# MIGARO. TECHNICAL REPORT 2023

ミガロ. テクニカルレポート 2023年

No.16

株式会社ミガロ.

# MIGARO.
# Technical Report 2023

ミガロ. テクニカルレポート 2023年

No.16

## CONTENTS

目次


ごあいさつ

SE論文

### Delphi/400

最新Delphi/400 11 Alexandriaで作成するアプリケーション開発入門 002
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 022
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 052
前坂 誠二／システム事業部 2課

### SmartPad4i

SmartPad4i ExcelJSで簡単！ Excel活用テクニック 070
國元 祐二／プロダクト事業部 技術支援課

### Valence

[Edit Grid]ウィジェット活用術 088
尾崎 浩司／プロダクト事業部 技術支援課

Backnumber 110
既刊号バックナンバー

# 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.製品を使った開発に関する技術情報をお届けする論文集で、このたび第16号を発刊する運びとなりました。これもひとえにお客様から頂戴するご支援とご愛顧の賜物と、心より感謝をしております。

さて、IBM iは、その安定性・堅牢性・継続性により多くの企業の基幹システムを支え続けておりますが、一方で、各種ソフトウェアパッケージ、データ分析用システム、クラウド型の開発・運用ツールなど、IBM i以外のシステムを導入される企業も年々増加し、IBM iと各種システムの連携や、「API接続」の重要性の説明を目にする機会も増えました。

弊社では、「API」という言葉が強く意識される遥か以前より、IBM iと各種システムを融合する技術の情報発信に努めてまいりました。帳票ツールやオープン系DBを始めとして、モバイル機器、IoT、クラウドサービス、AIチャットボットに至るまで、さまざまなシステムとIBM iを連携し、一つのアプリケーション内に統合する方法を本テクニカルレポートやセミナーなどで、度々ご紹介しております。

今回のテクニカルレポートでも、IBM iと他のITサービスの連携をテーマとしたSE論文が含まれています。メール・チャット連携を扱うDelphi/400論文、柔軟なExcel連携を扱うSP4i論文はいずれもオープンソースのライブラリを利用することにより連携機能を実現しています。他にも、Delphi/400は最新バージョンを使用したアプリケーション開発の基本、開発支援ツール作成、Valenceは表（グリッド）を本格的な入力業務に適用する方法をそれぞれご紹介しています。さまざまなテクニックを開発に活用いただくために、各論文は詳細な実装方法まで記載いたしました。本レポートが少しでも皆様の開発・保守のお役に立てば幸いです。

初号を発刊した2008年以来、長きにわたりテクニカルレポートの作成を続けてこられたのは、ひとえに多くのお客様やパートナー様からのご支援によるものと、この場をお借りして厚く御礼申し上げます。

2023年12月

株式会社ミガロ.
代表取締役社長
上甲 將隆

# Delphi/400

# 最新Delphi/400 11 Alexandriaで作成するアプリケーション開発入門

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**
**2. プロジェクトの新規作成**
**3. 継承フォームの作成とコーディング**
**4. データモジュールを活用したIBM iとの接続処理**
**5. 各画面のコーディング**
**6. まとめ**

## 1.はじめに

弊社がDelphi/400の取り扱いを開始してから20年以上が経過した。その間、弊社ホームページ上や各種テクニカルセミナー・テクニカルレポートにおいて、システムの運用効率やお客様の開発効率を向上する為のトピックを数多く公開してきた。
しかし、これまでに公開してきたトピックの中でも、基本的な開発手法に関する記事については、Delphi/400のバージョンや実行環境のOSバージョンが古いままのものも多い。さらに、データベースエンジンについても旧来のBDEやdbExpressを使用しているものが大半を占める。

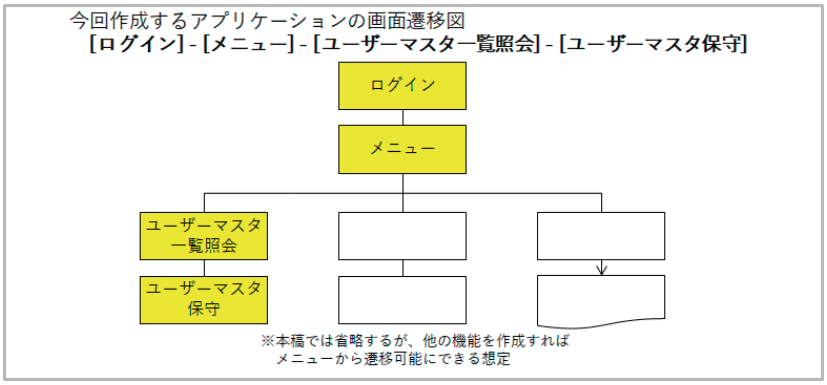
そこで本稿では、改めて2023年9月現在の最新版であるDelphi/400　11　Alexandriaを使用したアプリケーション開発手順の基本を紹介する。題材としてユーザーマスタファイルを参照、更新するアプリケーションを一から開発することを前提に具体的な手順を紹介したい。データアクセスについても最新のデータベースエンジンであるFireDAC接続を使用する。

初めてDelphi/400の開発に触れる方にとっては少々敷居が高く感じる部分があるかもしれないが、弊社で開催しているトレーニングコース(開発入門・開発基礎の各コース)修了レベルであれば問題はないだろう。逆に長年Delphi/400の開発に携わってきた方には、既知の内容も多いだろうが、その中からでも新しい発見があれば幸いである。

## 2.プロジェクトの新規作成

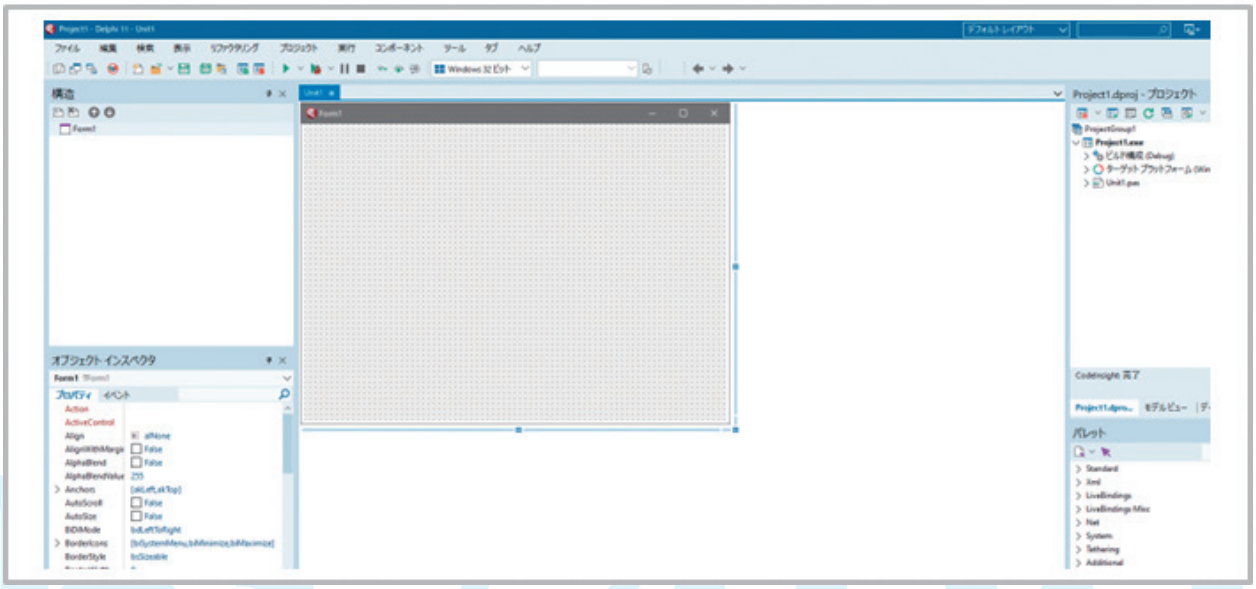
本稿でこれから作成するアプリケーションの画面遷移図を【図1】のようにまとめた。どのような画面が必要か、および画面間の遷移について事前に流れを明確にしておけば、必要な画面の過不足に後から気付くリスクを低減できる。

