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。

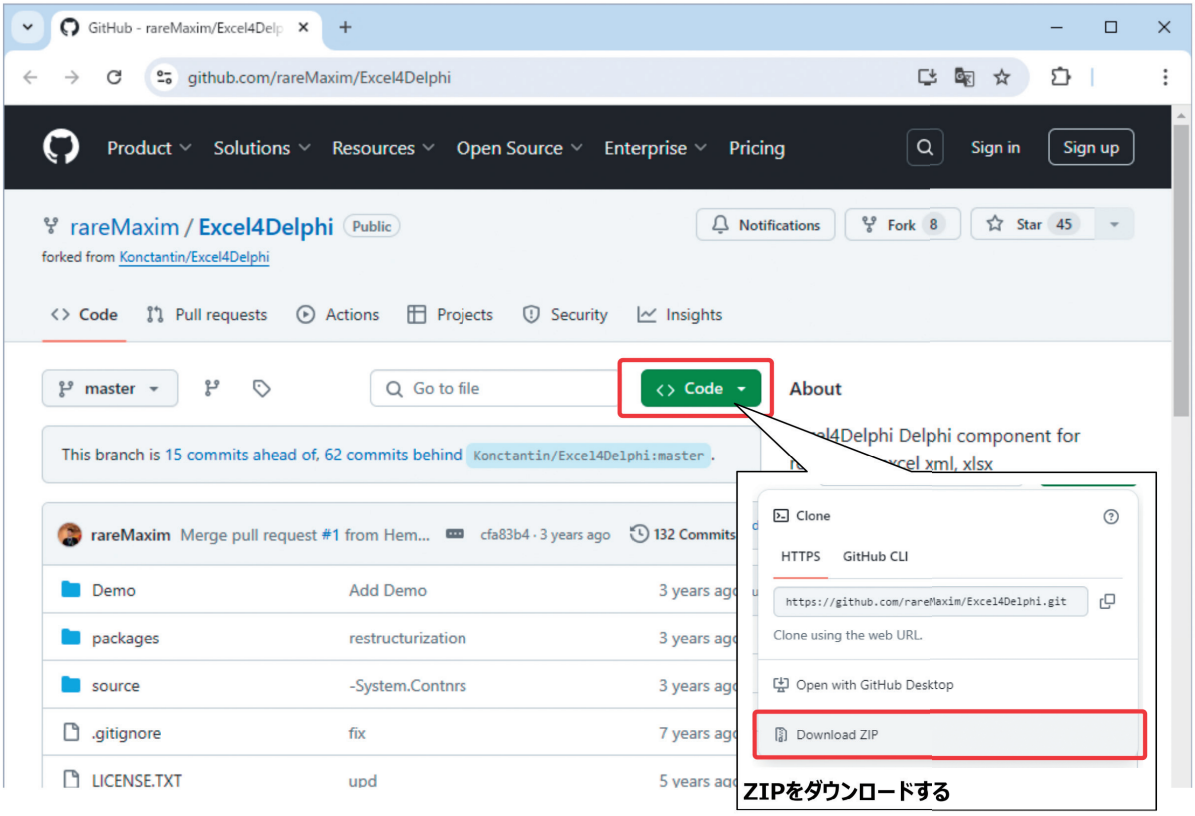
また今回使用するExcel4Delphiライブラリではファイル圧縮・解凍に「zlib」を使用しており、そのライセンス条項は「LICENSE.TXT」に記載されているため確認が必要である。

