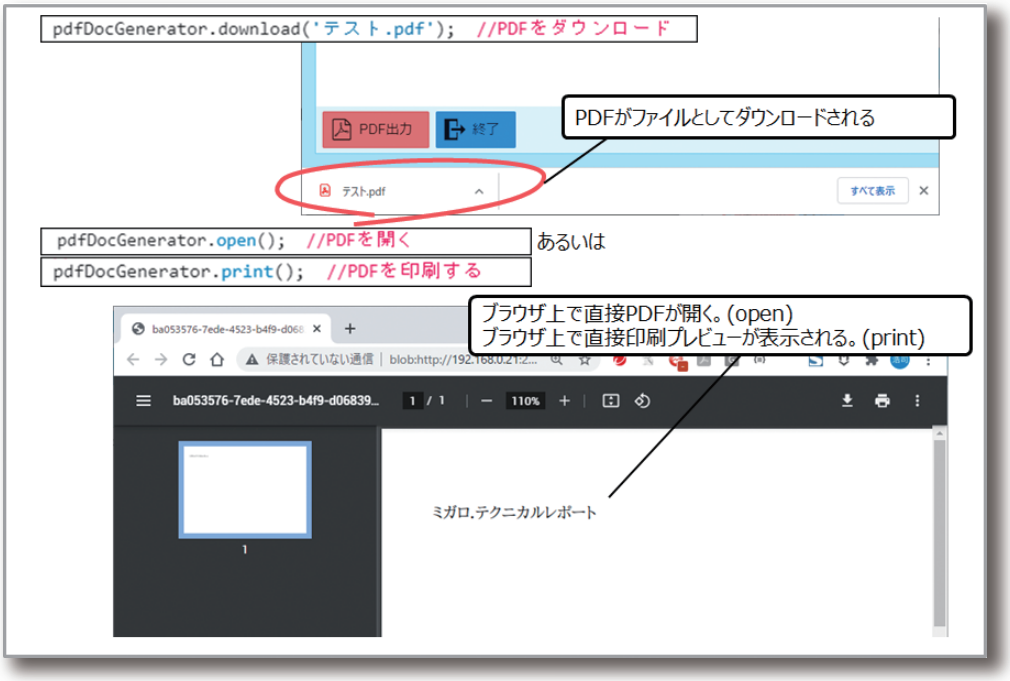
## ソース2 サンプルプログラム2:シンプルなPDF出力処理

```
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。

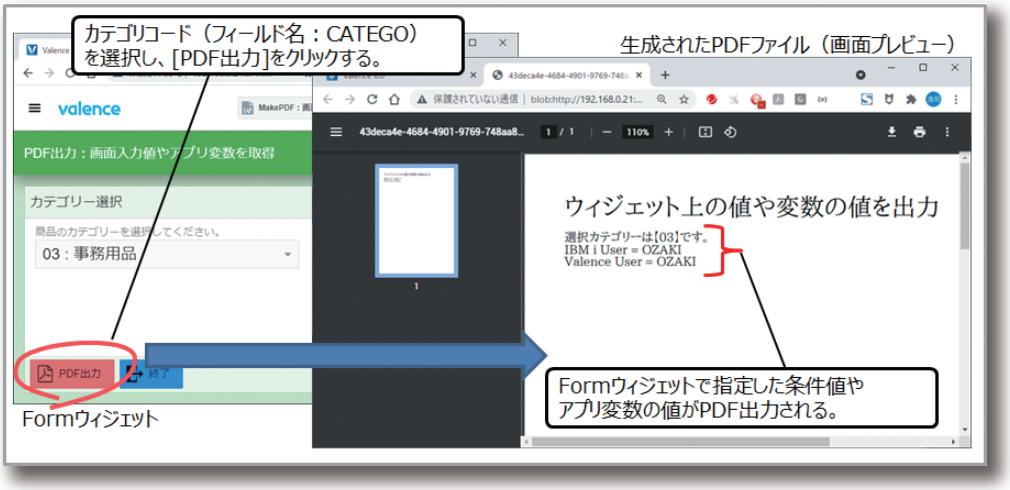
**図12　pdfmakeを使用したPDF出力方法**




### 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  });
```

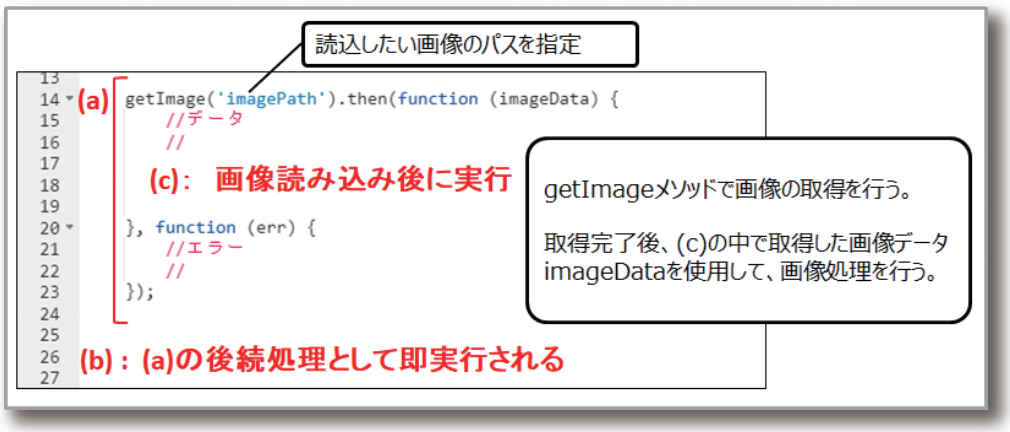
先程と同様"content"要素内でPDFレイアウトを作成しているのだが、ここでは、recというオブジェクト変数やgetAppVerというメソッドを使用していることがわかる。rec変数がウィジェット上に定義されたフィールドにアクセスする為に用意された変数で、getメソッドを使用すればウィジェット上のフィールド値を取得する事ができる。

