MIGARO. TECHNICAL REPORT 2024

ミガロ. テクニカルレポート 2024年
No.17

株式会社ミガロ.

MIGARO.
Technical Report 2024

ミガロ. テクニカルレポート 2024年

No.17

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、ミガロ.製品に関する技術情報をお届けする論文集で、このたび第17号を発刊する運びとなりました。これもひとえに皆様からのご支援とご愛顧の賜物と、心より感謝申し上げます。
企業の基幹システムを支えるIBM iですが、現在その取り巻く環境は大きく変化しています。5250エミュレーター環境のみで完結できていたシステムが、近年のクラウド型Webサービスの普及やスマートフォンやタブレットといったモバイル端末の業務活用ニーズの高まりにより、Web化やモバイル対応を求められるようになっております。

ミガロ.では、そんなIBM i環境のモダナイゼーションを支援する多彩なツールを提供しております。Delphi/400は、優れた統合開発環境により効率的なWeb・モバイル開発が可能です。SP4iはIBM i技術者が熟知するRPG／COBOLスキルを活かしたWeb・モバイル開発を実現いたします。Valenceはユーザーのニーズに迅速に対応可能なローコード開発環境を提供しております。それぞれのツールで開発アプローチは異なりますが、IBM i環境のWeb化、モバイル対応を推進できる点は共通です。

今回のテクニカルレポートでは、多様化するニーズに対応した最新技術を取り上げた論文をご用意しました。Delphi/400の論文では、Web環境の利用に適したOLEに頼らないExcel連携手法を紹介しています。また、Webフレームワーク「IntraWeb」の活用による操作性の良いUIの工夫や、タブレット端末に適したグリッドの構築方法を解説しています。SP4iの論文では、最新のCobos4i環境へ移行するメリットとマイグレーション手法を紹介します。Valenceは、ローコード開発の要となる「アプリ変数」に焦点を当て、その有効な活用方法を解説しています。各論文では詳細な実装方法も記載し、日々の開発にご活用いただける内容となっております。

2008年に初号を発刊して以来、多くの皆様のご支援を賜りながら、このテクニカルレポートを継続することができました。この場をお借りして、改めて厚く御礼申し上げます。本レポートが、今後の開発や保守にお役立ていただけましたら幸いです。

2024年12月

株式会社ミガロ.
代表取締役社長
上甲 將隆

Delphi/400

Excel4Delphi
ライブラリを活用したExcel操作術

株式会社ミガロ.
プロダクト事業部 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
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(現プロダクト事業部)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している。

1.はじめに

Delphi/400の業務アプリケーションにおいては、Excelのブックを出力または取込を行う機能を組み込むことが多いことと思う。主な方法としては、ExcelのOLE機能を使用したものがあり、Delphi/400のプログラムからExcelを起動して操作する。ただし、実行端末にExcelが導入されている必要がある。
サードパーティ製品でもExcelを扱うものはいくつか存在するが、Excelが導入されていない端末で動作が可能なものは少ない。

本稿ではExcel4Delphiというライブラリを使用して、Excel導入不要でXLSX形式(古いXLS形式は非対応)のブックの読み書きを行う手順を紹介する。

本稿の執筆にあたってはDelphi/400 11 Alexandria (Delphi 11.3)を使用しており、Excel4Delphiライブラリについての情報は2024年9月時点のものとなっている。

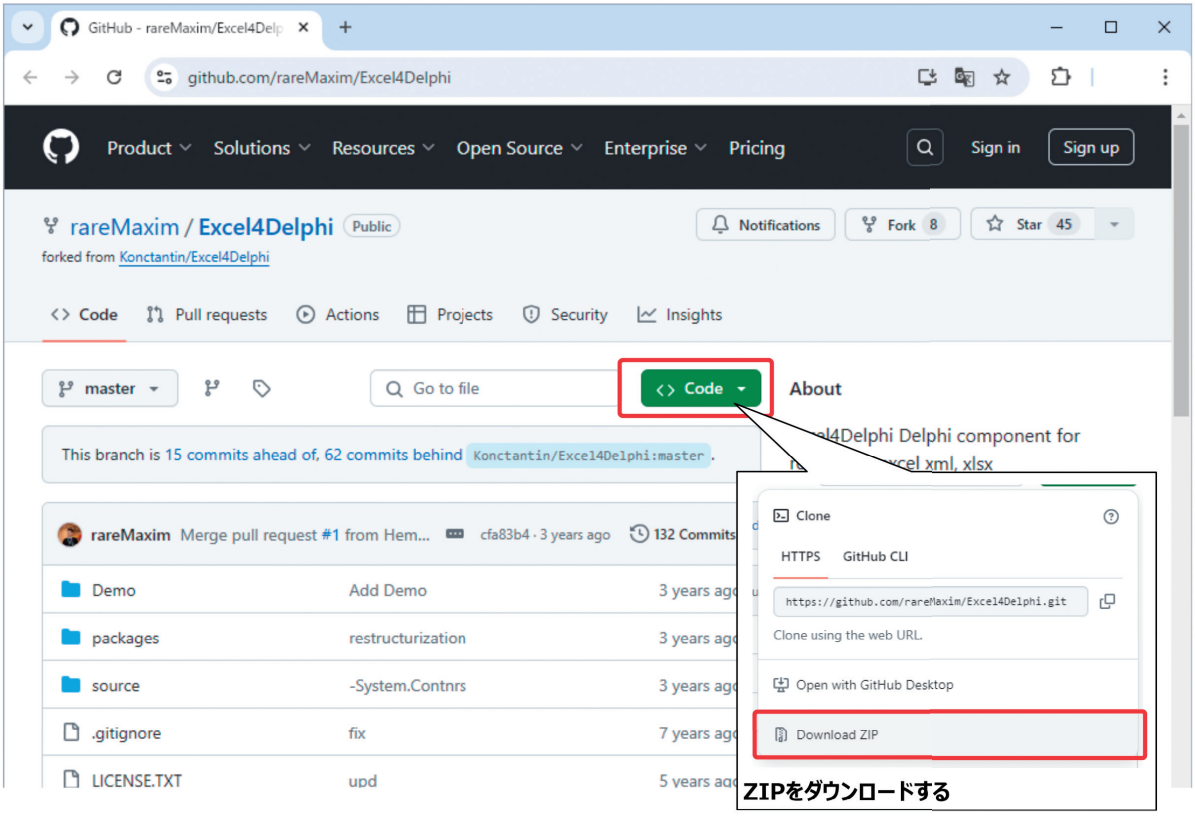
また今回使用するExcel4Delphiライブラリではファイル圧縮・解凍に「zlib」を使用しており、そのライセンス条項は「LICENSE.TXT」に記載されているため確認が必要である。

2. Excel4Delphiの概要と導入手順

今回使用するExcel4Delphiは、rareMaxim氏によって開発されたExcel入出力クラス集である。
Excel 2007以降のブック（XLSX・XLSMなど）は内部的にはXMLの集合体になっており、Excel4Delphiを使うとその読み書きをロジック内で行うことができる。
Excel4Delphiは厳密にはExcelブックを出力または取込するというよりはXMLの集合体を読み込み・生成するもので、OLE以外のExcelブックの出力または取込を行う機能の実装手段として利用することができる。（そのため、XMLではなくバイナリ形式のXLSブックには対応していない。）

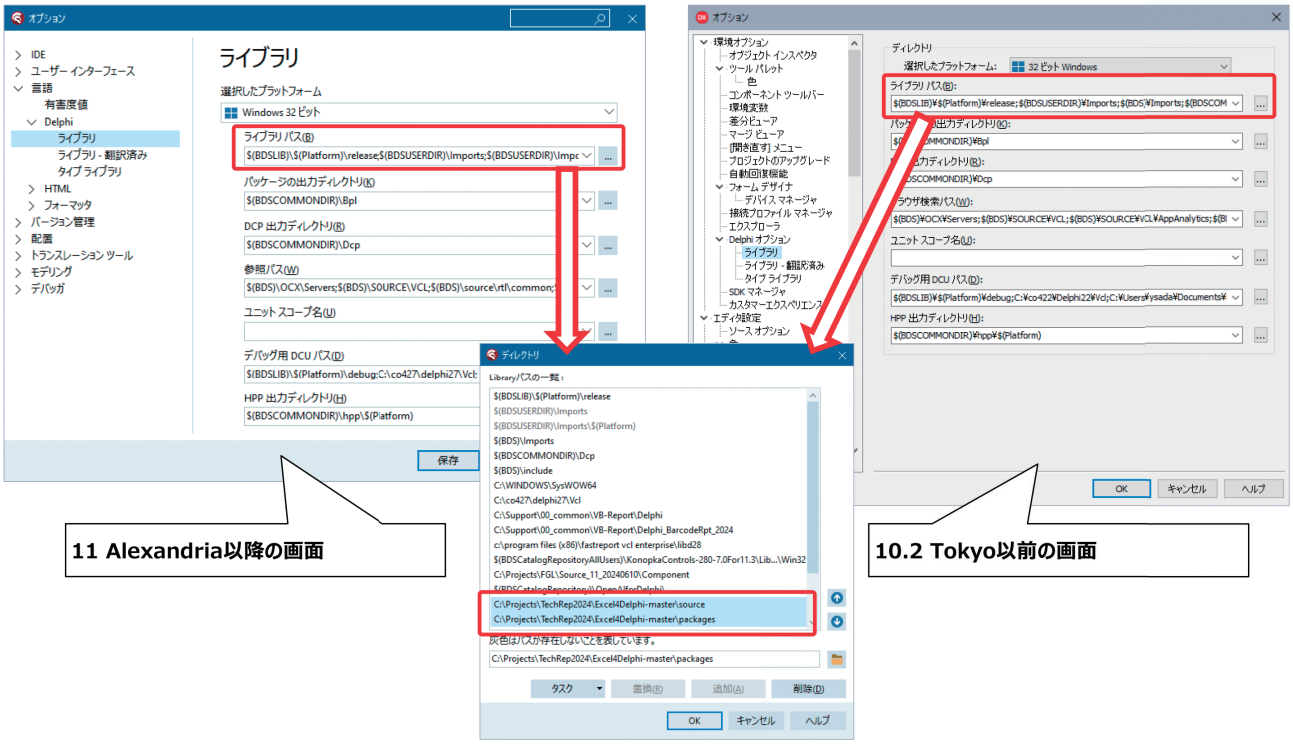
ここからは導入手順について説明していく。
まず、GitHubのウェブサイト内にある
https://github.com/rareMaxim/Excel4Delphi
にアクセスすると、ファイル一覧、英語の概要やサンプルロジックが記載されたページが表示される【図1】。内容や先述のライセンス条項を確認したあと、緑の「<> Code」メニューから『Download ZIP』ボタンを押すと、この一覧にあるExcel4Delphi関連ファイルがZIP形式で一括ダウンロードされる。

図 1　Excel4Delphiのダウンロード



取得したZIPファイルを任意のフォルダに解凍したら、各プロジェクトをコンパイルする際に必要になるため「Excel4Delphi-master」フォルダ内にある「source」というフォルダをDelphiのライブラリパスに追加しておく。ライブラリパスにフォルダを追加するには、IDE（統合開発環境）の「ツール」メニューの「オプション」を開き、【図2】の項目にフォルダを追加する。Delphi 11では10.2以前とオプション画面の配置が異なっているため、ご利用のバージョンの画面を参照頂きたい。
また「packages」というフォルダやその中に「Excel4DelphiLib.dpk」というパッケージが存在するが、Excel4Delphiではカスタムコンポーネントを使用していないためインストールしなくても支障はない。

図 2　ライブラリパスの設定



Excel4Delphiの準備が完了したら、まずは「Demo」フォルダにあるプロジェクト「d1.dpr」を開いて実行してみよう。プロジェクトをコンパイルして生成される「d1.exe」は画面を持たないため起動すると処理のみを行ってすぐに終了するが、終了後に同じフォルダ内に「file.xlsx」というブックが出力されている**【図3】**。このブックを開いてみると、A1～B3セルが結合され、「Hello」という文字がセットされていることが確認できる。この出力ロジックは「d1.dpr」内に記載されており、ソースを表示するとほんの十数行でこれだけのExcel出力ロジックが実装されているのを確認できるだろう**【図4】**。

図 3　デモプログラムで出力されるXLSXブック

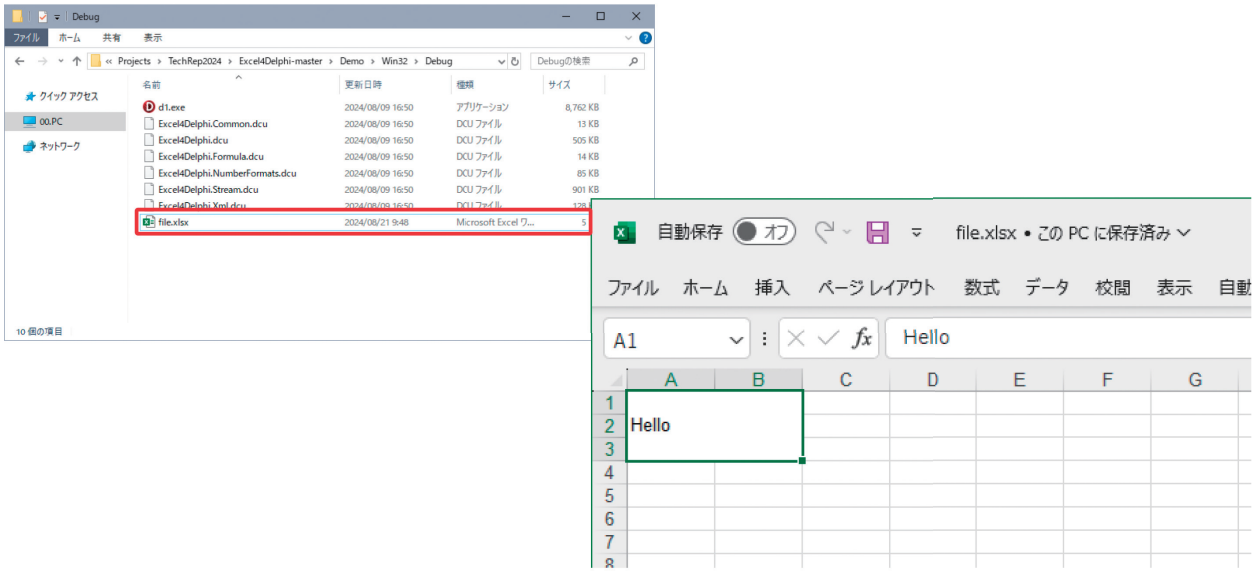


図 4　デモプログラムのロジック

ウェルカム ページ　d1

```
1  program d1;
2
3  {$APPTYPE CONSOLE}
4  {$R *.res}
5
6  uses
7    Excel4Delphi,
8    Excel4Delphi.Stream,
9    System.SysUtils;
10
11 procedure CreateNewBook;
12 // Creating new workbook
13 var
14   workBook: TZWorkBook;
15 begin
16   workBook := TZWorkBook.Create(nil);
17   try
18     workBook.Sheets.Add('My sheet');
19     workBook.Sheets[0].ColCount := 10;
20     workBook.Sheets[0].RowCount := 10;
21     workBook.Sheets[0].CellRef['A', 0].AsString := 'Hello';
22     workBook.Sheets[0].RangeRef['A', 0, 'B', 2].Merge();
23     workBook.SaveToFile('file.xlsx');
24   finally
25     workBook.Free();
26   end;
27 end;
28
29 begin
30   try
31     { TODO -oUser -cConsole Main : Insert code here }
32     CreateNewBook;
33   except
34     on E: Exception do
35       Writeln(E.ClassName, ': ', E.Message);
36   end;
37
38 end.
```

この部分がExcel出力ロジック

3. Excel4Delphiの基本的な使い方

3-1. Excel4Delphiを使ったブックの読み込み

本稿で紹介するExcel4Delphiは海外のライブラリであるため、日本語環境で問題なく使用できるか検証してみた。その結果、基本的にはそのまま使用できそうではあるが、いくつか工夫する必要があったため、そのポイントもあわせて紹介する。

本章ではExcel4Delphiを使って既存のXLSX形式のブックを読み込み、対象のセルにある値を読み込んでIBM iのワークファイルに更新するサンプルプログラムを作成していく。

今回使用するブックは例として、【図5】のようなフォーマットのものを用意する。また次項で使用する更新先のワークファイルを【図6】のDDSのようなレイアウトで作成しておく。

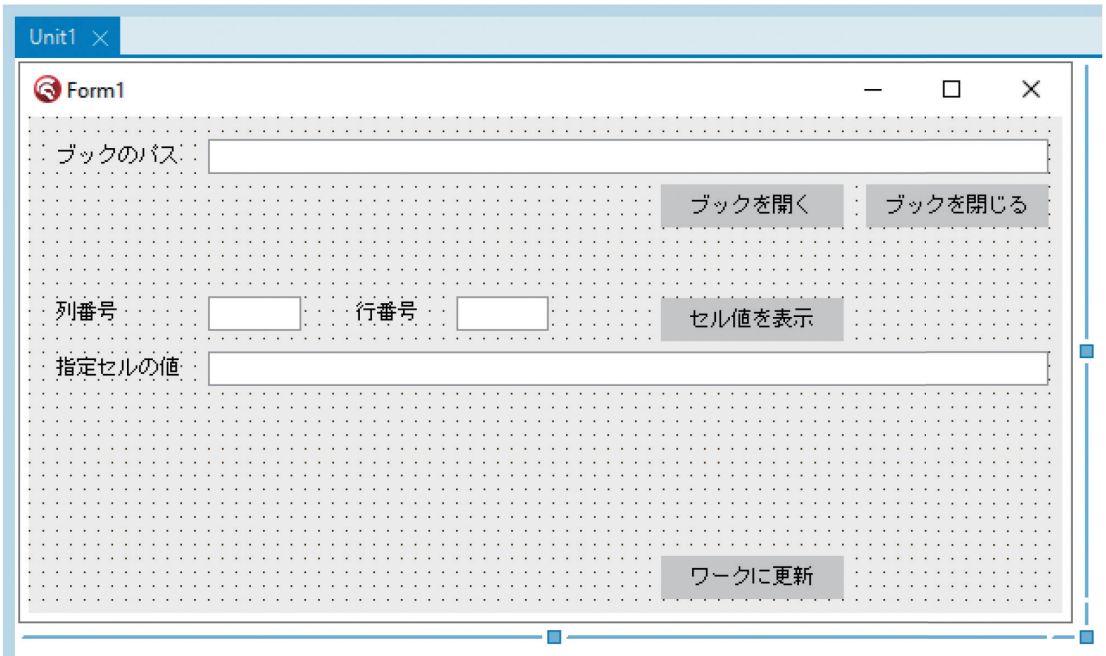
図 5　Excel4Delphiに読み込ませるブック

図 6　今回用のワークファイル(WKTR17)のレイアウト

```
A*************************************************************
A          R WKTR17R                   TEXT('テクニカルレポート２０２４')
A            WKHANO         8A        COLHDG('発注No.')
A            WKHADT         8P 0      COLHDG('発注日')
A            WKSICD         4A        COLHDG('仕入先コード')
A            WKSINM        520        COLHDG('仕入先名')
A            WKTNNM        120        COLHDG('ご担当者名')
A            WKTEL         15A        COLHDG('TEL')
A            WKFAX         15A        COLHDG('FAX')
A            WKHICD         5A        COLHDG('品番')
A            WKHINM        520        COLHDG('品名')
A            WKNOKI         8P 0      COLHDG('希望納期')
A            WKSURY         8P 0      COLHDG('数量')
A            WKTANI        120        COLHDG('単位')
A            WKNOCD         8A        COLHDG('納入場所コード')
A            WKNONM        520        COLHDG('納入場所名')
A            WKBIK1        520        COLHDG('備考１')
A            WKBIK2        520        COLHDG('備考２')
A          K WKHANO
A          K WKHADT
```

まずDelphiで新規プロジェクトを作成し、画面にコンポーネントを配置していく。ここでは各入力項目のためのTEdit、各種処理を行うTButton、および各キャプションのためのTLabelを配置する【図7】。

図 7　画面設計レイアウト



次にロジックを記述していく。宣言部のuses節に「Excel4Delphi」および「Excel4Delphi.Stream」を、Private宣言に変数『bOpened』(Boolean型)および『workBook』(TZWorkBook型)を宣言する。
TZWorkBookはブック全体を包括するクラスで、今回はコンポーネントのように使用するが、Excel4Delphiではカスタムコンポーネントは無く、付属のパッケージをインストールしてもパレット(ツールパレット)に項目は追加されない。そのため、今回はクラス(今回はTZWorkBook型の変数『workBook』)を画面生成時に一緒に生成し、クローズ時に一緒に解放するような記述としている。その際の処理を**【ソース1】**のように記述する。また、画面上部の「ブックを開く」ボタンを押した際の処理を**【ソース2】**のように記述し、「ブックを閉じる」ボタンを押した際の処理を**【ソース3】**のように記述する。

ソース 1

Excel4Delphi 画面の生成時とクローズ時

```
{*******************************************************************************
  画面生成時
*******************************************************************************}
procedure TForm1.FormCreate(Sender: TObject);
begin
  // TZWorkBookクラス生成
  workBook := TZWorkBook.Create(nil);
end;

{*******************************************************************************
  画面クローズ時
*******************************************************************************}
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  // TZWorkBookクラス解放
  workBook.Free();
end;
```

ソース 2

Excel4Delphi ブックを開く

```
{*******************************************************************************
  ブックを開くボタン
*******************************************************************************}
procedure TForm1.btnOpenBookClick(Sender: TObject);
begin
  // オープン済の場合は処理中断
  if bOpened then
  begin
    ShowMessage('開いているブックを先に閉じてください。');
    Abort;
  end;

  // ブックが存在しない場合は処理中断
  if not(FileExists(edtXLSX.Text)) then
  begin
    ShowMessage('指定されたファイルが存在しません。');
    Abort;
  end;

  // フルパスを指定してブックを開く
  workBook.LoadFromFile(edtXLSX.Text);

  // オープン済フラグをTrueに設定
  bOpened := True;

  ShowMessage('ブックを開きました。');
end;
```

ソース 3

Excel4Delphi ブックを閉じる

```
{*******************************************************************************
  ブックを閉じるボタン
*******************************************************************************}
procedure TForm1.btnCloseBookClick(Sender: TObject);
begin
  // 未オープンエラーの防止
  if not bOpened then
  begin
    ShowMessage('ブックが開かれていません。');
    Abort;
  end;

  // オープン済フラグをFalseに設定
  bOpened := False;

  ShowMessage('ブックを閉じました。');
end;
```

次に設計画面の中央部にある「セル値を表示」ボタンを押した際の処理を【ソース4】のように記述する。この処理の目的は、当サンプル内で取得するブック内の各セルの値を確認し、桁あふれや型不正などのエラーを防ぐことである。調査の結果、詳細は後述するが、いくつか工夫が必要なポイントがあった。

ソース 4

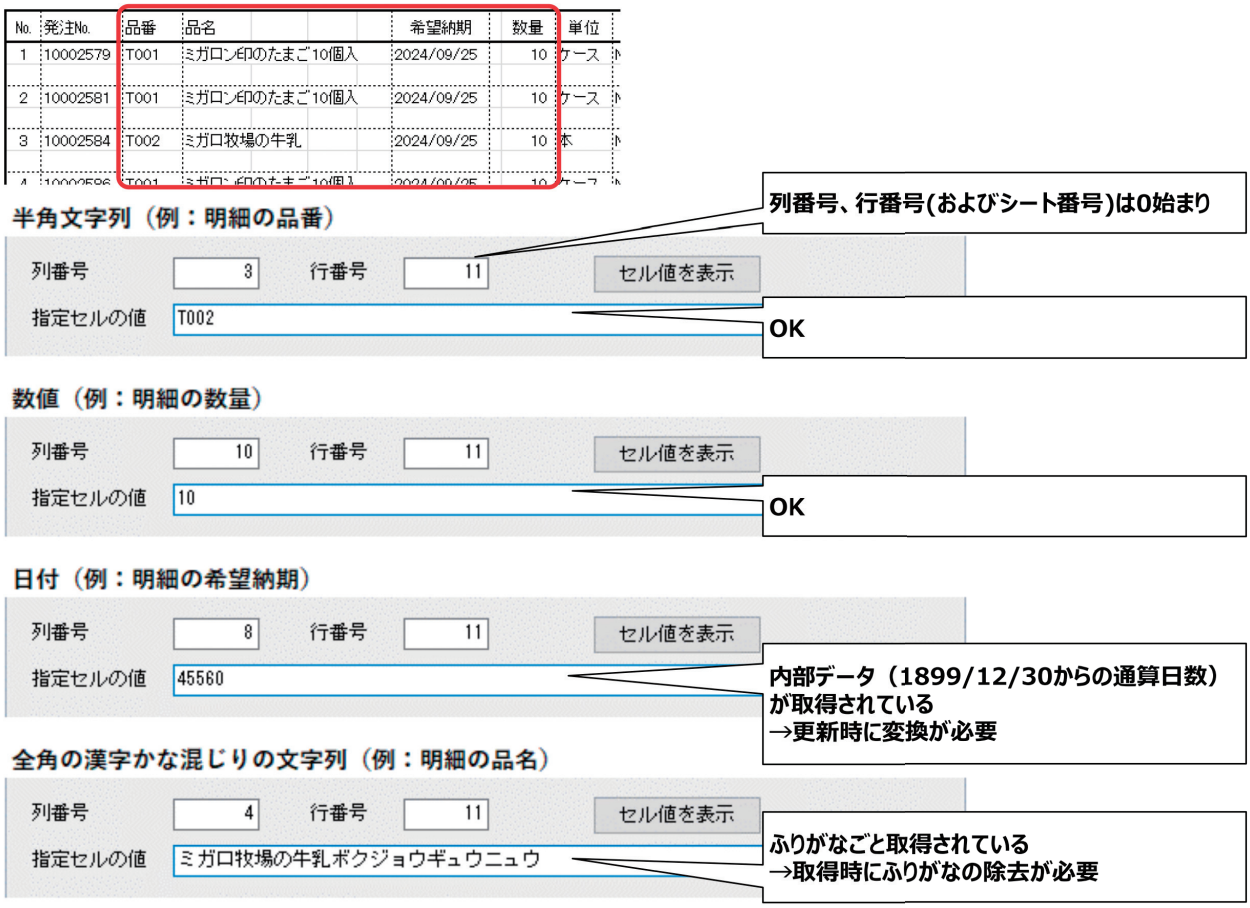
Excel4Delphi セル値を表示ボタン

```
{*******************************************************************************
  セル値を表示ボタン（取得結果を検証するための実装例）
*******************************************************************************}
procedure TForm1.btnCheckCellClick(Sender: TObject);
var
  iCol, iRow: Integer;
  sText: String;
begin
  // 未オープンエラーの防止
  if not bOpened then
  begin
    ShowMessage('ブックが開かれていません。');
    Abort;
  end;

  // 列番号と行番号を整数値にセット（それぞれ0始まり）
  iCol := StrToIntDef(edtCol.Text, -1);
  iRow := StrToIntDef(edtRow.Text, -1);
  if (iCol < 0) or (iRow < 0) then
  begin
    ShowMessage('列番号または行番号が正しくありません。');
    Abort;
  end;

  // 最初のシートにある指定セルの値を取得（シートも0始まり）
  sText := workBook.Sheets[0].Cell[iCol, iRow].AsString;
  edtText.Text := sText;
end;
```

ここまで記述できたら、プロジェクトをコンパイルして実行してみよう。ブックのパスに【図5】で示した注文書のフルパスを指定して開く。列番号および行番号（いずれも0始まり）を指定して「セル値を表示」ボタンを押すと、それぞれのセルの値を取得できていることが確認できる。なお今回は単一シートだが、シート番号も0始まりである。

取得値を一通りチェックしていると、想定に対して違和感があるセルが見つかるだろう【図8】。

図 8　データ型によって取得される値の違い



ここで問題になるセル値は日付値と漢字かな交じりの文字列である。前者はExcelの内部では日付に対応する数値でデータを保持しているため、ワークに更新する際は日付型に直す必要があることがわかった。そして後者はExcelで見えている文字列に加えて「ふりがな」まで取得されてしまっている。このままでは想定の値と異なるだけではなく更新時に桁あふれをおこす可能性もある。ここからはこの現象を改善するため、Excel4Delphiのソースを修正していく。

前章でも触れた通りXLSX形式ファイルはXMLの集合体なので、そのXMLの構造がわかればロジックから該当の部分を特定することができる。今回の場合はブックをコピーして拡張子を「XLSX」から「ZIP」に変えたものを作成し、それを任意の方法で解凍する。その中にある「sharedStrings.xml」というXMLの中に該当の部分が見つかった【図9】【図10】。

図 9　XLSXブック内のXML階層構造

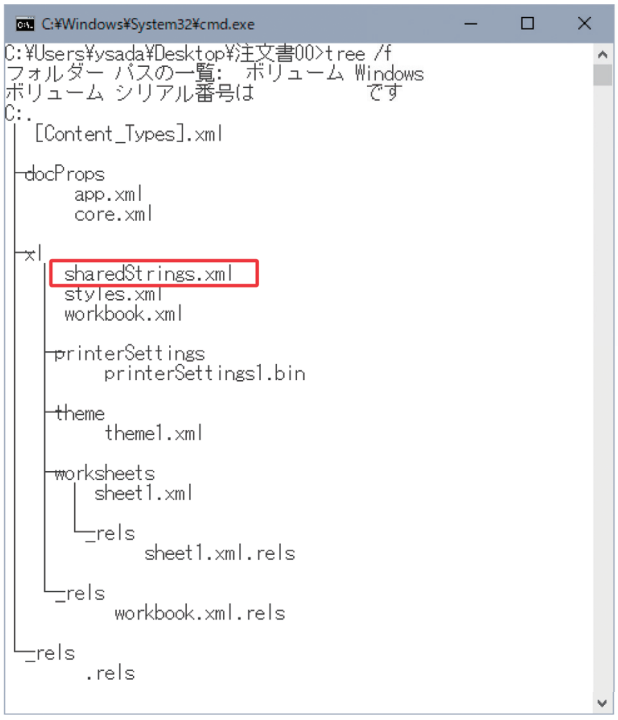
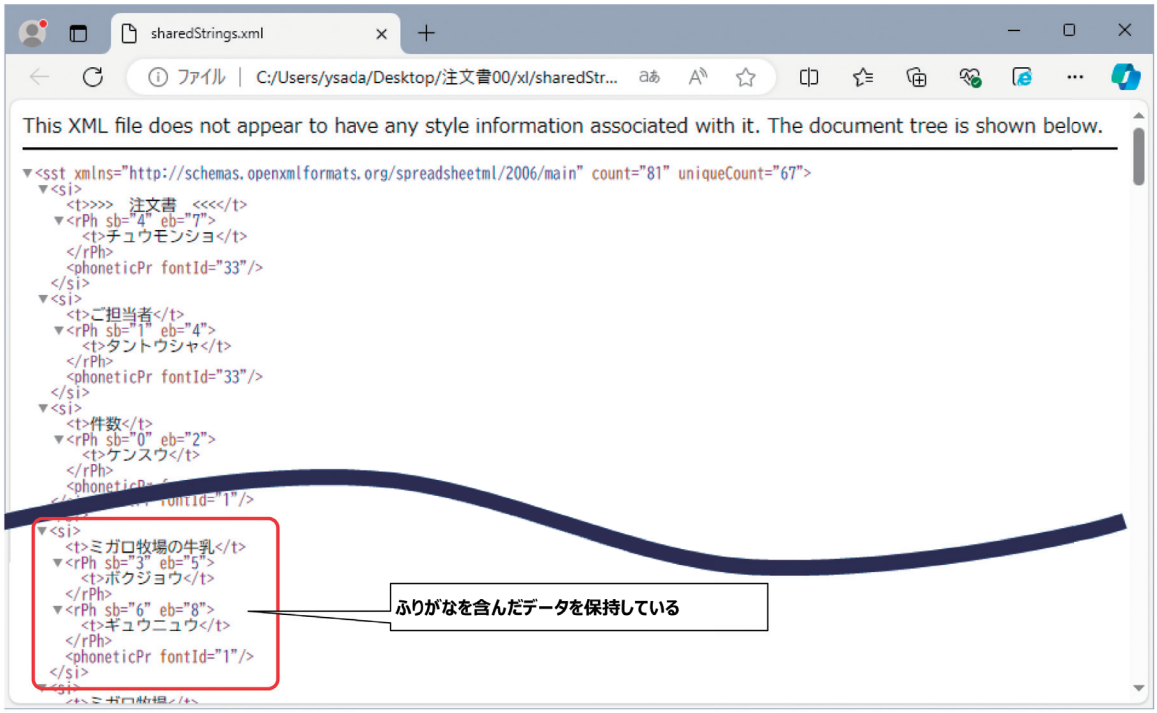


図 10　XML内の文字列セルのデータ部分



このXMLの構造をテキストベースで解読していくと、『<si>～～～</si>』の部分が1つのセルの値になっていて、その中にある『<t>～～～</t>』の部分を全て結合してセル値として返されているのが読み取れる。
さらにXML内の法則を探っていくと、『<rPh～～>～～</rPh>』という部分で、何文字目から何文字目までにふりがなを割り当てているという設定も読み取れる。従って、この部分をセルの値として返さないようにExcel4Delphiのソースを修正していけばよい。

Excel4Delphiでは、ブックのオープン時に内部の全ての情報が読み込まれている。そのため、Excel4Delphi側のソースでブックの読み込みを行っている「Excel4Delphi.Stream.pas」を必要に応じてバックアップを取った上で開く。その中のロジック(「ZEXSLXReadSharedStrings」関数内)に【ソース5】のように手を加えることで、『<t>～～～</t>』の部分を結合してセル値にするというロジックのうち『<rPh～～>～～～</rPh>』部分の中にある『<t>～～～</t>』を無視させることができる。
これでXLSXファイルにある各セルの内容をふりがな抜きで表示させることができる。