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。）

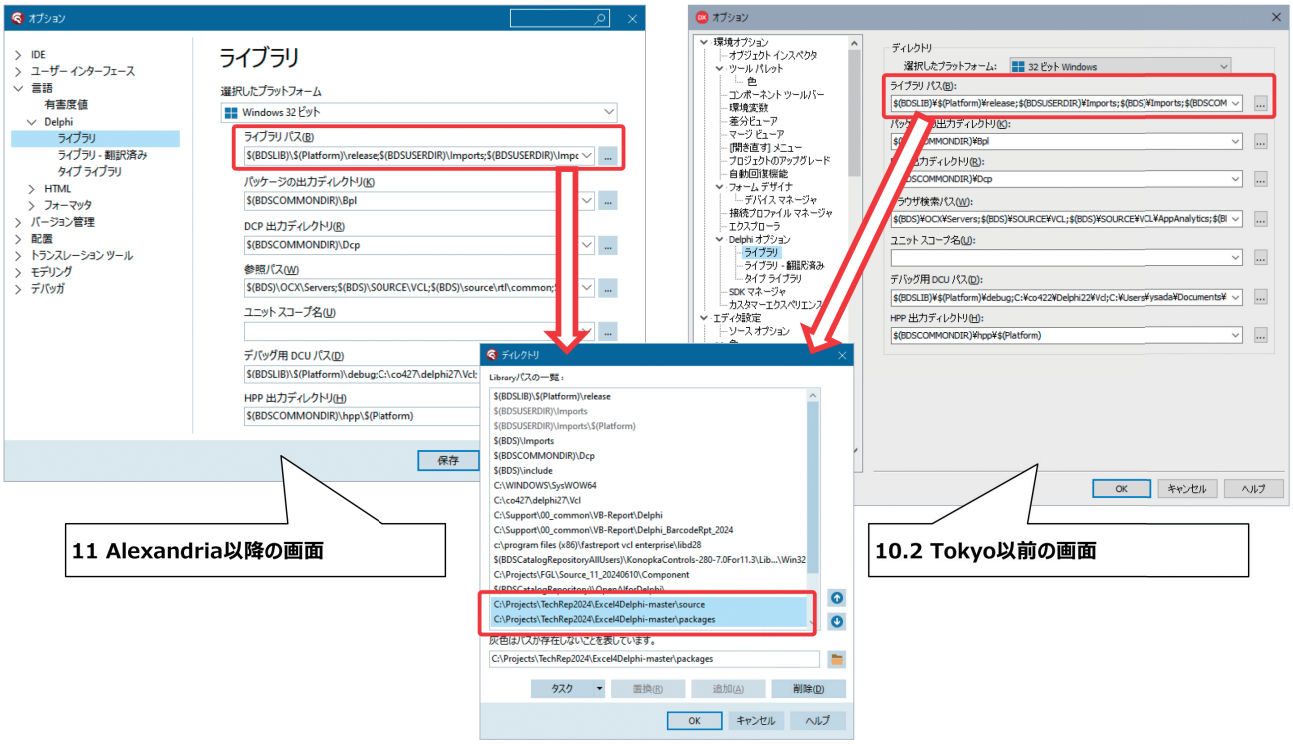
ここからは導入手順について説明していく。
まず、GitHubのウェブサイト内にある
https://github.com/rareMaxim/Excel4Delphi
にアクセスすると、ファイル一覧、英語の概要やサンプルロジックが記載されたページが表示される【図1】。内容や先述のライセンス条項を確認したあと、緑の「<> Code」メニューから『Download ZIP』ボタンを押すと、この一覧にあるExcel4Delphi関連ファイルがZIP形式で一括ダウンロードされる。