**図 1　開発するアプリケーションの画面遷移イメージ**




プロジェクトを新規作成する。Delphi/400の統合開発環境（IDE）を起動すると、ウェルカムページが表示される。ウェルカムページ内の「新規作成」または［ファイル］メニューの「新規作成」から「Windows　VCL　アプリケーション-Delphi」を選択すると、プロジェクトが新規作成され、【図2】のように新規フォーム（Form1）が表示される。フォームの名前（Nameプロパティ）は「frmBase」に変更しておく。

**図 2　プロジェクトの新規作成**

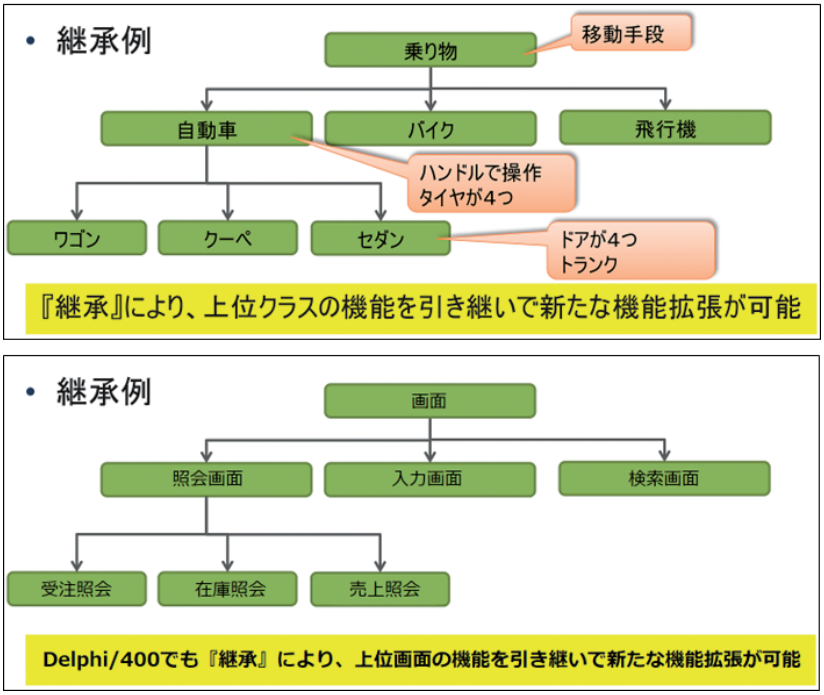


フォーム名の変更完了後、まずは先にフォームとプロジェクトを保存しておこう。保存先やファイル名は任意で指定可能であり、本稿では「BaseFrm.pas」「Project1.dproj」という名前で保存する。なお、ここで保存したプロジェクト名がコンパイルされた際の実行ファイル名（EXE名）になる。

## 3.継承フォームの作成とコーディング

次にフォームの継承について説明していく。
業務アプリケーションは、画面を機能や用途ごとに分類できる。同じ用途であれば、画面の構成や挙動を統一すると使い勝手が向上するだろう。統一化する際に役立つのが、フォームの「継承」である。本節では、フォームの継承について紹介する。
例えばプログラミングの世界以外でも【図3(上)】のようなものも継承の例といえる。この例では自動車や飛行機といった様々な乗り物があるが、大きく乗り物という一つの大きなカテゴリとして扱うことができる。これをDelphi/400のアプリケーション開発に当てはめると【図3(下)】のようになる。例えば「受注照会」「在庫照会」「売上照会」といった画面が存在するとき、いずれも参照するデータは異なるものの、『照会』するという機能は共通のため、一つの「照会画面」を継承することで、効率よく開発することが可能となる。

**図 3　継承の基本的な考え方**




今回作成するアプリケーションにおいて、各画面に共通して存在するものは何かを考えてみる。例えば本サンプルでは、画面名が記載されたパネルや「閉じる」ボタンは、全ての画面に配置したい。そこで、前章で保存したフォーム(frmBase)を継承元フォームとし、【図4】のように、各画面に共通で配置したいコンポーネントをセットしていく。

**図 4　継承元(frmBase)にセットするコンポーネント**

そしてフォーム表示時の処理(frmBaseのOnShowイベント)および「閉じる」ボタンを押した際の処理(btnCloseのOnClickイベント)を生成し、**【ソース1】**のように記述する。

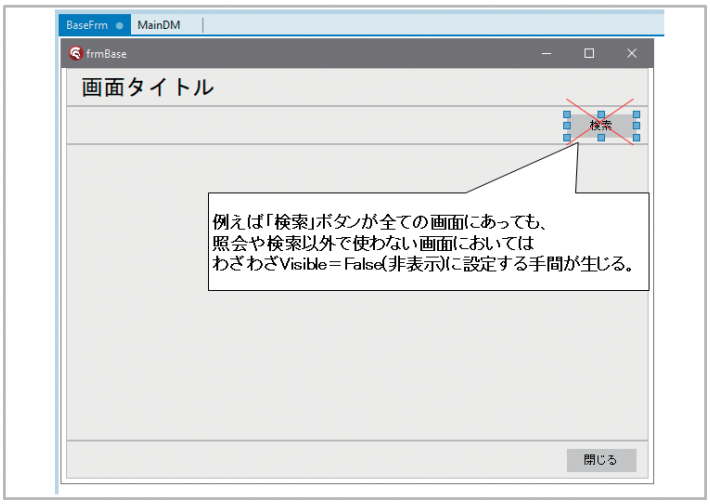
**ソース 1　基底画面のソース**

```
// 画面表示時処理
procedure TfrmBase.FormShow(Sender: TObject);
begin
  // 画面のタイトルを表示
  Self.Caption := lblTITLE.Caption;
end;

// 閉じるボタン押下時処理
procedure TfrmBase.btnCloseClick(Sender: TObject);
begin
  // 画面を閉じる
  Self.Close;
end;
```

ここでセットした項目やロジックは、継承先の各フォームではそのまま利用可能なため、改めてセット・記述する必要はない。また共通項目・ロジックに変更が発生した場合、継承元のフォームを変更すれば継承先にそれぞれ変更を加える必要はない。

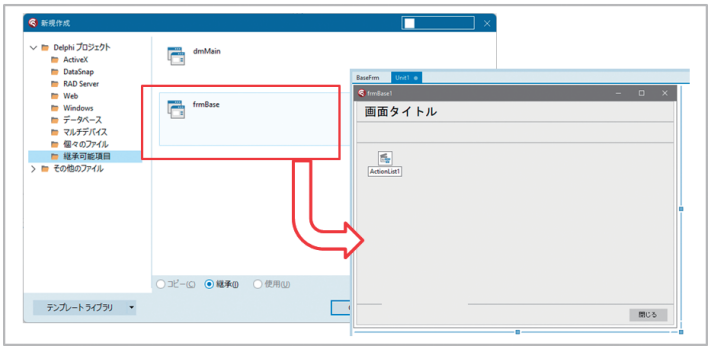
継承元フォーム作成時の注意点としては、よく使用するコンポーネントやロジックのみを継承元でセットする点である。継承元フォームで配置したコンポーネントは、継承先フォームでは削除することができない。例えば、使用頻度の低い項目・ロジックを継承元にセットすると、各画面で不要な項目の非表示や、不要なロジックを無効化する余分な手間が発生する**【図5】**。継承は便利である一方、継承先への制約となることも意識する必要がある。