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){
 1  //----- PDFMake を読込
 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      //---- 画像データ取得
12      getImage('/resources/images/valence_logo.png').then(function (imageData) {   4-①
13          //---- 画像取得成功
14          //---- PDFレイアウト作成
15          var docDefinition = {
16              //出力する内容
17              content: [
18                  {text: '画像の出力', fontSize: 20},
19                  {text: ' '},
20                  {image: imageData, width: 300}      // 画像      4-③
21              ],
22              //スタイル（書式）指定
23              defaultStyle: {
24                  font: 'Aozora'                      // 日本語フォント
25              }
26          };
27
28          //---- PDF生成
29          var pdfDocGenerator = pdfMake.createPdf(docDefinition);
30          //---- 生成したPDFを出力
31          pdfDocGenerator.open();     // ファイルを開く
32      }, function (err) {
33          //---- 画像取得エラー
34  });
35  }, function (err) {
36      //エラー
37  });
```

4-②

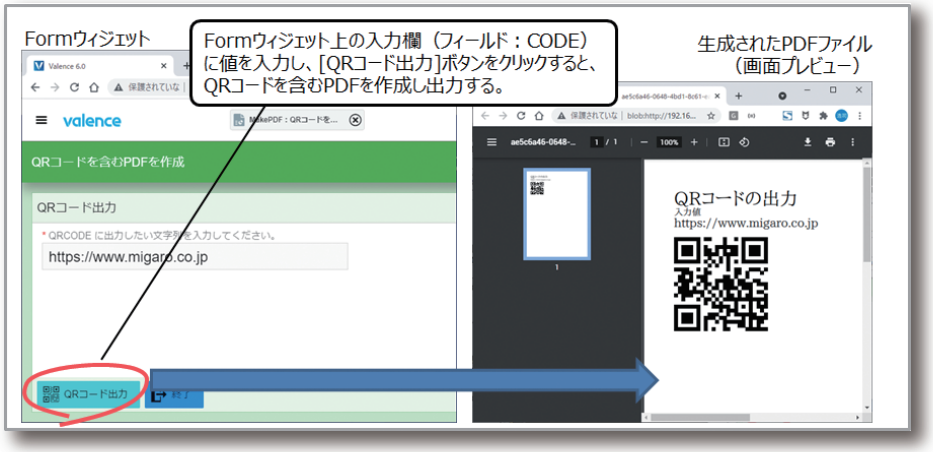
4-①が画像を読み込む部分となる。IFS上にあるValenceのロゴ画像（/resources/images/valence_logo.png）を取得している。画像が取得できれば、4-②の部分が実行される為、この部分でPDFファイルのレイアウト作成およびPDF生成を行っている。pdfmakeでは4-③のように画像項目（image）に取得した画像データ（imageData）をセットするだけで画像がPDFに出力できる。

Valence

### 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】**

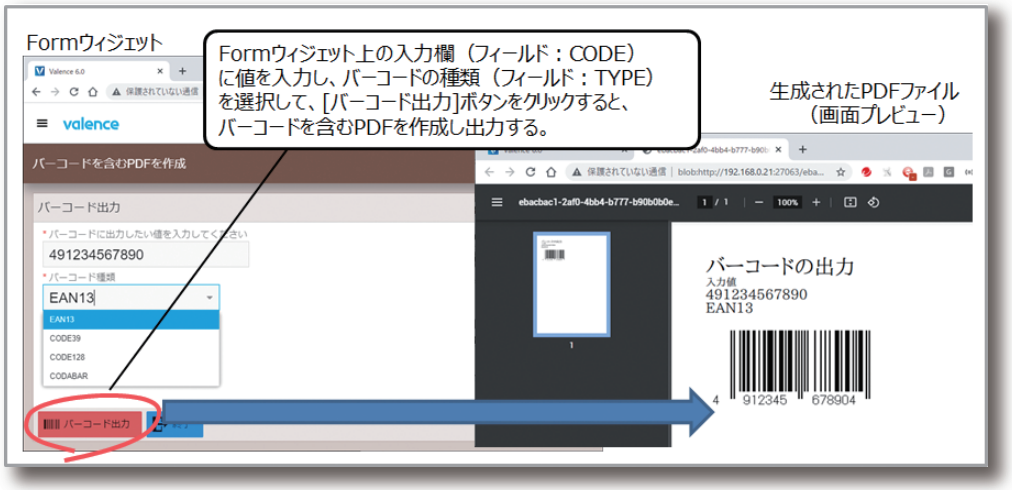
**図17 JavaScriptベースのバーコード作成ライブラリ**


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){