ソース 5

Excel4Delphi セル値の文字列からふりがなを除去

```
// ※「Excel4Delphi.Stream.pas」ユニット内（★★注釈箇所のみを追加する）
function ZEXSLXReadSharedStrings(var Stream: TStream; out StrArray: TStringDynArray;
out StrCount: Integer): Boolean;
var
  Xml: TZsspXMLReaderH;
  s: string;
  k: Integer;
  rs: TRichString;
  brPh: Boolean; // ★★ MIGARO ADD 判定用フラグ
begin
  Result := false;
  Xml := TZsspXMLReaderH.Create();
  try
【中　略】
              else if Xml.IsTagClosedByName('charset') then
                rs.Font.charset := StrToIntDef(Xml.Attributes['val'], 0);
            end;
          end;
// ★★ MIGARO ADD begin 「rPh」で始まるタグ内は後続処理で読ませない
          if Xml.IsTagStartByName('rPh') then
          begin
            brPh := True;     // 判定用フラグをTrueにする
          end;
// ★★ MIGARO ADD end
          if Xml.IsTagEndByName('t') then
          begin
            if (k > 1) then
              s := s + sLineBreak;
// ★★ MIGARO ADD begin 「rPh」で始まるタグ内では読まない
            if (brPh) then
            begin
              brPh := False; // 判定用フラグをFalseに戻す
            end
            else
// ★★ MIGARO ADD end  ↓↓この行は「rPh」で始まるタグ内以外でのみ読む
              s := s + Xml.TextBeforeTag;
          end;
          if Xml.IsTagEndByName('r') then
【中　略】
    Result := true;
  finally
    Xml.Free();
  end;
end; // ZEXSLXReadSharedStrings
```

3-2. 読み込んだブックの内容をワークに更新

前項で実装したロジックを基に、本項では必要なセルの値を読み取ってIBM iのワークファイルに更新する処理を作成する。画面にFireDAC接続を行うためのTFDConnection・TFDQuery・TFDPhysCO400DriverLinkを配置し、画面表示時処理の中にIBM iへの接続ロジックを記述する。また画面クローズ時にIBM iからの切断処理を記述する。
FireDACの接続や切断についての詳細は、過去のミガロ.テクニカルレポート「FireDAC実践プログラミングテクニック」を参考に設定して頂きたい(https://www.migaro.co.jp/tr/no11/tech/11_01_02.pdf)。

「ワークに更新」ボタンを押した時の処理を【ソース6】に記述する。ワークファイルは前項で作成した【図6】のレイアウトを使用しており、ヘッダー・明細いずれの項目も横持ちするようフィールド設計している。日付値の更新については【ソース6】で記載のように、5桁のデフォルト値(1899/12/30からの通算日数)をYYYYMMDD形式の8桁の整数に変換して更新している。更新結果は【図11】のようなイメージとなっている。

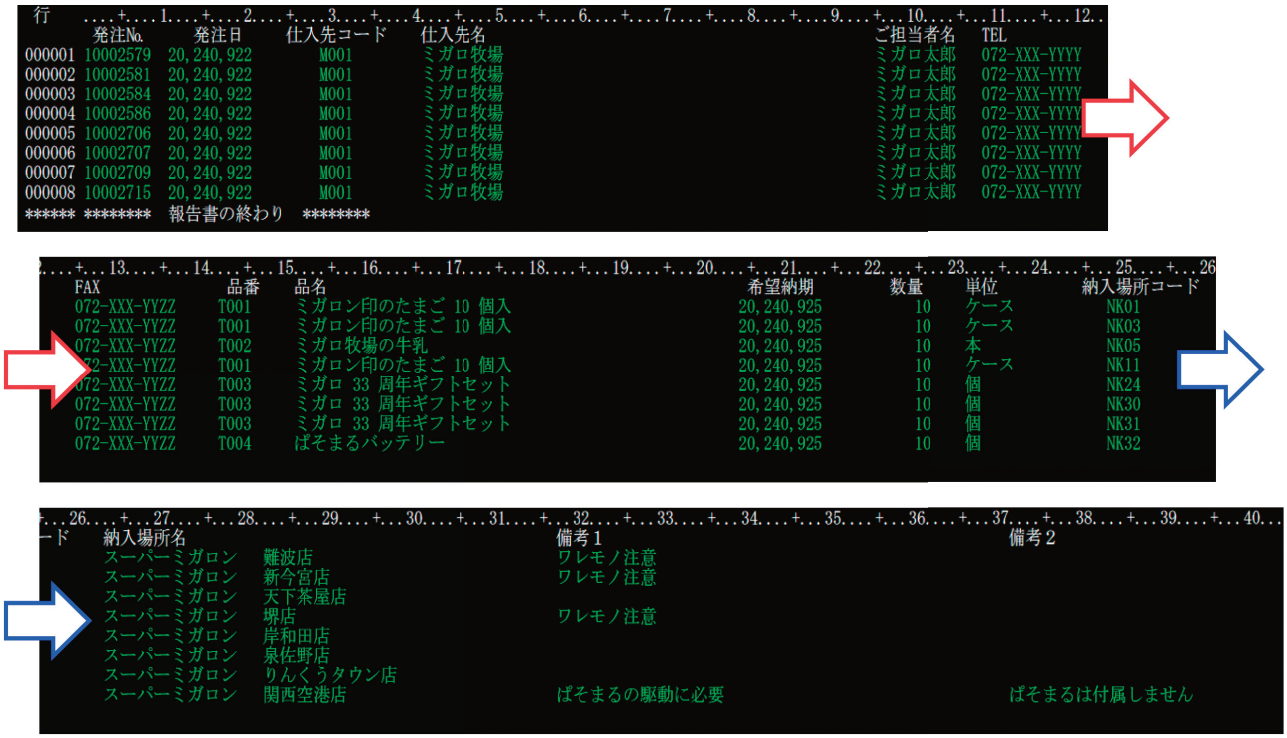
ソース 6

Excel4Delphi ワークに更新ボタン

```
{*****************************************************************************
  ワークに更新ボタン
*****************************************************************************}
procedure TForm1.btnXLStoXWRKClick(Sender: TObject);
var
  i: Integer;       // for文用
  sTEMP: String;    // 日付計算用
  dTEMP: TDateTime; // 日付計算用
begin
  // 未オープンエラーの防止
  if not bOpened then
  begin
    ShowMessage('ブックが開かれていません。');
    Abort;
  end;

  // ※※ワークファイルの初期化は省略、必要に応じて行う※※

  // 更新SQL設定
  qryU.SQL.Text := ' INSERT INTO YSADALIB/WKTR17 ( ' +
                   ' WKHANO, WKHADT, WKSICD, WKSINM, ' +
                   ' WKTNNM, WKTEL,  WKFAX,  WKHICD, ' +
                   ' WKHINM, WKNOKI, WKSURY, WKTANI, ' +
                   ' WKNOCD, WKNONM, WKBIK1, WKBIK2 ' +
                   ' ) VALUES ( ' +
                   ' :WKHANO, :WKHADT, :WKSICD, :WKSINM, ' +
                   ' :WKTNNM, :WKTEL,  :WKFAX,  :WKHICD, ' +
                   ' :WKHINM, :WKNOKI, :WKSURY, :WKTANI, ' +
                   ' :WKNOCD, :WKNONM, :WKBIK1, :WKBIK2) ';

  with workBook.Sheets[0] do  // 対象ブックの最初のシートを参照
  begin
    // ヘッダー項目をパラメータにセット
    qryU.ParamByName('WKSICD').AsString  := Cell[3, 2].AsString; // D3 仕入先CD
    qryU.ParamByName('WKSINM').AsString  := Cell[4, 2].AsString; // E3 仕入先名
    qryU.ParamByName('WKTNNM').AsString  := Cell[3, 3].AsString; // D4 ご担当者名
    qryU.ParamByName('WKTEL').AsString   := Cell[7, 3].AsString; // H4 TEL
    qryU.ParamByName('WKFAX').AsString   := Cell[7, 4].AsString; // H5 FAX
```

```
    // Q3 発注日（YYYYMMDD形式の整数に変換）
    sTEMP := Cell[16, 2].AsString;
    dTEMP := StrToIntDef(sTEMP, 0);
    qryU.ParamByName('WKHADT').AsInteger :=
                                    StrToInt(FormatDateTime('YYYYMMDD', dTEMP));

    // 明細項目をパラメータにセットし、更新を行う
    for i := 1 to 12 do // 8～31行目の明細部の値をセット
    begin
      if (Cell[2, (i*2)+5].AsString <> '') then // 発注No.が空の行を除く
      begin
        // C8～30 発注No.
        qryU.ParamByName('WKHANO').AsString  := Cell[2, (i*2)+5].AsString;
        // D8～30 品番
        qryU.ParamByName('WKHICD').AsString  := Cell[3, (i*2)+5].AsString;
        // E8～30 品名
        qryU.ParamByName('WKHINM').AsString  := Cell[4, (i*2)+5].AsString;
        // I8～30 希望納期（YYYYMMDD形式の整数に変換）
        sTEMP := Cell[8, (i*2)+5].AsString;
        dTEMP := StrToIntDef(sTEMP, 0);
        qryU.ParamByName('WKNOKI').AsInteger :=
                                      StrToInt(FormatDateTime('YYYYMMDD', dTEMP));
        // K8～30 数量
        qryU.ParamByName('WKSURY').AsInteger :=
                                    StrToIntDef(Cell[10,(i*2)+5].AsString, 0);
        // L8～30 単位
        qryU.ParamByName('WKTANI').AsString  := Cell[11,(i*2)+5].AsString;
        // M8～30 納入場所コード
        qryU.ParamByName('WKNOCD').AsString  := Cell[12,(i*2)+5].AsString;
        // N8～30 納入場所名
        qryU.ParamByName('WKNONM').AsString  := Cell[13,(i*2)+5].AsString;
        // O8～30 備考１
        qryU.ParamByName('WKBIK1').AsString  := Cell[14,(i*2)+5].AsString;
        // O9～31 備考２
        qryU.ParamByName('WKBIK2').AsString  := Cell[14,(i*2)+6].AsString;

        // 更新実行
        qryU.ExecSQL;
      end;
    end;

    ShowMessage('ワークに更新しました。');

    // ※※このあとRPGを呼び出す等の方法で更新をおこなう※※

  end;
end;
```

図 11　ワークへの更新結果イメージ

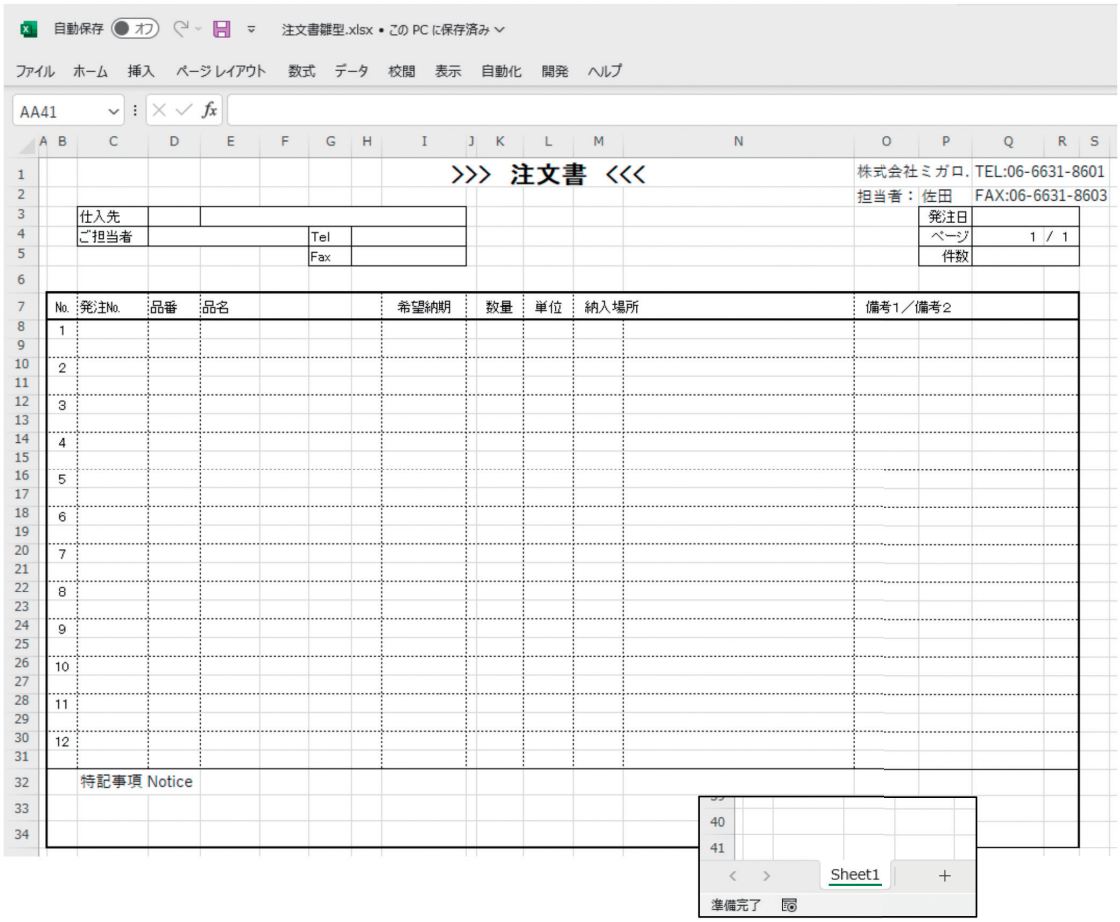


なおここではExcelから取得した文字列がワークのフィールド長に対して長すぎた場合の桁あふれチェックについては考慮していない。
必要に応じて弊社の技術Tipsにある桁あふれ対策の記事(https://www.migaro.co.jp/tips/2910/)を参照いただきたい。

3-3. ワークから読み込んだ内容をXLSXブックに更新

ここからは、IBM iのワークファイルから読み込んだ内容を、Excel4Delphiを使ってXLSXのブックに更新するサンプルを作成していく。今回はサンプルなので、前項で使用したワークファイルのレイアウトをそのまま流用する。また雛型となるXLSXのブックについても前項の注文書のフォーマットを流用し、**【図12】**のように値をクリアしたものを用意しておく。
画面にデータ参照用のTFDQueryと出力処理を記述するためのTButtonを配置し、データ参照およびExcel出力用のロジックを**【ソース7】**のように記述する。

図 12　注文書フォーマットの雛形イメージ



ソース 7

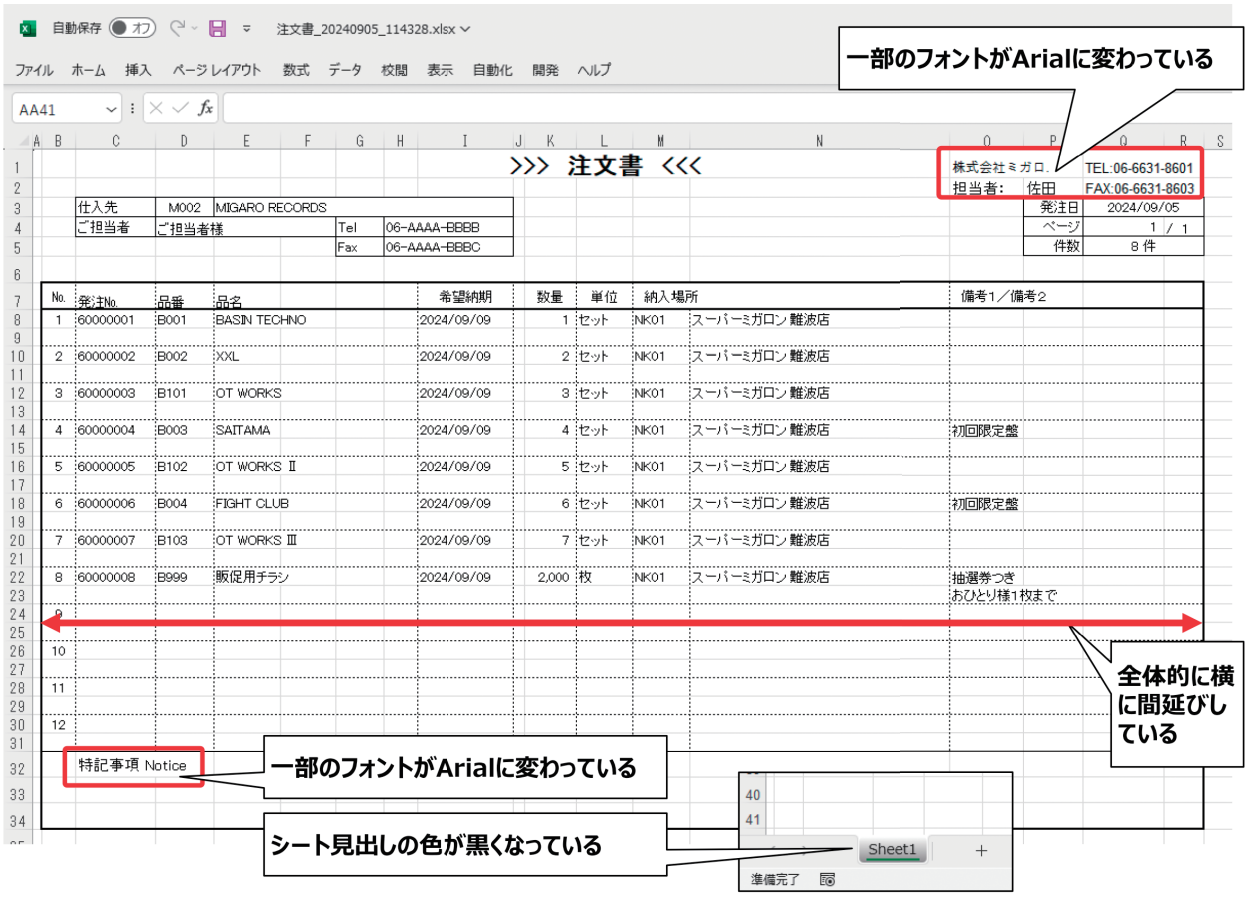
Excel4Delphi ワークから出力ボタン

```
{*******************************************************************************
  ワークから出力ボタン
*******************************************************************************}
procedure TForm1.btnWRKtoXLSXClick(Sender: TObject);
var
  iCnt: Integer;    // 明細行数保管用
  sTEMP: String;    // ファイル名計算用
begin

  // 取得SQL設定
  qryS.Close;
  qryS.SQL.Text := ' SELECT * FROM YSADALIB/WKTR17 ' +
                   ' WHERE (WKHADT = :WKHADT) ';

  // データ取得（今回はサンプルのため、特定発注日のデータのみを対象とする）
  qryS.ParamByName('WKHADT').AsInteger := 20240905;
  qryS.Open;

  // 雛型のブックを開く
  if bOpened then
  begin
    ShowMessage('現在開かれているブックを閉じます。');
  end;
```

```
  // フルパスを指定して雛型のブックを開く（EXEと同階層の固定ファイル名）
  workBook.LoadFromFile(ExtractFilePath(ParamStr(0)) + '注文書雛型.xlsx');
  bOpened := True;   // オープン済フラグ

  // 先頭シートのセルに値をセット
  with workBook.Sheets[0] do
  begin
    Cell[16,2].AsString := FormatFloat(
          'OOOO/OO/OO', qryS.FieldByName('WKHADT').AsInteger); // Q3 発注日
    Cell[3, 2].AsString := qryS.FieldByName('WKSICD').AsString; // D3 仕入先CD
    Cell[4, 2].AsString := qryS.FieldByName('WKSINM').AsString; // E3 仕入先名
    Cell[3, 3].AsString := qryS.FieldByName('WKTNNM').AsString; // D4 ご担当者名
    Cell[7, 3].AsString := qryS.FieldByName('WKTEL').AsString;  // H4 TEL
    Cell[7, 4].AsString := qryS.FieldByName('WKFAX').AsString;  // H5 FAX
    Cell[16,4].AsInteger := qryS.RecordCount;                   // Q5 件数

    iCnt := 0;
    while not(qryS.Eof) do
    begin
      Inc(iCnt);

      // C8～30 発注No.
      Cell[2, (iCnt*2)+5].AsString  :=   qryS.FieldByName('WKHANO').AsString ;
      // D8～30 品番
      Cell[3, (iCnt*2)+5].AsString  :=   qryS.FieldByName('WKHICD').AsString ;
      // E8～30 品名
      Cell[4, (iCnt*2)+5].AsString  :=   qryS.FieldByName('WKHINM').AsString ;
      // I8～30 希望納期
      Cell[8, (iCnt*2)+5].AsString  := FormatFloat(
                          'OOOO/OO/OO', qryS.FieldByName('WKNOKI').AsInteger);
      // K8～30 数量
      Cell[10,(iCnt*2)+5].AsInteger := qryS.FieldByName('WKSURY').AsInteger;
      // L8～30 単位
      Cell[11,(iCnt*2)+5].AsString  :=   qryS.FieldByName('WKTANI').AsString ;
      // M8～30 納入場所コード
      Cell[12,(iCnt*2)+5].AsString  :=   qryS.FieldByName('WKNOCD').AsString ;
      // N8～30 納入場所名
      Cell[13,(iCnt*2)+5].AsString  :=   qryS.FieldByName('WKNONM').AsString ;
      // O8～30 備考１
      Cell[14,(iCnt*2)+5].AsString  :=   qryS.FieldByName('WKBIK1').AsString ;
      // O9～31 備考２
      Cell[14,(iCnt*2)+6].AsString  :=   qryS.FieldByName('WKBIK2').AsString ;

      qryS.Next;
    end;
  end;

  // 書式の設定（【ソース8】参照）

  // ブックの保存処理
  sTEMP := '注文書_' + FormatDateTime('YYYYMMDD_HHNNSS', Now) + '.xlsx';
  workBook.SaveToFile(ExtractFilePath(ParamStr(0)) + sTEMP);

  bOpened := False;  // オープン済フラグ
  ShowMessage('ブックを保存しました。');
end;
```

プロジェクトをコンパイルして実行してみると、ワークファイルにあらかじめ登録しておいたデータが参照され、XLSXのブックにそのデータが【図13】のように出力されていることを確認できる。

図 13　そのまま帳票出力した結果



ここで雛型のブックとレイアウトを見比べてみると、以下のような見た目の差異が見つかった。

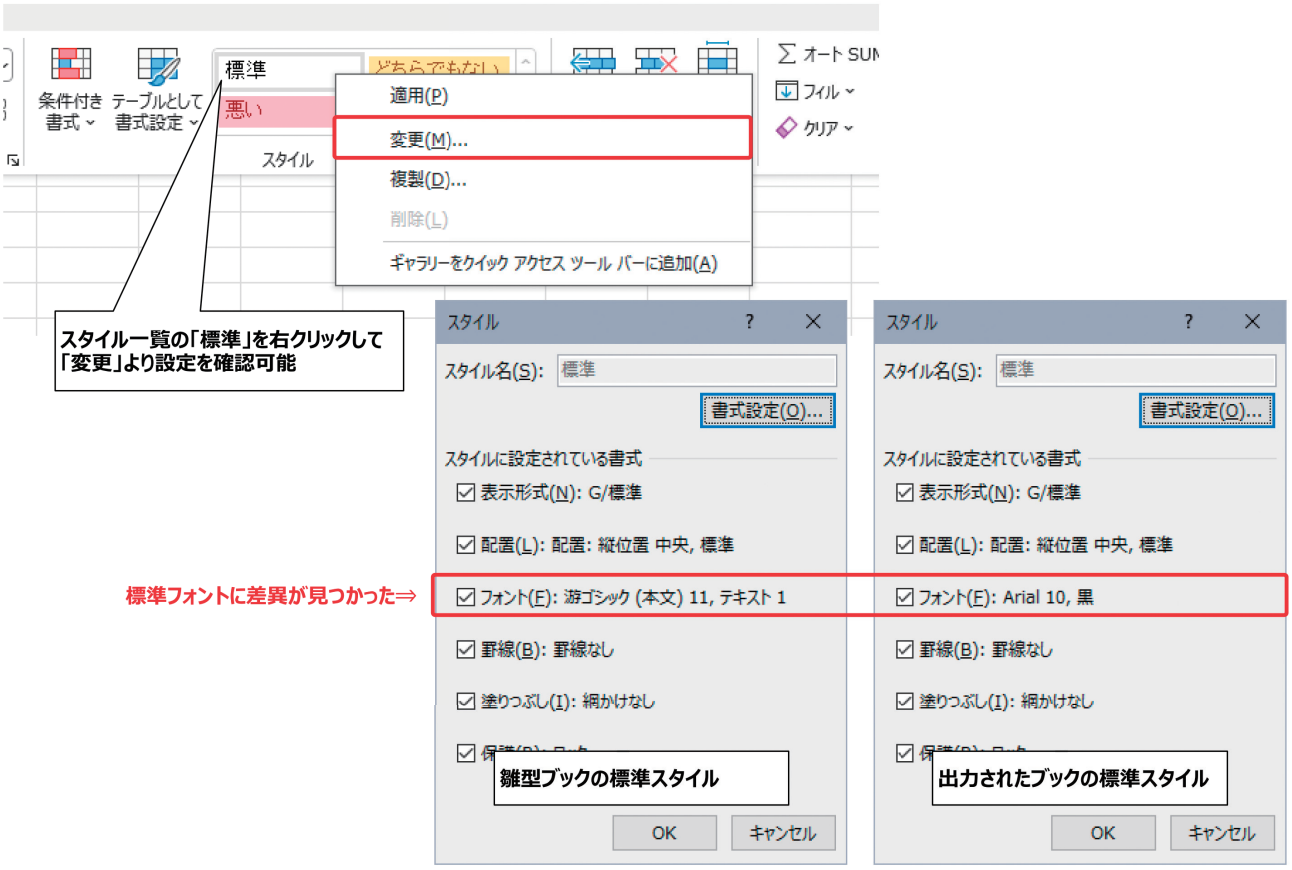
・標準設定のままのセルのフォントが欧文標準の「Arial」に変わり、全体的に横に間延びしている。

・シート見出しの色が黒に変わっている。

ここからは、これらの課題を1つずつ調査・解決していきたい。

まずは欧文フォントを元に戻していく。Excel4DelphiとExcelの仕組みを解析した結果、元のブックでスタイルが標準になっている（セルの書式設定がデフォルトのままの）セルにおいて発生するようだ。これを解決するには、標準スタイルのフォントを元のExcelと揃えればよい。Excelにおいて、各ブックの標準スタイルは【図14】の手順で参照や変更ができる。

図 14　ブックの標準スタイルの確認



元のブックが作成されたExcelのバージョンやOSの言語によって標準スタイルは異なるようだが、Excel4Delphiの標準スタイルと今回の元のブックで差異があると【図13】のような見た目になり、フォントや改ページ位置などに差異が発生する。

元のブックと出力したブックで標準スタイルの差異が確認できたら、個別ソース側でExcel出力前（保存の直前）に【ソース8】のようにロジックを追加する。これで出力されるブックの標準スタイルが元のブックと統一され、フォントや改ページ位置を揃えることができるだろう。全体的に間延びしていた各列の横幅もこの修正によって改善する。

ソース 8

Excel4Delphi 標準スタイルのフォントを元のブックと合わせる

```
// 標準スタイルのフォントを元のブックと合わせてフォントや印刷範囲を直す
  workBook.Styles.DefaultStyle.Font.Name := '游ゴシック';
  workBook.Styles.DefaultStyle.Font.Size := 11;
```

続いてシート見出しの色が黒くなっている現象を元に戻す。この原因はシート見出しの色を扱うXML内の「tabColor」という要素が、Excel4Delphiで保存時には色なしのシートも含めて必ず設定されていることにあるようだ(そのため、元から色を設定しているシートは正しい色で保存される)。元々見出しの色が無かったシートにはダミーの色が更新されているため、これを無効化していく。

修正手順としては、ブックへの書き込みを行っている「Excel4Delphi.Stream.pas」のロジック(「ZEXLSXCreateSheet」関数内)に【ソース9】のように手を加えると、元々見出しの色が無かったシートには「tabColor」要素を書き込ませないことで、シートの色を正常化することができる。

ソース 9

Excel4Delphi シート見出しの色が無いシートには設定しない

```
// ※「Excel4Delphi.Stream.pas」ユニット内（★★注釈箇所のみを追加する）
function ZEXLSXCreateSheet(var XMLSS: TZWorkBook; Stream: TStream; SheetNum: Integer;
  var SharedStrings: TStringDynArray; const SharedStringsDictionary: TDictionary<string,
Integer>;
  TextConverter: TAnsiToCPConverter; CodePageName: String; BOM: ansistring;
  const WriteHelper: TZEXLSXWriteHelper): Integer;
var
  Xml: TZsspXMLWriterH; // писатель
  sheet: TZSheet;
  procedure WriteXLSXSheetHeader();
  var
    s: string;
    b: Boolean;
    SheetOptions: TZSheetOptions;
    procedure _AddSplitValue(const SplitMode: TZSplitMode; const SplitValue: Integer;
const AttrName: string);
    var
      s: string;
      b: Boolean;
    begin
【中　略】
    end; // _AddSplitValue
    procedure _AddTopLeftCell(const VMode: TZSplitMode; const vValue: Integer; const
HMode: TZSplitMode;
      const hValue: Integer);
    var
      isProblem: Boolean;
    begin
【中　略】
    end; // _AddTopLeftCell
  begin
    Xml.Attributes.Clear();
    Xml.Attributes.Add('filterMode', 'false');
    Xml.WriteTagNode('sheetPr', true, true, false);
    if (sheet.TabColor <> 5) then //★★ MIGARO ADD シート見出しの色がある時のみ
    begin                         //★★ MIGARO ADD
    Xml.Attributes.Clear();
    Xml.Attributes.Add('rgb', 'FF' + ColorToHTMLHex(sheet.TabColor));
    Xml.WriteEmptyTag('tabColor', true, false);
    end;                          //★★ MIGARO ADD
    Xml.Attributes.Clear();
    if sheet.ApplyStyles then
【以下略】
```

色が無いシートにはダミーで「5」が設定されてしまうのを防ぐ

ここまでの修正を行うことで、【図15】のように元の雛型に近い書式設定のブックを生成することができた。内部でXMLを生成している都合上元のブックとミリメートル単位で完全に同じとはいかないが、内容の確認は十分に可能だろう。

図 15　標準スタイル設定後の出力ブックのイメージ

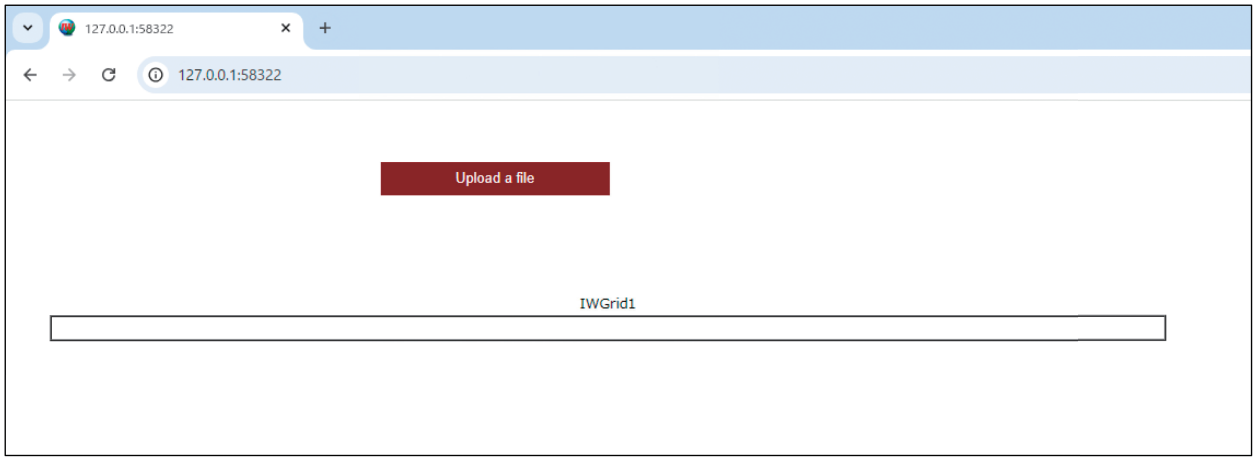
4. IntraWebでのExcel4Delphi活用

前章まではExcel4Delphiの概要や基本的な使い方を紹介したが、ここからは応用編として、OLEが使用できないIntraWebで想定される活用パターンを紹介する。

IntraWebのモジュールはWebサーバー上で動作するため、OLEを使ったExcelの取込や出力を行うことができない。リクエストの度にWebサーバー側でExcelを起動して処理を行うことが現実的でないためである。Excel4DelphiのようなExcelが不要なツールであれば、この課題を解決できる。

本項では、前章で取込に使用したXLSXブックをIntraWebの画面上に取り込んでグリッドに表示させるサンプルを作成する。まずはIntraWebのプロジェクトを新規作成し、IWForm1の画面上にファイルアップロードに使用するTIWFileUploaderと、取り込んだ結果を表示させるためのTIWGridを配置する。この時点でプロジェクトをコンパイルしてブラウザで実行すると【図16】のような画面が表示される。

図 16　IntraWeb　初期画面



ここからXLSXブックの取込を実装していく。まずは画面に配置したTIWGridのColumnCountプロパティを「7」に設定し、ソースのuses節に前章のサンプルと同じように「Excel4Delphi」「Excel4Delphi.Stream」を追加する。次にString変数「sFileName」をグローバル変数に定義し、TIWFileUploaderのファイルアップロード時の処理を【ソース10】のように記述する。今回はサンプルのため、今アップロードしたファイルの名前を便宜上グローバル変数に保持しておく目的である。また、TIWButton（取り込みボタン）を画面に配置し、そのOnClick処理を【ソース11】のように記述する。

ソース 10

Excel4Delphi IntraWeb でブックのアップロード完了時

```
{*******************************************************************************
  IWFileUploader ブックのアップロード完了時処理
*******************************************************************************}
procedure TIWForm1.IWFileUploader1AsyncUploadCompleted(Sender: TObject;
  var DestPath, FileName: string; var SaveFile, Overwrite: Boolean);
begin
  // 引数をグローバル変数に保管
  sFileName := FileName;    // アップロードしたファイル名
end;
```