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

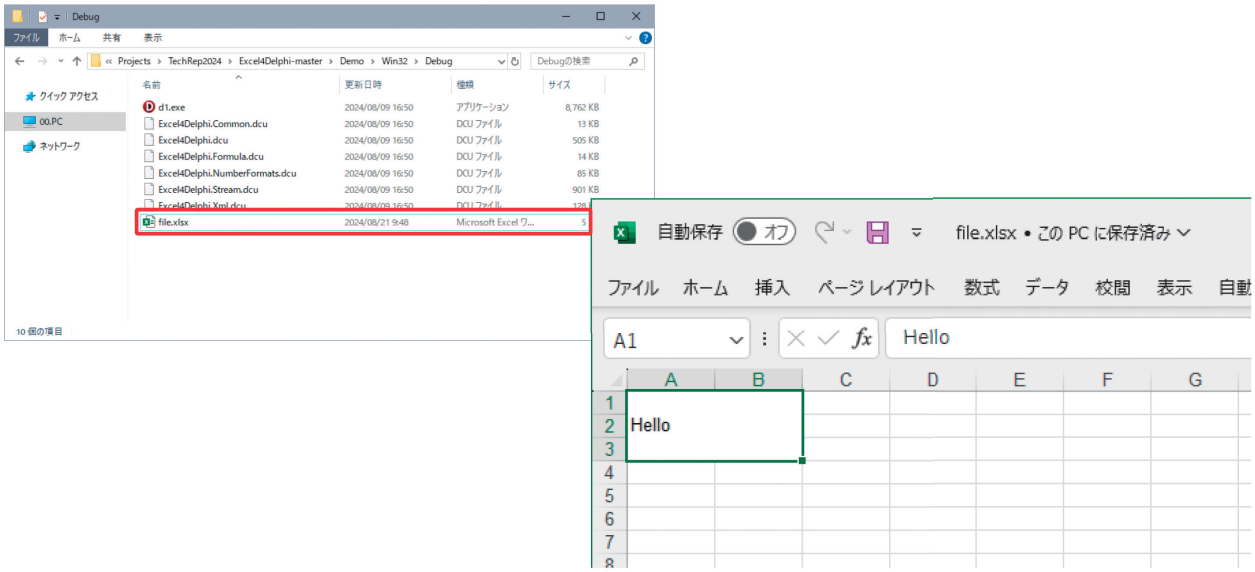


図 4　デモプログラムのロジック

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。

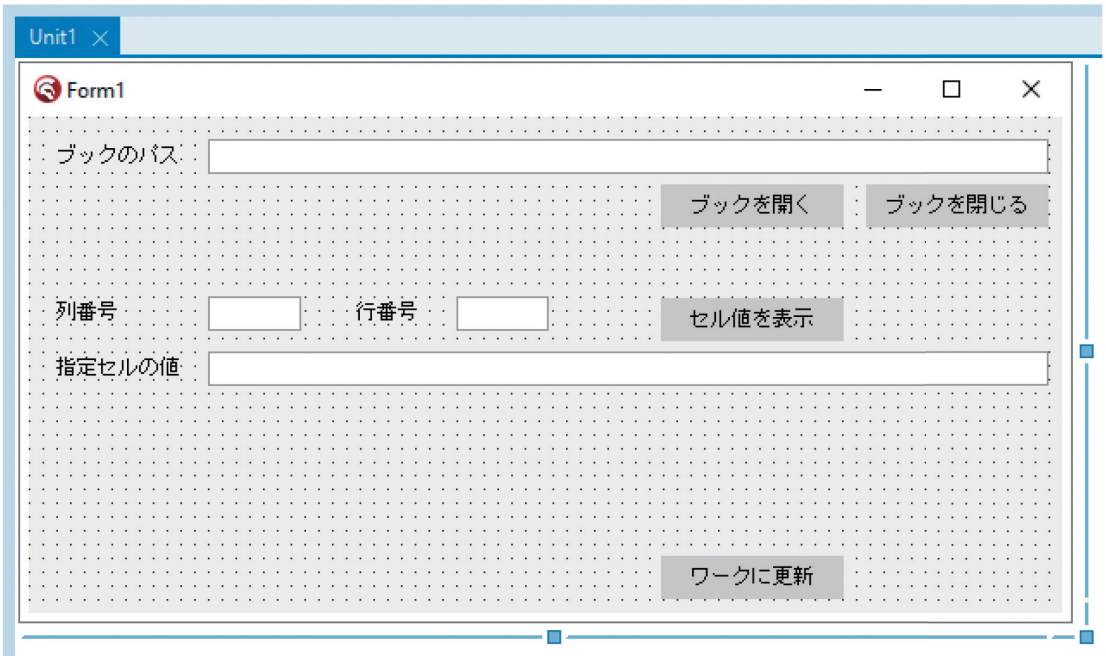
図 5　Excel4Delphiに読み込ませるブック

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