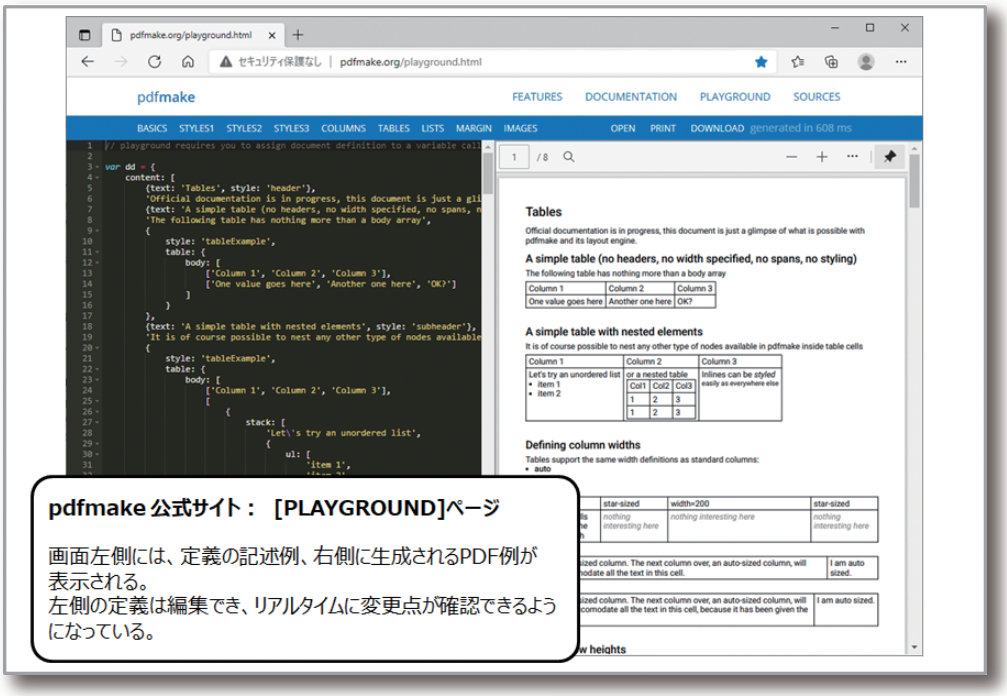
```
1  //---- PDFMakeを起動
2  loadPdfMake().then(function () {
3      //---- 日本語フォント定義
4      pdfMake.fonts = {
5          Aozora: {
6              normal: 'AozoraMinchoRegular.ttf',
7              bold:   'AozoraMincho-bold.ttf'
8          }
9      };
10
11     //----- フォーム上の入力値を取得
12     const CodeVal = rec.get('CODE');  // 入力したコード
13     const TypeVal = rec.get('TYPE');  // バーコード種類 (EAN13, CODE39, CODE128, CODABAR, ...)
14
15     //----- キャンバスを作成しバーコードを発行
16     var canvas = document.createElement('canvas');
17     JsBarcode(canvas, CodeVal,  {format: TypeVal});
18
19     //---- PDFレイアウト作成
20     var docDefinition = {
21         //出力する内容
22         content: [
23             {text: 'バーコードの出力', fontSize: 24},
24             {text: '入力値'},
25             {text: CodeVal, fontSize: 16},
26             {text: TypeVal, fontSize: 16},
27             {text: ' '},
28             {image: canvas.toDataURL(), width: 200, height: 100}  // バーコード画像
29         ],
30         //スタイル（書式）指定
31         defaultStyle: {
32             font: 'Aozora'//日本語フォント
33         }
34     };
35     //---- PDF生成
36     var pdfDocGenerator = pdfMake.createPdf(docDefinition);
37     //---- 生成したPDFを出力
38     pdfDocGenerator.open();   // PDFを開く
39 }, function(err) {
40     //エラー
41 });
```

6-①　6-②　6-③

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。

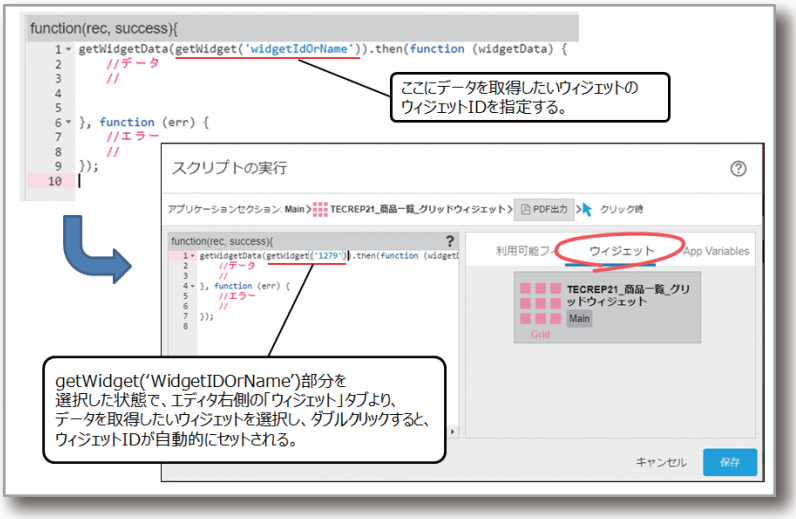