ソース 11

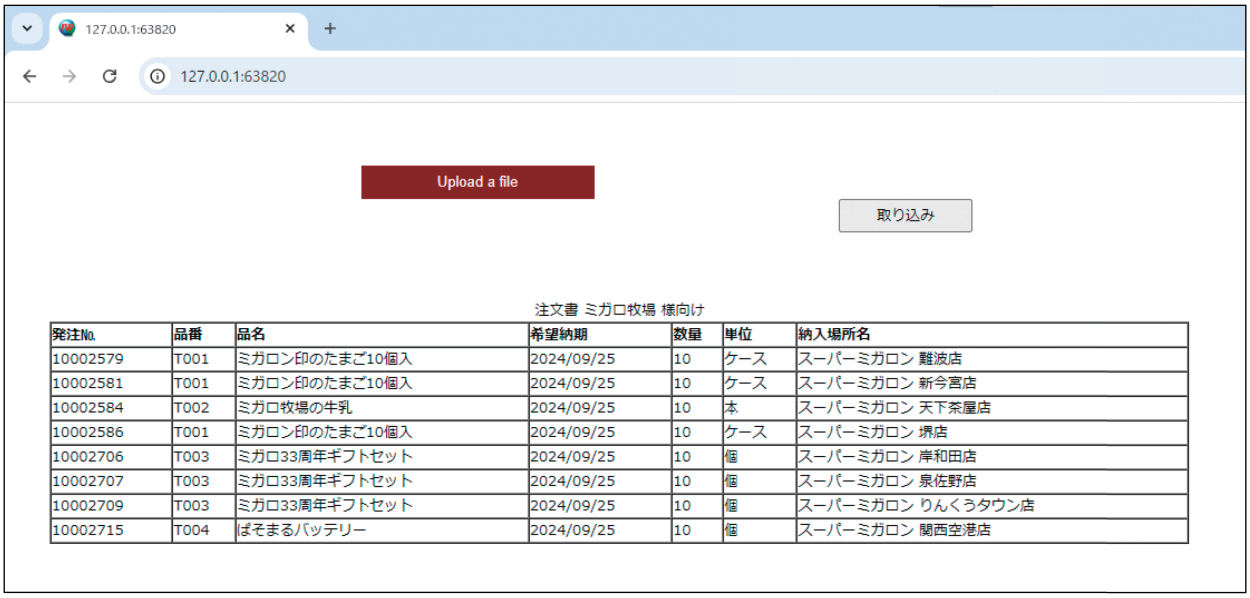
Excel4Delphi IntraWeb で取り込みボタン押下時

```
{*******************************************************************************
  IWFileUploader 取り込みボタン押下時
*******************************************************************************}
procedure TIWForm1.IWButton1Click(Sender: TObject);
var
  workBook: TZWorkBook; // Excel4Delphiのクラス
  i: Integer;       // for文用
  sTEMP: String;    // 日付計算用
  dTEMP: TDateTime; // 日付計算用
begin
  // ファイルの形式チェック （XLSX・XLSM以外をエラーにする）
  if (ExtractFileExt(UpperCase(sFileName)) <> '.XLSX') and
     (ExtractFileExt(UpperCase(sFileName)) <> '.XLSM') then
  begin
    WebApplication.ShowMessage('ファイルの形式が正しくありません。');
    Exit; // Abortすると画面にエラーが出るためExitで抜ける
  end;

  // 行カウント初期化
  IWGrid1.RowCount := 1;
  // グリッドのタイトル設定
  IWGrid1.Cell[0, 0].Text := '発注No.';
  IWGrid1.Cell[0, 1].Text := '品番';
  IWGrid1.Cell[0, 2].Text := '品名';
  IWGrid1.Cell[0, 3].Text := '希望納期';
  IWGrid1.Cell[0, 4].Text := '数量';
  IWGrid1.Cell[0, 5].Text := '単位';
  IWGrid1.Cell[0, 6].Text := '納入場所名';

  // TZWorkBookクラス生成
  workBook := TZWorkBook.Create(nil);
  try
    // キャッシュフォルダ内のパスを指定してブックを開く
    workBook.LoadFromFile(WebApplication.UserCacheDir + sFileName);

    // Excel4Delphiでブックを開き、明細に表示する
    // ※サンプルのため、ここでは一部フィールドのみ
    with workBook.Sheets[0] do  // 対象ブックの最初のシートを参照
    begin
      // E3 仕入先名（ここではグリッドのタイトルにする）
      IWGrid1.Caption := '注文書 ' + Cell[4, 2].AsString + ' 様向け';

      // 明細項目をパラメータにセットし、更新を行う
      for i := 1 to 12 do // 8～31行目の明細部の値をセット
      begin
        if (Cell[2, (i*2)+5].AsString <> '') then // 発注No.が空の行を除く
        begin
          // 行追加
          IWGrid1.RowCount := IWGrid1.RowCount + 1;
```

```
            // C8～30 発注No.
            IWGrid1.Cell[i, 0].Text := Cell[2, (i*2)+5].AsString;
            // D8～30 品番
            IWGrid1.Cell[i, 1].Text := Cell[3, (i*2)+5].AsString;
            // E8～30 品名
            IWGrid1.Cell[i, 2].Text := Cell[4, (i*2)+5].AsString;
            // I8～30 希望納期（書式を設定してセット）
            sTEMP := Cell[8, (i*2)+5].AsString;
            dTEMP := StrToIntDef(sTEMP, 0);
            IWGrid1.Cell[i, 3].Text := FormatDateTime('YYYY/MM/DD', dTEMP);
            // K8～30 数量
            IWGrid1.Cell[i, 4].Text := Cell[10,(i*2)+5].AsString;
            // L8～30 単位
            IWGrid1.Cell[i, 5].Text := Cell[11,(i*2)+5].AsString;
            // N8～30 納入場所名
            IWGrid1.Cell[i, 6].Text := Cell[13,(i*2)+5].AsString;
          end;
        end;
      end;

    finally
      FreeAndNil(workBook); // TZWorkBookクラス解放
    end;
  end;
```

ここまで実装した状態でプロジェクトをコンパイルしてブラウザで実行すると、【図17】のようにブックが取り込まれて内容を確認することができる。

図 17　IntraWeb XLSXブックを取り込んで明細に表示



発注No.	品番	品名	希望納期	数量	単位	納入場所名
10002579	T001	ミガロン印のたまご10個入	2024/09/25	10	ケース	スーパーミガロン 難波店
10002581	T001	ミガロン印のたまご10個入	2024/09/25	10	ケース	スーパーミガロン 新今宮店
10002584	T002	ミガロ牧場の牛乳	2024/09/25	10	本	スーパーミガロン 天下茶屋店
10002586	T001	ミガロン印のたまご10個入	2024/09/25	10	ケース	スーパーミガロン 堺店
10002706	T003	ミガロ33周年ギフトセット	2024/09/25	10	個	スーパーミガロン 岸和田店
10002707	T003	ミガロ33周年ギフトセット	2024/09/25	10	個	スーパーミガロン 泉佐野店
10002709	T003	ミガロ33周年ギフトセット	2024/09/25	10	個	スーパーミガロン りんくうタウン店
10002715	T004	ぱそまるバッテリー	2024/09/25	10	個	スーパーミガロン 関西空港店

読み込みについて紹介は本稿ではここまでとするが、前章で紹介した手順を活用することで、取り込んだ内容をIBM iへ更新するロジックを作成することもできるだろう。

最後に、Excel4Delphiで出力したXLSXブックをブラウザからダウンロードさせる方法を紹介する。

IntraWebでファイルをダウンロードする際は、キャッシュディレクトリにファイルを出力した後にTFileStreamを生成してMIMEタイプを指定の上、SendStreamを行う。例えばExcel4Delphiに付属している**【図3】【図4】**のデモプログラムで作成できるものと同じXLSXブックを出力する場合は**【ソース12】**のように記述することで、ブラウザからダウンロードできるようになる。

ソース 12

Excel4Delphi IntraWeb での XLSX ブック出力処理

```
{*******************************************************************************
  ブックの作成出力処理
*******************************************************************************}
procedure TIWForm1.IWButton2Click(Sender: TObject);
var
  sXLSXname: String;          // XLSXのファイル名
  workBook: TZWorkBook;       // Excel4Delphiのクラス
  Strm : TStream;             // XLSX出力ストリーム
begin
  // XLSXのファイル名
  sXLSXname := FormatDateTime('YYYYMMDD_HHNNSS', Now) + '.xlsx';

  // TZWorkBookクラス生成
  workBook := TZWorkBook.Create(nil);
  try
    // XLSXを生成してキャッシュに出力（ここではDemoと同じロジック）
    workBook.Sheets.Add('Sheet1');
    workBook.Sheets[0].ColCount := 10;
    workBook.Sheets[0].RowCount := 10;
    workBook.Sheets[0].CellRef['A', 0].AsString := 'Hello';
    workBook.Sheets[0].RangeRef['A', 0, 'B', 2].Merge();
    workBook.SaveToFile(WebApplication.UserCacheDir + sXLSXname);

    // キャッシュにストリーム生成
    Strm := TFileStream.Create(
      WebApplication.UserCacheDir + sXLSXname, fmOpenRead or fmShareDenyNone);

    // MIMEを指定し、XLSXをブラウザからダウンロードする
    WebApplication.SendStream(Strm, True,
      'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
      sXLSXname);

  finally
    FreeAndNil(workBook); // TZWorkBookクラス解放
    // ストリームは解放しない（ダウンロードできなくなる）
  end;
end;
```

5. まとめ

本稿では、海外のExcel4Delphiライブラリを用いてDelphiとExcelと連携する手法を紹介した。

Excelの仕組みがブラックボックスになっているため、Excelのインストール不要で利用できるツールやライブラリは少ない。今回紹介したようなツールは業務アプリケーションの開発や運用において役立つシーンが多くなるだろう。
本稿が、Delphi/400プログラムとExcelの連携強化の一助となれば幸いである。

Delphi/400

直感的に使える操作性を工夫した Webシステム開発（IntraWeb）の機能紹介

株式会社ミガロ.
システム事業部システム１課
田村　洋一郎

略　歴

生年月日：1983年9月27日
最終学歴：2006年　近畿大学　理工学部卒業
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：

RPGやDelphi/400などの開発経験を経て、現在は要件定義から安定稼働フォローまで、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
宮坂　優大

略　歴

生年月日：1982年11月19日
最終学歴：2006年　近畿大学　理工学部卒業
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：

主にDelphi/400を利用したシステムの受託開発・保守をメインに担当。設計から運用・フォローまでプロジェクト全般に関わることが多い。

株式会社ミガロ.
システム事業部システム１課
都地　奈津美

略　歴

生年月日：1989年8月19日
最終学歴：2012年　関西学院大学　理工学部卒業
入社年月：2012年4月　株式会社ミガロ.入社
社内経歴：2012年4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受託開発とシステム保守を担当している。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

Webシステムの需要は増加し、当社でも多くの受託開発を行っている。オンラインショッピングサイトなどの外部ユーザー向けWebシステム（ECサイト）では、さまざまなユーザーが利用するため、直感的に使える操作性が求められる。そこで本稿では、システム開発チームが実際の開発時に工夫し、実装したユーザーインターフェース機能を紹介する。

また今回紹介するWebシステムの開発手法は、Delphi/400を活用したIntraWebによるものである。IntraWebを使用すると、従来の GUI アプリケーションと同様にWeb システムを構築することができるため、ぜひ活用して頂きたい。

本稿では、以下のUI機能を紹介する。
【第2章】直感的に操作しやすいカレンダー機能
【第3章】視覚的に分かりやすいローディング画面
【第4章】フレームを使用した明細グリッド
今回の機能を実装したサンプルを活用して頂けるよう、本稿の最後にダウンロードURLを記載している。本サンプルはダウンロード可能であるため、本稿へのソースの記載は一部抜粋とする。また、サンプルは、スタンドアロン　モードを使用して作成している。
なお本稿では、Delphi/400 11 AlexandriaとIntraWeb 15を使用し、動作環境は、MicrosoftのEdgeを採用する。また、Delphi/400におけるIntraWebの基本的な開発手順について本稿では割愛するが、ミガロ. のサイト内に詳しく紹介している資料があるのでそちらを参照して頂きたい。
<参照先>
https://www.migaro.co.jp/ts/27th/Session2.pdf

2. 直感的に操作しやすいカレンダー機能

本章では、カレンダー機能を紹介する。IntraWebには標準でTIWCalendarというカレンダーコンポーネントが存在する**【図1】**。複雑なソースコードを記述することなく、カレンダー機能を実装できる便利なコンポーネントである。

図 1　TIWCalendarのイメージ

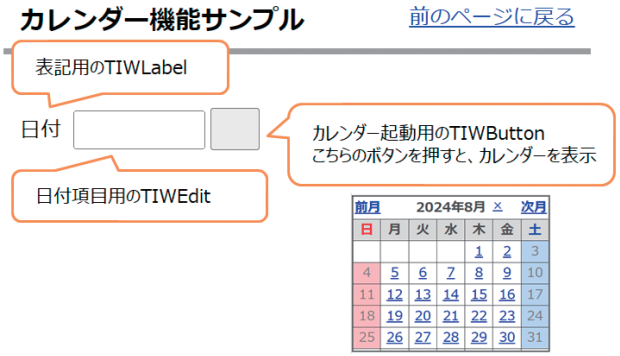
本章では、曜日毎に色やフォントを工夫し、直感的に操作しやすいカレンダー機能を実装する手法を紹介する。今回作成するカレンダーの完成イメージは**【図2】**の通りである。

図 2　今回紹介するカレンダーのイメージ

2-1.サンプル画面

カレンダー機能を実装するためのサンプル画面を【図3】の通りに作成する。

図 3　サンプル画面



本稿で紹介するサンプルプログラムでは、トップページ【図4】を作成し、各章のサンプル画面に遷移可能としている。各章のサンプル画面には、トップページに戻るための「前のページに戻る」(TIWLink)を画面右上に配置している。

図 4　トップページ

2-2.カレンダー機能の構成

カレンダー機能は、ソースコードが煩雑にならないよう、クラス化による処理の共通化を行う。カレンダーのメイン処理を集約した「TIWSMPLCalendar」クラス、画面を閉じて呼出元の画面に戻るための処理を集約した「TIWSMPLLink」クラスの2つを作成する。これらのクラスを使用することで、カレンダー全体の機能を実装している【図5】。

図 5　カレンダー機能の構成

2-3.カレンダー（TIWSMPLCalendar）クラス

カレンダー（TIWSMPLCalendar）クラスの宣言部を**【ソース1】**、レイアウト情報（罫線、色、フォント等）や選択不可とする入力制御を行う処理を**【ソース2】**の通りに記述する。

ソース 1

カレンダー（TIWSMPLCalendar）クラスの宣言部

```
TIWSMPLCalendar = class(TIWCalendar)
  private
    { Private 宣言 }
    FYear, FMonth: Integer;                                        // 年月

    procedure CalendarRenderCell(ACell: TIWGridCell;
                               const ARow, AColumn: Integer);      // カレンダーのセル内容セット処
理
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;              // 生成時処理（コンストラクタ）
  end;
```

ソース 2

カレンダーのセル内容セット処理

```
procedure TIWSMPLCalendar.CalendarRenderCell(ACell: TIWGridCell; const ARow,
  AColumn: Integer);
var
  dDate: TDateTime;      // 日付
  iDate: Integer;        // 日付（数値）
  sDate: String;         // 日付（文字）
  iDayOfWeek: Integer;   // 曜日
begin
  inherited;

  // CSSのセット
  Css := 'calendar';

  // ----- タイトル(年月)セット処理 -----
  // タイトル部の場合
  if (ARow = 0) and (AColumn = 1) then
  begin
    // タイトル部をセット：yyyy年mm月
    if (Pos('年', ACell.Text) = 0) then
    begin
      // 年月を保管
      FYear  :=
        StrToIntDef(Copy(ACell.Text, Pos(' ', ACell.Text) + 1, Length(ACell.Text)), 0);
      FMonth :=
        StrToIntDef(Copy(ACell.Text, 1, Pos(' ', ACell.Text) - 2), 0);

      // セット
      ACell.Text := IntToStr(FYear) + '年' + IntToStr(FMonth) + '月';
    end;
  end;
```

レイアウト用のCSSのセレクタを指定
CSS側でレイアウト情報(罫線、色、フォント等)を制御

タイトルに
年月を表示

```
  // ----- 選択不可の設定曜日 -----
  if (ARow > 1) and (ACell.Text <> '') then
begin
    // 日付
    iDate := (FYear * 10000) + (FMonth * 100) + StrToIntDef(ACell.Text, 0);
    sDate := FormatFloat('0000/00/00', iDate);
    dDate := StrToDateTime(sDate + FormatDateTime('hh:mm:ss', Now));

    // 曜日を取得
    iDayOfWeek := System.DateUtils.DayOfTheWeek(dDate);

    // CSS側で処理するため、特定のCSSをセット
    // 土曜日の場合、選択不可
    if (iDayOfWeek = 6) then
    begin
      Css := Trim(Css + ' cal_disable cal_blue');
    end
    // 日曜日の場合、選択不可
    else if (iDayOfWeek = 7) then
    begin
      Css := Trim(Css + ' cal_disable cal_red');
    end;
  end;
end;
```

土曜日、日曜日を判定し、対象のCSSのセレクタを指定CSS側でセルの色や選択不可を制御

次に、カレンダー（TIWSMPLCalendar）クラスの生成時処理にカレンダー全体の基本情報や描画イベントに【ソース2】の関数を指定する【ソース3】。

ソース 3

カレンダー（TIWSMPLCalendar）の生成時処理

```
constructor TIWSMPLCalendar.Create(AOwner: TComponent);
begin
  inherited;

  // 初期化
  Caption      := '';   // キャプション
  Text         := '';   // テキスト
  Height       := 205;  // 高さ
  Width        := 3     // 幅
  ZIndex       := 0;    // 位置

  // フォント名
  Font.FontName                := cFontName;
  CalendarFont.FontName        := cFontName;
  CalendarHeaderFont.FontName  := cFontName;
  // フォントサイズ
  Font.Size                 := cFontSize + 2;
  CalendarFont.Size         := cFontSize + 2;
  CalendarHeaderFont.Size   := cFontSize + 2;
  // フォント色
  Font.Color                 := cFontColor;
  CalendarFont.Color         := cFontColor;
  CalendarHeaderFont.Color   := cFontColor;

  // タイトルは太字
  CalendarHeaderFont.Style := [fsBold];
```

カレンダー全体の基本情報を指定

```
    // 色
    BorderColors.Color      := clNone;
    BorderColors.Light      := clNone;
    BorderColors.Dark       := clNone;
    BGColor                 := clNone;
    CalendarColor           := clNone;
    AlternateCalendarColor  := clNone;
    CalendarHeaderColor     := clNone;

    // キャプション
    CaptionPrevious  := '前月';
    CaptionNext      := '次月';

    // フレーム枠を描画
    UseFrame  := True;

    // 週始まりは日曜日
    WeekStarts  := wsSunday;

    // Asyncモード
    AsyncMode  := True;
    // AsyncイベントでのSubmit
    SubmitOnAsyncEvent  := False;

    // カレンダーのセル内容セット処理
    OnRenderCell  := CalendarRenderCell;
  end;
```

描画イベントに先程記述した「カレンダーのセル内容セット処理」を指定

2-4.画面を閉じるリンク(TIWSMPLLink)クラス

画面を閉じるリンク(TIWSMPLLink)クラスの宣言部を**【ソース4】**の通りに記述する。

ソース 4

画面を閉じるリンク (TIWSMPLLink) クラスの宣言部

```
TIWSMPLLink = class(TIWLink)
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;                    // 生成時処理(コンストラクタ)
  end;
```

次に、画面を閉じるリンク(TIWSMPLLink)クラスの生成時処理に基本情報を指定する【ソース5】。

ソース 5

画面を閉じるリンク（TIWSMPLLink）の生成時処理

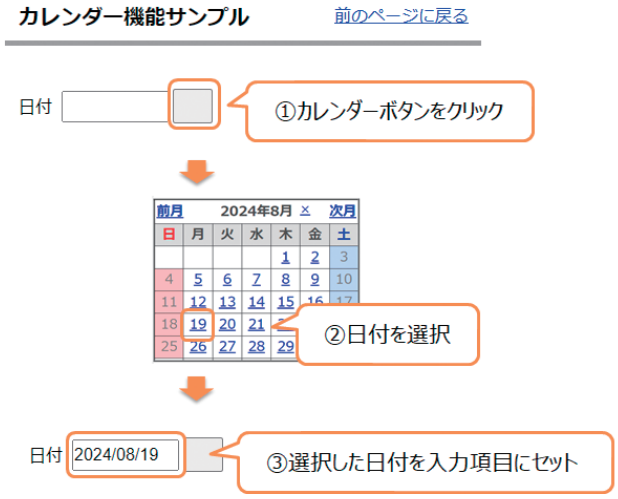
```
constructor TIWSMPLLink.Create(AOwner: TComponent);
begin
  inherited;

  // 初期化
  Font.FontName := cFontName;    // フォント名
  Font.Size     := cFontSize;    // フォントサイズ
  Font.Color    := cFontColor;   // フォント色
  Height        := 19;           // 高さ
  ZIndex        := 0;            // 位置

  // ボタンのAsyncClick時、画面をロック(ウエイトカーソルを表示)するよう変更
  LockOnAsyncEvents := [aeClick];

  // Asyncイベントでの Submit
  SubmitOnAsyncEvent := False;
end;
```

画面を閉じるリンクの基本情報を指定

2-5. カレンダーボタンクリック時処理

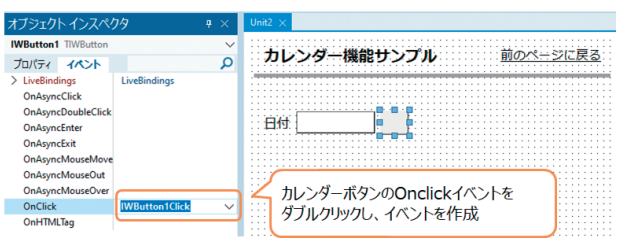
前述の2-3～2-4で実装したクラスを使用し、カレンダーボタンクリック時にカレンダーを表示、カレンダーの日付選択時に入力項目に日付をセットする処理を記述する【図6】。

図 6 サンプル画面の挙動イメージ



まず、カレンダーボタンのOnClickイベントを作成する【図7】。次に、前述で実装したクラスを使用し、カレンダーを作成する処理を記述する【ソース6】。

図 7 OnClickイベントの作成



ソース 6

カレンダーボタンのクリック時処理

```
procedure TIWForm2.IWButton1Click(Sender: TObject);
var
  dSetDate: TDateTime;     // 日付値(セット用)
  iDate: Integer;          // 日付値
  objParent: TWinControl;  // 親オブジェクト
  regParent: TIWRegion;    // メインRegion
  iTopSelf: Integer;       // Top位置
begin
  // 初期化
  objParent := IWForm2;  // 親オブジェクト
  iTopSelf  := 0;        // Top位置

  // 入力値を内部保持
  iDate := Self.DateValue;

  // セットする日付を内部保持
  if (TryStrToDate(FormatFloat('0000/00/00', iDate), dSetDate)) then
  begin
    dSetDate := dSetDate;
  end
  else
  begin
    dSetDate := Now;
  end;

  // 親オブジェクトを取得
  iTopSelf  := iTopSelf + objParent.Top;  // Top位置

  // メインRegion
  regParent := TIWRegion(objParent);

  // カレンダー関連の作成
  if (FDateCalendar = nil) then
  begin
    // ----- カレンダーの生成 -----
    FDateCalendar := TIWSMPLCalendar.Create(Self);
    with FDateCalendar do
    begin
      Parent                := regParent;
      Name                  := Self.Name + '_Calendar';
      BGColor               := clWebWhite;
      Left                  := Self.Left;
      Top                   := iTopSelf + IWButton1.Top + IWButton1.Height + 10;
      StartDate             := DateOf(dSetDate);
      OnDateChange          := CalendarDateChange;  // カレンダーの日付選択処理
      FreeNotification(Self);
    end;

    // ----- 画面を閉じるリンクの生成 -----
    FlnkClose := TIWSMPLLink.Create(Self);
    with FlnkClose do
    begin
      Parent       := FDateCalendar.Parent;
      Name         := Self.Name + '_Calendar_FlnkClose';
      Left         := FDateCalendar.Left + 185;
      Top          := FDateCalendar.Top  + 2;
      Width        := Height;
      Caption      := '×';
      Font.Color   := clWebBlue;
      Font.Size    := Self.Font.Size + 3;
      OnAsyncClick := CalendarlnkCloseAsyncClick;  // リンククリック処理
      FreeNotification(Self);
    end;
  end

  // カレンダー関連を破棄
  else
  begin
    CalendarDestroy;
  end;
end;
```

カレンダー
(TIWSMPLCalendar)
クラスを生成

画面を閉じるリンク
(TIWSMPLLink)
クラスを生成

カレンダー機能の実装は以上である。

3. 視覚的に分かりやすいローディング画面

WEBアプリケーションで、入力内容をサーバーへ送信する、ページ数の多いレポートを出力するなど、時間のかかる処理を行う場合があるだろう。その際、ブラウザ側はサーバーからのレスポンス待ちの状態となり、画面が固まっているように見えてしまう。そこで、ローディング画面の表示により処理中を示すことで、ユーザーの不安感を減らすことができる。本章では、Delphi/400とCSSを使用し、処理中にローディング画面を表示する方法を紹介する。

3-1.前提条件

本章では、商品マスタメンテナンス画面を想定し、検索時、更新時にローディング画面を表示させる。今回は**【図8】**のDDSより作成された商品マスタ（SHOHINM）を使用する。

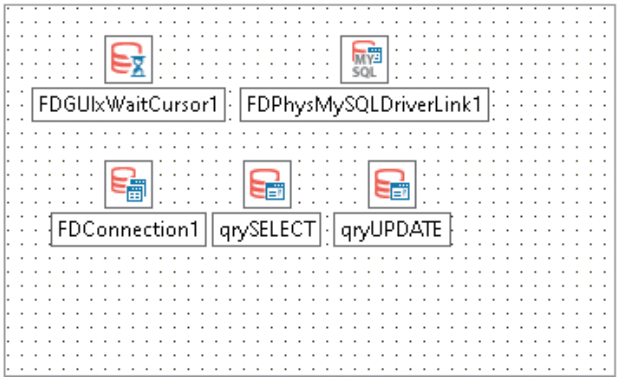
図 8　商品マスタのファイルレイアウト

```
A*****************************************************************************
A*        FILE-ID     :     SHOHINM
A*        FUNCTION    :     商品マスタ
A*****************************************************************************
A                                      UNIQUE
A           R SHOHINMR                 TEXT('  商品マスタ  ')
A             SHSHCD         10A       COLHDG('  商品コード  ')
A             SHSHNM         50O       COLHDG('  商品名  ')
A             SHSRNM         30O       COLHDG('  商品略称  ')
A             SHSRKN          9P 2     COLHDG('  仕入価格  ')
A             SHJURY          7P 2     COLHDG('  重量  ')
A             SHZKKB          1A       COLHDG('  在庫管理区分  ')
A             SHCRDT          8S 0     COLHDG('  作成日付  ')
A             SHCRTM          6S 0     COLHDG('  作成時間  ')
A             SHCRUS         10A       COLHDG('  作成ユーザー  ')
A             SHUPDT          8S 0     COLHDG('  作成日付  ')
A             SHUPTM          6S 0     COLHDG('  作成時間  ')
A             SHUPUS         10A       COLHDG('  作成ユーザー  ')
A           K SHSHCD
```

3-2.IBMiデータベースへの接続

接続先情報を設定するためのiniファイルを準備し、【図9】の通りにデータモジュールにコンポーネントを配置する。TFDConnectionのプロパティについては過去のミガロ.テクニカルレポート「FireDAC実践プログラミングテクニック」を参考に設定して頂きたい（https://www.migaro.co.jp/tr/no11/tech/11_01_02.pdf）。

図 9　データモジュール:コンポーネント配置



IBMiデータベースへ接続する処理【ソース7】を記述する。第4章のサンプル画面からも呼び出せるように、publicセクションで共通メソッドとして作成する。

ソース 7

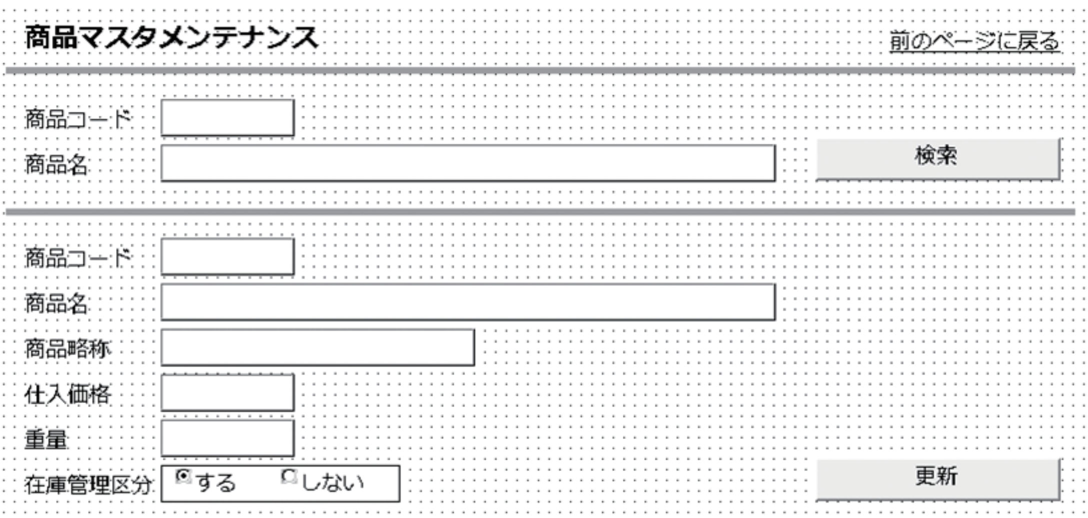
データモジュール接続開始処理

```
procedure TDataModule1.OpenDataBase;
begin
  // FireDAC接続
  if (not FDConnection1.Connected) then
  begin
    try
      FDConnection1.Params.Values['Database']     := IWServerController.Database;
      FDConnection1.Params.Values['ODBCAdvanced'] := 'LibraryOption=';
      FDConnection1.Params.Values['User_Name']    := IWServerController.User_Name;
      FDConnection1.Params.Values['Password']     := IWServerController.Password;
      FDConnection1.LoginPrompt := False;
      FDConnection1.Connected := True;
    except
      raise;
    end;
  end;
end;
```

3-3.商品マスタメンテナンス画面の作成

新規画面を作成し、【図10】の通りにコンポーネントを配置する。

図 10　商品マスタメンテナンス画面:コンポーネント配置



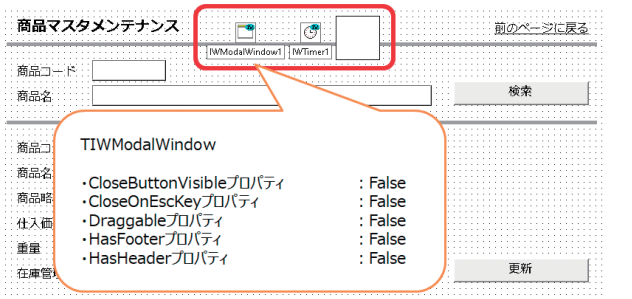
3-4.ローディング画面の作成

今回実装するローディング画面は、TIWFrame、TIWModalWindow、CSSを使用して作成する。まず、TIWFrameを新規作成し、**【図11】**の通りにコンポーネントを配置し、プロパティ値の設定を行う。また、ローディング中の文言表示用のプロパティを宣言し、画面から呼び出された際に、任意の文言を表示できるようにしておく。

図 11　ローディング画面(フレーム):コンポーネント配置

次に、3-3で作成した画面に、フレーム表示用のコンポーネントを追加で配置する**【図12】**。
TIWModalWindowを使用することで、フレームを画面へモーダル表示することができる。

図 12　ローディング画面(フレーム):コンポーネントの追加



ローディング画面表示用の共通関数を画面側に記述する**【ソース8】**。共通関数の引数には、ローディング画面表示中の文言、表示中に実行するイベントを準備し、画面側からの指定文言の表示、指定イベントの実行を行う。またTIWTimerのOnAsyncTimerメソッドに、対象イベントの実行、処理完了後にローディング画面を破棄する処理を記述する**【ソース9】**。

ソース 8

ローディング画面の表示

```
procedure TIWForm3.ShowLoadingFra(AEvent: TNotifyEvent;
  ALoadingStr: String = '');
var
  Frame1: TIWFrame1;
begin
  // ローディング画面の破棄
  IWModalWindow1.Close;
  FreeAndNilList(IWRegion4);

  // イベントのセット
  FAfterEvent := AEvent;
```

```
    // ローディング画面の生成
    Frame1              := TIWFrame1.Create(IWRegion4);
    Frame1.Parent
    Frame1.LoadingStr  := ALoadingStr;
    Frame1.SetSpinnerPosition;

    // フレーム(ローディング画面)を画面にレンダリングする
    IWModalWindow1.ContentElement := IWRegion4;
    IWModalWindow1.Show;

    // タイマー開始
    IWTimer1.Enabled := True;
  end;
```

ソース 9

TIWTimer の OnAsyncTimer メソッド

```
procedure TIWForm3.IWTimer1AsyncTimer(Sender: TObject;
  EventParams: TStringList);
begin
  // タイマー停止
  IWTimer1.Enabled := False;

  // ローディング画面表示中のイベントを実行
  if Assigned(FAfterEvent) then
  begin
    try
      FAfterEvent(nil);
    finally
      // ローディング画面の破棄
      FAfterEvent := nil;
      IWModalWindow1.Close;
      FreeAndNilList(IWRegion4);
    end;
  end;
end;
```