**図 5　継承元には必要な項目だけを配置する**



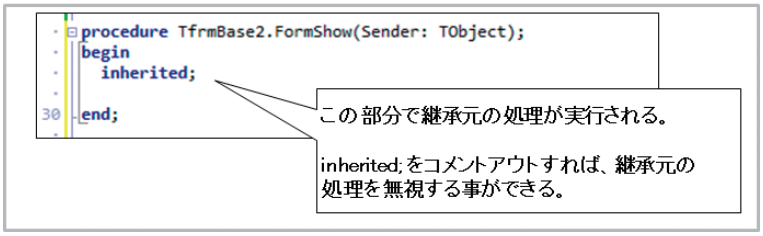

次に、継承先のフォームを作成する。[ファイル]メニューの「新規作成」から「その他」をクリックした後、表示される新規作成ダイアログの中から「継承可能項目」を選択する。すると、現在のプロジェクト内のフォームが一覧で表示されるので、この中から継承元にしたいフォーム(今回はfrmBase)を選択する。これにより、選択したフォームを継承した新しいフォームが作成される**【図6】**。

**図 6　継承先フォームの作成**



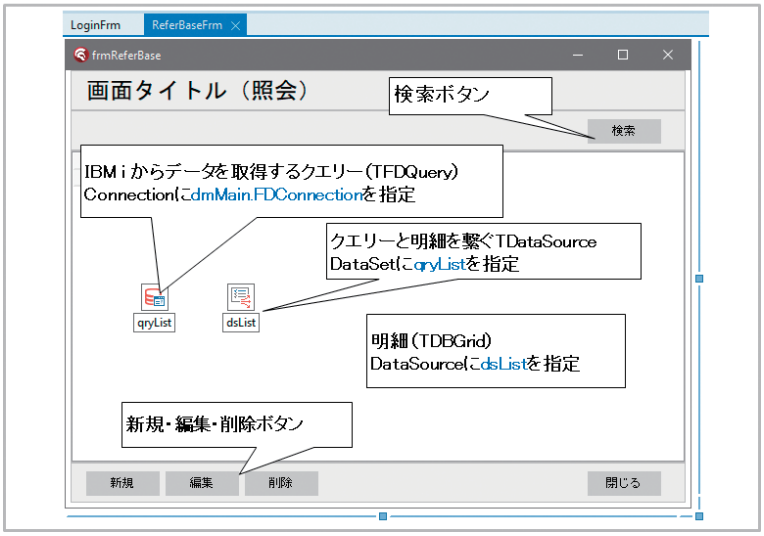
画面項目だけでなくロジックも継承されている。例えば今回は継承元フォームにて**【ソース1】**のロジックを記述したが、継承先フォームでは何も記述しなくても継承元と同じ処理が行われてフォーム表示時にタイトルがセットされる。
継承先ではフォーム表示時処理のロジックが**【図7】**のようになっており、『inherited;』と記述された部分で継承元の処理が呼び出されている。従って、この行をコメントアウトすると継承元の処理を実施しない。

**図 7　継承先のイベント**



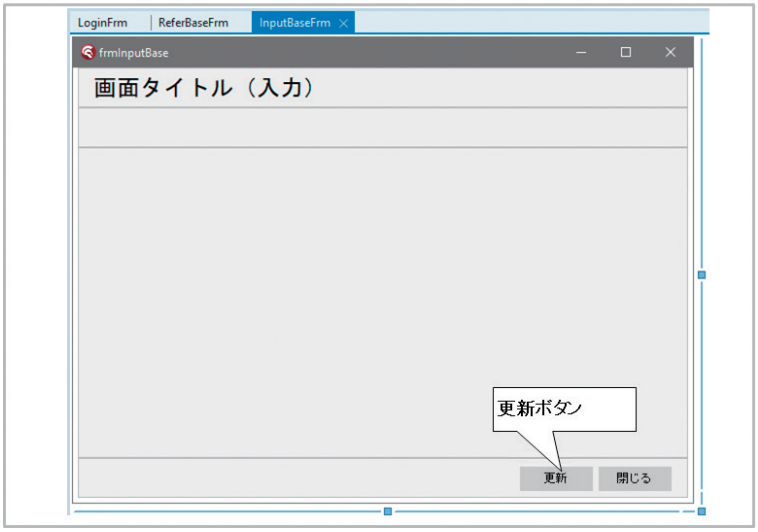

**【図6】**で作成したフォームは、照会画面の継承元として使用する。名前（Ｎａｍｅ）を「frmReferBase」に変更し、「ReferBaseFrm.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。続けて、照会画面で共通して使用するコンポーネントを**【図8】**のように配置する。

**図 8　照会画面の継承元にセットするコンポーネント**




また、入力画面の継承元も同様に作成し、フォーム名「frmInputBase」を設定し、ファイル名「InputBaseFrm.pas」として保存する。こちらにも**【図9】**のようにコンポーネントを配置する。

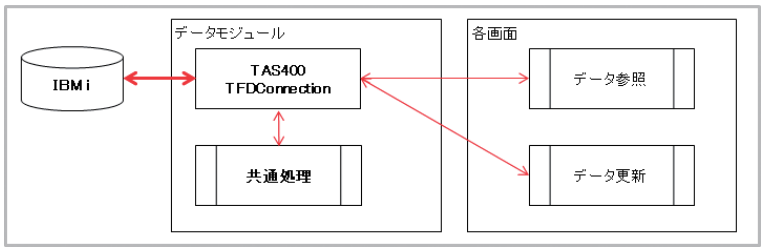
**図 9　入力画面の継承元にセットするコンポーネント**



## 4.データモジュールを活用したIBM iとの接続処理

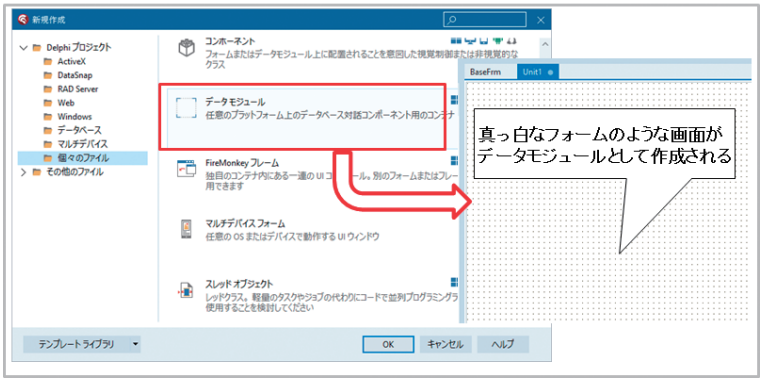
ここからはデータモジュールの作成方法を説明する。通常のフォームではなくデータモジュールを使用することで、データベースへのアクセスやデータ処理に関するロジックを専用の独立したモジュールに分離し、コーディングの簡略化や保守性の向上を図ることができる。また、同じデータモジュールを複数のフォームやユニットで共有し、一か所で管理することによって、コードの重複を避けて保守性や拡張性を高めることができる【図10】。

**図 10　データモジュールの役割**



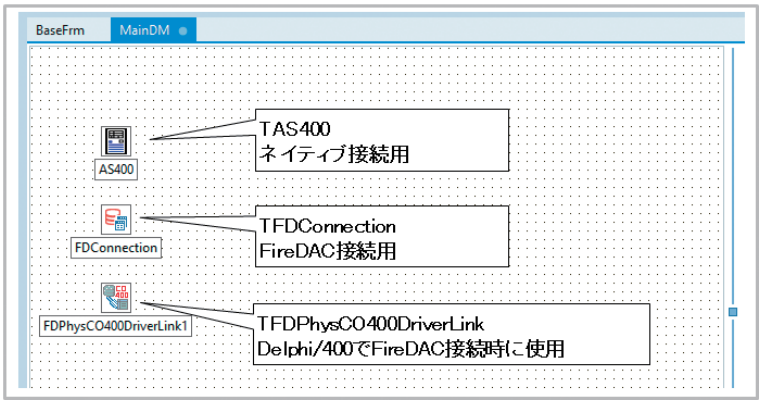

データモジュールを新規作成するには、[ファイル]メニューの「新規作成」から「その他」を選択し、新規作成の設定ダイアログから「個々のファイル」を選び、「データモジュール」を指定する【図11】。