### 5-1. ウィジェットデータの取得方法

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】**

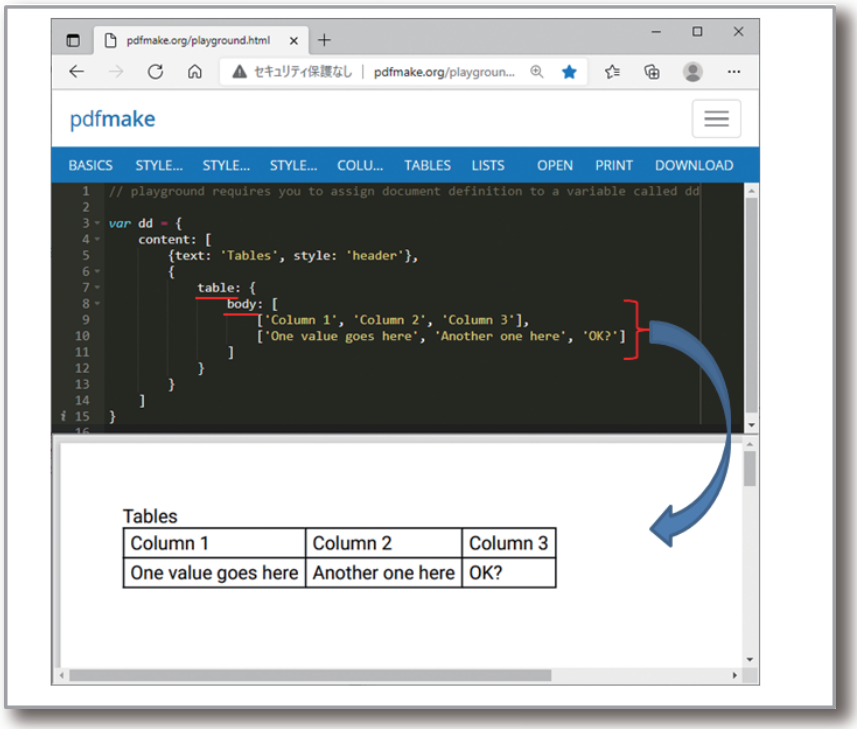
**図22　ウィジェットデータの取得処理（getWidgetData）**



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。

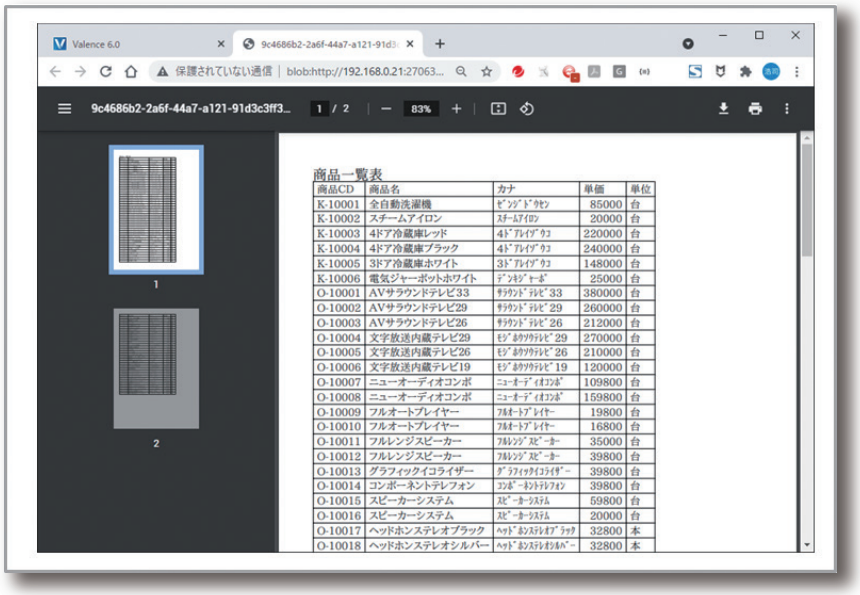
7-①で宣言しているtableBody変数が、動的にbody部を作成する為に用意した配列である。7-②では、表のタイトル行を作成している。7-③では、forループを使用して、wigdetData変数に格納されたウィジェットデータの配列をレコード単位で順番に取得しながら、tableBody変数に値をセットしている。そして、7-④の部分でtableBody変数をpdfmakeの表項目(table)のbody部にセットしている。

**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】

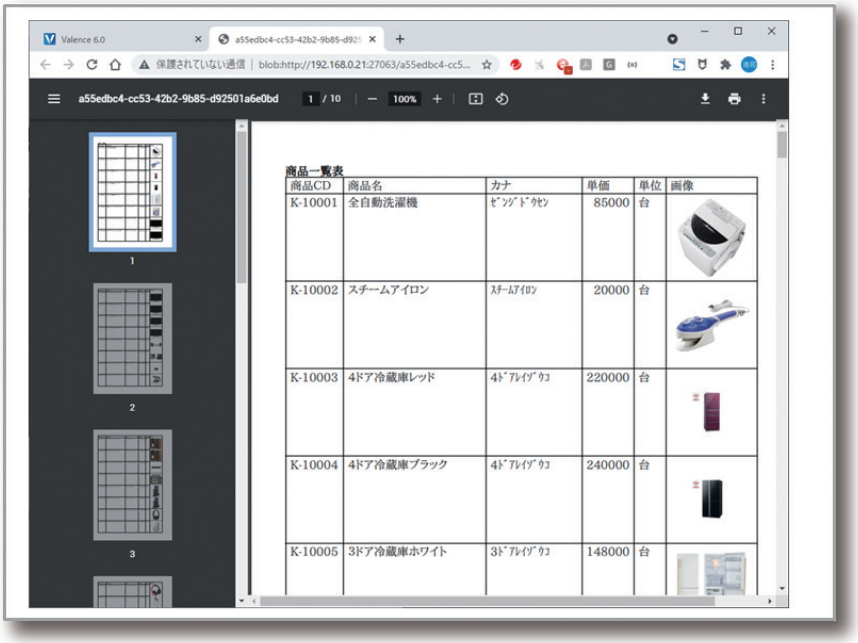
**図24　サンプルプログラム7によって生成されたPDF**




### 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){
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('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','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

            //----- 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
            });                                                                    8-②

            //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
                    width: ['auto','auto','auto','auto','auto','auto'],   // 列幅指定
                    body: tableBody                                 // 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);
        //---- 生成したPDFを出力
        pdfDocGenerator.open();   // PDFを開く
    }, function (err){
        //エラー
    });
}, function (err) {
    //エラー
});
```