ここで、TIWModalWindowのプロパティ・メソッドについて、ポイントとなる内容を一部抜粋して説明する。

(1)CloseButtonVisibleプロパティ
・True:フレーム右上の「×」ボタンを表示

(2)CloseOnEscKeyプロパティ
・True:Escapeキー押下でフレームを終了

(3)Draggableプロパティ
・True:マウスでフレームを移動できる

(4)HasFooter/HasHeaderプロパティ
・True:フレームのフッター/ヘッダー部を表示

(5)ContentElementプロパティ
・フレームをレンダリングするTControlを指定
※ここでは、【図12】で追加したTIWRegionとする

(6)Showメソッド
・ContentElementプロパティで指定したTControlに対し、フレームをレンダリングする

3-5.ローディング画面の組み込み

3-3で作成した画面の検索ボタンのOnAsyncClick処理、更新ボタンのOnAsyncClick処理のそれぞれで、3-4で作成したローディング画面表示用の共通関数を呼び出す**【ソース10～11】**。また、検索ボタンの押下時に商品マスタデータを画面にセットする処理、更新ボタン押下時に商品マスタを更新する処理を作成する。詳細なソースについてはここでは割愛するが、3-2で配置したTFDQueryを使用し、SQL文を発行して各処理を実行する。

ソース 10

検索ボタンの OnAsyncClick 処理

```
procedure TIWForm3.IWButton1AsyncClick(Sender: TObject;
  EventParams: TStringList);
var
  sLoadingStr: String;
begin
  sLoadingStr := '検索中です。<br>しばらくお待ちください。';
  ShowLoadingFra(IWButton1Click, sLoadingStr);
end;
```

ソース 11

更新ボタンの OnAsyncClick 処理

```
procedure TIWForm3.IWButton2AsyncClick(Sender: TObject;
  EventParams: TStringList);
var
  sLoadingStr: String;
begin
  sLoadingStr := '更新中です。<br>しばらくお待ちください。';
  ShowLoadingFra(IWButton2Click, sLoadingStr);
end;
```

3-6.CSSの組み込み

次に、CSSの定義について説明する。

(1)ローディング中の文字の中央揃え

・CSSファイルに**【図13】**のように記述する。また、3-4のローディング画面のスピナー用TIWLabelのCssプロパティに先ほど記述したセレクタ「center_hor　center_ver」を指定する。今回のように複数指定する場合は、セレクタ間を半角スペースで繋ぐ。

図 13　CSS:文字の中央揃えの設定

CSS：水平中央揃え

```
.center_hor {
  justify-content: center;
  height: 100%;
}
```

CSS：垂直中央揃え

```
.center_ver {
  display: flex;
  align-items: center;
}
```

実行結果：CSSの設定なし

検索中です。
しばらくお待ちください。

実行結果：CSSの設定あり

検索中です。
しばらくお待ちください。

(2)ローディング・スピナーのアニメーション

・CSSファイルに**【図14】**のように記述する。また、3-4のローディング画面のスピナーのアニメーション表示用TIWLabelのCssプロパティに先ほど記述したセレクタ「spinner」を指定する。

図 14　CSS:スピナーのアニメーション表示

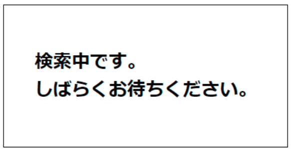
CSS：基本設定

```
/*スピナーの位置などの基本設定や、↓
 回転させるアニメーションの設定*/↓
.spinner {↓
  width: 32px;↓
  height: 32px;↓
  margin: 10px auto;↓
  border: 4px #ddd solid;↓
  border-top: 4px #2e93e6 solid;↓
  border-radius: 50%;↓
  animation: sp-anime 1.0s infinite linear;↓
}↓
```

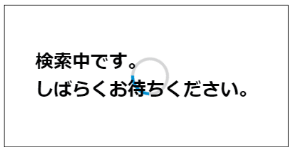
CSS：アニメーション

```
/*スピナーの角度を指定し、回転させる設定*/↓
@keyframes sp-anime {↓
  100% { ↓
    transform: rotate(360deg); ↓
  }↓
}↓
```

実行結果：CSSの設定なし



実行結果：CSSの設定あり



以上で、処理中にローディング画面を表示するプログラムは完成である。上記プログラムを実行し、検索ボタン押下時処理の結果を確認する。ボタン押下時に指定した文言のローディング画面が表示され、検索処理完了後はローディング画面が終了していることが確認できる**【図15〜17】**。

図 15　実行結果①:検索ボタン押下前

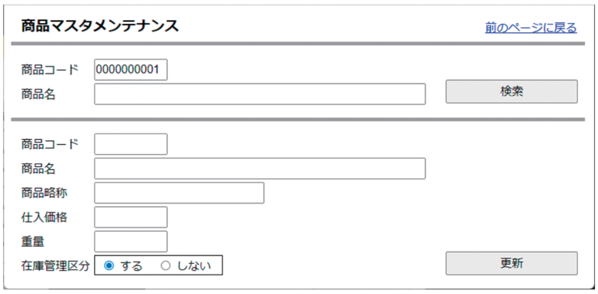


図 16　実行結果②:検索ボタン押下時

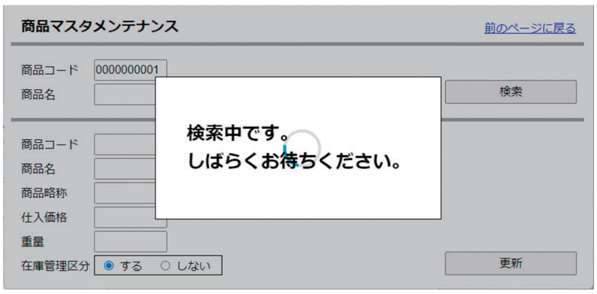
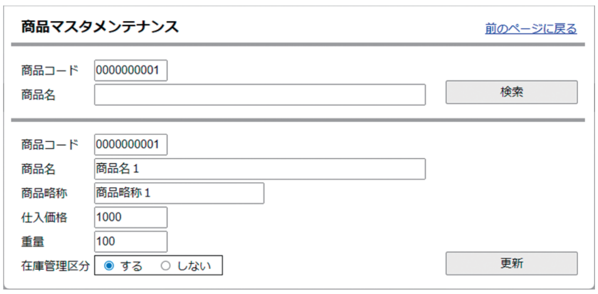


図 17　実行結果③:検索ボタン押下後



4. フレームを使用した明細グリッド

本章では、明細をフレーム形式で作成・使用する方法を紹介する。

4-1.フレームを使用する利点

IntrawebにはTIWGridという明細形式でデータを表示できるコンポーネントが存在する。TIWGridは、タイトルや項目間の幅など、実際にプログラムを実行するまで、その内容を確認することができず、少々扱い辛い印象を受ける【図18】。TIWGridでは、タイトルや幅など明細のビジュアルを統合開発画面上で確認することができないため、実行してビジュアルを確認する必要があり、レイアウト調整に時間がかかる【図19】。

今回紹介する明細のフレームでは、統合開発画面上でビジュアルを確認しながら、レイアウトを調整することができるため、効率よく開発することができる【図20】。

図 18 標準のIWGrid(開発画面)

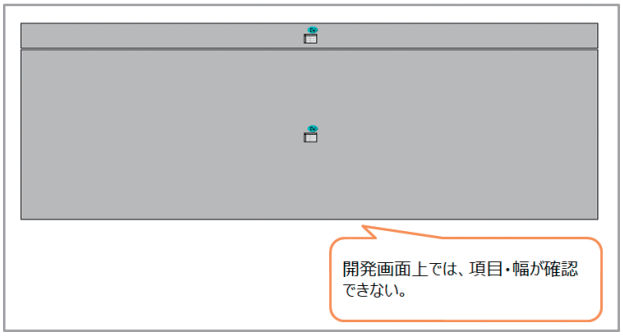


図 19 標準のIWGrid(実行画面)

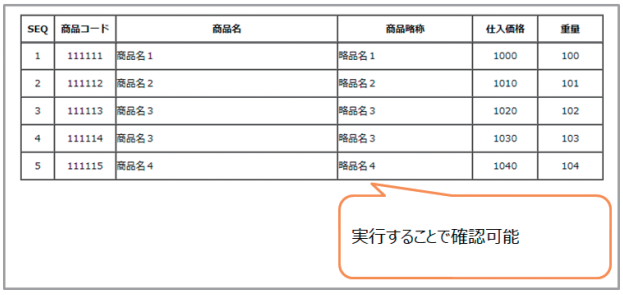
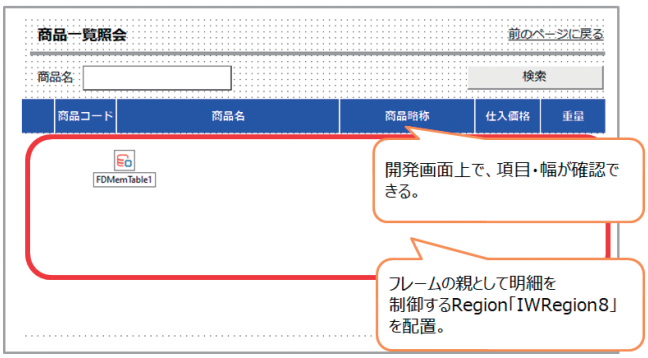


図 20 フレームを用いた場合の開発画面



そこで次節より、商品一覧照会を想定し、フレームを利用した明細形式のプログラム作成例を紹介する。

4-2.今回作成する画面について

今回作成するプログラムのコンポーネントの配置は**【図20】**の通りである。明細フレームの親として使用するIWRegion8を明細部に配置している。

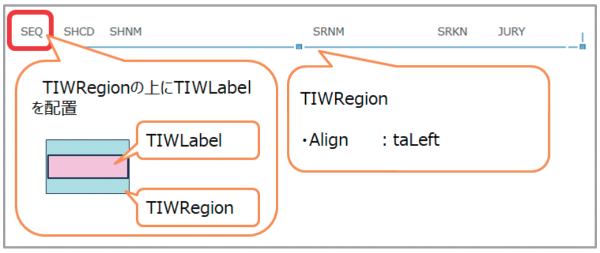
商品マスタについては、第3章と同様の商品マスタ(SHOHINM)を使用する。

4-3.明細フレームの作成

(1)コンポーネントの配置

・**【図21】**の通りにコンポーネントを配置する。

図 21　明細フレームの構成



(2)明細フレームのプロパティ宣言

・明細フレームには明細項目をそれぞれプロパティとして宣言する**【ソース12】**。

ソース 12

プロパティの宣言部

```
property SEQ : Integer  read GetSEQ  write SetSEQ;           // SEQ
property SHCD: String   read GetSHCD write SetSHCD;          // 商品コード
property SHNM: String   read GetSHNM write SetSHNM;          // 商品名
property SRNM: String   read GetSRNM write SetSRNM;          // 商品略称
property SRKN: Currency read GetSRKN write SetSRKN;          // 仕入価格
property JURY: Currency read GetJURY write SetJURY;          // 重量
```

(3)「GET項目名」メソッド

・各プロパティごとに、該当項目のLabelのCaptionを返却する処理を記述する**【ソース13】**。

ソース 13

「GET 項目名」メソッド

```
function TIWFrame2.GetSEQ: Integer;
begin
  Result := StrToIntDef(IWLabel1.Caption, 0);
end;
```

データ表示用Labelの値を受け取る

(4)「SET項目名」メソッド

・各プロパティごとに、該当項目のLabelのCaptionにプロパティ値をセットする処理を記述する【ソース14】。

ソース 14

「Set 項目名」メソッド

```
procedure TIWFrame2.SetSEQ(const Value: Integer);
begin
  IWLabel1.Caption := IntToStr(Value);
end;
```

受け取った値をデータ表示用Labelにセットする

以上で、明細フレームの準備は完了である。

4-4.メイン画面の作成

検索ボタン押下時、第3章と同様の方法でIBMiへ接続し、SQLを発行して画面で指定の条件で商品マスタのデータを取得する。取得したデータはTFDMemTableへ転送する【ソース15】。

ソース 15

データの取得処理

```
// 商品マスタを参照
qrySELECT.Close;
qrySELECT.SQL.Clear;
qrySELECT.SQL.Text := ' SELECT SHSHCD, SHSHNM, SHSRNM, SHSRKN, SHJURY '
                    + ' FROM SHOHINM ';

// 商品名
if (IWEdit1.Text <> '') then
begin
  qrySELECT.SQL.Text := qrySELECT.SQL.Text + ' WHERE SHSHNM LIKE :SHSHNM ';
  qrySELECT.ParamByName('SHSHNM').AsString := '%' + IWEdit1.Text + '%';
end;

// データ取得
qrySELECT.Open;

try
  // 取得したデータをTFDMemTableへ転送
  FDMemTable1.Close;
  FDMemTable1.AppendData(qrySELECT, False);
finally
  qrySELECT.Close;
end;
```

取得したデータをTFDMemTableへ転送

メイン画面では、次のような処理を記述する。

(1)明細の初期化

・明細フレームを解放するための共通関数【ソース16】を作成し、明細作成時の処理の先頭で実行し、明細フレームの初期化を行う。

ソース 16

FreeAndNilList 関数の内容

```
procedure TIWForm4.FreeAndNilList(ARegion: TIWRegion);
var
  i, iCnt: Integer;
  fraList: TFrame;
begin
  // コンポーネントの数を内部保持
  iCnt := ARegion.ComponentCount;

  // コンポーネントの数分、処理を行う
  for i := iCnt - 1 downto 0 do
  begin
    if (ARegion.Components[i] is TFrame) then
    begin
      fraList := TFrame(ARegion.Components[i]);
      FreeAndNil(fraList);
    end;
  end;
end;
```

指定したTIWRegion上にあるフレームを全て解放する

(2)明細作成処理

・TFDMemTableのレコードの数だけ明細フレームを生成する処理を記述する。

・レコード1行を1つのフレームに設定する。

(3)明細フレームのプロパティ設定

・取得したデータを4-3で宣言したフレームのプロパティにセットする【ソース17】。

ソース 17

FreeAndNilList 関数の内容

```
procedure TIWForm4.SetList;
var
  i: Integer;
  Frame2: TIWFrame2;
begin
  // 明細破棄・変数初期化
  FreeAndNilList(IWRegion8);
  FFraCnt := 0;
  Frame2 := nil;

  // 明細作成
  FDMemTable1.First;
  for i := 1 to FDMemTable1.RecordCount do
  begin
    Frame2 := TIWFrame2.Create(IWRegion8);
    with Frame2 do
    begin
      // Nameが重複するとエラーになるため、重複しないよう連番を与える
      Name := 'Frame2_' + IntToStr(i);
      Align := alTop;
      Parent := IWRegion8;
```

```
      // ＜明細データのセット＞
      // フレームに定義しているプロパティに値をセットする
      SEQ  := i;                                          // 明細連番
      SHCD := FDMemTable1.FieldByName('SHSHCD').AsString;  // 商品コード
      SHNM := FDMemTable1.FieldByName('SHSHNM').AsString;  // 商品名
      SRNM := FDMemTable1.FieldByName('SHSRNM').AsString;  // 商品略称
      SRKN := FDMemTable1.FieldByName('SHSRKN').AsInteger; // 仕入価格
      JURY := FDMemTable1.FieldByName('SHJURY').AsInteger; // 重量
      // 色をセット
      case (i mod 2) of
        1:   IWFrameRegion.Color := clWebWhite;  // 奇数行：白
        else IWFrameRegion.Color := $00A4FEF9;   // 偶数行：黄
      end;
    end;

    // 次レコードへ
    Inc(FFraCnt);
    FDMemTable1.Next;
  end;

  // 明細の高さを設定
  if (Frame2 <> nil) then
  begin
    IWRegion8.Height := Frame2.Height * FFraCnt;
  end;
end;
```

フレームを明細レコード数分生成。
同時に設定及びデータをセットする

(4)明細背景色の設定

・明細が見やすくするように奇数行は背景を白色、偶数行は背景を黄色に設定した。

(5)明細の高さを設定

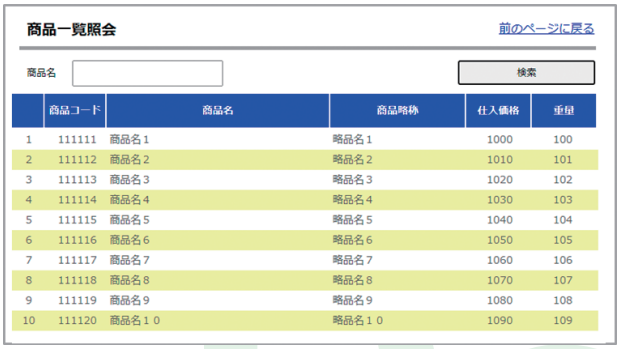
・明細フレームを生成したTIWRegionの高さを設定する。

以上で、フレームを利用した明細形式のプログラムは完成である。

4-5.プログラムの実行

上記プログラムを実行し、検索ボタン押下時処理の結果を確認する。ボタン押下時に商品マスタのデータが明細フレームに表示されることが確認できる【図22】。

図 22　検索ボタン実行後



	商品コード	商品名	商品略称	仕入価格	重量
1	111111	商品名１	略品名１	1000	100
2	111112	商品名２	略品名２	1010	101
3	111113	商品名３	略品名３	1020	102
4	111114	商品名４	略品名４	1030	103
5	111115	商品名５	略品名５	1040	104
6	111116	商品名６	略品名６	1050	105
7	111117	商品名７	略品名７	1060	106
8	111118	商品名８	略品名８	1070	107
9	111119	商品名９	略品名９	1080	108
10	111120	商品名１０	略品名１０	1090	109

本章では、商品一覧照会機能をフレームを利用して作成してきた。明細をフレーム形式で作成することで、設計画面で明細の高さや幅、色などを自由に設定することができる。似たような画面を展開する場合、フレームを利用することで、記述するロジックを削減する事ができ、開発工数を短縮する事が可能である。

5.さいごに

本稿では、ユーザーが直感的に操作することができるよう、システム開発チームが実際の開発時に工夫し、実装したユーザーインターフェース機能を紹介した。今回紹介した内容を参考に、必要な機能を追加するなど各自に合ったものにカスタマイズして頂くことも可能である。本稿を参考に、アプリケーション開発に役立てて頂ければ幸いである。

なお、今回紹介したプログラムのソース一式を以下よりダウンロード可能なので、是非活用して頂きたい。

https://www.migaro.co.jp/d4sample/2024report_intraweb.zip
（※ダウンロードには、Delphi/400メンテナンスページへのログインユーザー・パスワードが必要）

Delphi/400

Windowsタブレット向けカスタムグリッドの作成方法

株式会社ミガロ.
システム事業部 2課
前坂　誠二

略　歴

生年月日:1989年3月21日
最終学歴:2011年　関西大学　文学部卒業
入社年月:2011年04月　株式会社ミガロ. 入社
社内経歴:2011年04月　システム事業部配属

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を担当。Delphi、Delphi/400の開発経験を積みながら、日々スキルを磨いている。

1.はじめに

一覧形式でデータを照会する場合、グリッド形式のレイアウトがよく使用される。Delphi/400のVCLアプリケーションでは、TDBGridやTStringGridのコンポーネントを使用して容易にグリッド形式の照会画面が作成可能である。Delphi/400で作成したVCLアプリケーションは、PCだけでなくWindowsタブレットでも実行可能である。PCとタブレットで同一のアプリケーションが利用可能であることはDelphi/400アプリケーションの魅力のひとつである。しかし利用端末が異なるということは、利用するシーンや目的が異なることが多い。本稿のテーマであるタブレットの場合は、屋外で使用することが多く、別作業をしながら片手で操作するなどの場面が想定される。特に一覧形式の画面では、文字が細かくなりやすいため、より視認性や使い勝手のよさについて考慮する必要がある。

そこで本稿では、一覧形式の画面に注目し、Windowsタブレットでの利用に特化したアプリケーションの作成方法について紹介する。

2.作成する照会画面イメージ

まずは、本稿で作成するサンプルについて解説する。本稿では、商品を点検して状態を保存するという場面をイメージし、**【図1】**のような一覧画面を利用していると想定する。一覧画面の明細には、チェックボックスやリストボックス、カメラ起動の為のボタンを配置している。**【図1】**に対して、本稿の内容を実装した完成イメージは**【図2】**とする。

尚、本稿で紹介する内容はDelphi/400 11Alexandriaを使用し、フレームワークはFireMonkeyで作成する。

図 1　点検商品照会画面（変更前）

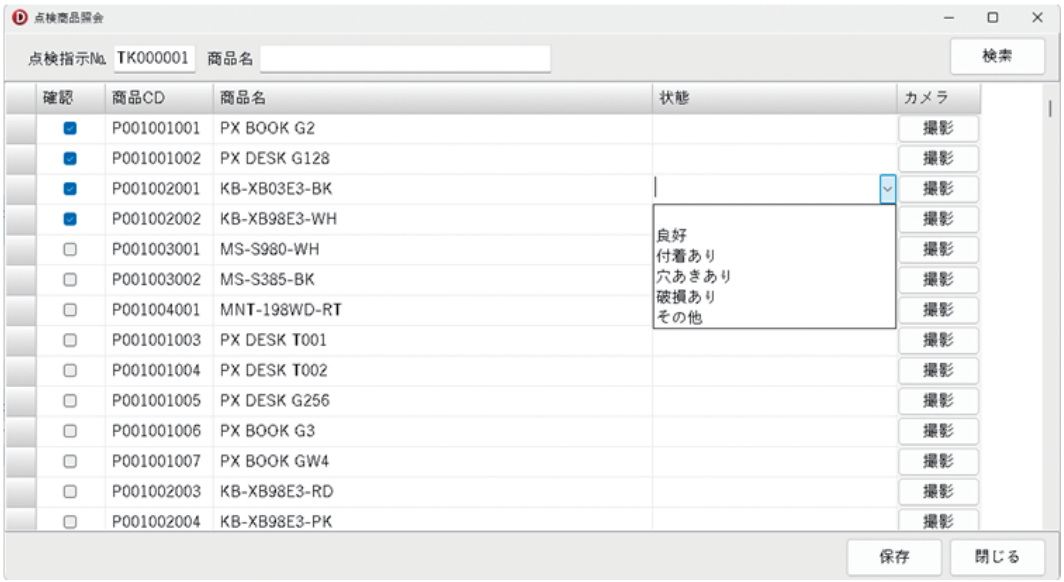


図 2　点検商品照会画面（変更後）

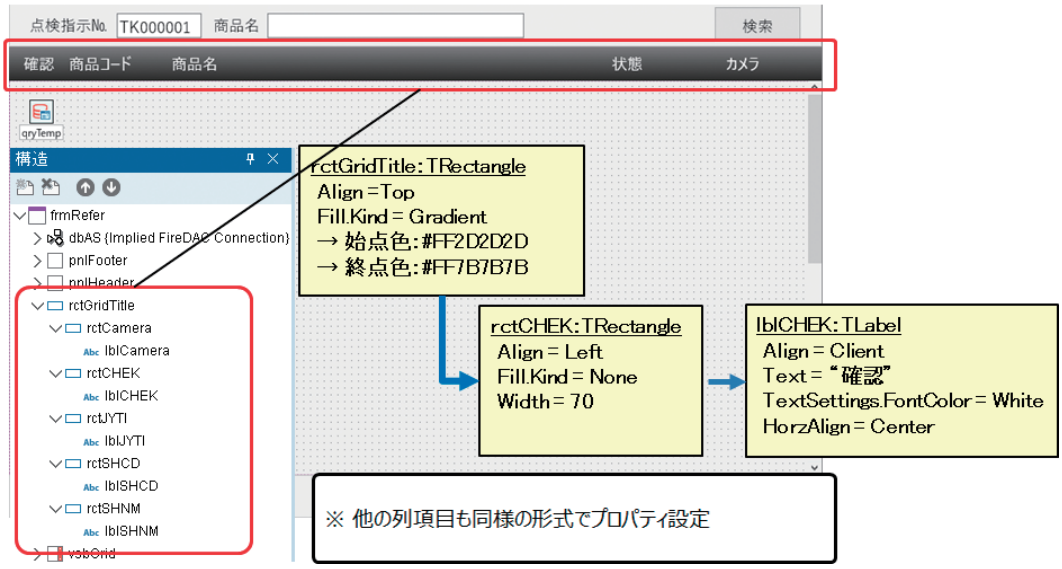
3.コンポーネントの配置

まずはコンポーネントの配置について説明する。本稿ではグリッドをメインテーマとして扱うためグリッド部分以外については説明を割愛する。

FireMonkeyは特徴として、コンポーネントに親子関係を持たせることができる。この特徴を活かし、コンポーネントの配置を行う。

明細タイトルは、列全体の色をコントロールするため、まずはTRectangleを配置し、子として各列用のTRectangle+TLabelを配置する【図3】。

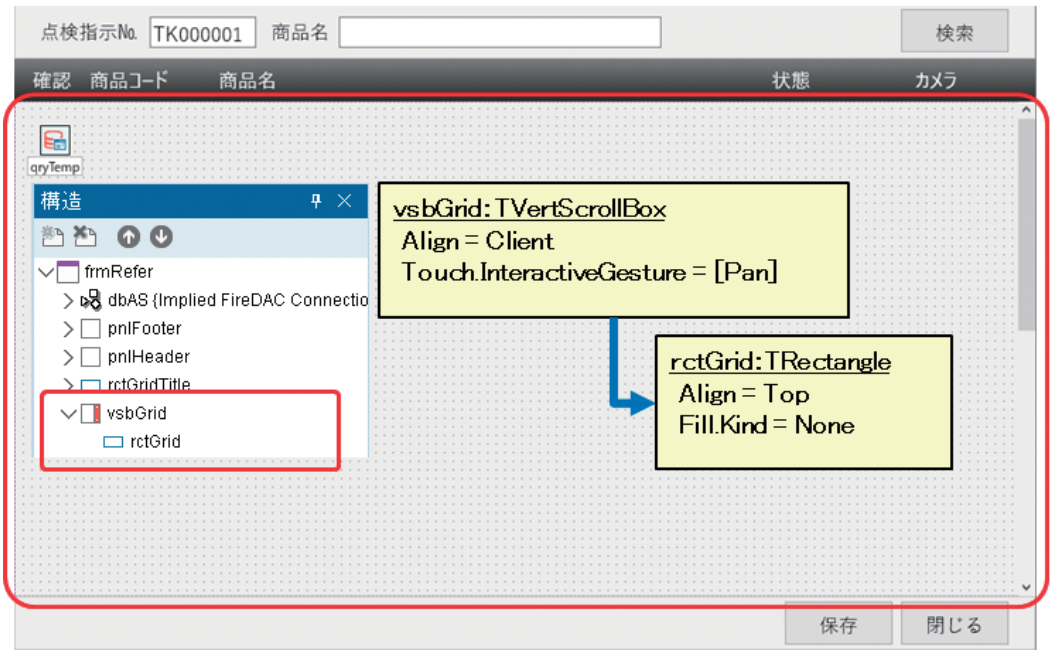
図 3　コンポーネント配置（明細タイトル）



データ内容を表示するためのコンポーネントは、TVertScrollBox→TRectangleコンポーネントを親子関係で配置する。TVertScrollBoxは、明細のスクロール可能にさせるために配置し（vsbGrid）、TRectangleは描画によるデータ表示を行うために配置する（rctGrid）。

この際、rctGridは［Fill］プロパティのKindをNoneとし、デフォルトの塗りつぶしを無しにする。［Align］はスクロール表示を行うため、Topの値に設定する。また、vsbGridのTouchプロパティにて［InteractiveGestures］の［Pan］をTrueにする。本プロパティをTrueにすることにより、タブレットで操作した際に指のスライドでのスクロールが可能となる**【図4】**。

図 4　コンポーネント配置（明細グリッド）



4.明細表示処理の実装

4-1.データ取得処理

まずは、データ取得処理の準備として今回実装に必要な変数・定数を定義する【ソース１】。
次に、btnSearchのOnClickイベントにて、TFDQueryでデータ取得→配列に保持する【ソース２】。その後、選択行を先頭にし、rctGridのHeightを行数×行の高さで設定している。
尚、本稿のサンプルでは【図５】のテーブルよりデータを取得する。

ソース１

変数定義

```
type
  // 明細データ
  TListData = record
    CHEK: Boolean;   // チェック
    SHCD: String;    // 商品CD
    SHNM: String;    // 商品名
    JYTI: String;    // 状態
  end;

  TfrmRefer = class(TForm)
    .
    .
    .

  private
    { private 宣言 }
    FListData: array of TListData; // 明細データの配列
    FSelectedRow: Integer; // 選択行保持
  public
    { public 宣言 }
  end;

    .
    .
    .
const
   cRowHeight = 55; // １明細の高さ
```

明細データ

内部保持用変数

定数

ソース２

データ取得処理

```
{*******************************************************************************
 目的: 検索ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.btnSearchClick(Sender: TObject);
var
  iCnt: Integer;
begin
  // データ取得
  qryTemp.Close;
  qryTemp.SQL.Text :=
    ' SELECT * FROM TRTKSJ ';
  qryTemp.Open;
  try
```

```
    qryTemp.First;
    while not qryTemp.Eof do
    begin
      // 配列へセット
      SetLength(FListData, Length(FListData) + 1);
      iCnt := Length(FListData) - 1;

      FListData[iCnt].SHCD  := qryTemp.FieldByName('TSSHCD').AsString; // 商品コード
      FListData[iCnt].SHNM  := qryTemp.FieldByName('TSSHNM').AsString; // 商品名

      qryTemp.Next;
    end;
  finally
    qryTemp.Close;
  end;

  // 選択行をセット ※0始まり
  FSelectedRow := 0;

  // スクロールを先頭位置にする
  vsbGrid.ScrollBy(0, Length(FListData) * cRowHeight);

  // rctGridの大きさを設定
  rctGrid.Height := Length(FListData) * cRowHeight;

  // 再描画
  rctGrid.Repaint;
end;
```

図 5 データ取得用テーブル

```
A*********************************************************
A*       FILE-ID    :    TRTKSJ
A*       FUNCTION   :    点検指示ファイル
A*********************************************************
A                                    UNIQUE
A          R TRTKSR                  TEXT(' 点検指示ファイル ')
A            TSSJNO        8A        COLHDG(' 点検指示No. ')
A            TSSHCD       10A        COLHDG(' 商品コード ')
A            TSSHNM       520        COLHDG(' 商品名 ')
A            TSTKZM        1A        COLHDG(' 点検済フラグ ')
A            TSJTCD        4A        COLHDG(' 点検状態コード ')
A          K TSSJNO
A          K TSSHCD
```

4-2.描画処理

4-1で配列に保持しているデータをループ処理にて順次描画する。処理はrctGridのOnPainting処理にて記述する**【ソース3】**。

ソース 3

描画処理（行の枠、テキスト描画）

```
{*******************************************************************************
 目的: 描画処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridPainting(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
var
  brsTemp: TBrush;
  rRowArea, rText: TRectF;
  iData, iStartRow, iEndRow: Integer;
  bBtnColor: TBrush;
begin
  // 現在のスクロール位置より表示開始行を計算
  iStartRow := Trunc(vsbGrid.ViewportPosition.Y / cRowHeight);
  if iStartRow < 0 then
       iStartRow := 0;

  // 現在のスクロール位置より表示最終行を計算
  iEndRow := Trunc(vsbGrid.ViewportPosition.Y / cRowHeight)
           + Trunc(vsbGrid.Height / cRowHeight);

  for iData := iStartRow to iEndRow do
  begin
    if Length(FListData) - 1 < iData then
          Exit;

    // 行の枠を描画 Str ---------------------------------------------
    rRowArea        := ARect;
    rRowArea.Left   := 1;
    rRowArea.Right  := rctGrid.Width;
    rRowArea.Top    := iData * cRowHeight + 1;
    rRowArea.Bottom := (iData + 1) * cRowHeight;

    brsTemp := TBrush.Create(TBrushKind.Solid, TAlphaColorRec.White);

    if FSelectedRow = iData then
      brsTemp.Color := TAlphaColor($FFFFDC20) // 選択行
    else if ((iData mod 2) = 0) then
      brsTemp.Color := TAlphaColor($FFEAF2FC) // 偶数行
    else
      brsTemp.Color := TAlphaColorRec.White;  // 奇数行

    Canvas.FillRect(
      rRowArea,      // 描画対象範囲
      0, 0,          // XRadius、YRadius：角の曲がり具合  0…四角
      [],            // Radiusを適用する角
      1,             // Opacity：透明度
      brsTemp);      // 色
    // 行の枠を描画 End ---------------------------------------------

    // テキスト描画 Str ---------------------------------------------
    Canvas.Fill.Color := TAlphaColorRec.Black;
    Canvas.Font.Size := 19;

    // 商品コード
    rText := rRowArea;
    rText.Top := rText.Top + 3;
    rText.Left  := rctSHCD.Position.X;
    rText.Right := rctSHCD.Position.X + rctSHCD.Size.Width;
    Canvas.FillText(
      rText,                  // 描画対象範囲
      FListData[iData].SHCD,  // 値
      True,                   // WordWrap：True - 改行あり
      1,                      // Opacity：透明度
      [],                     // テキストを読む方向 ※ヘルプにて値なしを推奨
      TTextAlign.Leading,     // 水平方向の配置  Leading：左揃え
      TTextAlign.Center);     // 垂直方向の配置  Center ：中央
```

①

②

③

```
    商品名も商品コードと同手順で描画
    // テキスト描画 End ------------------------------------------
  end;
end;
```

処理の流れとしては以下の通りである。
①現在表示しているスクロール位置より
画面に表示する開始行と終了行を計算する
②各明細データ単位に行の区切り枠を描画する
③明細の値を描画する
本稿で実装している描画処理では、まずTRectF型の変数を定義し、Left、Right、Top、Bottomの値を指定して、描画の対象範囲を定める。その後、各々の描画処理を実装している。
行の区切り枠では、rRowArea変数に描画範囲をセットした後、選択行、偶数行、奇数行による色の指定を行い、Canvas.FillRect処理で描画を実施している。
明細の値では、rText変数に明細タイトルの位置と幅に合わせて描画範囲をセットした後、FillText処理で値を表示している。列の幅を大きくしたいときや位置を変更したい場合は、明細タイトルの幅や位置を変更するだけで、明細の描画も付随して変更されるという仕組みである。