**図 11　データモジュールの新規作成**



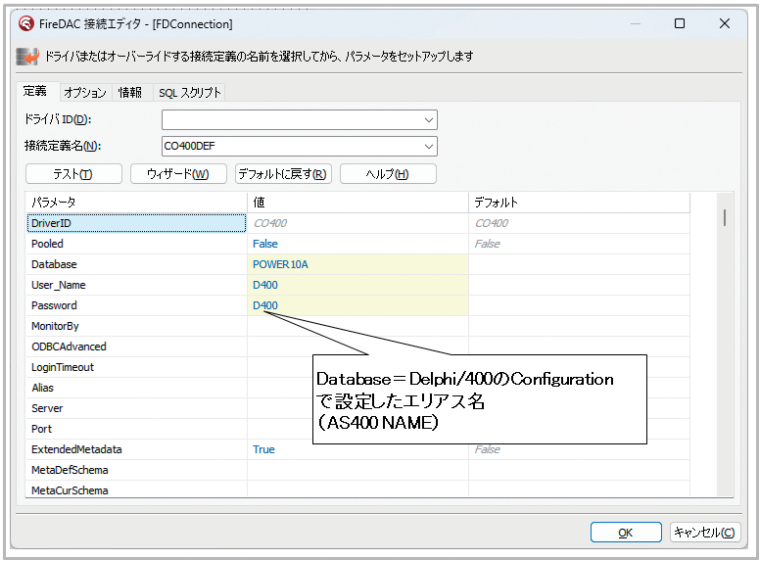

作成したデータモジュールの名前（Ｎａｍｅ）を「ｄｍＭａｉｎ」とし、「MainDM.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。dmMainにはデータベース接続に使用する各コンポーネントを配置していく。今回はFireDAC接続用コンポーネントとしてTFDConnection・TFDPhysCO400DriverLinkを、ネイティブ接続コンポーネントとしてTAS400をフォームに配置する【図12】。

**図 12　データモジュールの項目配置例**



Delphi/400でIBMi(AS/400)に接続するため、今回はFireDAC接続とネイティブ接続を併用する。本稿ではデータアクセスに全てSQLを使用しており、プログラム内ではFireDAC接続しか利用していないが、ネイティブ接続は、CL、RPG等の呼出やIBM iコマンド実行が必要な際に容易に利用できるため、接続部分は併用する形で紹介した。
FireDAC接続については過去のテクニカルレポートなどでも何度か紹介している(※)ため、そちらも参照頂きたい。
コンポーネントを配置後、TFDConnectionをダブルクリックしてFireDAC接続エディタを起動し、接続設定を【図13】のように行う。ここで実際のエリアス名・ユーザー・パスワードを入力して「テスト」ボタンを押すと、設定した内容で接続テストを実行できる。
(※https://www.migaro.co.jp/ts/19th/Session3.pdf や
https://www.migaro.co.jp/tr/no11/tech/11_01_02.pdf を参照)

**図 13　FireDAC接続エディタの設定例**




また、dmMainの画面上で何もない場所をダブルクリックして生成される、dmMainのOnCreateイベント(TdmMain.DataModuleCreate)の中でIBM iへの接続処理を記述していく**【ソース2】**。

**ソース 2　データモジュール生成時のソース**

```
// データモジュール生成時処理
procedure TdmMain.DataModuleCreate(Sender: TObject);
var
  sALIAS, sUSER, sPASS, sLIB : String;
begin
  // 接続パラメータの設定（実際には固定値やINIファイルなどを使用）
  sALIAS := 'POWER10A'; // 接続先AS/400名
  sUSER  := 'D400';     // サインオンユーザー
  sPASS  := 'D400';// サインオンパスワード
  sLIB   := '';// デフォルトライブラリ（*LIBL指定時はブランク）

  // 初期処理（切断）
  FDConnection.Connected := False;
  AS400.Active   := False;

  // AS400コンポーネント接続定義
  AS400.Userid       := sUSER;  // サインオンユー
  AS400.PWD          ザー := sPASS;  // サインオンパ
  AS400.PLUAlias := sALIAS;ド// 接続先AS/400名

  // データベースコンポーネント接続定義
  with FDConnection do
  begin
    Params.Values['ConnectionDef'] := 'CO400DEF';
     Params.Values['User_Name']                  // サインオンユーザー
      Params.Values['PASSWORD'] := sUSER; := sPASS;      // サインオンパスワー
      Params.Values['Database']      := sALIAS;     ド // 接続先AS/400名
    Params.Values['ODBCAdvanced']  := 'LibraryOption=' + sLIB; // デフォルトライブラリ
  LoginPrompt := False;
  end;

  // 接続処理
  FDConnection.Connected := True;
  AS400.Active   := True;
end;
```

## 5.各画面のコーディング

前章までででプログラムの下地ができたので、ここから各機能の画面を作成していく。

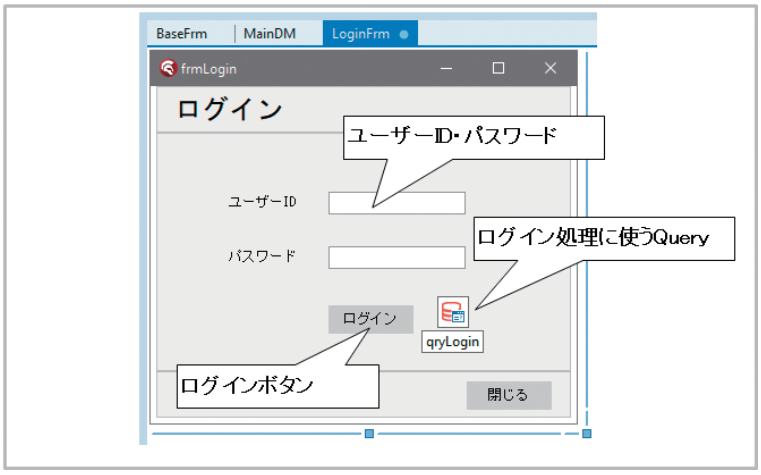
### 5-1. ログイン・メニュー画面

今回のプログラムでは「ログイン」→「メニュー」→「一覧照会画面」→「入力画面」といった画面遷移を想定する。まずは、ログイン画面を作成する。

ログイン画面は照会画面でも入力画面でもないため、基底フォーム(frmBase)を継承元として新規作成する。新規作成したログイン画面は名前(Name)を「frmLogin」とし、「LoginFrm.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。

ログイン画面のフォームでは**【図14】**のように、ユーザーIDとパスワードの入力欄、そしてログインボタンを配置する。

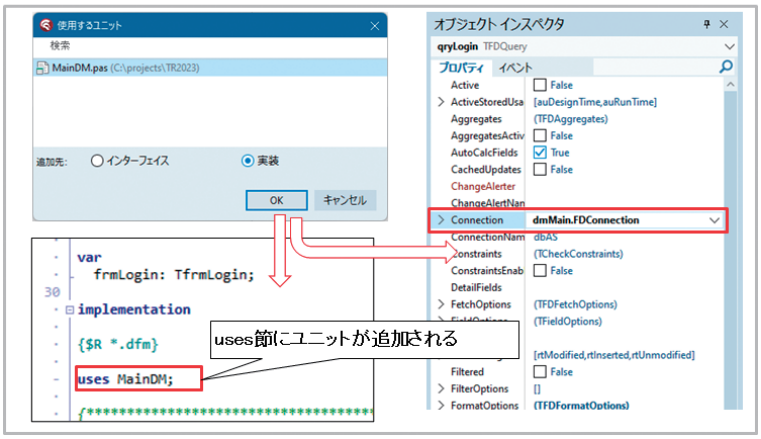
**図 14 ログイン画面の項目**




また、TFDQueryをフォーム内の任意の場所に配置する。前章でデータベース関連のコンポーネントはデータモジュールに配置すると記載したが、各個別フォームでしか使用しないTFDQueryについては、そのフォーム内に配置したほうが良いだろう。