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){
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('1279')).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, alignment: '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
                });
            }
        }
```

9-②　9-③

次レコードが無い場合、9-④へ進む

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  9-④                 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  });
```

9-① 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

# MIGARO. Technical Report

## 既刊号バックナンバー

**電子版・書籍(紙)媒体で提供中!**
**https://www.migaro.co.jp/contents/support/technical_report/**

## No.1 2008年秋

### お客様受賞論文

● 最優秀賞

**直感的に理解できるシステムを目指して**
**―情報の“見える化”の取り組み**
石井 裕昭 様／豊鋼材工業株式会社

● ゴールド賞

**運用部間にサプライズをもたらしたDelphi/400**
春木 治 様／株式会社ロゴスコーポレーション

● シルバー賞

**JACi400使用によるWebアプリケーション**
**開発工数削減**
中富 俊典 様／日本梱包運輸倉庫株式会社

**Delphi/400 を利用したWeb受注システム**
飯田 豊 様／東洋佐々木ガラス株式会社

● 優秀賞

**Delphi/400による**
**販売管理システム(FAINS)について**
藤田 建作 様／株式会社船井総合研究所

**技研化成の新基幹システム再構築**
藤田 健治 様／技研化成株式会社

### SE論文

**はじめてのDelphi/400プログラミング**
畑中 侑／システム事業部 システム2課

**Delphi/400とExcelとの連携**
中嶋 祥子／ RAD事業部 技術支援課

**連携で広がるDelphi/400 活用術**
尾崎 浩司／システム事業部 システム2課

**フォーム継承による効率向上開発手法**
吉原 泰介／ RAD事業部 技術支援課

**APIを利用した出力待ち行列情報の取得方法**
鶴巣 博行／ RAD事業部 技術支援課

**DelphiテクニカルエッセンスQ&A 集**
吉原 泰介／ RAD事業部 技術支援課

**JACi400を使ってRPGでWeb画面を制御する方法**
松尾 悦郎／システム事業部 システム2課

**あなたはブラインドタッチができますか?**
福井 和彦／システム事業部 システム1課

## No.2 2009年秋

### お客様受賞論文

●最優秀賞

**JACi400で既存Webサービスの内製化を実現**
佐々木 仁志 様／株式会社ジャストオートリーシング

●ゴールド賞

**.NET環境でのDelphi/400の活用**
福田 祐之 様／林兼コンピューター株式会社

●シルバー賞

**5250で動作する「中古車在庫照会プログラム」の**
**GUI 化**
佐久間 雄 様／株式会社ケーユー

●優秀賞

**Delphiによる輸入システム「MISYS」の再構築**
秦 榮禧 様／株式会社モトックス

**Delphi/400による物流システムの再構築**
仲井 学 様／西川リビング株式会社

**Delphi/400で開発し**
**3台のオフコンを1台のIBM iへ統合**
島根 英行 様／シルフ

### SE論文

**JACi400環境でマッシュアップ!**
岩田 真和／ RAD事業部 技術支援課

**Delphi/400を利用したはじめてのWeb開発**
福岡 浩行／システム事業部 システム2課

**Delphi/400を使用した**
**Webサービスアプリケーション**
尾崎 浩司／システム事業部 システム3課

**Delphi/400によるネイティブ資産の応用活用**
吉原 泰介／ RAD事業部 技術支援課 顧客サポート

**RPGでパフォーマンスを制御**
松尾 悦郎／システム事業部 システム1課

**MKS Integrityを利用したシステム開発**
宮坂 優大・田村 洋一郎／システム事業部 システム1課

# No.3 2010年秋

## お客様受賞論文

●最優秀賞

**建物のクレーム情報管理システム「アフターサービスDB」について**

大橋 良之　様／東レ建設株式会社

●ゴールド賞

**Delphi/400 で「写真管理ソフト」と「スプールファイルのPDF化ソフト」を自社開発**

寒河江 幸喜　様／日綜産業株式会社

●シルバー賞

**Delphi/400で鉄鋼受発注業務を統一し鉄鋼EDIも実現**

柿本 直樹　様／合鐵産業株式会社

●優秀賞

**Delphi/400で EIS（Executive Information System）の高速化**

小島 栄一 様／西川計測株式会社

**イントラでのPHP-Delphi-RPG連携**

仲井 学 様／西川リビング株式会社

**Delphi/400を使った取引先管理システム**

大崎 貴昭 様／森定興商株式会社

## SE論文

**Delphi/400 ローカルキャッシュ活用術**

中嶋 祥子／ RAD事業部 技術支援課

**Delphi/400 帳票開発ノウハウ公開**

尾崎 浩司／システム事業部 システム3課