4-3.明細タップ処理

本章のタップ処理では、タップ時に選択行の色の変更を行う。色の指定については4-2の処理で実装済みであるためrctGridのOnMouseDownイベントにてFSelectedRowの変数値を変更し、再描画を行うだけで実装可能である**【ソース4】**。

ソース 4

明細タップ処理（選択行の変更）

```
{*******************************************************************************
 目的: 明細タップ時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
  // タップ位置(Y座標)から選択行を計算
  FSelectedRow := Trunc(Y) div cRowHeight;

  // 再描画
  rctGrid.Repaint;
end;
```

4-4.スクロールバーの自動表示切替え

スクロールバーは明細をスライドしたタイミングだけ表示させるように実装する。ロジックとしては画面生成時処理に1行追加するだけで実装可能である【ソース5】。

ソース 5

スクロールの自動表示設定

```
{*******************************************************************************
 目的: 画面生成時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.FormCreate(Sender: TObject);
begin
  // スクロールの自動表示
  vsbGrid.AniCalculations.AutoShowing := True;
end;
```

5.チェックボックスの実装

5-1.描画処理

本稿でのチェックボックスは、チェックONの場合は黒丸で塗りつぶし、チェックマークが表示される見た目で作成する。rctGridのOnPaintingに処理を追加する【ソース6】。

ソース 6

描画処理（チェックボックス）

```
{*******************************************************************************
 目的: 描画処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridPainting(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
var
  brsTemp: TBrush;
  rRowArea, rText: TRectF;
  rCheck: TRectF;
  iData, iStartRow, iEndRow: Integer;
  pDrawPoint1, pDrawPoint2: TPointF;
begin
  表示開始行、終了行計算【ソース3】

  for iData := iStartRow to iEndRow do
  begin
    if Length(FListData) - 1 < iData then
      Exit;
    行の枠、テキスト描画【ソース3】
```

```
    // チェックボックス描画 Str ------------------------------------
    // チェックの外側描画
    rCheck := rRowArea;
    rCheck.Left   := rctCHEK.Position.X + rctCHEK.Width / 4; // Left   ：開始位置
    rCheck.Right  := rCheck.Left + 32;                       // Right  ：幅
    rCheck.Top    := rRowArea.Top + 11;                      // Top    ：開始位置
    rCheck.Bottom := rRowArea.Bottom - 11;                   // Bottom：高さ

    // 描画する線の設定値
    Canvas.Stroke.Kind  := TBrushKind.Solid;
    Canvas.Stroke.Color := TAlphaColorRec.Black;
    Canvas.Stroke.Thickness  := 0.5;
    Canvas.DrawRect(
      rCheck,               // 描画対象範囲
      16, 16,               // Radius：横半径、縦半径
      AllCorners,           // Radiusを適用する角
      1,                    // Opacity：透明度
      TCornerType.Round);   // 角の種類

    // チェックONの場合
    if FListData[iData].CHEK then
    begin
      // チェックの中身描画
      Canvas.Fill.Color := TAlphaColorRec.Black;
      Canvas.FillArc(
        PointF(
          rCheck.Left + 16,                        // 円の中心位置：X
          Trunc(cRowHeight / 2) + rRowArea.Top),   // 円の中心位置：Y
        PointF(16, 16),                            // Radius：横半径、縦半径
        0, 360,                                    // 塗りつぶし角度
        1);                                        // Opacity：透明度

      // チェック線左
      pDrawPoint1.X := rCheck.Left + 6;            // 始点のX
      pDrawPoint1.Y := rCheck.Top  + 9;            // 始点のY
      pDrawPoint2.X := pDrawPoint1.X + 7;          // 終点のX
      pDrawPoint2.Y := pDrawPoint1.Y + 16;         // 終点のY
      Canvas.Stroke.Thickness  := 3.5;             // 線の太さ
      Canvas.Stroke.Color := TAlphaColorRec.White; // 線の色
      Canvas.DrawLine(pDrawPoint1, pDrawPoint2, 1);

      // チェック線右
      pDrawPoint1.X := pDrawPoint2.X;              // 始点のX
      pDrawPoint1.Y := pDrawPoint2.Y;              // 始点のY
      pDrawPoint2.X := rCheck.Left + 32;           // 終点のX
      pDrawPoint2.Y := rCheck.Top + 2;             // 終点のY
      Canvas.Stroke.Thickness  := 3.5;             // 線の太さ
      Canvas.Stroke.Color := TAlphaColorRec.White; // 線の色
      Canvas.DrawLine(pDrawPoint1, pDrawPoint2, 1);

      // 値を元に戻す
      Canvas.Stroke.Color := TAlphaColorRec.Black;
      Canvas.Stroke.Thickness  := 1;
    end;
    // チェックボックス描画 End --------------------------------------
  end;
end;
```

①

②

③

処理の流れとしては以下の通りである。
①外側の円を描画する
②チェックONの場合に、黒丸で塗りつぶしを行う
③チェックONの場合に、チェックマークを描画する
①の処理では、前章での描画処理と同様に、まずは描画対象範囲を指定する。その後DrawRect処理により描画処理を実施する。円の描画は、XRadius、YRadiusの引数値でどれくらいの大きさの円を描くかを決定する。XRadius、YRadiusの値が円の縦と横の半径にあたるため、設定した値に対して、Rectの値についても調整する必要がある。例えば、本サンプルの場合は各Radiusを16と設定しているため、RectのLeft-Right間、Top-Bottom間は32とならなければ、いびつな円として描画される。

②、③の処理はチェックONとした場合に実施する。②の黒丸での塗りつぶしはFillArc処理によって行う。こちらはDrawRect処理とは異なり、Rectの指定は不要であるが描画したい円の中心を起点に、塗りつぶし範囲を指定する必要がある。本サンプルでは、外側の円と同一の大きさで塗りつぶしを行っている。③のチェックマークの描画は、2本の線を組み合わせて実装している。それぞれ線の始点と終点、線の太さと色を設定した後、DrawLine処理により、線を描画している。

5-2.明細タップ処理

明細タップ時に、チェックボックスのONとOFFを切り替える処理を実装する。
タップしたポイントのX座標が”確認”列の範囲内であれば、配列の値を切り替える処理を追加するだけで実装完了である【ソース7】。

ソース 7

明細タップ処理（チェックボックス処理）

```
{*******************************************************************************
 目的: 明細タップ時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
  // タップ位置(Y座標)から選択行を計算
  FSelectedRow := Trunc(Y) div cRowHeight;

  // チェックボックスタップ処理 Str ---------------------------------------
  if (Trunc(X) >= rctCHEK.Position.X)                    and
     (Trunc(X) <= (rctCHEK.Position.X + rctCHEK.Width)) then
  begin
    FListData[FSelectedRow].CHEK := not (FListData[FSelectedRow].CHEK);
  end;
  // チェックボックスタップ処理 End ---------------------------------------

  // 再描画
  rctGrid.Repaint;
end;
```

処理追加

6.ボタンの実装

6-1.描画処理

“カメラ”列のボタンの描画を行う。ボタン内のテキストには“撮影”の文言を設定する。rctGridのOnPaintingに処理を追加する**【ソース8】**。

ソース 8

描画処理(ボタン)

```
{*******************************************************************************
 目的: 描画処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridPainting(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
var
  brsTemp: TBrush;
  rRowArea, rText: TRectF;
  rCheck: TRectF;
  rBtn: TRectF;
  iData, iStartRow, iEndRow: Integer;
  pDrawPoint1, pDrawPoint2: TPointF;
  bBtnColor: TBrush;
begin
  表示開始行、終了行計算【ソース3】
  for iData := iStartRow to iEndRow do
  begin
    if Length(FListData) - 1 < iData then
      Exit;
    行の枠、テキスト描画【ソース3】
    チェックボックス描画【ソース6】
    // ボタン描画 Str ----------------------------------------------
    rBtn := rRowArea;
    rBtn.Left   := rctCamera.Position.X;
    rBtn.Right  := rctCamera.Position.X + rctCamera.Size.Width;
    rBtn.Top    := rBtn.Top + 6;
    rBtn.Bottom := rBtn.Bottom - 6;

    // ボタン色
    bBtnColor := TBrush.Create(TBrushKind.Solid, TAlphaColor($FFE0E0E0));

    // ボタン描画
    Canvas.FillRect(
      rBtn,                 // 描画対象範囲
      5, 5,                 // XRadius、YRadius：角の曲がり具合  0…四角
      AllCorners,           // Radiusを適用する角
      1,                    // Opacity：透明度
      bBtnColor);           // 色

    // ボタン枠描画
    Canvas.Stroke.Kind  := TBrushKind.Solid;
    Canvas.Stroke.Color := TAlphaColorRec.Gray;
    Canvas.DrawRect(
      rBtn,                 // 描画対象範囲
      5, 5,                 // XRadius、YRadius：角の曲がり具合  0…四角
      AllCorners,           // Radiusを適用する角
      1,                    // Opacity：透明度
      TCornerType.Round); // 種類
```

①

②

```
    // 文字描画
    Canvas.Fill.Color := TAlphaColorRec.Black;
    Canvas.FillText(rBtn, '撮 影', True, 1, [], TTextAlign.Center, TTextAlign.Center);
    // ボタン描画 End ------------------------------------------------
  end;
end;
```

③

処理の流れとしては以下の通りである。
①指定範囲でボタンの形で塗りつぶしを行う
②ボタンの枠を描画する
③ボタンに表示する文言を描画する
①の処理では、前章での描画処理と同様に、まずは描画対象範囲を指定する。その後、FillRect処理にてボタンの形に描画する。本サンプルでは、少し丸みを帯びた形で描画するため、XRadiusとYRadiusの値を5で指定している。例えば、各Radiusの値を0で指定した場合は四角の形のボタンとなる。
②の処理では、FillRect処理で①で作成したボタンの枠を描画している。
③の処理では、FillText処理で描画したボタン上に文字をセットしている。文字の描画については、第４章と同様の手順である。

6-2.明細タップ処理

明細タップ時に、”撮影”ボタン押下時の処理を実装する。タップしたポイントのX座標が”カメラ”列の範囲内であれば、カメラ起動を行う【ソース９】。本サンプルでは、カメラ起動の処理を実装しているが、TButtonのOnClickイベントをイメージいただき、各々の処理を実装すればよい。

ソース９

明細タップ処理(ボタン処理)

```
{*******************************************************************************
 目的: 明細タップ時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
  // タップ位置(Y座標)から選択行を計算
  FSelectedRow := Trunc(Y) div cRowHeight;

  チェックボックスタップ処理【ソース7】

  // カメラボタンタップ処理 Str ------------------------------------------
  if (Trunc(X) >= rctCamera.Position.X)                    and
     (Trunc(X) <= (rctCamera.Position.X + rctCamera.Width)) then
  begin
    // *** カメラ起動 *** //
    ShellExecute(0, 'OPEN', PChar('microsoft.windows.camera:'), nil, nil, SW_SHOWMAXIMIZED);
  end;
  // カメラボタンタップ処理 End ------------------------------------------

  // 再描画
  rctGrid.Repaint;
end;
```

7.リストボックスの実装

7-1.描画処理

“状態”列のリストボックスの描画を行う。リストボックスはボタンの描画実施後に下向きの三角マークを描画して表示している【ソース１０】。

ソース 10

描画処理(リストボックス)

```
{*******************************************************************************
 目的: 描画処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridPainting(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
var
  brsTemp: TBrush;
  rRowArea, rText: TRectF;
  rCheck: TRectF;
  rBtn: TRectF;
  iData, iStartRow, iEndRow: Integer;
  pDrawPoint1, pDrawPoint2: TPointF;
  bBtnColor: TBrush;
  TempPoints: TPolygon;
begin
  表示開始行、終了行計算【ソース3】
  for iData := iStartRow to iEndRow do
  begin
    if Length(FListData) - 1 < iData then
      Exit;
    行の枠、テキスト描画【ソース3】
    チェックボックス描画【ソース6】
    ボタン描画【ソース8】
    // リストボックス描画 Str ---------------------------------------
    rBtn := rRowArea;
    rBtn.Left   := rctJYTI.Position.X;
    rBtn.Right  := rctJYTI.Position.X + rctJYTI.Size.Width;
    rBtn.Top    := rBtn.Top + 6;
    rBtn.Bottom := rBtn.Bottom - 6;

    // ボタン色
    bBtnColor := TBrush.Create(TBrushKind.Solid, TAlphaColor($FFE0E0E0));

    // ボタン描画
    Canvas.FillRect(
      rBtn,                 // 描画対象範囲
      5, 5,                 // XRadius、YRadius：角の曲がり具合　0…四角
      AllCorners,           // Radiusを適用する角
      1,                    // Opacity：透明度
      bBtnColor);           // 色

    // ボタン枠描画
    Canvas.Stroke.Kind  := TBrushKind.Solid;
    Canvas.Stroke.Color := TAlphaColorRec.Gray;
    Canvas.DrawRect(
      rBtn,                 // 描画対象範囲
      5, 5,                 // XRadius、YRadius：角の曲がり具合　0…四角
      AllCorners,           // Radiusを適用する角
      1,                    // Opacity：透明度
      TCornerType.Round); // 種類
```

①

```
    // 下三角配置設定
    SetLength(TempPoints, 4);
    TempPoints[0] := PointF(rctJYTI.Position.X + rctJYTI.Size.Width - 22, rBtn.Top    + 14); // 左支点
    TempPoints[1] := PointF(rctJYTI.Position.X + rctJYTI.Size.Width -  4, rBtn.Top    + 14); // 右支点
    TempPoints[2] := PointF(rctJYTI.Position.X + rctJYTI.Size.Width - 13, rBtn.Bottom - 12); // 下支点
    TempPoints[3] := PointF(rctJYTI.Position.X + rctJYTI.Size.Width - 22, rBtn.Top    + 14); // 左支点

    // 下三角描画
    Canvas.Fill.Color := TAlphaColorRec.Black;
    Canvas.FillPolygon(TempPoints, 1); // [0]→[1]→[2]→[3]で支点を結んで▼描画

    // 文字描画
    Canvas.Fill.Color := TAlphaColorRec.Black;
    Canvas.FillText(rBtn, FListData[iData].JYTI, True, 1, [], TTextAlign.Leading, TTextAlign.Center);
    // リストボックス描画 End ------------------------------------
  end;
end;
```

②

③

処理の流れとしては以下の通りである。

①第６章と同様の手順でボタンを描画する

②下三角(▼)を描画する

③リストボックスに表示する文言を描画する

①と③の処理については、第６章と同様のイメージでボタンと文言の描画処理を行う。

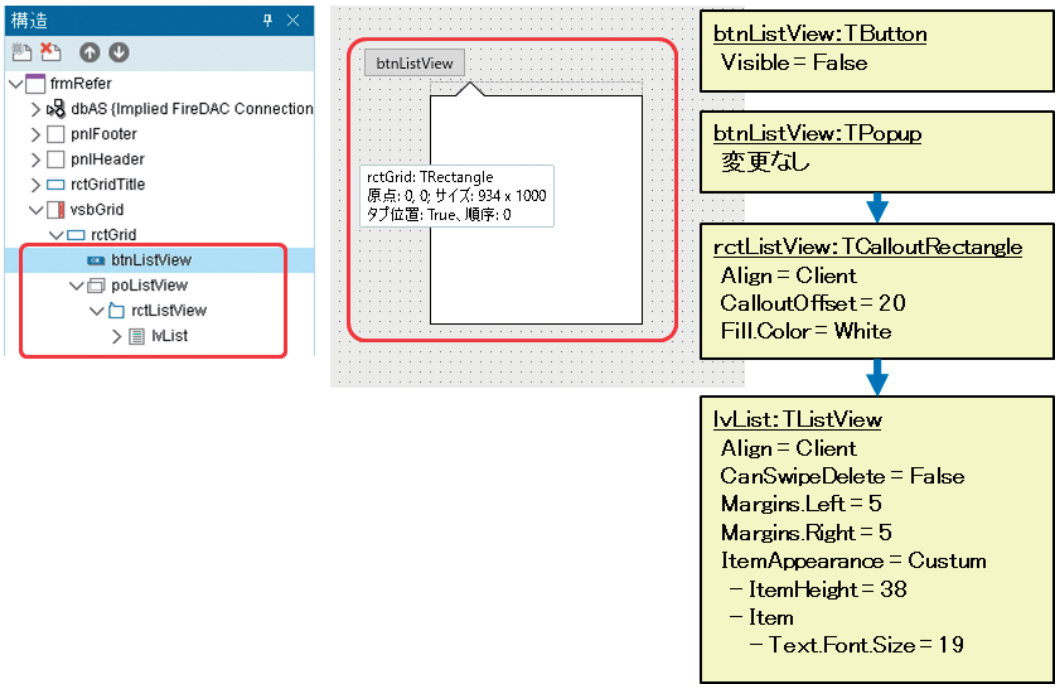
②の処理では、下三角を描画するためにFillPolygon処理を実施する。TempPoints変数では三角形を形成するために各支点を設定している。

下三角は記号文字でも表現可能であるが、実行端末の使用フォントに依存する可能性がある。しかし、②の処理のように図形で描画処理を行うと実行端末による影響は受けない。

7-2.コンポーネントの配置

本サンプルでは、明細タップ時に選択リストを表示する。リストの内容表示は別途コンポーネントを配置して実装する【図６】

図 6　コンポーネント配置(リスト)



7-3.リスト内容の取得

画面生成時に【図7】のテーブルよりデータを取得する。取得した内容をlvListのItemsに取得内容を追加する【ソース11】。

図 7　データ取得用テーブル(リスト取得)

```
A*****************************************************************
A*       FILE-ID    :    MSSHTJ
A*       FUNCTION   :    商品点検状態マスタ
A*****************************************************************
A                                      UNIQUE
A         R MSSHTR                     TEXT(' 商品点検状態マスタ ')
A           STJTCD         4A          COLHDG(' 状態コード ')
A           STJTNM        320          COLHDG(' 状態名 ')
A         K STJTCD
```

ソース 11

リストデータ取得処理

```
{*******************************************************************************
 目的: 画面生成時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.FormCreate(Sender: TObject);
begin
  // スクロールの自動表示
  vsbGrid.AniCalculations.AutoShowing := True;

  // リスト内容取得
  lvList.Items.Clear;
  lvList.Items.Add.Text := '';
  qryTemp.Close;
  qryTemp.SQL.Text :=
    ' SELECT * FROM MSSHTJ ';
  qryTemp.Open;
  try
    while not qryTemp.Eof do
    begin
      lvList.Items.Add.Text :=
        qryTemp.FieldByName('STJTNM').AsString;
      qryTemp.Next;
    end;
  finally
    qryTemp.Close;
  end;

  // ポップアップ非表示
  poListView.Visible := False;
end;
```

処理追加

7-4.リスト表示処理

btnListViewのOnClickイベントにてリスト内容の表示処理を記述する【ソース１２】。poListViewの表示場所のターゲットをbtnListViewとし、ポップアップ表示させる。また、表示させる向きをbtnListViewの表示位置で決定する処理を記述している。

ソース 12

リスト内容表示処理

```
{*******************************************************************************
 目的: リスト表示処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.btnListViewClick(Sender: TObject);
var
  sY: Single;
begin
  poListView.PlacementTarget := btnListView;
  poListView.Popup;

  // ボタンのY座標を保持
  sY := btnListView.Position.Y;

  // CalloutPositionの設定
  if TCustomPopupForm(TMyPopupForm(poListView).PopupForm).Top < sY then
  begin
    // ポップアップを上向きに表示
    rctListView.CalloutPosition := TCalloutPosition.Bottom;
    lvList.Margins.Top    := 5;
    lvList.Margins.Bottom := 15;
  end
  else
  begin
    // ポップアップを下向きに表示
    rctListView.CalloutPosition := TCalloutPosition.Top;
    lvList.Margins.Top    := 15;
    lvList.Margins.Bottom := 5;
  end;
end;
```

7-5.明細タップ処理

明細タップ時に、“状態”列のリストボックス押下時の処理を実装する。タップしたポイントのX座標が“状態”列の範囲内であれば、リスト表示を行う【ソース１３】。
7-4でリスト表示処理を実装したbtnListViewをタップしたポイントへ移動させ、btnListViewClickイベントを動作させている。しかしbtnListViewボタンはVisible:=Falseで設定しており、画面上は表示されない。そのため、ポップアップのみがリストボックス上で表示されているように見える仕組みである。

ソース 13

明細タップ処理(リストボックス)

```
{*******************************************************************************
 目的: 明細タップ時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmRefer.rctGridMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
  // タップ位置(Y座標)から選択行を計算
  FSelectedRow := Trunc(Y) div cRowHeight;
    チェックボックスタップ処理【ソース7】
    ボタンタップ処理【ソース9】
  // 状態リストボックスタップ処理 Str ------------------------------------
  if (Trunc(X) >= rctJYTI.Position.X)                  and
     (Trunc(X) <= (rctJYTI.Position.X + rctJYTI.Width)) then
  begin
    btnListView.Position.X := rctJYTI.Position.X + Trunc(rctJYTI.Width / 3);
    btnListView.Position.Y := Y - vsbGrid.ViewportPosition.Y - Trunc(cRowHeight / 2);
    btnListViewClick(nil);
  end;
  // 状態リストボックスタップ処理 End ------------------------------------

  // 再描画
  rctGrid.Repaint;
end;
```

8.おわりに

本稿ではトピックとなる処理を1つずつ分けてご紹介したため、難易度が高く感じられたかもしれない。しかし、実際の総ステップ数としてはわずか500ステップ程度で作成している。さらにフレーム化や共通関数化などを行うと、複数画面作成したい場合も容易に対応可能である。
また、本稿を活用すると、データ取得ロジックは共通化し、画面の見た目のみをPCとタブレットで切り替えるといったことも可能である。例えば、データ上でログイン情報と使用端末を紐づけておけば、ログインによって自動で画面を切り替えるといったことも実現できる。ぜひ、本稿が画面設計における課題解決への一助となれば幸いである。

Smart Pad 4i

SmartPad4iからCobos4i環境へのマイグレーション

株式会社ミガロ.
プロダクト事業部　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
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(現プロダクト事業部)
　　配属

現在の仕事内容:

Cobos4i(SP4i)、Valenceの製品試験やサポート業務、導入支援などを担当している。

1.はじめに

2021年11月にSmartPad4iの後継として、Cobos4iの発売を開始した。
Cobos4iはフロントエンドを HTMLやJavaScript、CSSで開発、バックエンドにはRPG/COBOLを使用してWebアプリケーションを開発できるツールである。

Cobos4iでは、統合開発環境(IDE)としてEclipseが採用されている。
Cobos4iはIDE上で一貫して開発できるため、SmartPad4iに比べてアプリケーション開発を効率よく進めることができる。
そこで本稿では、Cobos4i でのアプリケーション開発のメリットと、SmartPad4iからCobos4iへのマイグレーション方法について紹介する。

2.Cobos4iとは？

2-1. Cobos4iの特徴

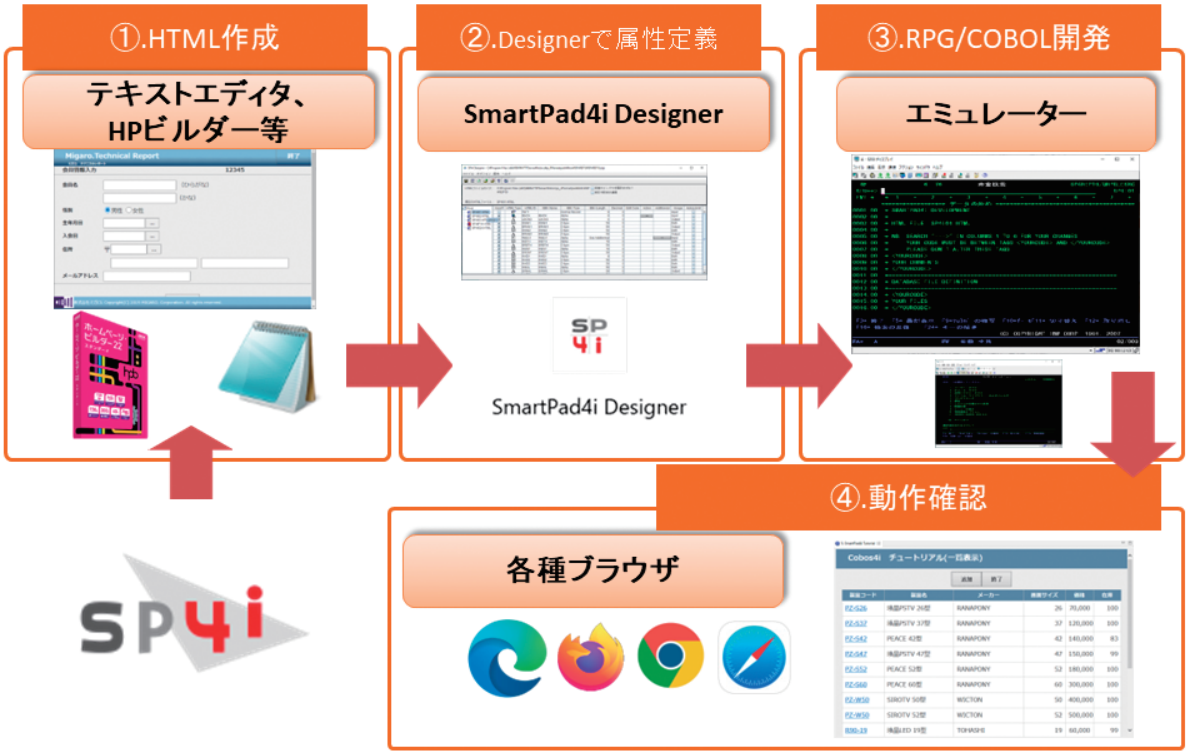
Cobos4iの開発環境はEclipseのプラグインとして組み込まれたIDEと、スタンドアローンのApache　Tomcatで構成されている。
※Apache Tomcatは、オープンソースで公開されている、Javaサーブレットを実行可能なアプリケーションサーバー

SmartPad4iでサポートしているWebサーバー環境はIBM WebSphere Application Serverのみであったが、Cobos4iからは、動作環境にWebサーバー環境としてApache Tomcatもサポートしている。
Apache Tomcatはオープンソースのため無償で利用可能というメリットがある。また、Javaアプリケーションの動作環境として一般的に使用されているため、環境構築の情報も豊富である。
また、SmartPad4iをご使用いただいているユーザー向けとなるが、Apache　Tomcatサーバー環境の構築手順は、Cobos4iのマニュアルに含められているため「Windows Tomcat環境作成マニュアル」を参考頂きたい。

開発面では、IDE上で開発ができるため、SmartPad4iの開発よりも効率が良くなる。

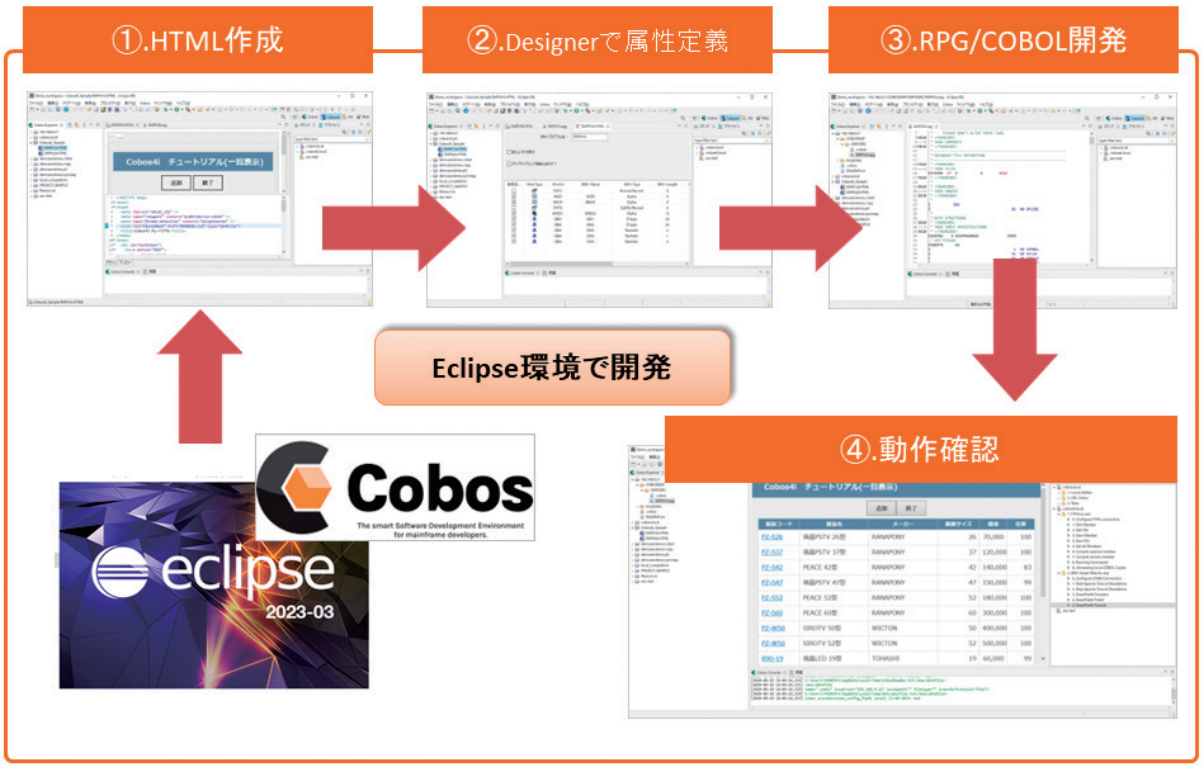
従来のSmartPad4iでのアプリケーション開発は次のような手順【図1】で、複数のツールを必要とする。
①HTMLを作成
②DesignerでHTML画面上のフィールド属性を定義
③エミュレーターからRPG/COBOLでビジネスロジックを作成
＆RPG/COBOLのコンパイル
④ブラウザを使用して動作確認

図 1　SmartPad4iでの開発



Cobos4iでは、IDE上で【図1】の手順がすべて行えるため、スムーズにアプリケーションを開発できる。
また、Apache Tomcatをスタンドアローンで動作させることでIBMiとWindows開発環境で動作確認まで完結できる。【図2】

図 2　Cobos4iの開発はすべてEclipse上で作業可能



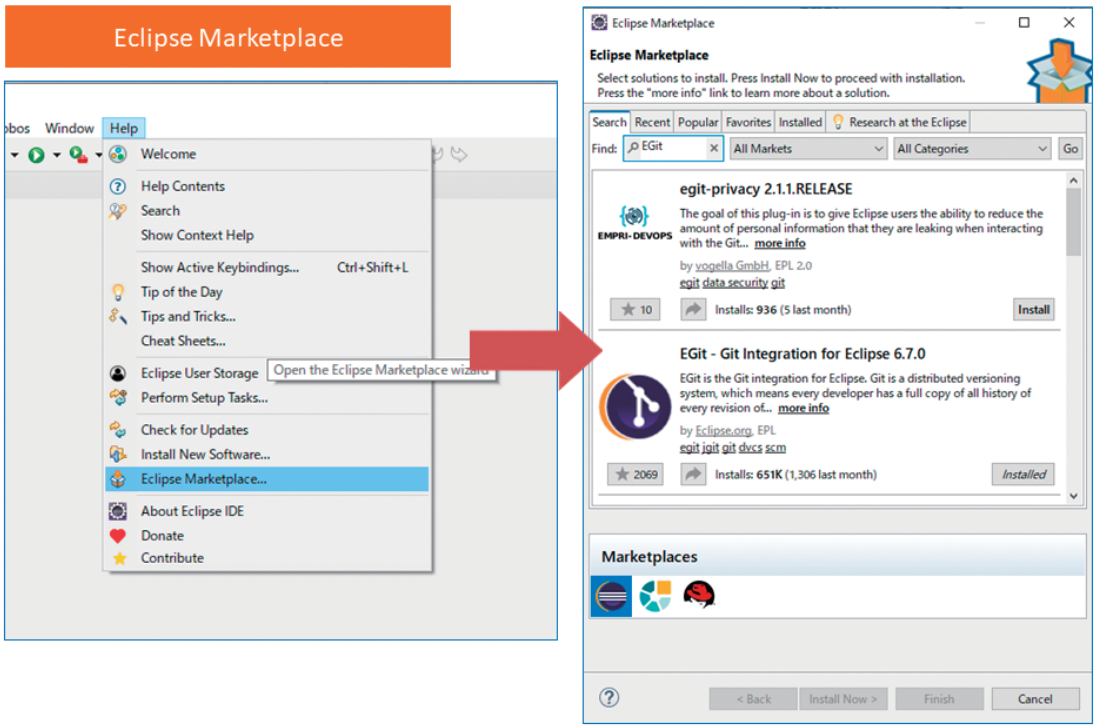
2-2. Eclipseプラグイン

Cobos4iはEclipseプラグインを導入して利用することも可能だ。
例えば、Gitのようなソフトウェアのプラグインを利用できる。
※Gitはソースコードや変更履歴を管理するために使われる、代表的な分散型バージョン管理システム

Gitのプラグインを導入することで、RPG/COBOLのソースコードをバージョン管理できる。
Gitの導入例は過去のテクニカルレポートにまとめているので参照していただきたい。
https://www.migaro.co.jp/tr/no13/tech/13_01_04.pdf