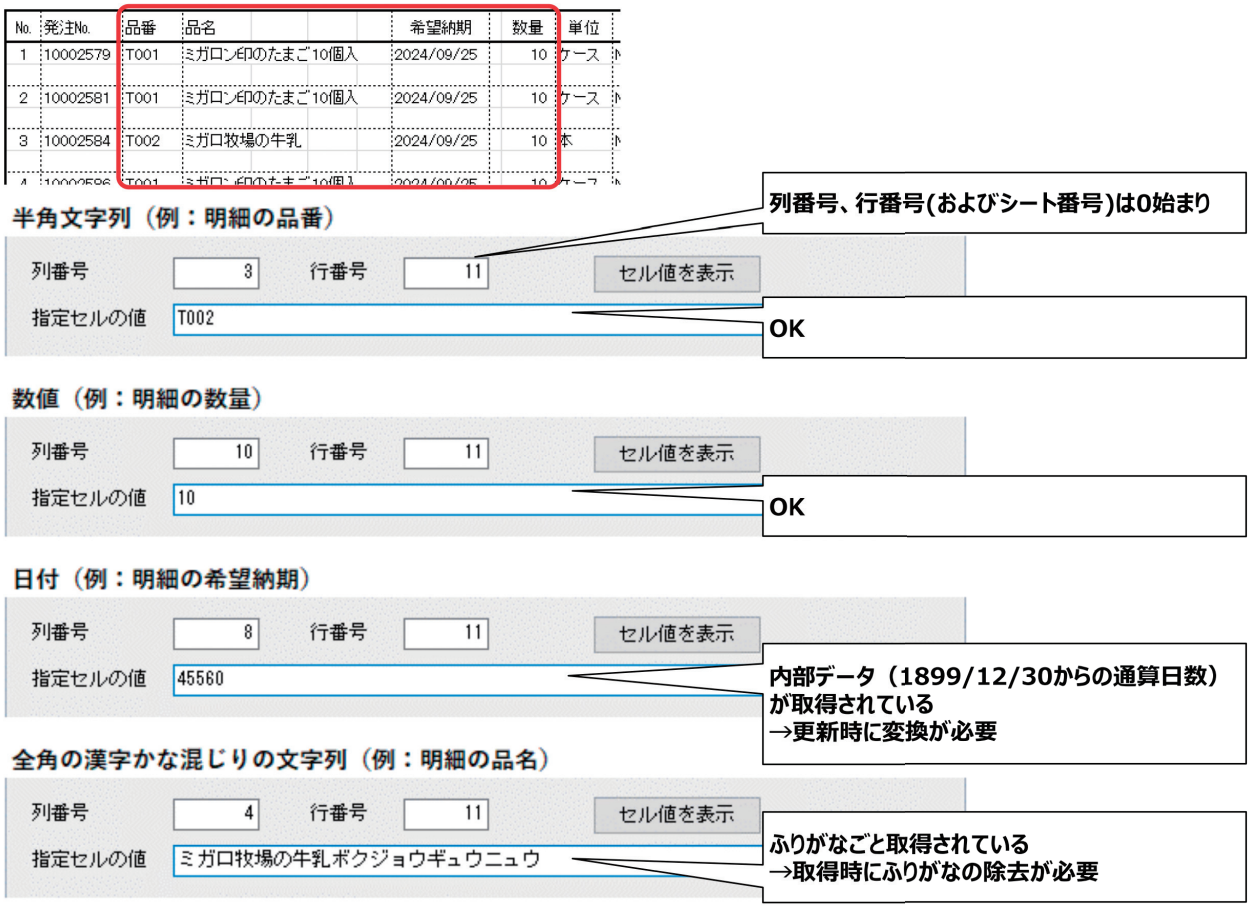
Excel4Delphi セル値を表示ボタン

```
{*******************************************************************************
  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

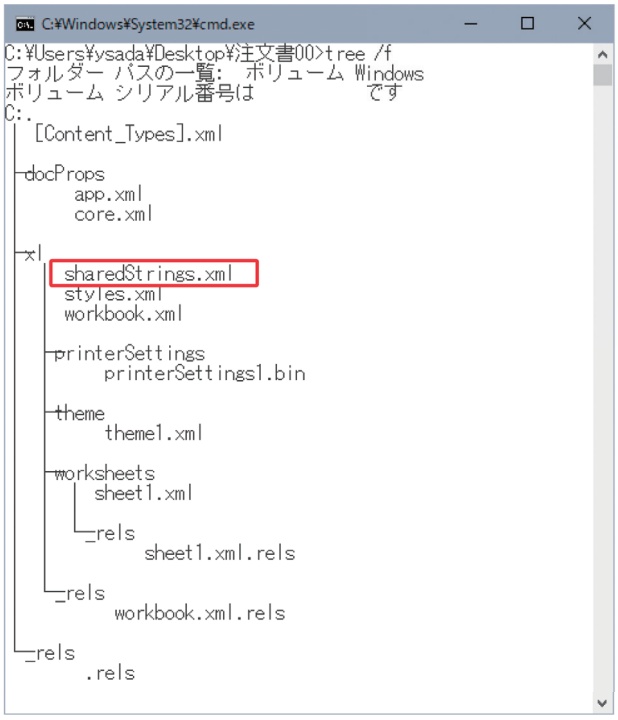