**Delphi/400でドラッグ＆ドロップを制御**

辻林 涼子／システム事業部 システム2課

**Delphi/400のモジュールバージョン管理手法**

前田 和寛／システム事業部 システム2課

**Delphi/400 WebからのPDF出力**

福井 和彦・清水 孝将／システム事業部システム3課・システム2課

**Delphi/400でFlash 動画の実装**

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

# No.4 2011年秋［創立20周年記念号］

## お客様受賞論文

●最優秀賞

**全社の経費処理業務を効率化した「e総務システム」**

鈴木 英明 様／阪和興業株式会社

●ゴールド賞

**「Web進捗管理システム」でリアルタイム性を実現**

堀内 一弘 様／エスケーロジ株式会社

●シルバー賞

**「営業奨励金申請書」をたった2日間で開発**

簑島 宏明 様／株式会社ケーユーホールディングス

**液体輸送における「配車支援システム」の構築**

桂 哲 様／ライオン流通サービス株式会社

## SE論文

**グラフ活用リファレンス**

中嶋 祥子／ RAD事業部 技術支援課

**Webサービスを利用して機能UP！**

福井 和彦・畑中 侑／システム事業部 システム2課

**OpenOffice実践活用**

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

**VCL for the Web 活用TIPS紹介**

尾崎 浩司／システム事業部 プロジェクト推進室

**JC/400でJavaScript 活用**

清水 孝将／システム事業部 システム1課

**jQuery連携で機能拡張**

國元 祐二／ RAD事業部 技術支援課 顧客サポート

# MIGARO. Technical Report
# 既刊号バックナンバー

## No.5 2012年秋[創刊5周年記念]

### お客様受賞論文

【部門1】

●最優秀賞

**JC/400による取引先とのWeb-EDI システム構築**

久保田 佳裕 様／極東産機株式会社

●ゴールド賞

**DelphiとExcelを使用した帳票コストの削減**

大久保 治高 様／合鐵産業株式会社

**もっと見やすく、もっと使いやすい画面を**

新谷 直正 様／株式会社アダル

【部門2】

●優秀賞

**Delphi/400で確認業務の効率化**

為国 順子 様／ベネトンジャパン株式会社

**取引先申請システムでの稟議書作成ワークフロー**

大崎 貴昭 様／森定興商株式会社

**Delphi/400でIBM iのストアードプロシージャを利用し、SQL処理を高速化**

島根 英行 様／シルフ

### SE論文

**InstallAwareを使ったDelphi/400運用環境の構築**

中嶋 祥子／ RAD事業部 技術支援課 顧客サポート

**カスタマイズコンポーネント入門 Delphi/400開発効率向上**

前田 和寛／システム事業部 システム2課

**Delphi/400スマートデバイスアプリケーション開発**

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

**DataSnapを使用した3層アプリケーション構築技法**

尾崎 浩司／システム事業部 プロジェクト推進室

**JC/400でポップアップウィンドウの制御＆活用ノウハウ**

清水 孝将・伊地知 聖貴／システム事業部 システム1課

【創刊5 周年記念】

**ミガロ.SE座談会 ―お客様と共に歩む、お客様への熱い思い**

## No.6 2013年秋

### お客様受賞論文

【部門1】

●最優秀賞

**自社用開発フレームワークの構築**

駒田 純也 様／ユサコ株式会社

●ゴールド賞

**Delphi/400でCTI開発および関連機能組み込み**

仲井 正人 様／株式会社スマイル・ジャパン

●シルバー賞

**IBM WebFacingからJC/400への移行・リニューアル手法**

八木 秀樹 様／極東産機株式会社

**Delphi/400とDelphiを利用したIBM i資源の有効活用**

小山 祐二 様／澁谷工業株式会社

**発注システムをVBからDelphiへ移植しリニューアル**

川島 寛 様／株式会社タツミヤ

【部門2】

●優秀賞

**5250画面を使用せずにAS/400スプールファイルをコントロールする**

白井 昌哉 様／太陽セメント工業株式会社

**Delphi/400を利用した承認フロー導入によるIT内部統制構築**

塚本 圭一 様／ライオン流通サービス株式会社

### SE論文

**FastReportを使用した帳票作成入門**

尾崎 浩司／ RAD事業部 営業推進課

**Delphi/400で開発する64bitアプリケーション**

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

**Webコンポーネントのカスタマイズ入門**

佐田 雄一／システム事業部 システム1課

**Indyを利用したメール送信機能開発**

辻野 健・前坂 誠二／システム事業部 システム2課

**Windowsテキストファイル操作ノウハウ**

小杉 智昭／システム事業部 プロジェクト推進室

**JC/400 Webアプリケーションのユーザー管理・メニュー管理活用術**

吉原 泰介・國元 裕二／ RAD事業部 技術支援課 顧客サポート

# No.7 2014年秋

## お客様受賞論文

【部門1】

●最優秀賞

**Delphi/400による生産スケジューラの再構築**

柿村 実 様／東洋佐々木ガラス株式会社

●ゴールド賞

**Delphi/400およびDelphiを利用した
オンライン個人別メニューの構築**