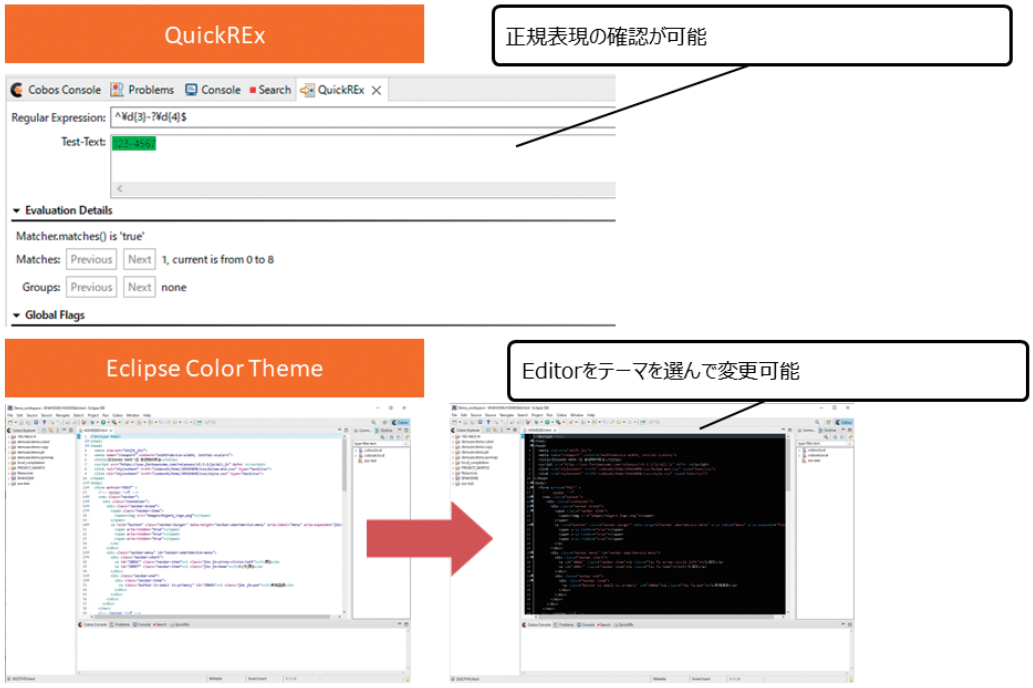
開発者が利用したいプラグインを導入することができる自由度の高さも、Cobos4iの魅力だ。
Eclipseのプラグイン導入はEclipseのメニュー[Help]>[Eclipse Marketplace]から行える。【図3】

図 3 Eclipseプラグイン



例えば、QuickRExというプラグインでは、HTMLやJavaScriptで使用する正規表現のチェックをIDEで実施できる。他には、Eclipse Color Themeを使用するとEditorを様々なテーマから選択して変更可能だ。【図4】

図 4 Eclipseプラグイン例



Eclipseには、様々なオープンソースのプラグインが存在するため、開発を効率化するプラグインを使用して、開発環境を自由にカスタマイズしてほしい。

2-3. SmartPad4iとの互換性

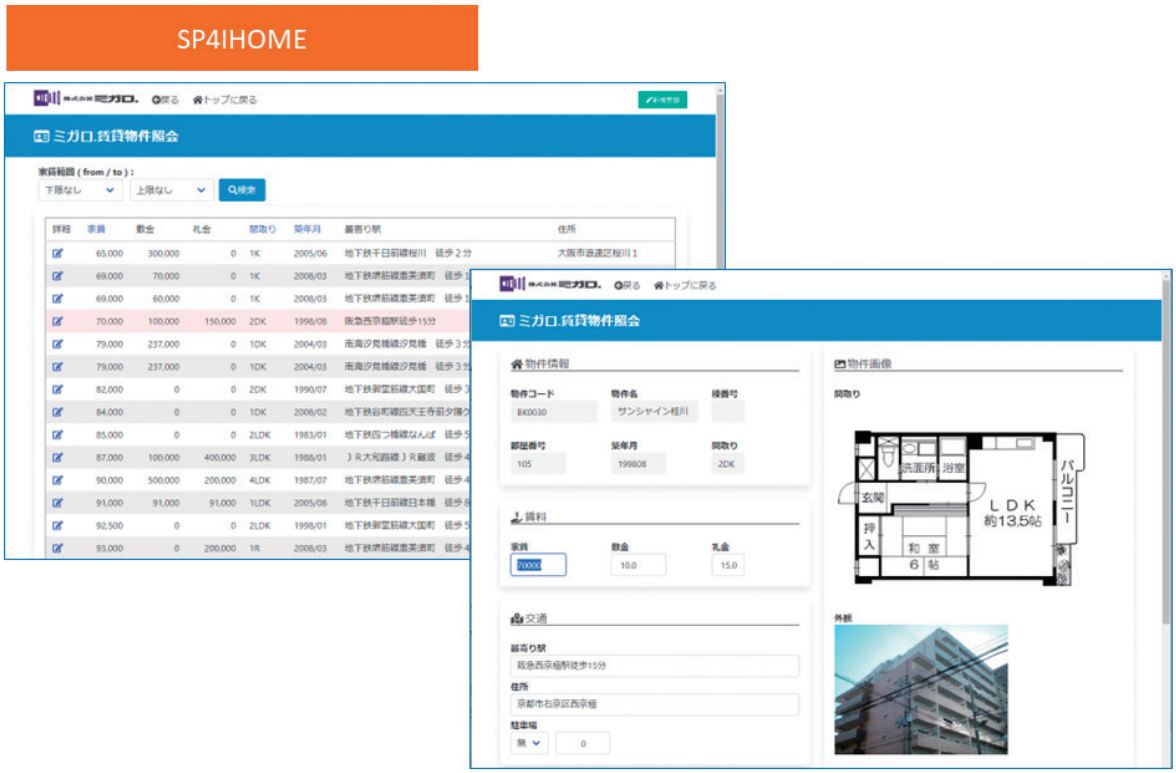
Cobos4iはSmartPad4iの後継開発ツールのため、SmartPad4iで作成したアプリケーションをCobos4iに取込み、引き続き開発することができる。
また、SmartPad4iで開発したアプリケーションもCobos4i環境で実行可能だ。

SmartPad4iアプリケーションからCobos4iアプリケーションへのマイグレーションはIDE上の操作で簡単に実行できる。

3.Cobos4iへのマイグレーション

SmartPad4iで開発したサンプルのアプリケーション（SP4IHOME）をCobos4iにマイグレーションする手順について説明する。**【図5】**

図 5　サンプルのSmartPad4iアプリケーション



マイグレーションはIDE上の操作だけで簡単に実施可能だ。また、Cobos4i用にRPG/COBOLのソースコードやJavaScriptの書換えの必要はなく、SmartPad4iのアプリケーションをCobos4iで動作させることが可能だ。

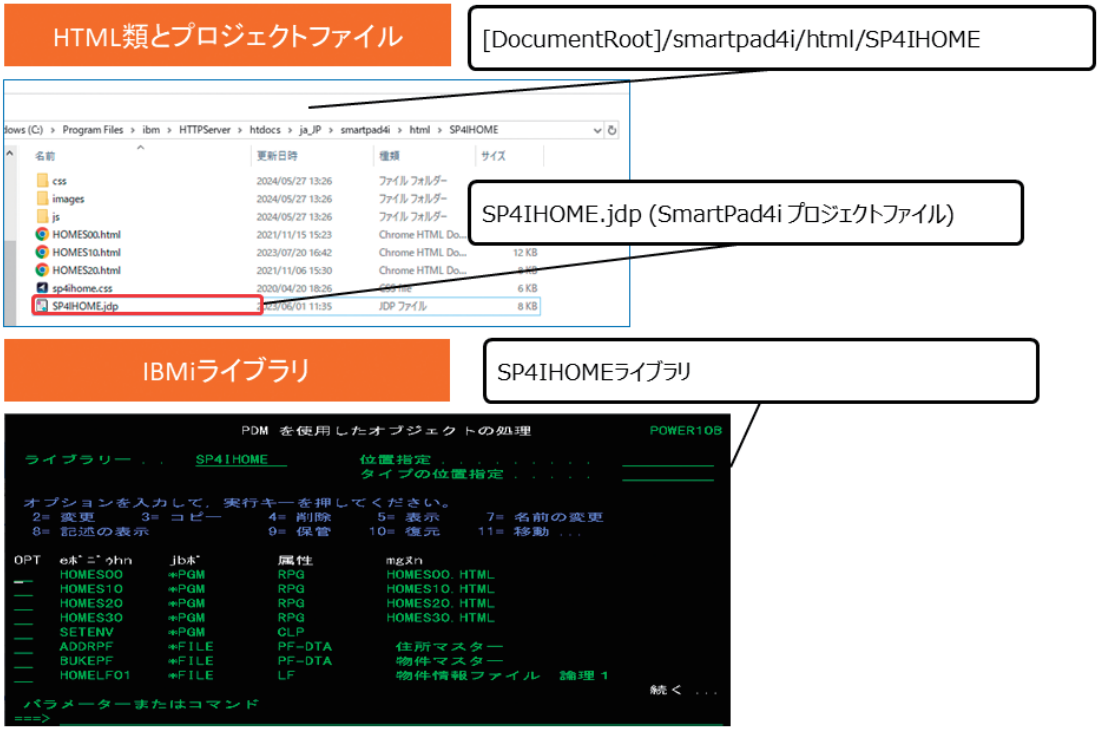
3-1. プロジェクトのマイグレーション

SmartPad4iのアプリケーション開発時には、フロントエンド/バックエンド間で入出力する項目の属性定義をSmartPad4i Designerで行う。
SmartPad4i Designerで定義した内容はプロジェクトファイル(*.jdp)として保存される。
Cobos4iにはSmartPad4i Designerで開発したプロジェクトファイル(*.jdp)をCobos4i Designerのデータに変換する機能が備わっているため、簡単にSmartPad4iのプロジェクトをCobos4iのプロジェクトにマイグレーション可能だ。

次のような環境で開発した、SmartPad4iアプリケーションをCobos4iへマイグレーションする手順を例として説明する。

◆IBMiプログラム/ソースを管理するライブラリ
SP4IHOME
◆HTML/CSS/JavaScript
[DocumentRoot]/smartpad4i/html/SP4IHOME
※Windows版のWebSphere Application Server環境で開発している場合、[DocumentRoot]は一般的に以下のようになる。
(C:\Program Files\ibm\HTTPServer\htdocs\)
◆プロジェクトファイル
[DocumentRoot]/smartpad4i/html/SP4IHOME/SP4IHOME.jdp
【図6】

図 6　サンプルのSmartPad4iのプロジェクト



SP4IHOMEライブラリにプログラムとソースを展開して作成しているSmartPad4iアプリケーションの場合、[DocumentRoot]/smartpad4i/html/SP4IHOME フォルダにHTMLが格納されている。
また、入出力の属性を定義したiniファイルは[DocumentRoot]/smartpad4i/infos/SP4IHOMEに保存されている。
この両フォルダを、既存の開発環境から、Cobos4i開発環境(Tomcatサーバー)のWebディレクトリに配置する。

◆HTML
C:\SystemObjects\Cobos4i\Tomcat\webapps\cobos4i\html\SP4IHOME

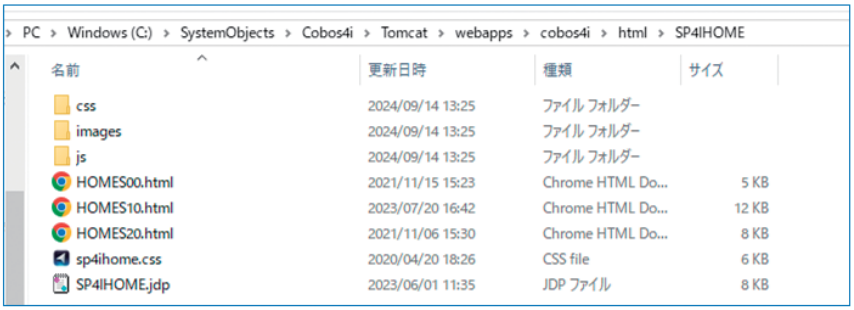
◆iniファイル
C:\SystemObjects\Cobos4i\Tomcat\webapps\cobos4i\infos\SP4IHOME

次に、プロジェクトファイル(*.jdp)をテキストエディタで開き、Baseセクションのパスを変更する。

C:\ProgramFiles\IBM\HTTPServer\htdocs\ja_JP\smartpad4i\html\SP4IHOME
→
C:\SystemObjects\Cobos4i\Tomcat\webapps\cobos4i\html\SP4IHOME\
【図7】

図 7　サンプルのSmartPad4iのプロジェクト

C:¥SystemObjects¥Cobos4i¥Tomcat¥webapps¥cobos4i¥html¥SP4IHOME



SP4IHOME.jdp

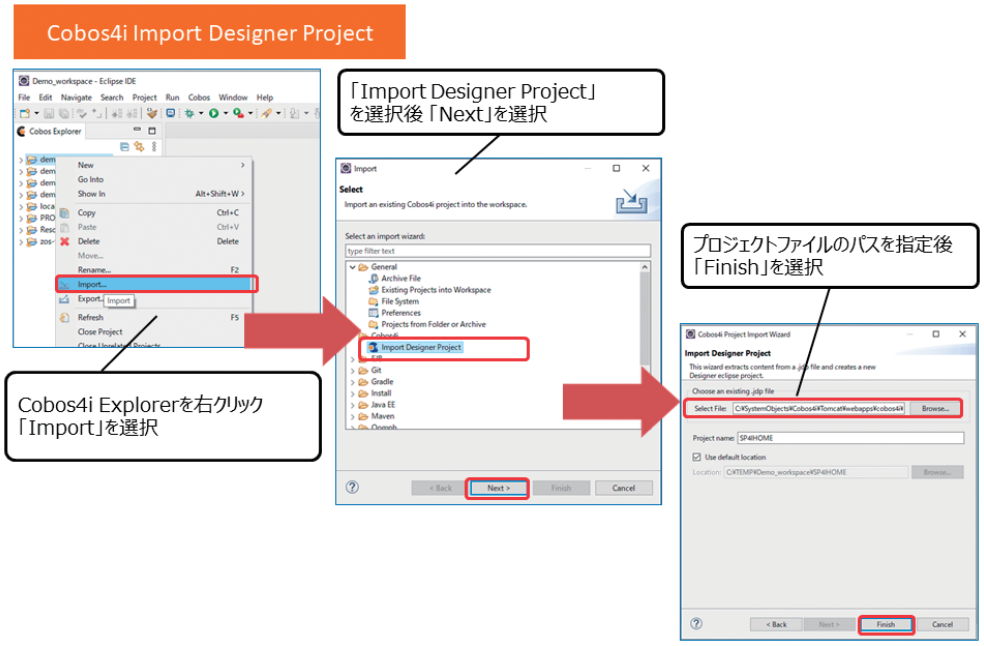
```
[MAIN]
Base=C:¥Program Files¥IBM¥HTTPServer¥htdocs¥ja_JP¥smartpad4i¥html¥SP4IHOME
```

```
[MAIN]
Base=C:¥SystemObjects¥Cobos4i¥Tomcat¥webapps¥cobos4i¥html¥SP4IHOME
```

次に、Cobos Explorer を右クリックして表示されるメニューから［Import］を選択して、［Cobos4i］>［Import Designer Project］を実行する。

インポートウィザードでプロジェクトファイル（*.jdp）を指定後［Finish］ボタンで Cobos4i プロジェクトへの取込みが完了する。**【図 8】**

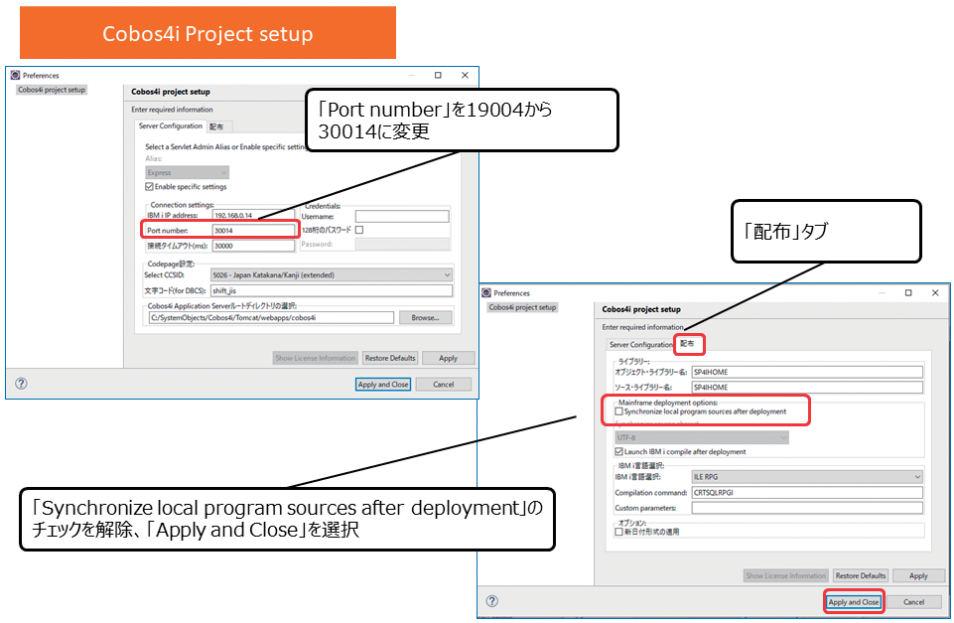
図 8 SmartPad4iプロジェクトファイルをインポート



変換後、Cobos4iのプロジェクト設定画面が開くため、SmartPad4iのポート番号19004からCobos4iのポート番号30014に変更する必要がある点に注意してほしい。

最後に、［配布］タブに移動して、［Synchronize local program sources after deployment］のチェックを解除後、［Apply and Close］を選択するとCobos4 Designerプロジェクトの設定が完了する。**【図9】**

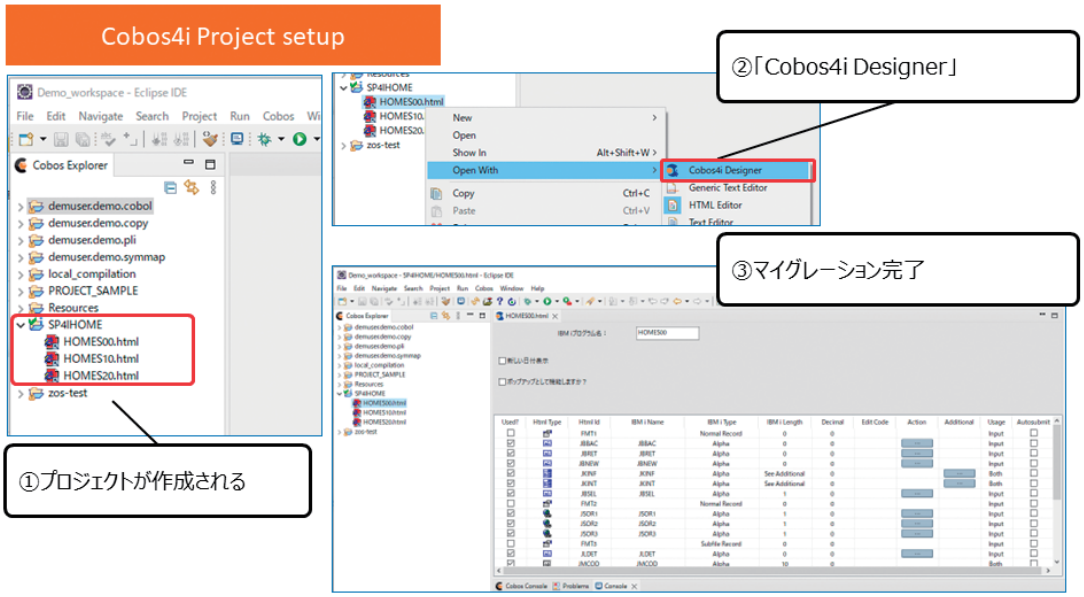
図 9 Cobos4i接続設定



Cobos4iエクスプローラーからプロジェクト内のファイルを右クリックして表示されるメニューから、Cobos4i Designerでオープンすると、SmartPad4i Designerのプロジェクトが Cobos4iに取込まれていることを確認できるだろう。

以上で、SmartPad4i Designerのプロジェクトから Cobos4i Designerプロジェクトへのマイグレーションが完了できた。**【図10】**

図 10　Cobos4iプロジェクトに変換



3-2. IBMiプログラムソースコードの取込み

Designerの定義が読み込まれたので、次にIBMiのSP4ISMPライブラリに存在するソースコードQRPGSRCのメンバーをCobos4i開発環境にダウンロードする。

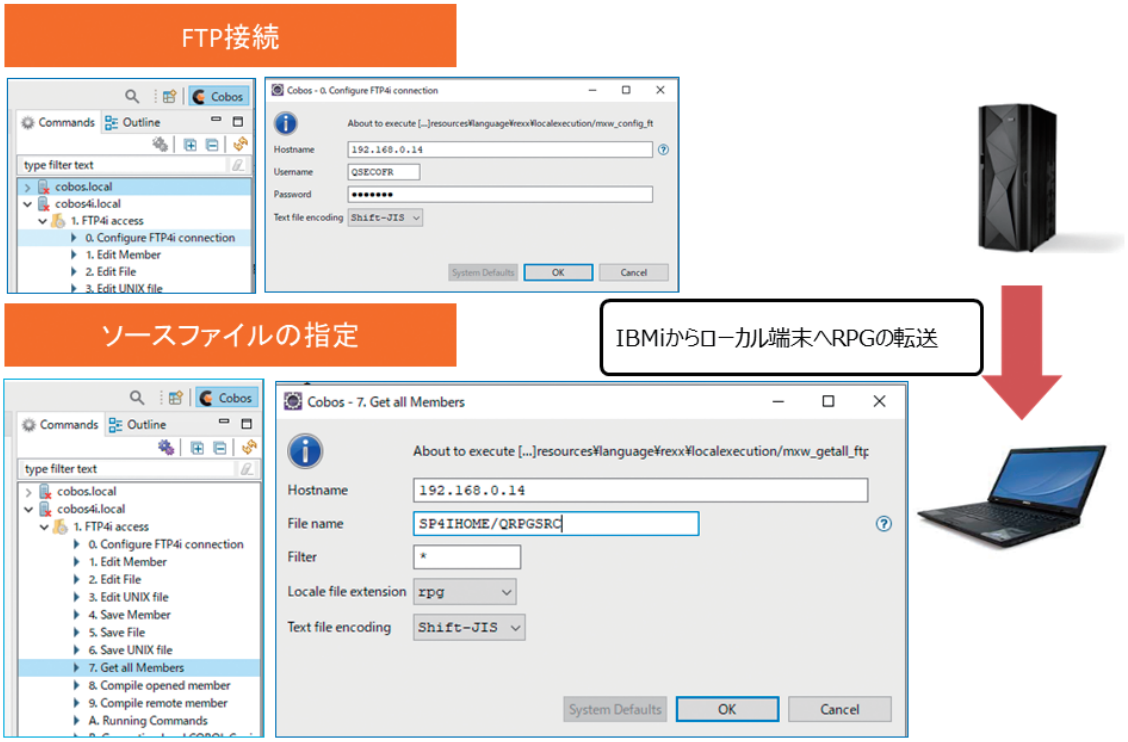
コマンドビューの[cobos4i local]ノードを開き、[1.FTP4i access]をさらに展開する。

[0.Configure FTP4i connection]をダブルクリックすると、IDEとIBMi間でFTP接続設定が表示されるため、IBMiのユーザープロファイル・パスワードと転送時のエンコード(Shift_JIS)を指定する。

FTP接続設定完了後、[1.FTP4i access] > [7.Get all Members]を選択して表示されるダイアログでIBMiのソースファイル名とソースの文字コードを設定する。

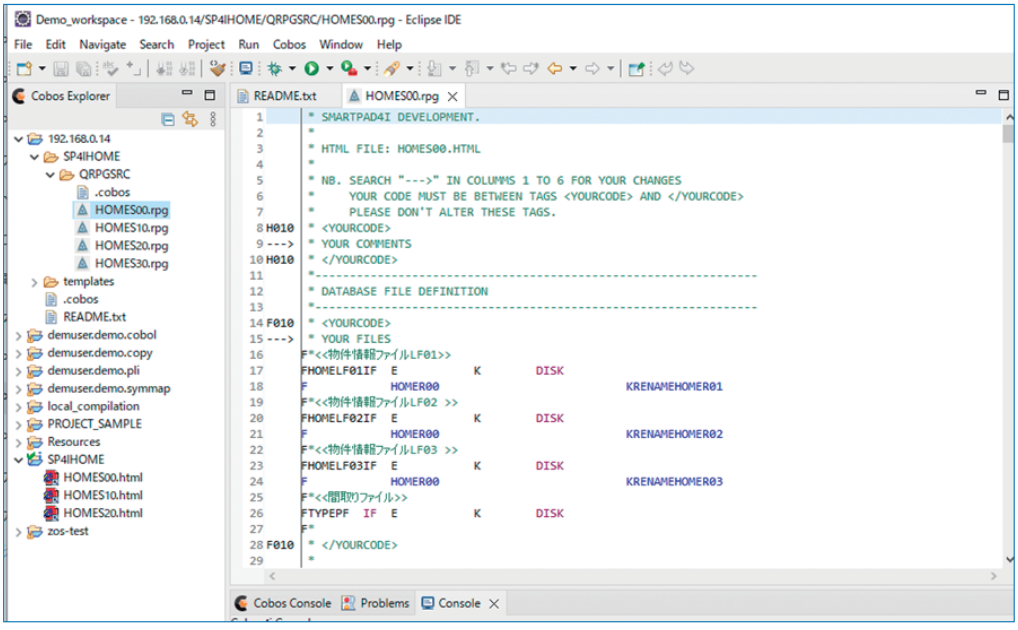
File nameにSP4IHOME/QRPGSRCを入力、文字コードはFTP接続と同じShift_JISを選択して、[OK]ボタンをクリックすると、IDEにRPGソースが取込まれる。**【図11】**

図 11 IDEにソースの取り込み



IDEに取込まれたソースコードはRPG　Editor上で編集可能だ。【図12】

図 12 IDEでソースを編集

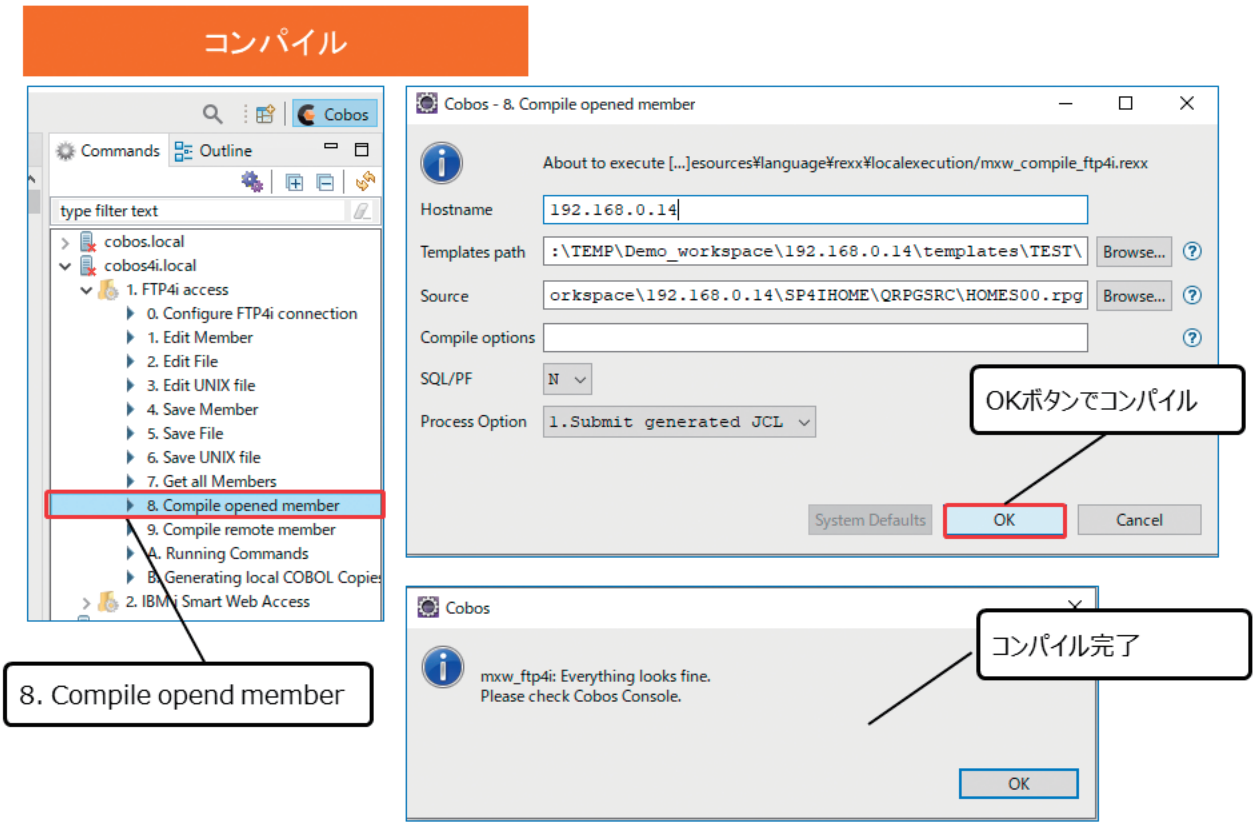


3-3. 統合開発環境上からのコンパイル

Cobos4iでは、IDE上で直接IBMiプログラムをコンパイル可能である。

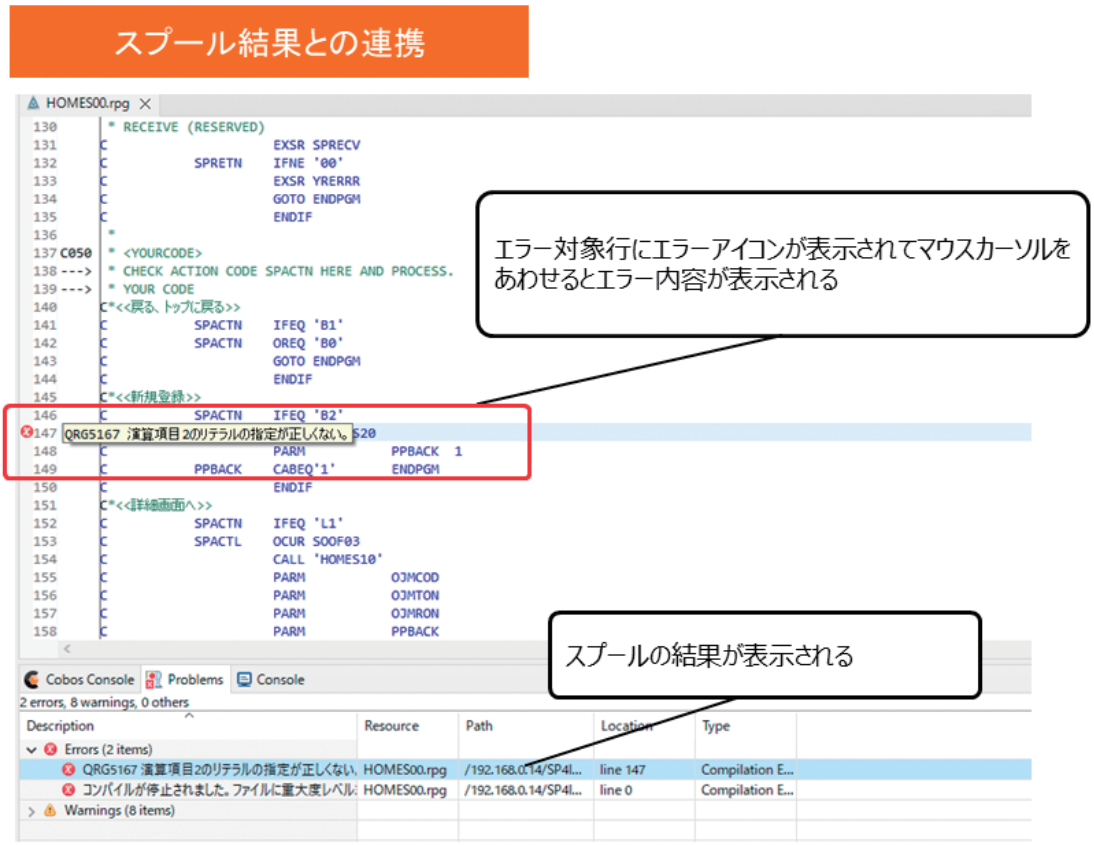
コマンドビューの[1.FTP4i access]から、[8.Compile opend member]を実行して、表示されるダイアログから[OK]ボタンをクリックすることでコンパイルが完了する。**【図13】**

図 13　IDEからコンパイルの実行



また、コンパイルエラーがあった場合、コンパイル実行時に出力されるスプールファイルから、エラー原因がエディター上に出力される。**【図14】**

図 14　IDEでデバッグ



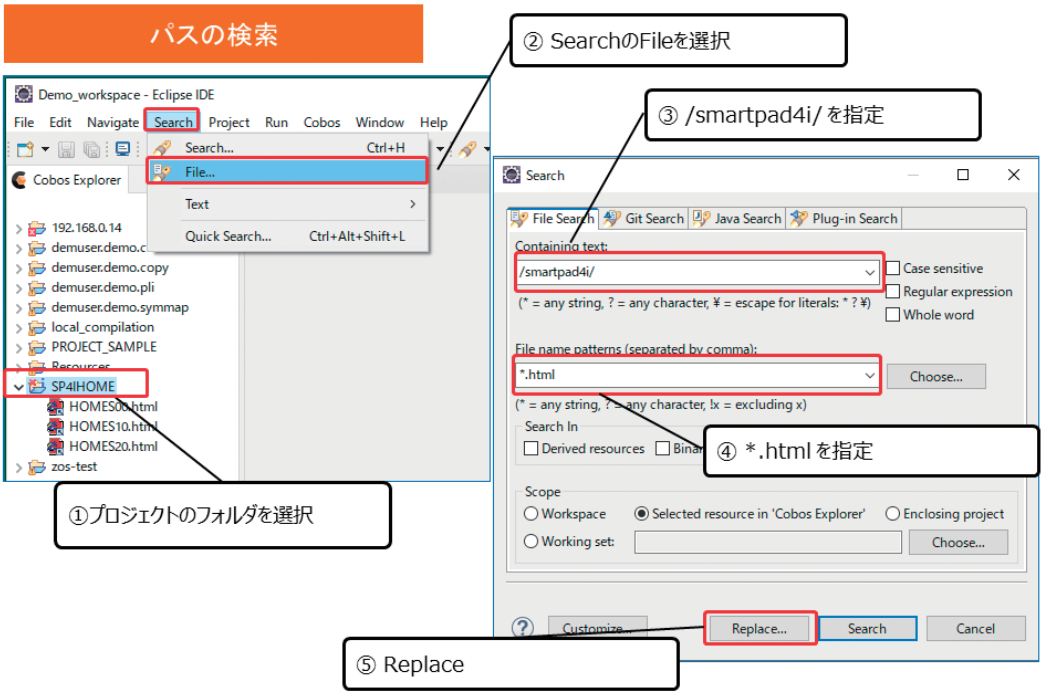
通常、エミュレーターで開発している場合には、スプールを確認してコンパイルエラーの原因を特定する手順が必要となる。Cobos4iではコンパイルエラー時にスプールの確認する手間が不要なため、この点も開発効率の向上につながっている。