ログインボタンの処理ではデータベースに接続する。ファイルメニューの「使用するユニット」から「MainDM.pas」を選択し、第4章で作成したdmMainの参照設定を行う。するとuses節に「MainDM」が追加され、ログイン画面からdmMainを参照できるようになる**【図15(左)】**。この参照設定によって、ログイン画面に配置したTFDQueryの設定を**【図15(右)】**のように行うことができる。

**図 15 使用するユニット**



設計画面で配置したログインボタンをダブルクリックすると、ログインボタンを押した際の処理を記述できる。ここには【ソース3-①】のようにユーザーマスタを参照して、入力された値が正しいかチェックを行うロジックを記述する。(ユーザーマスタのファイルレイアウトは【図16】のように定義されているものとする。)

## 図 16　今回のユーザーマスタのレイアウト

DSPFMT　　レ コ ー ド 設 計 書　　日付　23/08/28
時刻　13:09:03

| 物理ファイル | YSADALIB/USERMASTER | 様式名 | USERMR | レコード長 | 81 |
|---|---|---|---|---|---|
| 様式記述 | テクレポ２０２３ユーザー | | | | |

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／欄見出し |
|---|---|---|---|---|---|---|---|
| _ | USERID | 8 | A | 1 ANN | 1 | 8 | ユーザーＩＤ |
| _ | USERNM | 32 | 0 | | 9 | 40 | ユーザー名 |
| _ | USRKNM | 16 | 0 | | 41 | 56 | ユーザー略称 |
| _ | USPASS | 8 | A | | 57 | 64 | パスワード |
| _ | USBUSY | 16 | 0 | | 65 | 80 | 部署名 |
| _ | USDLFG | 1 | A | | 81 | 81 | 削除フラグ |

## ソース 3　ログイン画面のログイン処理ソース

```
// ログインボタン押下時処理
procedure TfrmLogin.btnLoginClick(Sender: TObject);
begin
  inherited;

  //【ログイン処理】-------------------------------------------------------------
  // 入力チェック
  if (edtUSER.Text = '') or (edtPWD.Text = '') then
  begin
    ShowMessage('ユーザーIDまたはパスワードが未入力です。');
  Abort; // 処理中断
  end;

  // ユーザーマスタを参照
  qryLogin.Close;
  qryLogin.SQL.Clear;
  qryLogin.SQL.Add('  SELECT * FROM YSADALIB/USERMASTER ');
  qryLogin.SQL.Add('  WHERE USDLFG = '''' ');  // 論理削除済でない
  qryLogin.SQL.Add('  AND   USERID = ' + QuotedStr(edtUSER.Text)); // IDが一致
  qryLogin.Open;

  if (qryLogin.Bof and qryLogin.Eof) then // 対象レコード無し
  begin
    ShowMessage('ユーザーが存在しません。');
  Abort; // 処理中断
  end;
  if (qryLogin.FieldByName('USPASS').AsString <> edtPWD.Text) then
  begin
     ShowMessage('パスワードが間違っています。');
     Abort; // 処理中断
  end;

  //【メニュー画面への遷移】-----------------------------------------------------
  frmMenu := TfrmMenu.Create(Self); // メニュー画面を生成
  try
    frmMenu.ShowModal;              // メニュー画面を表示
  finally
    FreeAndNil(frmMenu);            // メニュー画面を閉じた後は解放する
  end;
end;
```

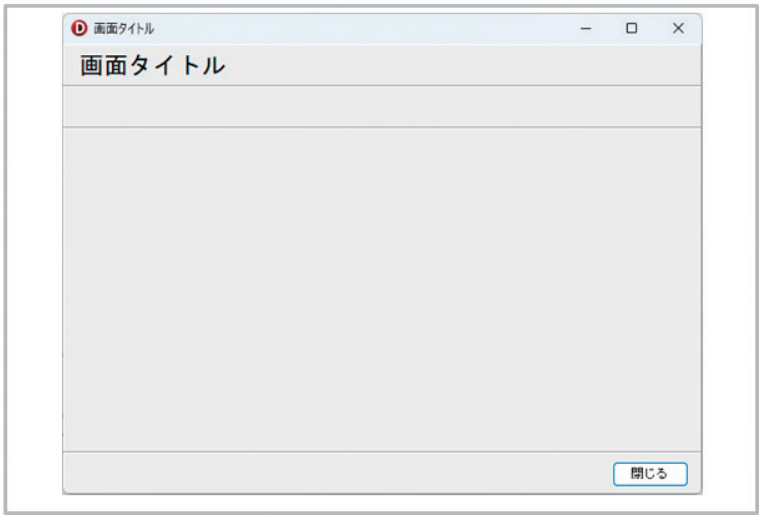
①

②

なお、各ソース内で使用している関数「QuotedStr」は、文字列をシングルクォーテーション(' ')で囲むための関数である。ソース内で『'AND USERID='+QuotedStr(edtUSER.Text)』と記述している箇所で、『'AND USERID='''+edtUSER.Text+''' '』と記述することも可能である。しかし、この方法では、囲まれる文字列にシングルクォーテーションが含まれているときに文字列が終わる位置を正しく判定できず、エラーになってしまうことがある。QuotedStrを使うとその調整を自動で行ってくれる。

ここで、作成したプログラムをコンパイルして実行してみよう。すると【図17】のように最初に作成したfrmBaseが表示される。

**図 17 初回実行結果の画面**




これはプロジェクトの設定でメインフォームにfrmBaseが指定されているためである。ログイン画面を最初に表示させるには、プロジェクトオプションでの設定が必要である。アプリケーションの「フォーム」の項目で、新規作成された各フォームやデータモジュールなどは【図18】では左側の「自動生成フォーム」に入っている。「使用可能フォーム」へ移すことで、自動生成の対象から外すことができる。プログラムは起動すると「自動生成フォーム」の上から順にフォームを生成する。本稿では「dmMain」「frmLogin」の順で設定する。(dmMainがログイン画面よりも先なのは、最初に暗黙のサインオンを行うため)

なお、この自動生成フォームのうち最初(一番上)に設定されたフォームがメインフォームとなるが、このメインフォームが閉じられるとアプリケーションは終了する。

改めてプログラムをコンパイルして実行すると、ログイン画面が最初に表示されるようになる【図19】。

**図 18 自動生成フォームの設定**

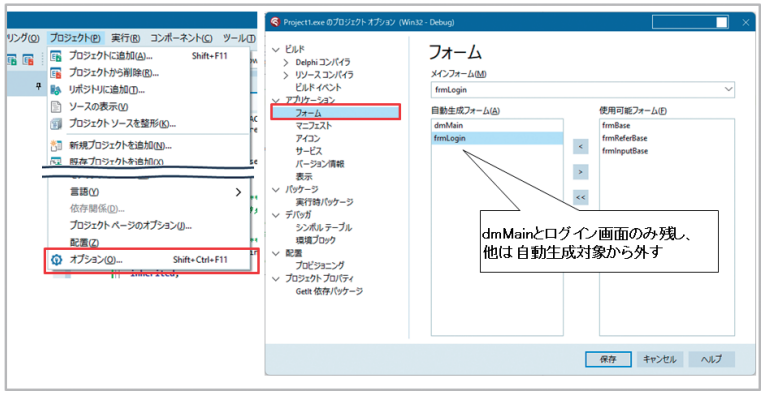
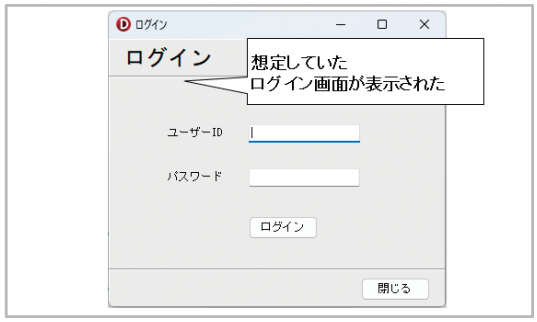