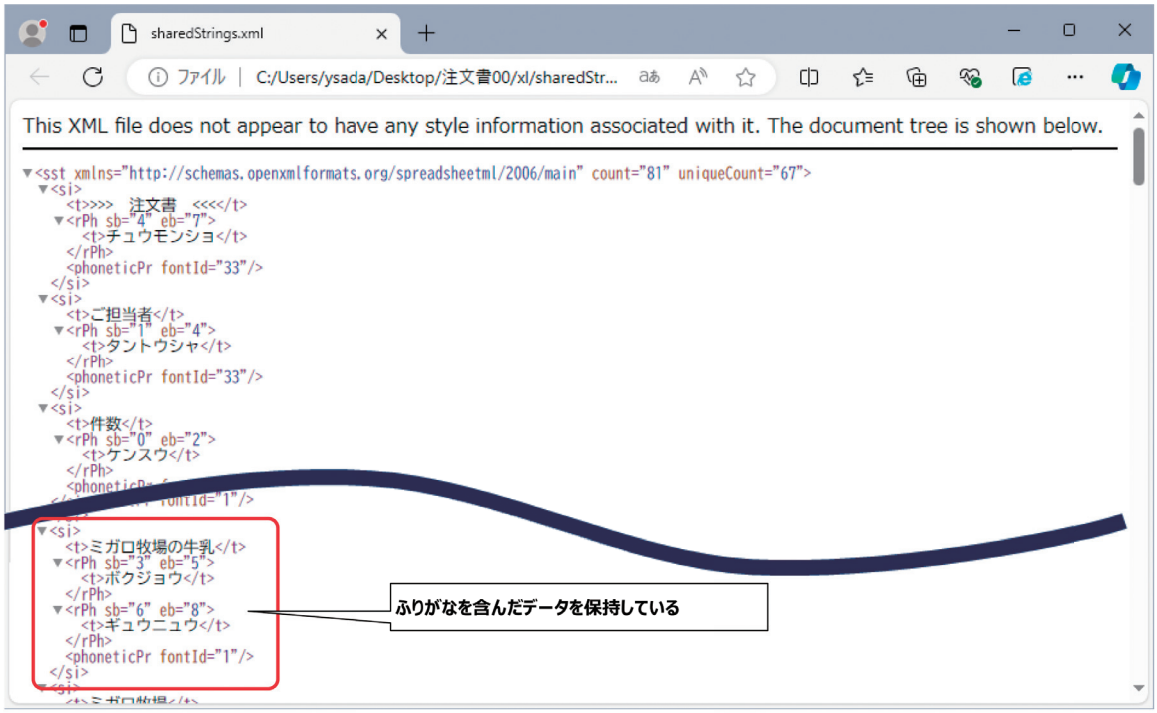


図 10　XML内の文字列セルのデータ部分



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。

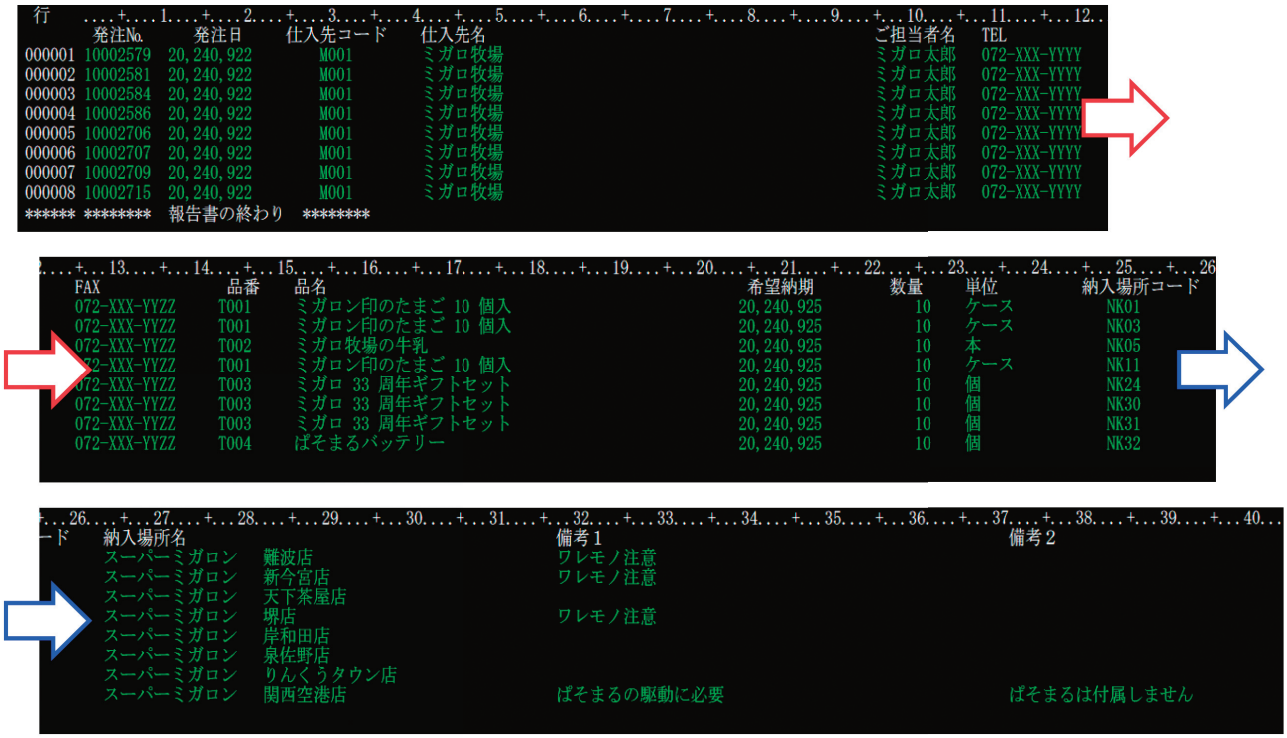
ソース 6

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

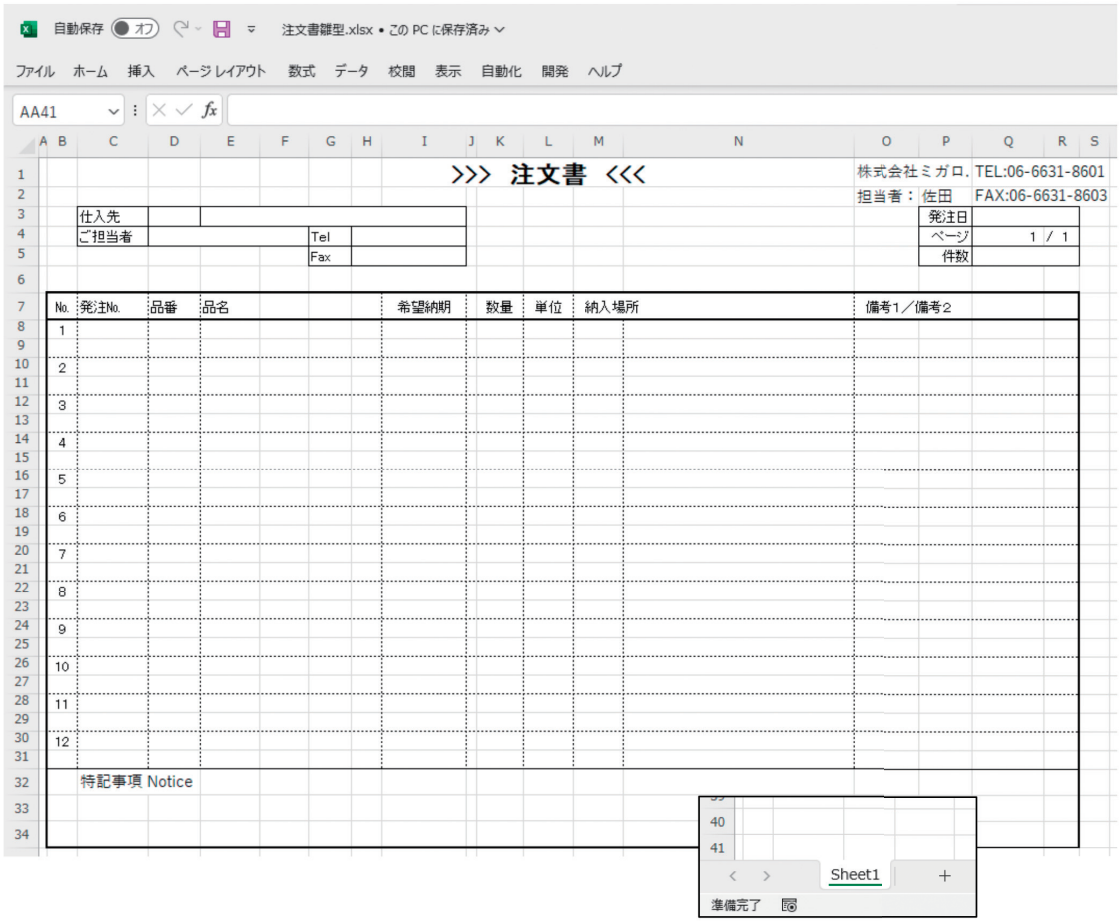