小山 祐二 様／澁谷工業株式会社

●シルバー賞

**IBM iとDelphi/400のコラボレーション**

新谷 直正 様／株式会社アダル

●シルバー賞

**荷札発行システムリプレースについて**

仲井 学 様／西川リビング株式会社

【部門2】

●優秀賞

**Delphi/400バージョンアップのための
クライアント環境構築**

普入 弘 様／株式会社エイエステクノロジー

●優秀賞

**外出先からメールでリアルタイム在庫を問い合せ**

島根 英行 様／シルフ

## SE論文

**iOS/Androidネイティブアプリケーション入門**

吉原 泰介／ RAD事業部 技術支援課

**ファイル加工プログラミングテクニック**

小杉 智昭／ システム事業部 プロジェクト推進室

**FastReportを使用した帳票作成テクニック**

前坂 誠二／ システム事業部 システム2課

**大量データ処理テクニック**

佐田 雄一／システム事業部　システム１課

**スマートデバイスWEBアプリケーション入門**

尾崎 浩司／ RAD事業部 営業推進課
國元 祐二／ RAD事業部 技術支援課

# No.8 2015年秋

## お客様受賞論文

【部門1】

●最優秀賞

**iPod Touchの業務利用開発と検証**

石井 裕昭 様／豊鋼材工業株式会社

●ゴールド賞

**ブランク加工図管理システムの構築**

小山 祐二 様／澁谷工業株式会社

●シルバー賞

**Delphi/400でスプールファイル管理
（WRKSPLF コマンドの活用）**

三好 誠 様／ユサコ株式会社

●シルバー賞

**予算管理システムの構築**

川島 寛 様／株式会社タツミヤ

●シルバー賞

**送状データ送信システムのWeb化について**

仲井 学 様／西川リビング株式会社

【部門2】

●優秀賞

**繰り返しDB参照時の
ClientDataSetのFirst 機能について**

牛嶋 信之 様／株式会社佐賀鉄工所

●優秀賞

**IBM iのカレンダーを基準に他のシステムを稼働**

福島 利昭 様／株式会社ランドコンピュータ

## SE論文

**フレームを利用した開発手法**

前坂 誠二／システム事業部 システム2課

**Windowsタブレット用に
カスタムソフトウェアキーボードを実装**

福井 和彦／システム事業部 プロジェクト推進室

**マルチスレッドを使用したレスポンスタイム向上**

尾崎 浩司／ RAD事業部 営業・営業推進課

**AndroidアプリケーションのNFC機能活用**

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

**スマートデバイス開発で役立つ画面拡張テクニック**

國元 祐二／ RAD事業部 技術支援課 顧客サポート

# MIGARO. Technical Report
# 既刊号バックナンバー

## No.9 2016年秋

### お客様受賞論文

【部門1】

●最優秀賞

**IBM iの見える化で実現するアジャイル開発**

吉岡 延泰 様／日本調理機株式会社社

●ゴールド賞

**Windows Like 5250への道のり**

小山 祐二 様／澁谷工業株式会社

●シルバー賞

**Delphiプログラム管理ソフトの開発**

牛嶋 信之 様／株式会社佐賀鉄工所

【部門2】

●優秀賞

**Delphi/400を利用した定型業務のPDF化**

佐藤 岳 様／ライオン流通サービス株式会社

●優秀賞

**ちょい足しモバイル**

仲井 正人 様／株式会社スマイル・ジャパン

●優秀賞

**AS/400の受注データをWebで社員に公開**

福島 利昭 様／株式会社ランドコンピュータ

### SE論文

**iOSモバイルアプリ開発のデザイニングテクニック**

前坂 誠二／システム事業部 システム2課

**新データベースエンジンFireDACを使ってみよう！**

福井 和彦／システム事業部 プロジェクト推進室

**Delphi/400 最新プログラム文法の活用法**

尾崎 浩司／ RAD事業部 営業・営業推進課

**FastReportを活用した電子帳票作成テクニック**

宮坂 優大／システム事業部 システム1課

**Beacon技術によるIoT活用の第一歩**

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

**Web&ハイブリッドアプリ開発で役立つ IBM i&ブラウザデバッグテクニック**

國元 祐二／ RAD事業部 技術支援課 顧客サポート

## No.10 2017年秋

### Migaro.Technical Report 創刊10周年記念

**パートナー様からの祝辞**

武藤 和博 様／日本アイ・ビー・エム株式会社

藤井 等 様／エンバカデロ・テクノロジーズ日本法人

Serge Charbit 様／ SystemObjects Corporation

飯田 恭子 様／アイマガジン株式会社

**お客様からの祝辞・お客様の声**

石井 裕昭 様／豊鋼材工業株式会社

牛嶋 信之 様／株式会社佐賀鉄工所

大崎 貴昭 様／森定興商株式会社

川島 寛 様／株式会社タツミヤ

久保田 佳裕 様／極東産機株式会社

駒田 純也 様／ユサコ株式会社

小山 祐二 様／澁谷工業株式会社

寒河江 幸喜 様／日綜産業株式会社

佐々木 仁志 様／株式会社ジャストオートリーシング

佐藤 岳 様／ライオン流通サービス株式会社

白井 昌哉 様／太陽エコブロックス株式会社

仲井 学 様／西川リビング株式会社

福島 利昭 様／株式会社ランドコンピュータ

**お客様座談会**

石井 裕昭 様／豊鋼材工業株式会社

駒田 純也 様／ユサコ株式会社

寒河江 幸喜 様／日綜産業株式会社

仲井 学 様／西川リビング株式会社

上甲 將隆／株式会社ミガロ.