**図 19 改めてEXE起動**

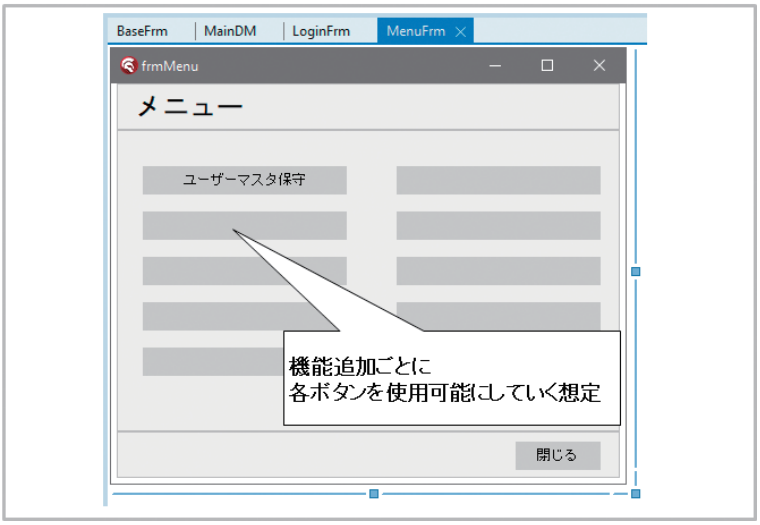


次にメニュー画面を作成する。先ほどのログイン画面と同様に基底フォーム(frmBase)を継承して新規作成し、名前(Ｎａｍｅ)を「frmMenu」とし、「MenuFrm.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。また、プロジェクトオプションの「自動生成フォーム」に追加されている「frmMenu」を除外する。

ログイン画面からメニュー画面に遷移する処理を記述する。IDEでログイン画面のソースを表示したら、先ほどと同様に「使用するユニット」を使って今作成した「MenuFrm.pas」への参照を可能にし、画面遷移ロジックを記述する**【ソース3-②】**。

メニュー画面は、**【図20】**のように各機能へ遷移するためのボタンを配置し、遷移先の画面を作成後に**【ソース4】**のように画面遷移の処理を記述する。

**図 20　メニューの画面項目**



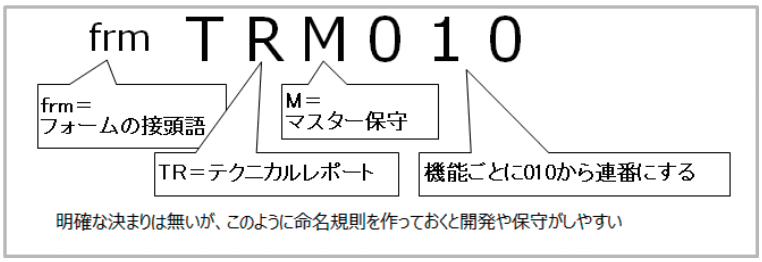

**ソース 4　メニュー画面のボタン押下時ソース**

```
// ユーザーマスタ保守ボタン押下時処理
procedure TfrmMenu.Button1Click(Sender: TObject);
begin
  inherited;

  // ユーザーマスター覧照会画面を生成frmTRM010 :=
  TfrmTRM010.Create(Self);
  try
    // ユーザーマスター覧照会画面に遷移
    frmTRM010.ShowModal;
  finally
   // 画面を閉じた後は解放
   FreeAndNil(frmTRM010);
   end;
end;
```

## 5-2. 一覧照会画面

次に、ユーザーマスタ保守の一覧照会画面を作成する。新規作成メニューの「継承可能項目」から、照会画面の継承元(frmReferBase)フォームを継承して新しいフォームを作成する。個別画面ではフォーム名には、命名規則を設けると保守性が高まるだろう。本稿では**【図21】**のように命名規則を設け、名前(Name)を「frmTRM010」とし、「TRM010Frm.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。また、メニュー画面から当画面に遷移するロジックを記述し、前項で解説したプロジェクトオプションの「自動生成フォーム」に追加されている「frmTRM010」を除外する。

**図 21　フォームの命名規則の例**



今回は【図16】のユーザーマスタを一覧照会する機能を作成する。
作成したフォーム(frmTRM010)のタイトルラベルを「ユーザーマスター覧照会」とし、検索条件の項目を【図22】のように追加する。

**図 22　マスター覧照会画面の項目**


続いて検索ボタンをダブルクリックし、検索処理のロジックを【ソース5】のように記述する。今回はユーザーIDを完全一致、ユーザー名を部分一致で絞り込むSQLを記述している。また、削除フラグがオフ(空白)のレコードのみを対象としている。データセット(qryList)を開いて明細を表示し、対象データが0件の場合はエラーを表示してデータセットを閉じる(明細も閉じる)。

**ソース 5　ユーザーマスター覧照会画面の検索処理ソース**

```
// 検索ボタン押下時処理
procedure TfrmTRM010.btnSearchClick(Sender: TObject);
begin
  inherited;
  with qryList do
  begin
    Close;
    SQL.Clear;
    SQL.Add(' SELECT * FROM YSADALIB/USERMASTER ');
    SQL.Add(' WHERE USDLFG = '''' ');  // 論理削除済でない

    if (edtUSERID.Text <> '') then     // ユーザーIDが一致（完全一致）
    begin
      SQL.Add(' AND USERID = ' + QuotedStr(edtUSERID.Text));
    end;
    if (edtUSERNM.Text <> '') then     // ユーザー名が一致（部分一致）
    begin
      SQL.Add(' AND USERNM LIKE ' + QuotedStr('%' + edtUSERNM.Text + '%'));
    end;

    Open;

    if (Bof and Eof) then
    begin
      Close; // 開いた明細を閉じる
      ShowMessage('対象データが存在しません。');
      Abort; // 処理中断
    end;
  end;
end;
```

ここまで実装できたら、プロジェクトをコンパイルしてEXEの動作を確認してみよう。ユーザーの一覧が【図23】のように表示されているのを確認できるだろう。

【図23】では事前にエミュレータのDFUで登録したデータが2件表示されている。以下の4点を追加で実装すると、照会画面としてより利便性がよくなるだろう。

①明細のタイトル部にフィールドIDが記載されているが、システムを利用する方がわかりやすいタイトルに変更したい。

②パスワードが見えてしまっているため、この列を非表示にしたい。

③選択しているセルだけではなく、選択行全体の色を変更するようにしたい。

④明細から他の項目にフォーカスが移ると選択行が元の色に戻ってしまうため、常に選択行が色でわかるようにしたい。

これらの課題を解決するため、プログラムを変更していく。

まずは課題①および②を解決するため、フィールドの情報をIBM iから取得した上でタイトルや非表示などの設定を行っていく。

フォームのTFDQuery（qryList）のSQLプロパティに、ロジックでセットしているものと同じSQLをセットする。（今回の場合は『SELECT * FROM YSADALIB/USERMASTER』。WHERE以降は不要）

この状態でqryListのActiveプロパティをTrueに変更すると、【図24】のように設計画面上の明細が開かれた状態になる。qryListを右クリックしてフィールドエディタを表示し、右クリックメニューから「すべてのフィールドを追加」を選択すると、指定のフィールド情報を取得することができる【図25】。