なおここではExcelから取得した文字列がワークのフィールド長に対して長すぎた場合の桁あふれチェックについては考慮していない。
必要に応じて弊社の技術Tipsにある桁あふれ対策の記事(https://www.migaro.co.jp/tips/2910/)を参照いただきたい。

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

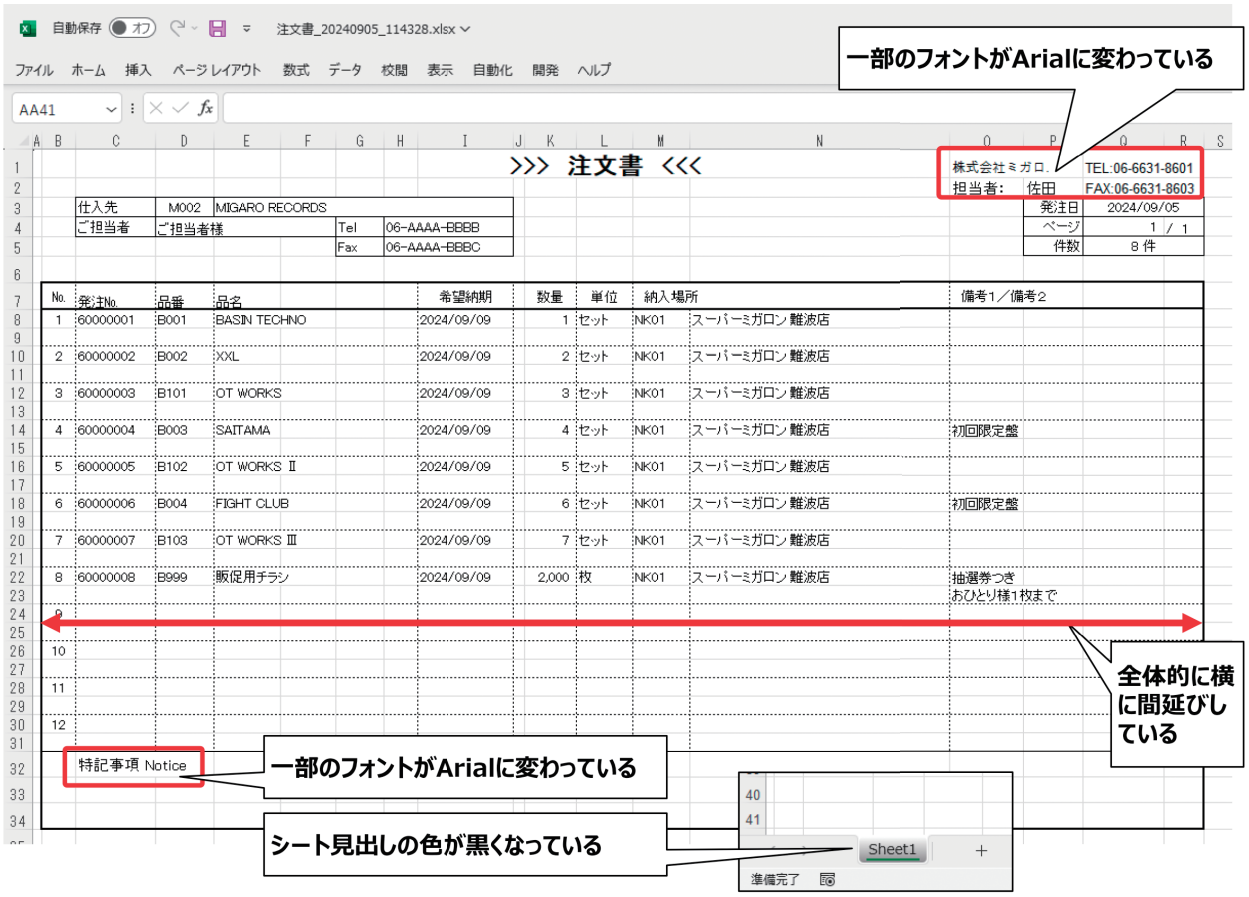
Excel4Delphi ワークから出力ボタン