3-4. HTMLの編集（置換）

Cobos4iの実行環境は[DocumentRoot]/cobos4iに展開されている。

SmartPad4iの実行環境は[DocumentRoot]/smartpad4iのため、作成しているアプリケーションのHTMLに/smartpad4i/のパス記述がある場合には、/cobos4i/に置換する必要がある。置換方法は下記の手順で、IDEの機能を使用して簡単に行える。**【図15】**

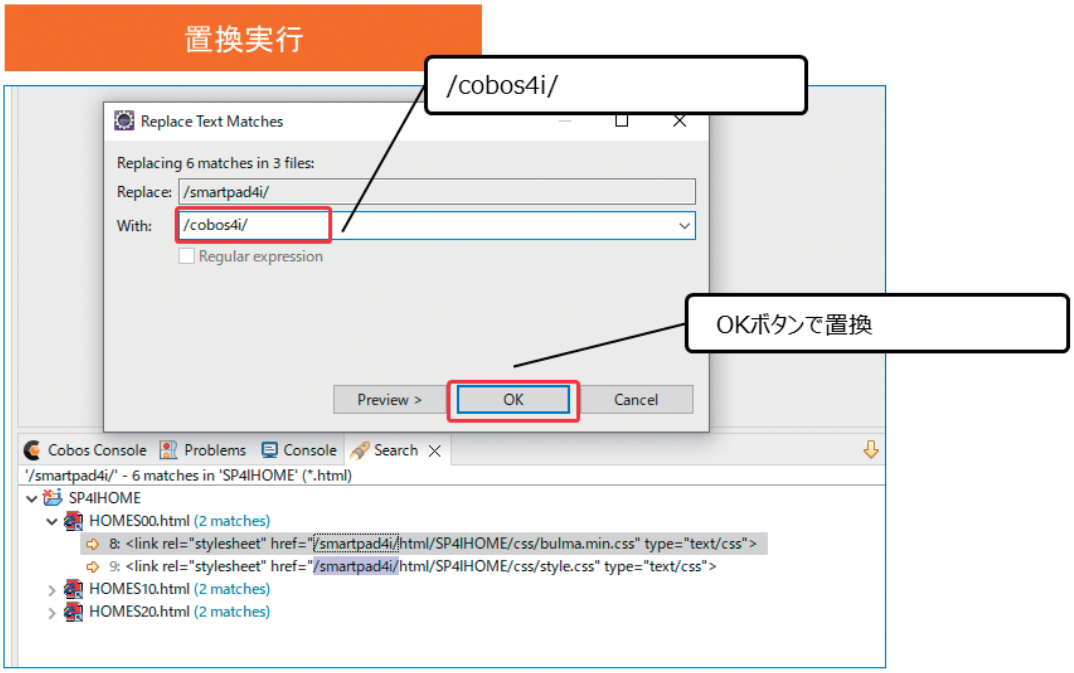
図 15 HTMLの置換



①Cobos Explorerで置換対象のプロジェクトフォルダを選択
②メニューの[Search]> [File]を選択
③Searchダイアログで置換対象の文字列を指定する[Containing text]に/smartpad4i/を入力
④対象ファイルの拡張子を指定する[File name patterns]に*.htmlを入力
⑤[Replace]ボタンを選択して実行

検索結果が表示された後、[With]に置換後の文字列を入力して[OK]ボタンをクリックすると、HTML内のパスが一括置換できる。**【図16】**

図 16 HTMLの置換

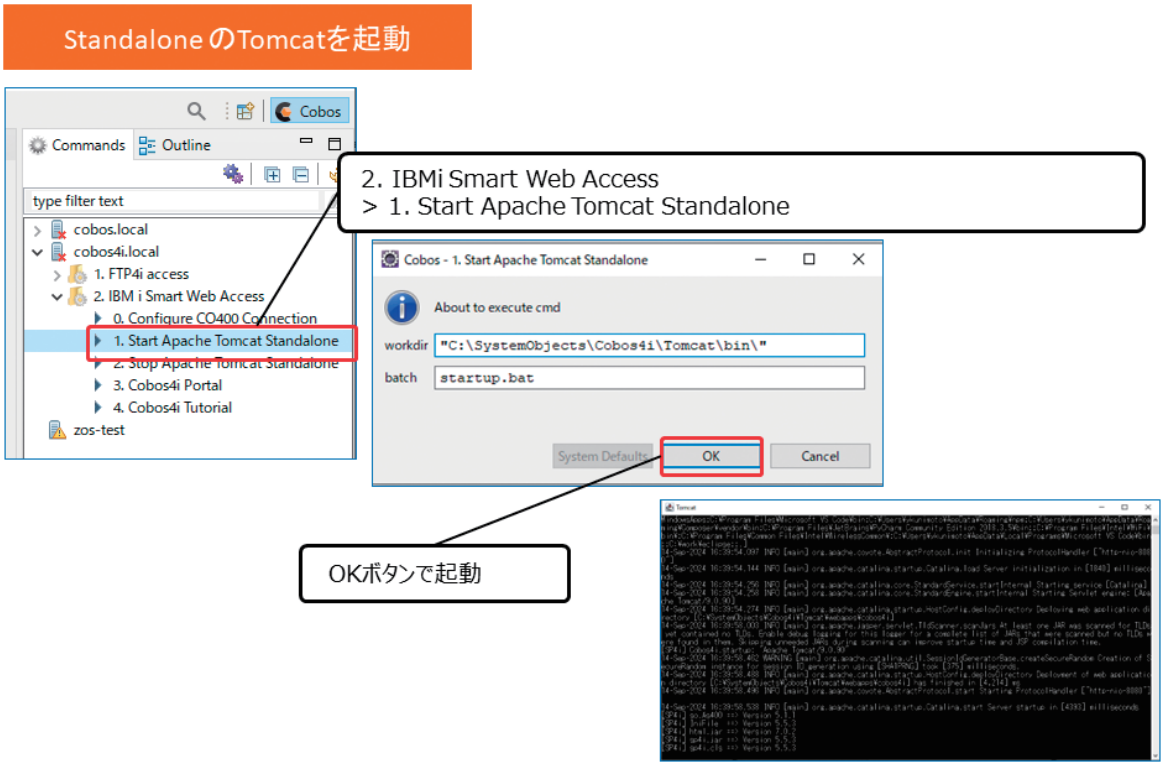


3-5. 動作確認

Cobos4iの開発環境には、スタンドアローンのApache Tomcatサーバーが含まれているため、サーバーを起動して動作確認をIDE上で行える。

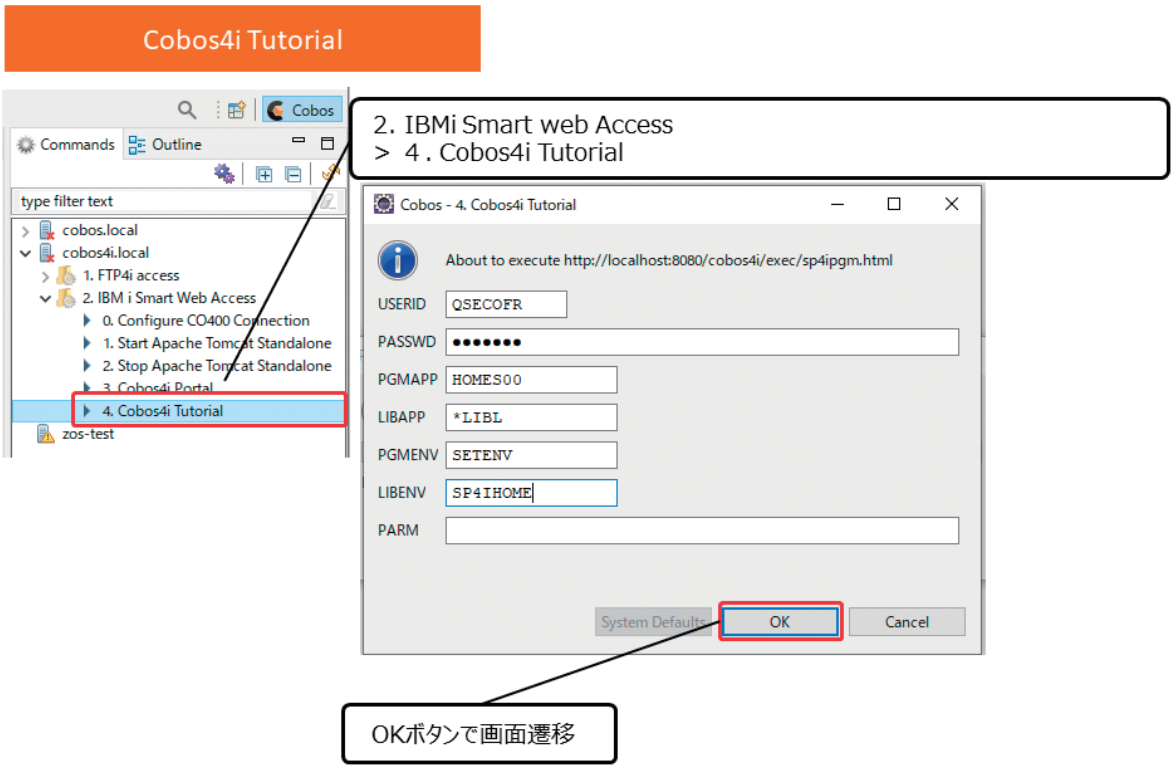
コマンドビューの[2.IBM i Smart Web Access]> [1.Start Apache Tomcat Standalone]を実行する。ダイアログに表示される[OK]ボタンをクリックするとApache Tomcatのサーバーが起動する。**【図17】**

図 17 Tomcatの起動



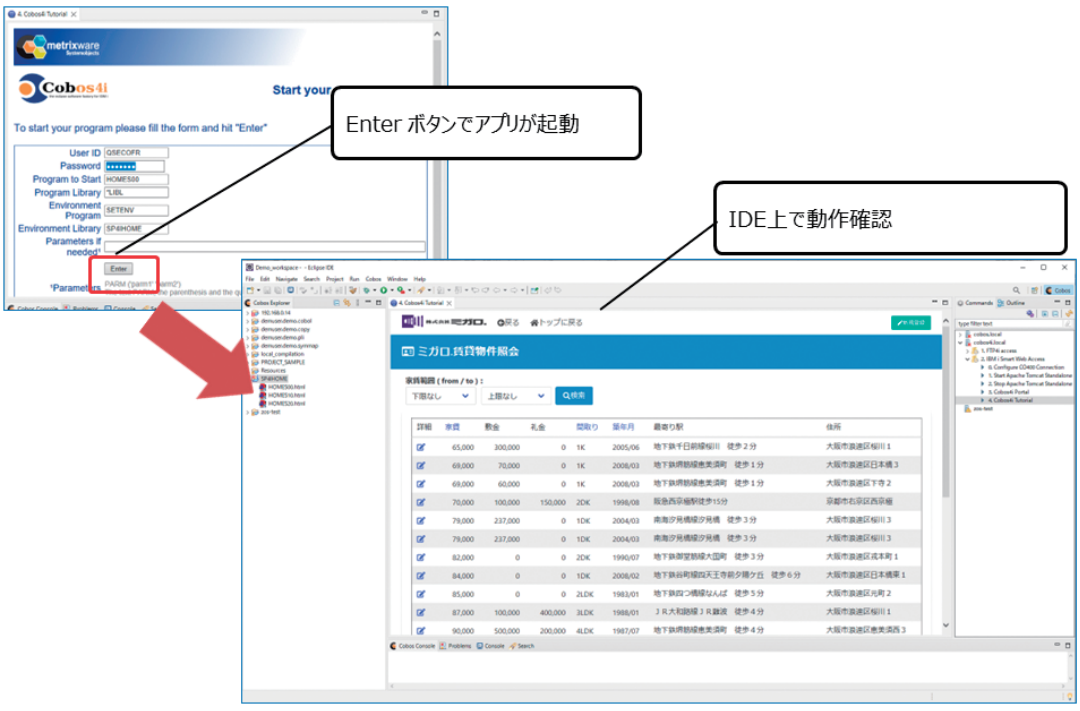
開発したアプリケーションを起動するには、コマンドビューの[2.IBMi Smart Web Access]> [4.Cobos4i Tutorial]を実行、ダイアログが表示されるので、ユーザープロファイル、パスワード、プログラム名、環境設定用のプログラム名、ライブラリを指定して[OK]ボタンを選択する。【図18】

図 18 IDE上で動作確認 ①



OKボタンをクリックすると、画面が遷移して、Cobos4iアプリケーション起動用のWeb画面がIDE上で開くため、Web画面上で[Enter]ボタンをクリックすると、開発したアプリケーションを実行することができる。**【図19】**

図 19 IDE上で動作確認 ②



4.おわりに

Cobos4iは運用サーバー環境としてApache　Tomcatがサポートされるようになった点は大きな利点だ。
また、Cobos4iのアプリケーション開発では、コーディング、コンパイル、デバッグ等の開発工程をIDE上ですべて行えることがお分かりいただけたと思う。

IDEやプラグインを使用することで、開発効率やプログラムの保守性を向上させることができるため、SmartPad4iからCobos4iへのマイグレーションについて、ご検討いただければ幸いである。

Valence

Valence App Builder
アプリ変数の基本と応用テクニック

株式会社ミガロ.
プロダクト事業部　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
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（現プロダクト事業部）配属

現在の仕事内容：
ミガロ. が取り扱う3つの開発ツールのセミナー講師や技術支援を主に担当している。

1.はじめに

IBM i環境に特化したモダナイゼーションツールValenceは、2018年8月に登場したValence5.2からローコード開発機能である「App Builder」が追加され、Webアプリの開発生産性が大幅に向上した。それまでは、Senchaと呼ばれるJavaScriptベースのUIフレームワークを使用して、画面やイベント処理を全て独自開発する必要があったが、「App Builder」の登場により、容易な定義のみでWebアプリが作成できるようになった。

ただ、Valence5.2当時の「App Builder」は、まだ機能面で不十分なところも多く、痒い所に手が届かない部分があったのも事実である。そんな「App Builder」の機能が大きく向上したのが、2020年12月に登場したメジャーバージョンアップ版であるValence6.0である。大幅な機能拡張の中でも特にインパクトの大きかったのが、本稿で取り上げる「アプリ変数」の導入である。所謂「変数」がWebアプリ内で保持できるようになったのである。

それでも「アプリ変数」が登場した当初は、使いどころは限定されていると感じていたのだが、その後のバージョンアップで機能が拡張されていく中、この「アプリ変数」こそがValenceアプリ作成の肝となるという事がわかってきた。「App Builder」に「アプリ変数」が加わった事で、本当の意味での「ローコード開発ツール」になったといっても過言ではない。

本稿ではこの「アプリ変数」に注目し、基本的な使用方法から活用例、応用的なテクニックなどを紹介したい。なお本稿では執筆時点（2024年8月）の最新版であるValence6.2.20240514.0を使用している。本稿公開時には更なるバージョンアップによりアプリ変数の更なる活用方法が追加されているであろう。また、本稿で紹介している各種画面等がバージョンアップにより変更されている可能性がある点についてはご了承頂きたい。

2.アプリ変数とは？

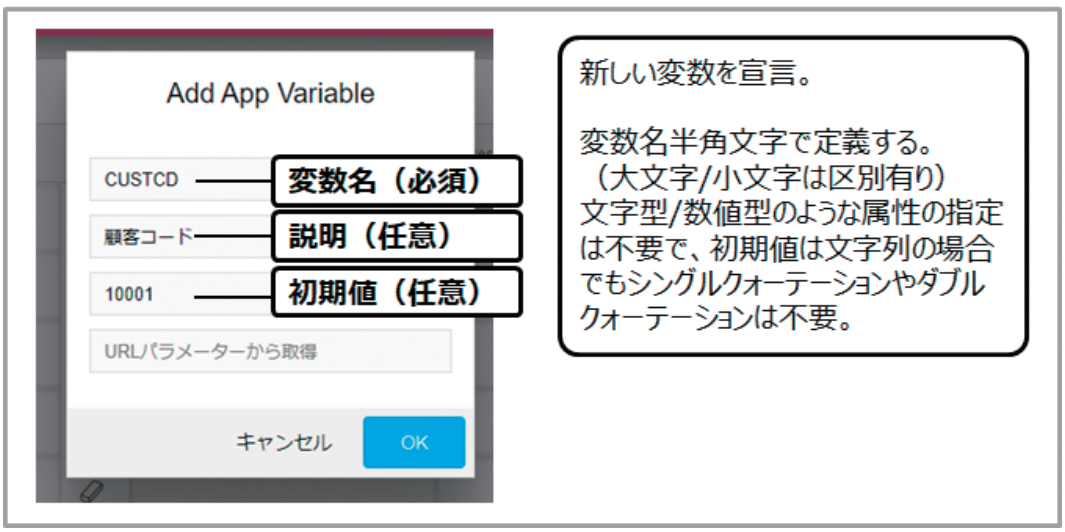
App Builderにおけるアプリ開発手順には、[①データソース作成]→[②ウィジェット作成]→[③アプリケーション作成]という3つのステップがある。「アプリ変数」は、その名のとおり[③アプリケーション作成]時に定義ができる。

アプリケーション編集画面の左上にある3つのアイコンの一つである[アプリ変数]ボタンをクリックすれば、アプリ変数を宣言する為の画面が表示される。**【図1】**

図 1　アプリ変数　宣言画面

新しい変数を追加する場合、アプリ変数宣言画面の右上部にある[Add]ボタンをクリックすると、**【図2】**のような変数宣言画面が表示される。変数名は、半角文字で定義すればよい。また、変数に対する説明や初期値を設定する事もできる。

図 2　新規アプリ変数の宣言



なお、アプリ変数宣言画面には、予めいくつかの青い背景色がついたアプリ変数が宣言されている事がわかる。これらは、[nab～]から始まるデフォルトアプリ変数である。デフォルトアプリ変数の一覧は、**【図3】**のとおりである。

図 3　デフォルトアプリ変数一覧

アプリ変数宣言画面に設定可能なデフォルトアプリ変数一覧

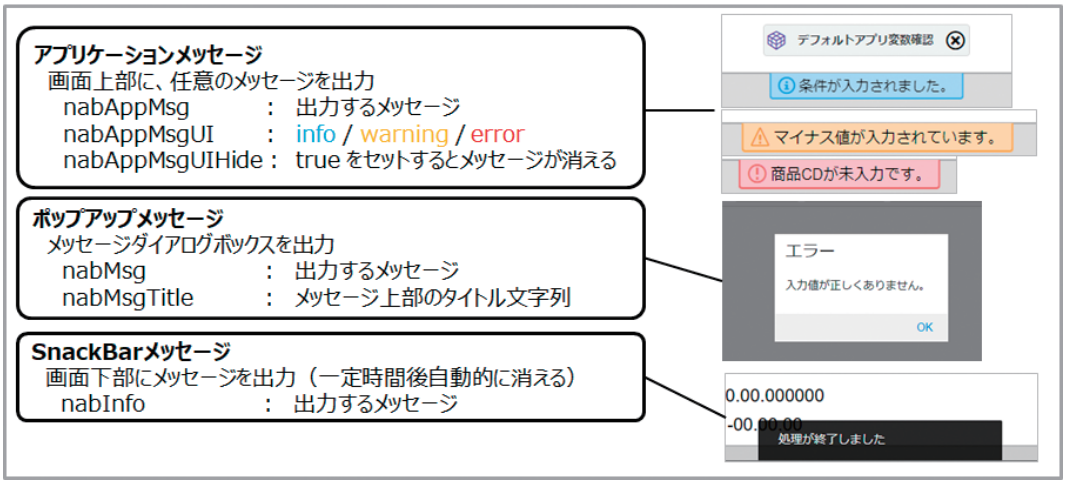
変数名	概要	備考
nabAppMsg	アプリケーションメッセージを表示	
nabAppMsgHide	アプリケーションメッセージを隠す	true：隠す
nabAppMsgUI	アプリケーションメッセージUI	info：情報、warning：警告、error：エラー
nabChangeAppName	アプリケーション名を変更	PCのみ有効
nabFireLocalEvent	指定された名称のローカルイベントを実行	
nabHideAppBar	アプリケーションタイトルバーを隠す	true：隠す
nabHideBreadcrumbs	パンくずリストを隠す	true：隠す
nabInfo	SnackBar メッセージを表示	
nabMsg	ポップアップメッセージを表示	
nabMsgTitle	ポップアップメッセージのタイトル	
nabPromptBeforeClose	アプリケーション終了時に表示するメッセージ	

例えば、[nabPromptBeforeClose]に"このまま終了すると、未保存データは破棄されます。" と初期値を設定すると、実行中のアプリを終了しようとした時に、必ず終了確認メッセージが表示されるようになる。**【図4】**

図 4　デフォルトアプリ変数設定例

その他デフォルトアプリ変数には、アプリケーション上に、任意のメッセージが出力できるものがあり、3種類のメッセージ出力方法が用意されている。これらのアプリ変数に値を設定した場合の実行イメージは【図5】のとおりである。

図 5　デフォルトアプリ変数　メッセージ出力例



また、デフォルトアプリ変数には、アプリケーション実行時に自動的に値がセットされるものもある。例えば[nabUser]は、ログインしているユーザー名を取得できるアプリ変数である。これらのアプリ変数は値の取得のみ可能な読取専用の変数となる。主な読取専用のアプリ変数は【図6】のとおりである。

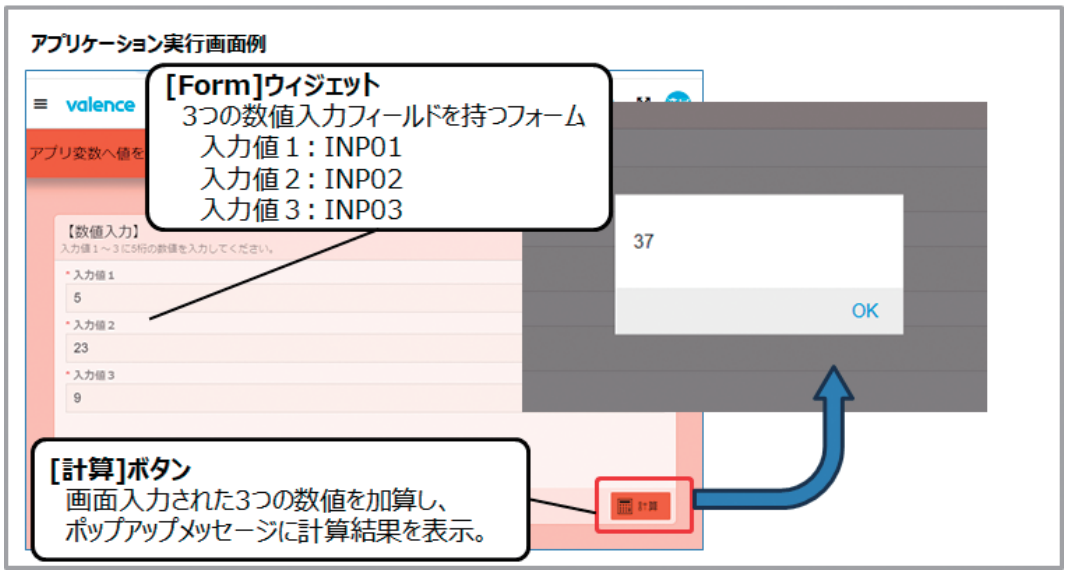
図 6　読取専用　デフォルトアプリ変数一覧

アプリケーション実行時に自動的にセットされる読取専用のデフォルトアプリ変数

変数名	概要	備考
nabActiveFormHelperCnt	Formヘルパープログラム(RPG)実行中カウント	FormHelperプログラム実行中は、1となる
nabCurrenctSection	現在開いているセクション名	
nabIbmI	IBMi接続ユーザー名	ログインユーザー情報
nabUserID	ValenceのユーザーID（数値4桁）	ログインユーザー情報
nabUser	Valenceのユーザー名	ログインユーザー情報
nabUserEmail	Valenceユーザーの登録メールアドレス	ログインユーザー情報
nabIsDesktop	PCブラウザでの実行かどうか	PCブラウザでの実行の場合、true
nabIsMobile	モバイルアプリでの実行かどうか	モバイルアプリでの実行の場合、true
nabIsPhone	スマートフォンからの実行かどうか	スマートフォンからの実行の場合、true
nabIsTablet	タブレットからの実行かどうか	タブレットからの実行の場合、true
nabNullDate	IBMi日付型のNULL値	“0001-01-01”が格納
nabNullTimeStamp	IBMiタイムスタンプ型のNULL値	“0001-01-01-00.00.00.000000”が格納
nabNullTimestampShort	IBMiタイムスタンプ型（秒迄）のNULL値	”0001-01-01-00.00.00”が格納

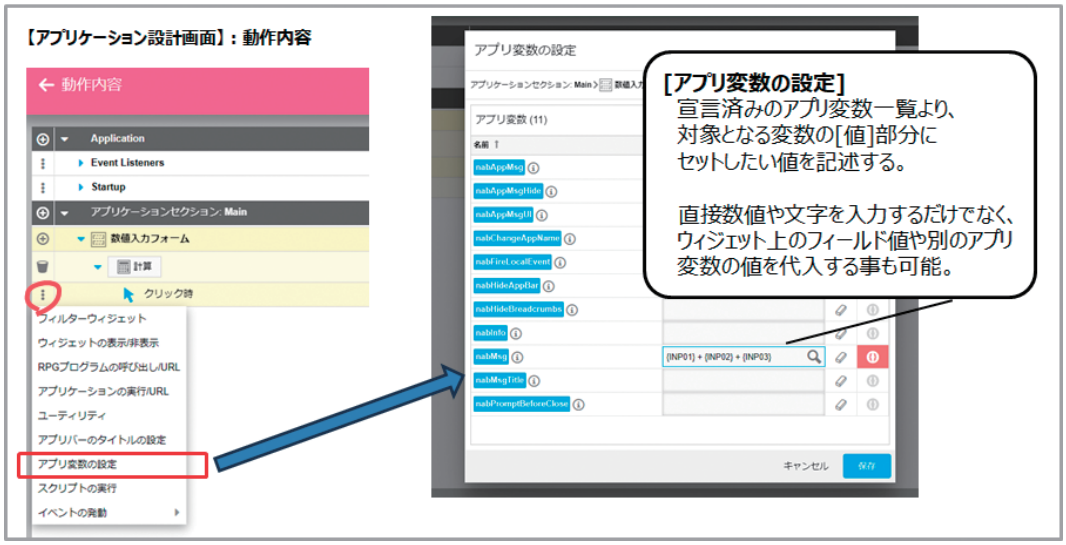
読取専用のアプリ変数以外は、もちろんアプリケーション実行中に値を代入する事も可能である。具体例を紹介する。3つの数値入力フィールド（[INP01]～[INP03]）をもつ[Form]ウィジェットがあり、[Form]の下部に追加した[計算]ボタンをクリックした時に、3つの数値を合計し、計算結果をポップアップメッセージに出力するアプリケーションを考える。**【図7】**

図 7　アプリ変数へ値をセットするプログラム例



App Builderでは独自のボタン追加やアクション設定は、「③アプリケーション作成」の[動作内容]にて定義する。アプリ変数への値の代入もアクションとして定義できる。具体的な手順としては、[Form]ウィジェット上に追加した[計算]ボタンの[クリック時]アクションとして[アプリ変数の設定]を追加する。アプリ変数の設定画面には、定義済みのアプリ変数が一覧表示されるので、目的のアプリ変数に代入したい値を記述すればよい。**【図8】**

図 8　アプリ変換に値をセット



値には直接数値や文字列を記述する事もできるが、入力欄右側の虫眼鏡アイコンをクリックすると、ウィジェットのフィールド値や別のアプリ変数の値を選択する事もできるので、利用してほしい。

今回は、3つの入力値を加算した結果をポップアップメッセージとして出力したい為、デフォルトアプリ変数である[nabMsg]に"{INP01}+{INP02}+{INP03}"と代入している。ここで注意点だが、値の記述後に設定画面の右側にあるアイコン(Expression)を有効にしてほしい。これは値部分を計算式として処理する為のオプション設定である。この設定を有効にしない場合、変数には文字列がそのまま代入される為、例えば"5+23+9"のような文字列がアプリ変数にセットされてしまう。

このようにアプリ変数は、App Builderにおいてアプリ単位の変数として利用できるわけだが、実はRPGプログラムの連携を行えば、RPG側でもアプリ変数の値の参照や代入が可能となる。次にRPGプログラムにおけるアプリ変数の使用方法について紹介する。

なお、本稿ではApp Builderにおける RPGプログラムとの連携手順について詳細は触れないが、2019年度のテクニカルレポート「Valence App Builder RPG連携テクニック」(https://www.migaro.co.jp/tr/no12/tech/12_01_06.pdf)にて詳しく解説しているので、そちらも参照してほしい。

ここで紹介するのは、**【図9】**のようなファイルレイアウトを持つ商品マスタ(MPRDTP)に新規レコードを登録する為の入力アプリである。

図 9 商品マスタ(MPRDTP)ファイルレイアウト

DSPFMT　　レ コ ー ド 設 計 書　　日付 24/09/13　時刻 18:52:57

物理ファイル	TECREP2024/MPRDTP	様式名	MPRDTPR	レコード長	46
様式記述	商品マスター				

5= 詳細

選択	項目名	桁数	属性	キー順	開始	終了	テキスト記述
_	PRPRCD	5	A	1 ANN	1	5	商品コード
_	PRPRNM	32	O		6	37	商品名
_	PRUNPR	7 0	S		38	44	単価
_	PRCTCD	2	A		45	46	カテゴリＣＤ

このアプリは、**【図10】**のようなシンプルな[Form]ウィジェットを使用した画面である。

図 10 商品マスタ新規登録アプリ

先程の例と同様、[登録]ボタンの[クリック時]アクションにて、[アプリ変数の設定]を追加し、画面上の4つの入力値をそれぞれアプリ変数にセットしている。そして、[RPGプログラムの呼出し]アクションを追加し、"REP24PG01"というプログラムを呼び出すようにしている。

呼び出されるRPGプログラムは、**【ソース1】**である。このプログラムは、テンプレートソース(EXNABBTN)をコピーして作成したものである。1-①のようにEXNABBTNに用意されたApp Builder専用APIであるGetAppVarメソッドを使用すれば、メソッドの引数としてアプリ変数名を指定するだけで、対象の変数値を文字列として取得できる。なお、アプリ変数は定義上、文字型や数値型といったデータ属性の区別がなく、全て変数値は文字列として取得される為、RPG側で数値項目として処理する場合は、数値型にキャスト(型変換)する必要がある事に注意してほしい。

ソース 1 **REP24PG01：商品マスタ新規登録**

```
 /copy qcpylesrc,vvHspec
** ----------------------------------------------------------------
** REP24PG01:商品マスタ　新規登録
** ----------------------------------------------------------------
F* ----------------------------------------------------------------
F* ファイル定義
F* ----------------------------------------------------------------
F*<商品マスタ>
FMPRDTP    UF A E           K DISK
F*
** ----------------------------------------------------------------
 /include qcpylesrc,vvNabBtn
** ----------------------------------------------------------------
** program start
** ----------------------------------------------------------------
 /free
  Initialize();
  Process();
  CleanUp();
  *inlr=*on;
 /end-free
** ----------------------------------------------------------------
p Process         b
d                 pi
D*---変数宣言
D VPRCD           S              5A
D VPRNM           S             32A
D VUNPR           S              7  0
D VCTCD           S              2A
C*
 /free
   //アプリ変数の値を取得
   VPRCD = GetAppVar('PRCD');                    //商品ＣＤ
   VPRNM = GetAppVar('PRNM');                    //商品名
   VUNPR = %DEC(%CHAR(GetAppVar('UNPR')):7:0);   //単価
   VCTCD = GetAppVar('CTCD');                    //カテゴリＣＤ
 /end-free
C*
C*---商品マスタの項目をセット
```

1-①

```
C                   MOVEL     VPRCD          PRPRCD
C                   MOVEL     VPRNM          PRPRNM
C                   Z-ADD     VUNPR          PRUNPR
C                   MOVEL     VCTCD          PRCTCD
C*---商品マスタに書き込む
C                   WRITE     MPRDTPR
C*
 /free
   //レスポンスの返却
   SetResponse('success':'true');
   SetResponse('info':'商品マスタの登録が完了しました');
 /end-free
p                   e
 /include qcpylesrc,vvNabBtn
```

また、**【ソース1】**はアプリ変数値の取得のみだが、EXNABBTNでは、アプリ変数に値を代入する場合は、SetResponseメソッドにキーとして'appVar'をセットして使用する。Valence6.2のApp Builderには8種類のテンプレートが用意されているが、テンプレート毎にアプリ変数の入出力方法が異なり、SetAppVarメソッドを使用するものもある。テンプレート毎のアプリ変数の入出力メソッドを**【図11】**にまとめたので、参考にしてほしい。