**図 23　マスター覧照会画面を動かしてみた結果と課題**

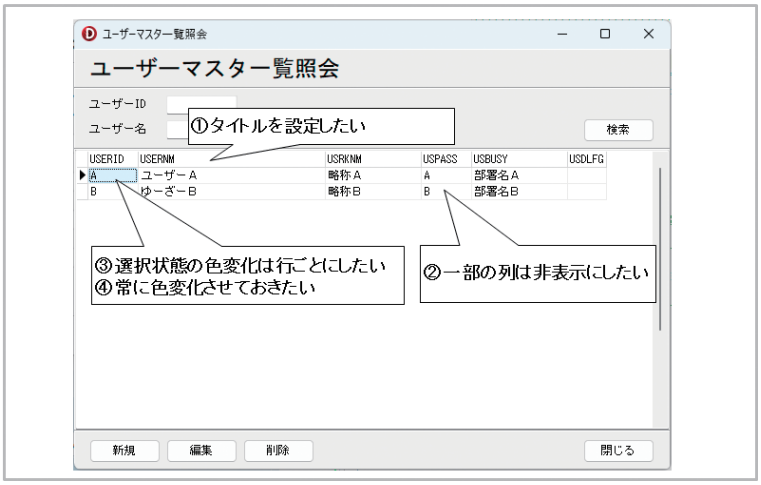
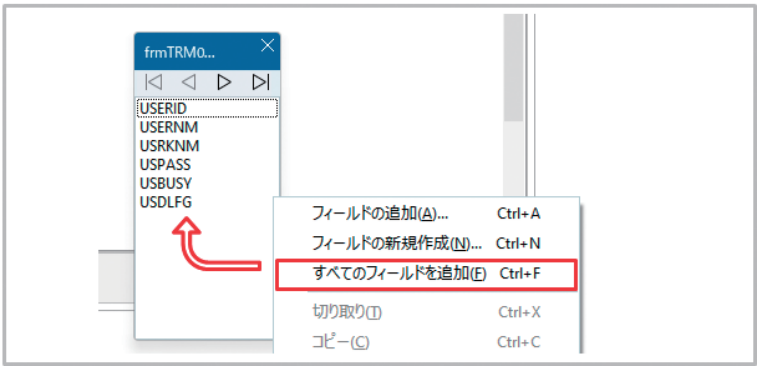


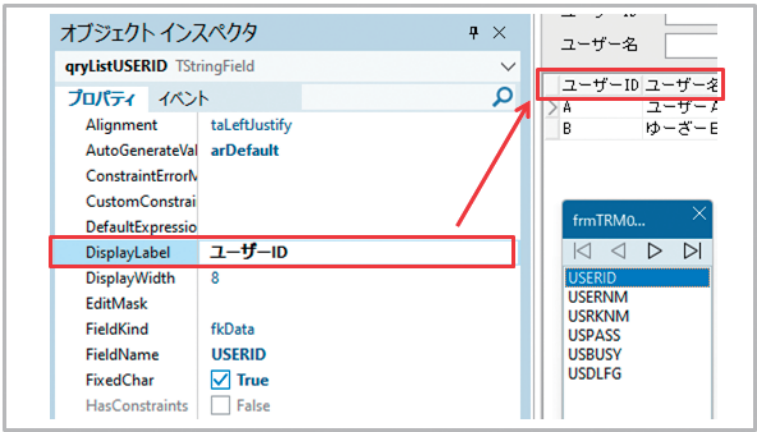

**図 24　設計画面上で明細を開く**


**図 25　qryListにフィールドを追加する**



取得したフィールドを選択し、プロパティ一覧からDisplayLabelプロパティの値を変更すると、明細のタイトルを変更できる【図26】。また、フィールドのプロパティ一覧からVisibleプロパティをFalseに変更することで、明細からそのフィールドを非表示に設定できる。

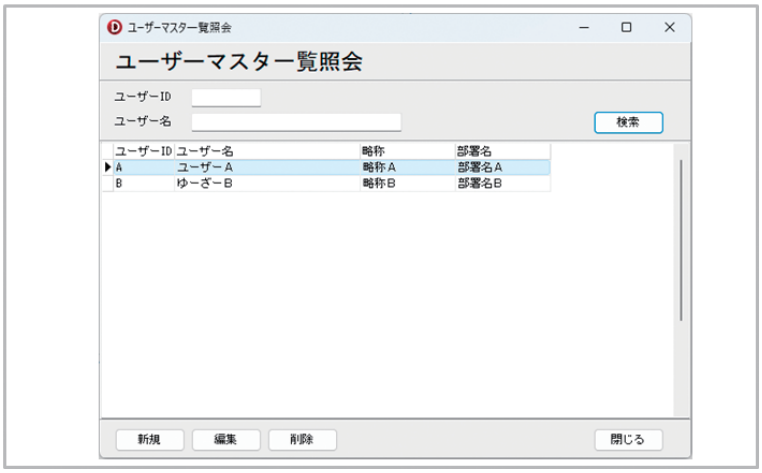
**図 26　明細タイトルの設定**




この方法を行う際の注意事項は、SQLを実行するタイミングでdmMain.FDConnectionを使ってIBM iに接続するため、【図13】のエリアス名、ユーザー、パスワードが接続可能な設定にしておく点である。またフィールド情報を取得した後はdmMain.FDConnectionのConnectedプロパティをFalseに変更し、設計画面からの接続を切断しておく必要がある。（Trueになっていると、次回Delphiの設計画面でこのフォームを開いた際にもIBM iへ接続を試行してしまう）

次に課題③および④については、明細TDBGrid（dbgList）の設定を変更すればよい。dbgListのOptionsプロパティにて、dgRowSelectをTrueに設定すると選択範囲が行単位になる。またdgAlwaysShowSelectionをTrueに設定すると常時選択行の色が変わった状態になる。再度プログラムをコンパイルして起動すると、明細が【図27】のように表示され、課題が解決していることを確認できる。

**図 27　設定変更後の一覧照会画面イメージ**



### 5-3.入力画面

本項では、ユーザーマスタのデータを新規登録・編集・削除するための入力画面（ユーザーマスタ保守）を作成する。
まず前章までに作成した入力画面の継承元（frmInputBase）を継承して、新しい入力用のフォームを作成する。今回は名前（Name）を「frmTRM020」とし、「TRM020Frm.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。また、プロジェクトオプションの「自動生成フォーム」に追加されている「frmTRM020」を除外する。
**【図28】**のように画面項目を配置し、使用するユニットからdmMainを参照設定する。TFDQueryについては、今回は参照用と更新用の2つを配置する。

次にフォームの変数定義（Public宣言）に「USERID」と「MODE」という2つのプロパティ変数を記述する。プロパティ変数は読み取りと書き込みを制御できる。**【ソース6-①】**のように記述し、Shift+Ctrl+Cキーを押すことで、**【ソース6-②】**のようにread・write部がPrivate宣言部に自動生成される。他のユニットからプロパティを参照・更新しようとすると、それぞれread・write部で指定した変数またはメソッドを使って読み取り・書き込みされる**【図29】**。