```
{*******************************************************************************
  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。

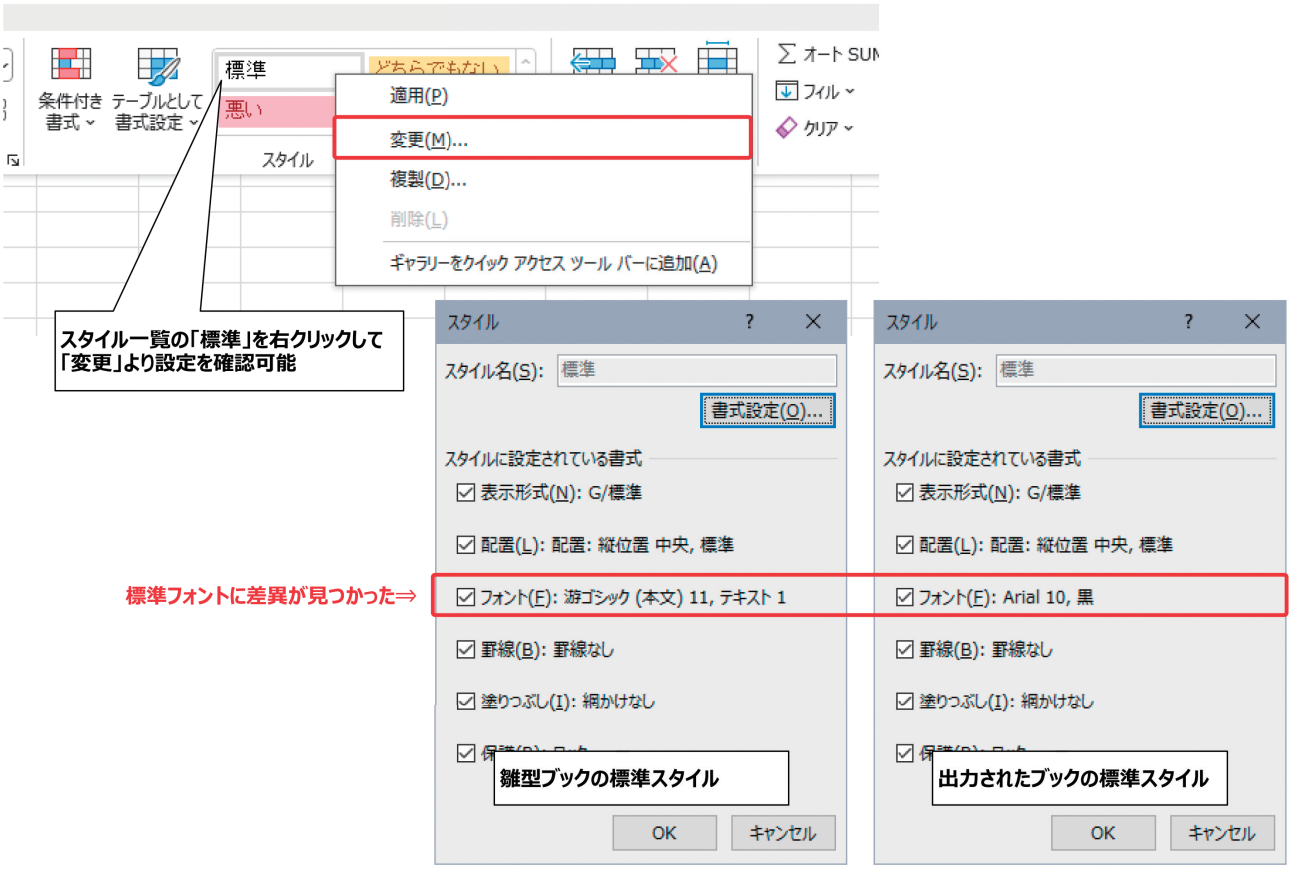
・標準設定のままのセルのフォントが欧文標準の「Arial」に変わり、全体的に横に間延びしている。