図 11　RPGでのアプリ変数入出力メソッド一覧

ソースファイル名	機能	概要	アプリ変数の値取得	アプリ変数への値代入
EXNABBTN	ボタンクリック	ボタンのクリックや、行クリック等のイベント処理を記述するテンプレート	GetAppVar('[変数名]')	SetResponse('appVar':'[変数名]':'[値]')
EXNABCLOSE	アプリ終了	アプリケーション終了時実行される終了処理を記述するテンプレート	GetAppVar('[変数名]')	アプリ終了時の処理の為、代入は無し
EXNABDS	データソース実行前	データソースを開く直前に実行したい処理を記述するテンプレート	GetAppVar('[変数名]')	データソースに対するRPGの為、代入は無し
EXNABFHLP	フォームヘルパー	Formウィジェットへの初期値セットや入力項目の値変更時のイベント処理を記述するテンプレート	GetAppVar('[変数名]')	SetAppVar('[変数名]':'[値]')
EXNABFLT	フィルター	フィルター処理実行時に独自のフィルター条件を記述するテンプレート	GetAppVar('[変数名]')	SetAppVar('[変数名]':'[値]')
EXNABIV	フィルター初期値	フィルター条件項目への初期値セットを記述するテンプレート	GetAppVar('[変数名]')	SetAppVar('[変数名]':'[値]')
EXNABSTART	アプリ起動	アプリケーション起動時に実行される初期処理を記述するテンプレート	GetAppVar('[変数名]')	SetAppVar('[変数名]':'[値]')
EXNABVAL	EditGridチェック	EditGridウィジェットで行追加・行変更・行削除において独自処理を記述するテンプレート	GetAppVar('[変数名]')	SetResponse('appVar':'[変数名]':'[値]')

本節では、基本的なアプリ変数の宣言方法と使用方法を紹介したが、次節からは具体的なアプリ変数の活用方法を紹介する。

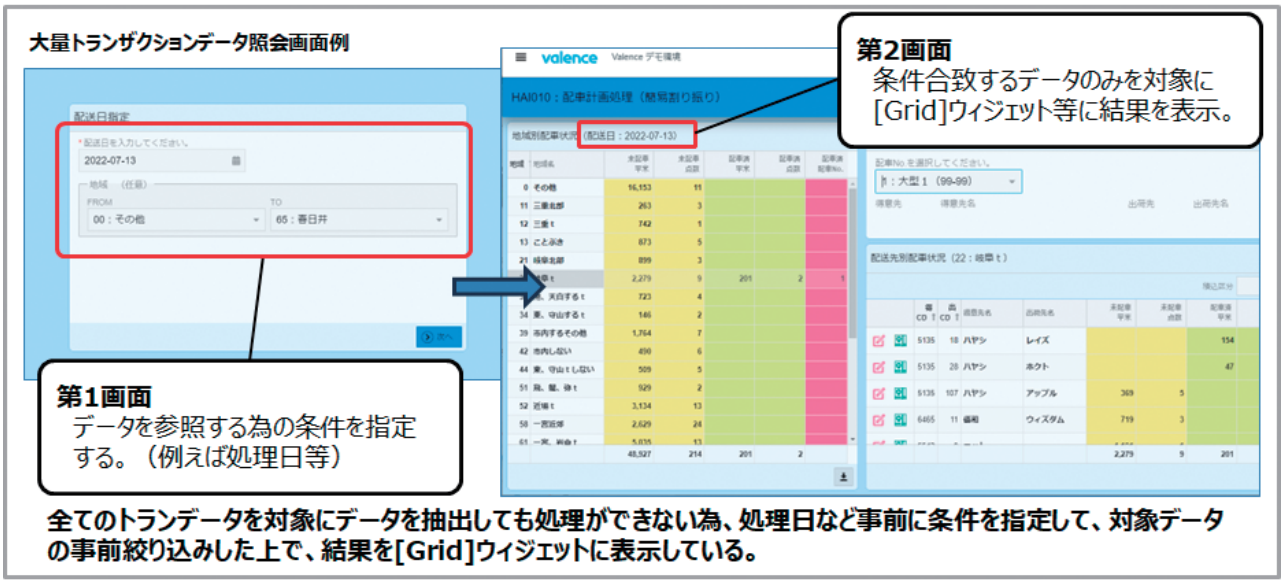
3. データソースにおけるアプリ変数の活用

表形式にデータが一覧表示できる[Grid]ウィジェットは、App Builderでのアプリ開発で最も使用されるウィジェットの一つである。例えば、商品マスタや社員マスタといったようなマスタの照会画面等であれば、[Grid]ウィジェットの定義のみで問題なく作成できるであろう。データソースは対象となるマスタファイルをそのまま使用して作成し、[Grid]ウィジェット上の「フィルタ」定義により、ユーザーによる絞り込み条件を設定できるような工夫をすれば大抵の場合問題ないだろう。

しかし、大量のデータをもつ受注ファイルのようなトランザクションファイルの場合はどうだろうか？トランザクションファイルの全てのデータにアクセスする単純なデータソースをそのまま使用してしまうと、数十万件から場合によっては数百万件を対象とした一覧照会画面を作成する事になってしまう。もちろん[Grid]ウィジェットの「フィルタ」定義も使用できるが、大量のデータを都度実行時にフィルタで絞り込みするとなると、処理速度等で課題が発生する可能性が高い。つまり、大量データをもつトランザクションファイルを[Grid]ウィジェット等に出力する場合は、予めデータソースの時点である程度条件の絞り込みが出来ている事が望ましい。

一般的にこのようなトランザクションデータを照会するようなアプリを開発する場合、まず始めに条件検索画面で絞り込み条件を指定し、その条件に合致するデータが一覧表示されるような動きを実装する事が多い。例えば、App Builderで作成した出荷手配アプリの実装例が、**【図12】**である。

図 12　大量トランザクションデータを照会するアプリ例



この例では、第1画面で処理日付等を条件指定した上で、[次へ]ボタンをクリックすると、入力した条件に該当するトランザクションデータが集計されて第2画面が表示される。このようにトランザクションデータを照会する場合は、何らかの検索条件を指定する事が一般的である。こういったアプリを作成する場合どうすればよいだろうか？
App Builderにおいて[Grid]ウィジェット等にデータを表示する為には元となるデータソースが必要となる。実はこのデータソースの絞り込み条件部分にアプリ変数を使用する事が出来る為、条件に合致するデータだけを抽出できる動的なデータソースを作成する事ができるのである。
具体例を紹介する。ここでは**【図13】**のようなファイルレイアウトを持つ受注ファイル(FSODRP)を対象に条件に合致するデータのみを参照する受注照会アプリの作成を検討したい。

図 13　受注ファイル(FSODRP)ファイルレイアウト

DSPFMT　　レ コ ー ド 設 計 書　　日付 24/09/14　時刻 10:27:53

物理ファイル	TECREP2024/FSODRP	様式名	FSODRPR	レコード長	44
様式記述	受注ファイル				

5= 詳細

選択	項目名	桁数	属性	キー順	開始	終了	テキスト記述／欄見出し
_	ORORNO	7 0	S	1 ASN	1	7	受注番号
_	ORCSCD	5 0	S		8	12	顧客コード
_	ORORDT	8 0	S		13	20	受注日
_	ORPRCD	5	A		21	25	商品コード
_	ORQTY	4 0	S		26	29	数量
_	ORUNPR	7 0	S		30	36	単価
_	ORSHDT	8 0	S		37	44	出荷日

App　Builderではウィザードを使用してデータソースを作成する際、[④フィルタ]にて絞り込み条件を指定する。**【図14】**は、特定の顧客コード(10001)のみのデータに絞り込みする場合の設定例である。

図 14　データソース作成:④フィルタ

この[④フィルタ]の設定で固定値として設定した"10001"の部分を「アプリ変数」に変更すれば、アプリケーション実行時に動的に絞り込み条件が設定できるようになる。ただ、残念ながら、この設定方法で値として記述できるのはあくまで固定値のみで、アプリ変数は設定できない。では、どうすればよいだろうか？

[④フィルタ]画面の右上にある[SQL]ボタンをクリックする。すると、絞り込み条件をSQL文で定義するマニュアルモードに切り替わる。App Builderにおけるデータソースの実体はSQLである為、このようにSQL文を記述するモードも用意されているのである。[④フィルタ]は、SQLの「WHERE句」にあたる。定義方法は次の通りである。マニュアルモードに切り替えると画面右側にSQLエディタが表示される。次に左側のフィールド一覧から[顧客コード]を選択すると、エディタ上に"F1.ORCSCD=77777"と表示される。右辺の"77777"部分はダミー値である為、この部分にアプリ変数を割り当てるように変更すればよい。例えばアプリ変数名が[CUSTCD]だとすると、右辺には"F1.ORCSCD=vvIn_virtual('CUSTCD', '0', 'num', 5, 0) "と記述すればよい。**【図15】**

図 15　④フィルタ：SQLモード

実はこのvvIn_virtualメソッドがApp Builderに用意された専用APIでSQL文の中にアプリ変数値を代入するものである。一見構文が複雑そうに見えるかもしれないが、そこまで難しくはない。アプリ変数自体は定義上、データ型や桁数を持っていないが、SQLにおいては対象となるフィールドには、もちろん属性や桁数がある為、それにあわせて、アプリ変数の値をキャスト（型変換）するのがポイントとなる。vvIn_virtualメソッドのパラメータについての詳細は、**【図16】**のとおりである。

図 16 vvIn_virtualパラメーター一覧

vvIn_virtualは、データソースにおけるSQLに、アプリ変数値をセットする専用API

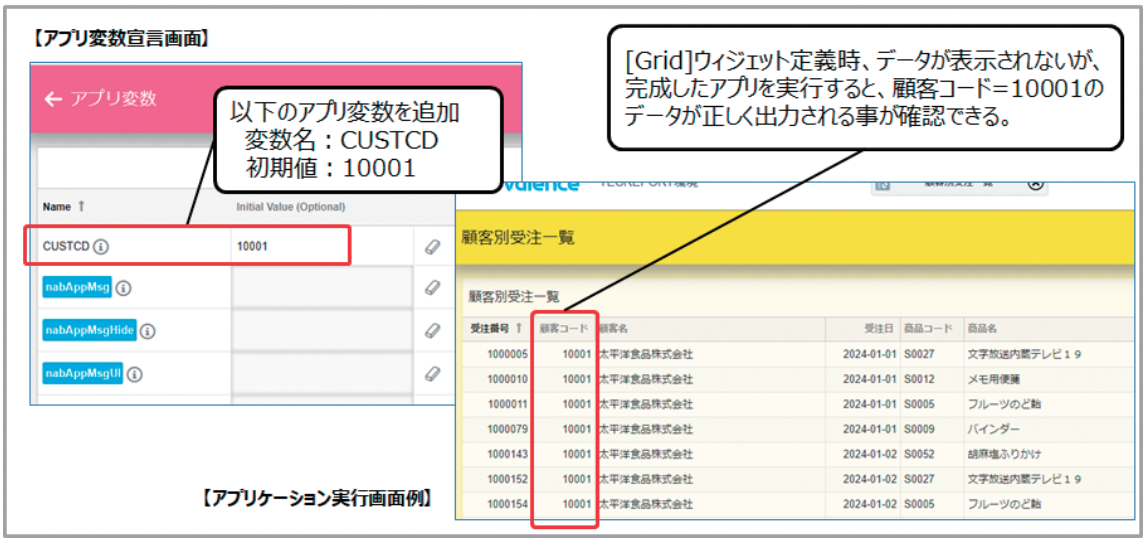
パラメータ（引数）	概要	備考
第1パラメータ	アプリ変数名	シングルクォーテーションで括る事（例: 'CUSTNO'）
第2パラメータ	データソースの定義時はアプリ変数が存在しない為、アプリ変数未定義時に、代わりにセットされる初期値。データソースのプレビュー時は、初期値にて実行される。	数値を指定する場合も、シングルクォーテーションで括る事（初期値が数値0の場合、'0'と記述）
第3パラメータ	セットする値のデータ属性（'num', 'char', 'date'）のいずれかを指定	'num' ： 数値型項目の場合 'char' ： 文字型項目の場合 'date' ： 日付型項目の場合
第4パラメータ	セットする値の桁数	
第5パラメータ（省略可）	セットする値の小数桁数	'num'（数値型）の場合のみ使用。数値型で省略時は、0がセットされる
第6パラメータ（省略可）	セットする値自体に、シングルクォーテーション（引用符）を含むかどうかの指定。	true ：括る false ：括らない 'char'（文字型）/'date'（日付型）で省略時は、trueの扱いとなる。 （例：値が ABC の場合、'ABC'とセットされる）

今回の設定例ではアプリ変数が存在しない場合の初期値として'0'を指定している為、[④フィルタ]設定後に、[⑦プレビュー]に進んだ際、「結果はありません」と表示されるはずであるが、これで正解である。(顧客コード=0の受注データが存在しない為である。)ここでもし、エラーが表示される場合は、vvIn_virtualメソッドの定義が正しくない為、見直してほしい。

データソースの定義が完成したら、このデータソースを元に[Grid]ウィジェットを定義し、定義した[Grid]ウィジェットを使用するアプリケーションを作成すればよい。アプリケーション作成において、新規のアプリ変数[CUSTCD]を初期値[10001]で定義した上で完成したアプリを実行してみてほしい。

[Grid]ウィジェットの定義時にはデータは表示されなかったはずだが、アプリケーションから実行すると、顧客コード=10001のデータが一覧表示される事がわかる。**【図17】**

図 17 データソースにアプリ変数が適用されたアプリ実行例



このようにアプリ変数を定義すると、アプリケーションから呼び出されるデータソースに対して、アプリ変数の値を受け渡す事ができるようになるのである。
なお、この例ではアプリ変数[CUSTCD]の値が0の場合、顧客コード=0のデータを抽出しようとする為、対象データ無しとなってしまう。値が0の場合は、顧客コードによる絞り込みは行わず、0以外の場合だけ顧客コードによる絞り込みを行いたい場合はどうすればよいだろう。これはSQLの書き方で解決できる。**【ソース2】**のような記述を行う事により、アプリ変数[CUSTCD]の値が0以外の場合だけ絞り込みを行う事が可能である。

ソース 2 **④フィルタ：アプリ変数を含む条件指定例**

```
/*-------------------------------------------------------------------------*/
/* 例１：アプリ変数[CUSTCD]が0以外の場合、顧客ＣＤ(ORCSCD)の絞り込みを行う */
/*-------------------------------------------------------------------------*/
(vvIn_virtual('CUSTCD', '0', 'num', 5, 0) = 0) OR
(F1.ORCSCD =  vvIn_virtual('CUSTCD', '0', 'num', 5, 0))


/*-------------------------------------------------------------------------*/
/* 例２：複合条件の指定例                                                  */
/*    アプリ変数[CUSTCD]が0以外の場合、顧客ＣＤ(ORCSCD)の絞り込み          */
/*    アプリ変数[PRDTCD]がブランク以外の場合、商品ＣＤ(ORPRCD)の絞り込み    */
/*-------------------------------------------------------------------------*/
((vvIn_virtual('CUSTCD', '0', 'num', 5, 0) = 0) OR
(F1.ORCSCD =  vvIn_virtual('CUSTCD', '0', 'num', 5, 0)))
AND
((vvIn_virtual('PRDTCD', '', 'char', 5) = '') OR
 (F1.ORPRCD =  vvIn_virtual('PRDTCD', '', 'char', 5)))
```

なお、**【ソース2】**には、複合条件の場合の指定例も提示しているので参考にしてほしい。

次に条件指定を行う検索画面を検討する。例えば**【図18】**のような画面である。

図 18 **検索条件指定フォーム例**

App Builderによるアプリ開発ではウィジェットを定義する場合、[Info]ウィジェットのような一部のユーティリティウィジェットを除き、データを表示する為に元となるデータソースが必要となる。つまり何らかのデータベースファイルを必ず使用する。

ただ**【図18】**は、[Form]ウィジェットで作成しているのだが、この条件指定フォームは何らかのデータベースの値を表示するものではない。では、こういったデータベースに関連付かない単なる入力フォームを定義する場合はどうすればよいのだろうか？

もちろん、入力フォームの項目にあわせたファイルレイアウトをDDSで作成すれば、それをそのままデータソースとして定義する事も可能ではある。ただ条件指示画面毎にDDSを都度作成するのは効率が良くないであろう。

そんな時に活用できるのが、SQLにおける「ダミーテーブル」の活用である。主なRDBMSには、「ダミーテーブル」というのが用意されており、通常のテーブルを用いずにSQL文から何らかの計算項目を取得する仕組みが用意されている。(例えばOracleの場合、[DUAL]表がダミーテーブルとなる。)

実はこのダミーテーブルは、IBMiが採用するDB2においても使用可能である。IBMiのDB2では、SYSIBMライブラリにあるSYSDUMMY1テーブルを使用すればよい。

例えば、システム日付値を取得するだけのSQLを作成したいとする。SQLを定義する場合も何らかのファイルが必要(「FROM句」に相当する。)だが、このような場合もちろん物理ファイルは存在しない為、**【ソース3】**のような記述をすればよいのである。

ソース 3 **ソース3 ダミーテーブルSQL記述例**

```
 /*-------------------------------------------------------------------*/
 /* 例：ダミーテーブルを使用して、システム日付値を取得するSQL例       */
 /*-------------------------------------------------------------------*/
SELECT CURRENT DATE FROM SYSIBM/SYSDUMMY1
```

このダミーテーブルは、App Builderにおけるデータソース作成でももちろん使用できる。ウィザードモードの場合、[①ファイル]にて、ライブラリに"SYSIBM"、ファイルに"SYSDUMMY1"を記述すれば、通常のファイルと同様に追加可能である。追加後、[③カラム]において、電卓アイコンの[計算項目の追加]ボタンより、独自の計算項目として入力フィールドを設定すればよい。この計算項目に設定する記述例を**【ソース4】**に示すので参考にしてほしい。

ソース 4 **③カラム：計算項目によるフィールド定義例**

```
/*-------------------------------------------------------------------*/
/* [③カラム]：計算項目設定例                                        */
/*     (基本形)  CAST([初期値] AS [データ型(桁数)])                  */
/*-------------------------------------------------------------------*/
//---- 5桁(小数0桁)の数値項目で、初期値を0と定義する場合
CAST(0 AS NUMERIC(5, 0))

//---- 10桁の文字項目で、初期値をブランクと定義する場合
CAST('' AS CHAR(10))

//---- 日付項目で、初期値をブランクとして定義する場合
CAST('0001-01-01' AS DATE)

//---- 日付項目で、システム日付を初期値として定義する場合
CURRENT DATE

//---- 数値8桁の数値項目で、初期値をシステム日付値の数値として定義する場合
CAST(VARCHAR_FORMAT(CURRENT DATE, 'YYYYMMDD') AS NUMERIC(8, 0))
```

条件指定フォームで入力したい項目数分だけ、計算項目を定義すればデータソースの定義は完了である。[⑦プレビュー]まで進めると、1レコードのダミー項目をもつデータソースが定義されるはずだ。後はこのダミーのデータソースを元に通常通り[Form]ウィジェットを定義すれば、**【図18】**のような検索画面は作成可能である。条件検索フォームが完成したら、この[Form]ウィジェットをアプリケーションに追加し、[検索]ボタンの[クリック時]処理にて、[アプリ変数の設定]アクションと[ウィジェットの表示/非表示]アクションを設定すれば良い。[ウィジェットの表示/非表示]アクションでは、結果を表示する[Grid]ウィジェットに対し、[表示]を選択のうえ、[データ読み込み]オプションを有効にすれば、アプリ変数にセットされた値を使用して[Grid]ウィジェットが再読込できるようになる。

本節では、アプリ変数を使用した動的な条件絞り込みが可能なデータソースの作成について紹介した。トランザクションデータを照会する画面は、よくあるアプリの作成パターンとなる為、今回紹介した手順を参考にしてほしい。

4.アプリ変数応用テクニック

前節ではアプリ変数の基本的な活用として動的なデータソースを定義する方法を紹介したが、他にもアプリ変数は色々な活用が可能である。ここでは主な3つの応用例を紹介したい。

1つ目は、モバイルアプリ開発における応用例である。ValenceにはiOS/Androidに対応した「Valence Mobile」が用意されており、App Builderアプリも容易にモバイル対応できる。本稿では具体的なモバイルアプリの開発手順は割愛するが、モバイルアプリ開発に関しては2022年のテクニカルレポート「Valenceモバイルアプリケーション開発テクニック」(https://www.migaro.co.jp/tr/no15/tech/15_01_05.pdf)に詳しくまとめているので、そちらを参照してほしい。

モバイルアプリを作成する時の留意点として、iPhone等のスマートフォンの画面サイズがある。縦画面での利用が前提のスマートフォンは、[Grid]ウィジェットにおいて複数のカラムを表示させようとすると、1カラムあたりの表示列幅が小さくなり見づらくなってしまう。もちろん、スマートフォン専用のアプリとして、[Grid]ウィジェットの表示カラム数を予め少なくしておけば問題ないが、1つのアプリをデスクトップ用とモバイル用で共用したい場合もあるだろう。このような場合にアプリ変数が役立つ。

具体的な手順を紹介する。アプリケーションのワークスペース上に配置した[Grid]ウィジェット上にマウスを合わせると、[リンク]と表示される為、これを選択する。この[リンク]は、ウィジェットの見た目や挙動をアプリ変数により動的に変更するオプション機能である。[Grid]ウィジェットの[リンク]を選択すると、「アプリ変数にリンク」画面が表示される為、左側の一覧より「カラムを非表示」を選択する。すると、右側にフィールド一覧が表示される為、モバイルから実行する際にだけ、非表示にしたいカラムについて、デフォルトアプリ変数[nabMobile]を設定すれば良い。**【図19】**

図 19　ウィジェットに対する　アプリ変数とのリンク設定

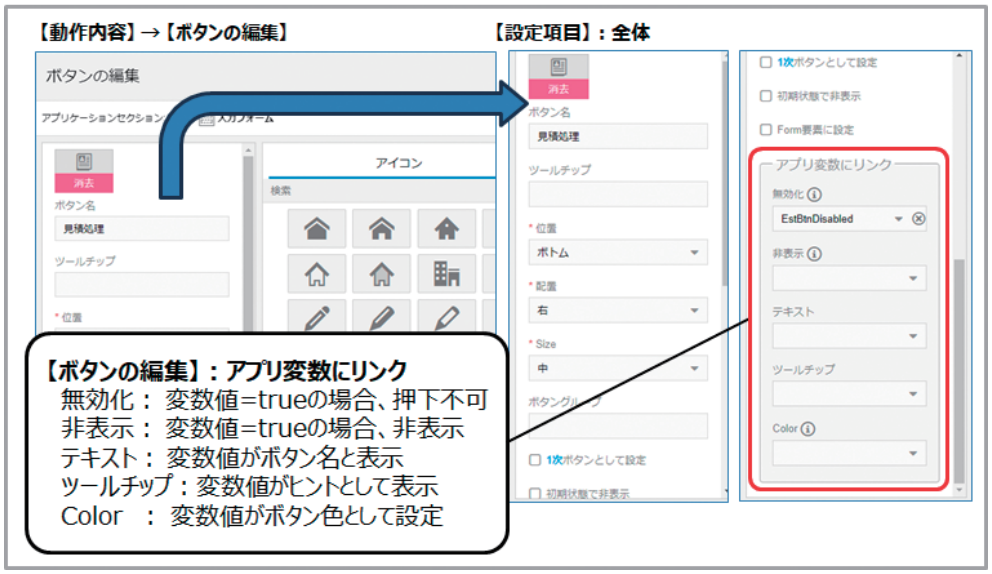
[nabMobile]変数は、モバイルアプリから実行した際に自動的に"true"が設定される読取専用のアプリ変数である。このリンク設定を行ったアプリを実行した時の実際の画面が【図20】である。PC/モバイルいずれの端末で実行しても、違和感の無い画面が表示される事がわかるだろう。

図 20　[Grid]ウィジェット　アプリ変数　リンク機能例

このように、アプリケーション設計画面上に配置した各ウィジェットにはアプリ変数へのリンク機能が用意されている為、アプリ変数を割り当てる事で、ウィジェットの見た目や挙動を制御できるので、是非色々試してほしい。

2つ目は動的なボタンの押下可否制御についてである。App Builderにおいて独自のボタンは[動作内容]の中で追加できる。追加するボタンの設定画面には、ボタンの制御に対するアプリ変数の割り当てが可能になっており、ボタンの表示/非表示や押下可否等が制御できるようになっている。例えば、**【図21】**は、[Form]ウィジェット上に配置した[見積処理]ボタンの押下可否を設定するアプリ変数として[EstBtnDisabled]を割り当てたものである。[EstBtnDisabled]の値がtrueになった場合に、[見積処理]ボタンは押下不可となる。

図 21　ボタンの編集画面



では、この[見積処理]ボタンの押下可否をアプリケーション実行時に状況に応じて制御する場合どうすればよいだろう。今回は、**【図22】**のように[Form]ウィジェットに配置した文字1桁のフィールド「処理区分」項目（ラジオボタンとして、"1":[見積]、"2":[受注]を割当）で、値"1"が選択された場合のみ、[見積処理]ボタンが押下できるように制御する方法を紹介する。

図 22　処理区分の選択画面例

まず、[処理区分]フィールドで選択された値("1"or"2")を保持する為のアプリ変数[INPVAL]を追加し、[Form]ウィジェットの　[リンク]を選択し、「アプリ変数にリンク」を開く。「フィールドの値を設定」から[処理区分]のフィールドについて、アプリ変数[INPVAL]を割り当てる。(これにより、画面上で選択した値が自動的にアプリ変数[INPVAL]へ設定する事ができる。)

次に、アプリ変数宣言画面を開き、一覧に表示された[EstBtnDisabled]行の右端にある「Expression」列に**【ソース5】**のような式をセットすればよい。

ソース 5　**ソース5 アプリ変数宣言画面：Expression設定例**

```
 /*-------------------------------------------------------------------*/
 /* アプリ変数宣言画面                                                */
 /*   EstBtnDisabled：  [INPVAL]値が1以外の場合、True                 */
 /*   OrdBtnDisabled：  [INPVAL]値が1の場合、True                     */
 /*-------------------------------------------------------------------*/
//---- EstBtnDisabled([見積処理]ボタン押下可否フラグ)
{[INPVAL]}!='1'

//---- OrdBtnDisabled([受注処理]ボタン押下可否フラグ)
{[INPVAL]}=='1'
```

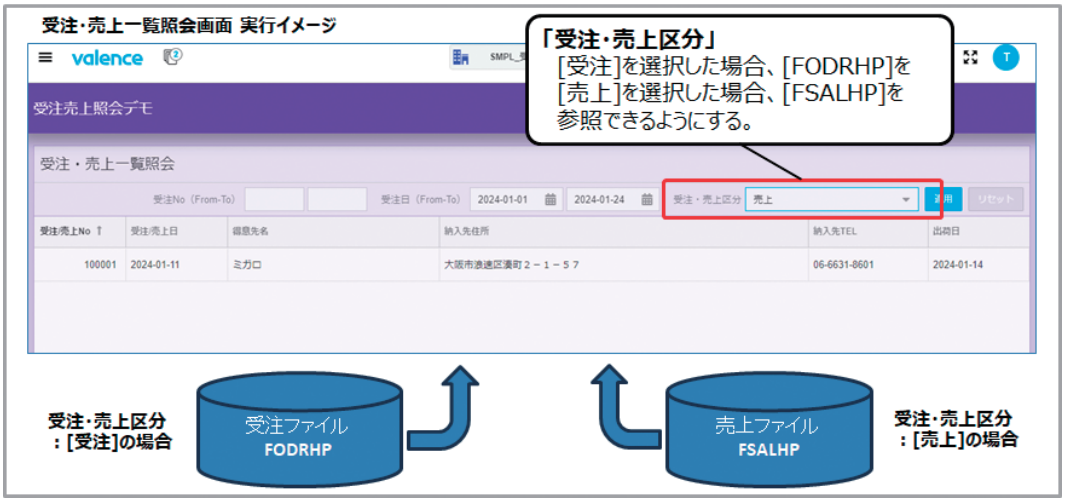
この例では[INPVAL]≠"1"の場合、[EstBtnDisabled]=trueが自動的にセットされる。("1"の場合はfalse。)

App　Builderにおける「アプリ変数のリンク」では、この例のようにtrue/false値をセットするものが多い為、「Expression」列で、論理式が定義出来る事を理解しておこう。

3つ目の応用例を紹介する。前節にてデータソースにおけるアプリ変数としてvvIn_virtualメソッドを使用したが、ここではこのメソッドの応用例を紹介する。

受注ファイル(FODRHP)と売上ファイル(FSALHP)という2つの物理ファイルがあると仮定する。異なるファイルだが、ファイルレイアウトは同一であると考えてほしい。作成するアプリは**【図23】**のような「受注・売上一覧照会」画面である。

図 23　**受注売上一覧照会　画面例**



ウィジェットのフィルタ条件にある「受注・売上区分」にある選択値により、参照するファイルを動的に変更する仕組みを作成する。App Builderのウィジェットにデータを表示する為には参照元のデータソースを定義するわけだが、ウィザードモードを使用したデータソース作成における[①ファイル]では、ファイル名には固定値を設定する必要がある。では、今回のように条件に応じて参照するファイルを動的に変更するにはどうすればよいだろうか？

データソース作成には、ウィザードモード以外にマニュアルモード（SQL）が用意されている。所謂抽出条件をSQL文で直接記述できるモードである。ウィザードモードでデータソースの新規作成画面を起動後、画面左下にある「自由形式のSQLステートメントを入力」を選択すると、マニュアルモードに切り替わる。このモードでは画面左側のエディタ上に直接SQLを記述するのだが、画面右側でファイル→フィールドを選択すれば、ある程度までSQLを自動作成できる。**【図24】**

図 24　データソース作成 マニュアル（SQL）モード

ウィザードモードでは、[③カラム]の計算項目や、[④フィルタ]にてvvIn_virtual メソッドを使用する事ができるが、マニュアルモードではSQL文の任意の箇所でvvIn_virtual メソッドが使用できる。つまり、SQLにおけるファイル選択（FROM句）にアプリ変数を設定できるのである。今回の場合、**【ソース6】**のような記述が可能である。6-①のようにFROM句部分にvvIn_virtualメソッドを定義する事により、参照するファイル名を動的に変更できるようになる。

ソース 6　ソース6 SQL文中にアプリ変数を使用

```
SELECT
  ohorno,
  ohordt,
  ohcunm,
  ohnyad,
  ohnytl,
  ohspdt
FROM
  vvin_virtual('TBLNAME', 'FODRHP', 'char', 10, NULL, 'false')   6-①
ORDER BY
  ohorno
```

アプリ変数[TBLNAME]に受注の場合は"FODRHP"を、売上の場合は"FSALHP"がセットできれば良い。
今回の場合、[Grid]ウィジェットの絞り込み条件を定義する「フィルタ」タブを設定する。「フィルタ」タブでは通常データベース上の項目を選択して絞り込み条件を定義できるが、データソースにアプリ変数を設定した場合、アプリ変数[TBLNAME]も条件として設定できるのである。そこで、[TBLNAME]に対して「ドロップダウン」設定を行えば、フィルタの選択によって参照するファイルを変更する事が実現できるのである。**【図25】**

図 25　[Grid]ウィジェット:フィルタでアプリ変数を設定

本節では3つの応用例を紹介したが、アプリ変数が多様な場面で活用できる事がご理解頂けたであろう。

5.さいごに

本稿では App Builder 開発における「アプリ変数」に焦点を絞って紹介してきた。アプリ上に保持できる単なる変数であるが、データソースやウィジェットにリンクする事により多彩な応用が出来るのである。本稿で紹介しきれていないリンクも多数あるので、色々試してみてほしい。
また、ミガロのホームページ上には「ミガロ．技術 Tips」というサイトを公開している。Valence に関しての記事は以下の URL からアクセスできる。
【Valence 技術 Tips】
https://www.migaro.co.jp/tips/category/valence/

技術 Tips 記事のトピックとして、アプリ変数を取り上げたものを多数掲載しているので、これらも是非参考にしてほしい。Valence App Builder はローコード開発ツールとして今後も更なる機能拡張を予定しているが、あらゆる場面でアプリ変数を駆使する事になるだろう。是非本稿を参考にアプリ変数への理解を深めてほしい。

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

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
一情報の"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

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

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 ［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 [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

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

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

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

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

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

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

No.12 2019年

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

MIGARO. Technical Report
既刊号バックナンバー

No.13 2020年

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

No.14 2021年

SE論文

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
畑中 侑／システム事業部 システム2課

IntraWebを使用したWeb開発のTips紹介
福井 和彦　石山 智也／システム事業部 システム1課

FireDACを活用した Delphi/400ロジック最新化テクニック
佐田 雄一／RAD事業部 技術支援課

洗練されたUIデザインを簡単に実現！ HTML作成テクニック
國元 祐二／RAD事業部 技術支援課

Valenceにおける帳票出力について
尾崎 浩司／RAD事業部 技術支援課

No.15 2022年

SE論文

Delphi/400によるデジタルサイネージアプリの開発
宮坂 優大　石山 智也／システム事業部 1課

アプリケーションテザリングと モバイルカメラを利用した業務の効率化
前坂 誠二 ／システム事業部 2課

Delphi/400 11 Alexandriaによる 最新モバイルアプリ開発術
佐田 雄一／プロダクト事業部 技術支援課

SmartPad4iで電子サインを実現する方法
國元 祐二／プロダクト事業部 技術支援課

Valence モバイルアプリケーション開発テクニック
尾崎 浩司／プロダクト事業部 技術支援課

No.16 2023年

SE論文

最新Delphi/400 11 Alexandriaで作成する アプリケーション開発入門
佐田 雄一／プロダクト事業部 技術支援課

Delphi/400アプリケーション向け 開発支援ツールの作成方法
都地 奈津美／システム事業部 1課

業務アプリケーションと メール・チャットサービスの連携
前坂 誠二／システム事業部 2課

SmartPad4i ExcelJSで簡単！ Excel活用テクニック
國元 祐二／プロダクト事業部 技術支援課

[Edit Grid]ウィジェット活用術
尾崎 浩司／プロダクト事業部 技術支援課

MIGARO.
Technical Report 2024

No.17 2024年

2024年12月6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