司会　飯田 恭子 様／アイマガジン株式会社

## お客様受賞論文

【部門1】

●最優秀賞

**貸金庫と鍵のマッチング業務をDelphi/400で実現**
**―文字認識データと基幹システムデータを統合**

佐藤 正 様／株式会社富士精工本社

●ゴールド賞

**Windowsタブレット導入による**
**工作部材料受入業務改革**

小山 祐二 様／澁谷工業株式会社

【部門2】

●優秀賞

**Delphi/400を利用した**
**各拠点PINGコマンド簡素化**

松垣 秀昭 様／ライオン流通サービス株式会社

**汎用的な帳票出力画面**

牛嶋 信之 様／株式会社佐賀鉄工所

**バーコードリーダー読み取り後、**
**次の入力位置にカーソルを自動遷移させる技術**

上総 龍央 様／キョーラクシステムクリエート株式会社

**IBM iのスプールファイル参照機能を**
**Webで構築**

福島 利昭 様／株式会社ランドコンピュータ

## SE論文

**デスクトップアプリケーション開発でも役立つ**
**FireMonkey活用入門**

尾崎 浩司／ RAD事業部 営業・営業推進課

**Delphi/400バージョンアップに伴う**
**文字コードの違いと制御**

宮坂 優大／システム事業部 システム1課

**FastReportへの**
**効率的な帳票レイアウトコンバート**

畑中 侑／システム事業部 システム2課

**IBM iトリガー機能を活かした**
**セキュリティログ対応**

八木沼 幸一／システム事業部 プロジェクト推進室

**アプリケーションテザリングを利用した**
**PC&モバイルアプリケーション連携**

吉原 泰介／ RAD事業部 技術支援課 顧客サポート

**SmartPad4iの運用で役立つWEB サーバー機能**

國元 祐二／ RAD事業部 技術支援課 顧客サポート

# No.11 2018年秋

## お客様受賞論文

【部門1】

●最優秀賞

**Excelテンプレートを使用した帳票出力機能の開発**

駒田 純也 様／ユサコ株式会社

●ゴールド賞

**SP4iの活用による製品検査チェックシステムの構築**

八木 秀樹 様／極東産機株式会社

【部門2】

●優秀賞

**配車支援システムをDelphi/400で再構築**

村上 稔明 様／ライオン流通サービス株式会社

**一般シール受注入力業務のDelphi/400化**

寺西 健一 様／大阪シーリング印刷株式会社

**Delphi/400による無線ハンディターミナルの**
**データ集約の仕組みの実装**

寺西 健一 様／大阪シーリング印刷株式会社

## SE論文

**OLEを利用したExcel出力の**
**パフォーマンス向上手法**

薬師 尚之／システム事業部 システム2課

**FireDAC実践プログラミングテクニック**

佐田 雄一／システム事業部 システム1課

**RESTによるWebサービスを活用した**
**機能拡張テクニック**

尾崎 浩司／ RAD事業部 営業・営業推進課

**Google Maps Platformを使用した**
**アプリケーション開発テクニック**

福井 和彦・小杉 智昭／システム事業部 プロジェクト推進室

**RAD Serverを使った**
**新しい多層アプリケーション構築**

吉原 泰介／ RAD事業部 技術支援課

**JC/400からSP4iへのマイグレーションノウハウ**

吉原 泰介・國元 祐二／ RAD事業部 技術支援課

# MIGARO. Technical Report
# 既刊号バックナンバー

## No.12 2019年秋

### お客様受賞論文

【部門1】

●最優秀賞

**Delphi/400による**
**電子帳簿保存法スキャナ保存制度への挑戦**

石井 裕昭 様／豊鋼材工業株式会社

●ゴールド賞

**出荷業務従事者のモチベーションアップ大作戦**
**—Delphi/400で基幹データの見える化**

池田 純子 様／錦城護謨株式会社

●ゴールド賞

**iPhoneを利用した**
**宝飾品の販売系システムをSP4i で構築**

島本 佳昭 様／株式会社大月真珠

【部門2】

●優秀賞

**SmartPad4iによる、導入後の改修を意識した設計**

西村 直也 様／株式会社ジャストオートリーシング

●優秀賞

**Valence を使用した 温度・湿度ログ表示**
**－サーバー室内温度・湿度変化の見える化－**

中谷 佳史 様／株式会社保健科学西日本

●優秀賞

**RPGソースコードをValence File Editorで改修**

福島 利昭 様／株式会社ランドコンピュータ

●特別寄稿論文

**統合開発環境Cobosのご紹介**
**—RPG・SP4iの効率的な開発に**

### SE論文

**IBM iデータベースへのFTP データ転送手法の紹介**

田村 洋一郎・宮坂 優大・都地 奈津美／システム事業部 システム1課

**FireMonkeyの活用**
**カメラコンポーネントを使ったアプリ**

畑中 侑／システム事業部 システム2課

**Subversionを使用したDelphiソース管理**

福井 和彦／システム事業部 プロジェクト推進室

**Enterprise Connectorsを利用した**
**クラウド連携テクニック**

佐田 雄一／ RAD 事業部 技術支援課

**SmartPad4iのインターフェース機能拡張**

國元 祐二／ RAD事業部 技術支援課

**Valence App Builder RPG連携テクニック**

尾崎 浩司／ RAD事業部 技術支援課

## No.13 2020年秋

### SE論文

**Windowsタブレット向け**
**VCLアプリケーション作成テクニック**

都地 奈津美／システム事業部 システム1課

**iOSモバイルアプリケーションによる**
**ファイル閲覧機能作成**

前坂 誠二／システム事業部 システム2課

**Delphi 10シリーズ**
**VCLプログラム開発の最新トピックス**

佐田 雄一／ RAD事業部 技術支援課

**Eclipse（Cobos4i）を使用したSmartPad4i開発術**

國元 祐二／ RAD事業部 技術支援課

**Valence最新バージョン 進化のポイント**

尾崎 浩司／ RAD事業部 技術支援課

# MIGARO.
# Technical Report 2021

**No.14 2021年**

**2021年12月1日 初版発行**

発行

株式会社ミガロ.

〒556-0017

大阪府大阪市浪速区湊町 2-1-57

難波サンケイビル 13F

TEL：06（6631）8601

FAX：06（6631）8603

https://www.migaro.co.jp/

発行人

上甲 將隆

編集協力・印刷

芳武印刷株式会社

株式会社ミガロ.

https://www.migaro.co.jp/


**本社**

〒556-0017

大阪市浪速区湊町2-1-57

難波サンケイビル13F

TEL:06(6631)8601

FAX:06(6631)8603

**東京事業所**

〒100-0013

東京都千代田区霞が関3-7-1

霞が関東急ビル2F

TEL:03(5510)5701

FAX:03(5510)5702