・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

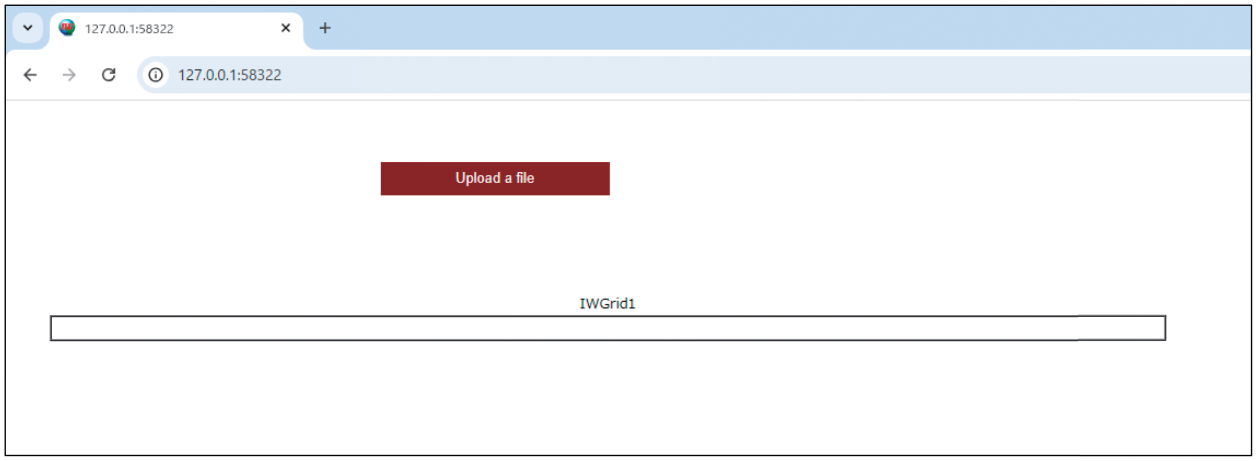
4. IntraWebでのExcel4Delphi活用

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

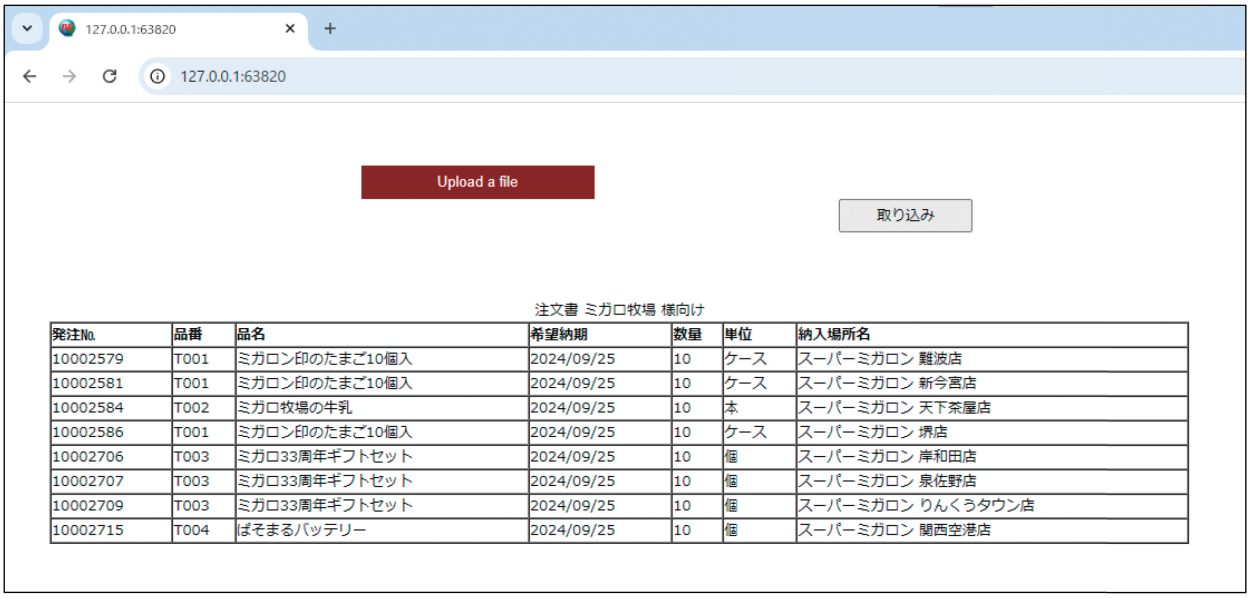
Excel4Delphi IntraWeb で取り込みボタン押下時

```
{*******************************************************************************
  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。