**図 28　ユーザーマスタ保守画面の項目**

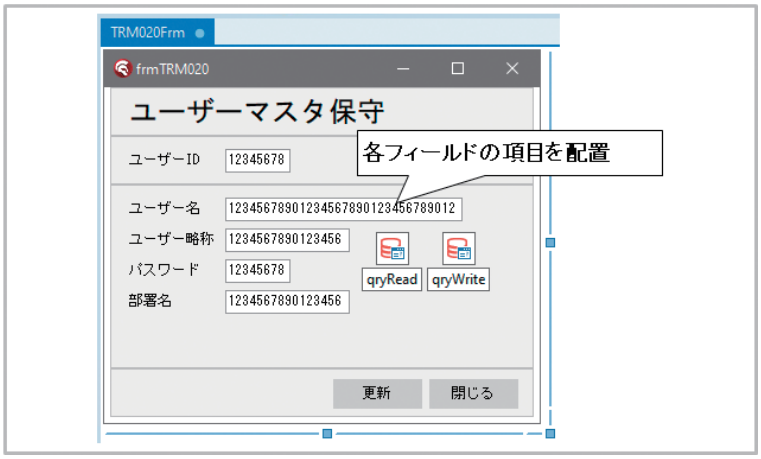
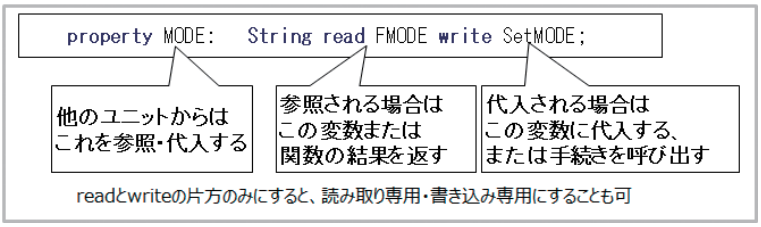



**図 29　プロパティ変数のイメージ**




**ソース 6　ユーザーマスタ保守画面 プロパティ変数の設定**

```
// 【Shift+Ctrl+C 押下前】
// （宣言部）
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    property USERID: String;
    property MODE:   String;
  end;

//---------------------------------------------------------------------
// 【Shift+Ctrl+C 押下後】
// （宣言部：ソースの冒頭）
  private
    { Private 宣言 }
    FMODE: String;
    FUSERID: String;
    procedure SetMODE(const Value: String);
    procedure SetUSERID(const Value: String);
  public
    { Public 宣言 }
    property USERID: String read FUSERID write SetUSERID;
    property MODE:   String read FMODE write SetMODE;
  end;

// （実装部：ソースの「implementation」より後）

procedure TfrmTRM020.SetMODE(const Value: String);
begin
  FMODE := Value;
end;

procedure TfrmTRM020.SetUSERID(const Value: String);
begin
  FUSERID := Value;
end;
```

この画面は前項の一覧照会画面から呼び出されて遷移する前提なので、フォーム表示時処理(FormShowイベント)にて【ソース7】のように記述する。

**ソース 7　ユーザーマスタ保守画面 表示時処理ソース**

```
// 画面表示時処理
procedure TfrmTRM020.FormShow(Sender: TObject);
begin
  inherited;
  // 新規モードの場合
  if (FMODE = 'N') then
  begin
    edtUSERID.Clear;             // ユーザーＩＤ
    edtUSERNM.Clear;             // ユーザー名
    edtUSRKNM.Clear;             // ユーザー略称
    edtUSPASS.Clear;             // パスワード
    edtUSBUSY.Clear;             // 部署名
    chkUSDLFG.Checked := False; // 削除フラグ
  end
  // 編集または削除モードの場合
  else
  begin
    with qryRead do
    begin
      Close;
      SQL.Clear;
      SQL.Add(' SELECT * FROM YSADALIB/USERMASTER ');
      SQL.Add(' WHERE USERID = ' + QuotedStr(FUSERID));     // ユーザーID
      Open;
      try
        edtUSERID.Text := FieldByName('USERID').AsString;   // ユーザーID
        edtUSERNM.Text := FieldByName('USERNM').AsString;   // ユーザー名
        edtUSRKNM.Text := FieldByName('USRKNM').AsString;   // ユーザー略称
        edtUSPASS.Text := FieldByName('USPASS').AsString;   // パスワード
        edtUSBUSY.Text := FieldByName('USBUSY').AsString;   // 部署名
        // USDLFGの値が"D"の場合削除フラグオンとする
        chkUSDLFG.Checked := (FieldByName('USDLFG').AsString = 'D');
        // ユーザーIDは変更不可
        edtUSERID.Enabled := False;
        // 削除モードの場合はほかの項目も変更不可とし、ボタンの文言を変える
        if (FMODE = 'D') then
        begin
          edtUSERNM.Enabled := False;
          edtUSRKNM.Enabled := False;
          edtUSPASS.Enabled := False;
          edtUSBUSY.Enabled := False;
          chkUSDLFG.Enabled := False;
          btnPost.Caption   := '削除';
        end;
      finally
        Close; // 参照に使ったQueryを閉じる
      end;
    end;
  end;
end;
```

また、ユーザーマスター一覧照会画面で「使用するユニット」からfrmTRM020を参照設定し、「新規」「編集」ボタンを押した際の処理を【ソース8】のように記述する。「削除」ボタンの処理は「編集」ボタンのソースとほぼ同じで、frmTRM020.MODE := 'E';の部分をfrmTRM020.MODE := 'D';に変更すればよい。

**ソース 8　ユーザーマスター一覧照会画面から保守画面への遷移処理**

```
// 新規ボタン押下時処理
procedure TfrmTRM010.btnNEWClick(Sender: TObject);
begin
  inherited;
  // 新規モードでユーザーマスタ保守画面を開く
  frmTRM020 := TfrmTRM020.Create(Self);
  try
    frmTRM020.MODE   := 'N';   // 画面モード＝「N」（新規）
    frmTRM020.USERID := '';    // ユーザーID

    // 画面遷移し、更新（削除）して戻ってきたら画面を再検索する
    if (frmTRM020.ShowModal = mrOk) then
    begin
      btnSearch.Click;
    end;
  finally
    FreeAndNil(frmTRM020);
  end;
end;

// 編集ボタン押下時処理
procedure TfrmTRM010.btnEDITClick(Sender: TObject);
begin
  inherited;
  // 明細が開いていない場合、編集対象が無いので処理中断
  if (not qryList.Active) then Abort;

  // 編集モードでユーザーマスタ保守画面を開く
  frmTRM020 := TfrmTRM020.Create(Self);
  try
    frmTRM020.MODE   := 'E';   // 画面モード＝「E」（編集）
    frmTRM020.USERID := qryList.FieldByName('USERID').AsString; // ユーザーID

    // 画面遷移し、更新（削除）して戻ってきたら画面を再検索する
    if (frmTRM020.ShowModal = mrOk) then
    begin
      btnSearch.Click;
    end;
  finally
    FreeAndNil(frmTRM020);
  end;
end;
```

変数「MODE」の値によって、新規モード・編集モード・削除モードそれぞれの場合に異なる動作や画面項目の入力制御を行う。大まかには、以下のような違いをもたせる。

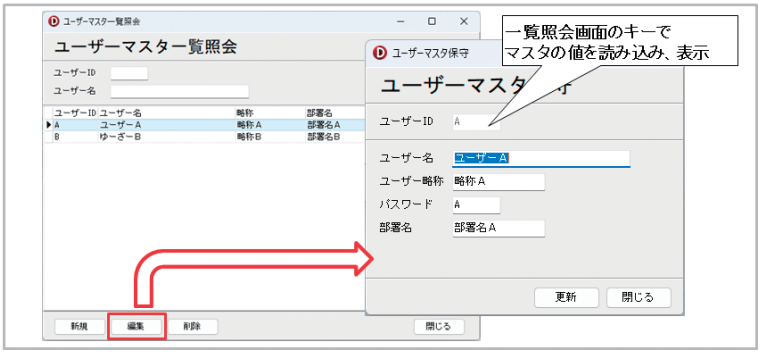
-新規モード（MODE＝"N"）：各入力項目をクリアした状態で画面を表示する。

-編集モード（MODE＝"E"）：キー（USERID）をもとに各入力項目に値をセットした状態で画面を表示する。ただしキーだけは変更不可とする。

-削除モード（MODE＝"D"）：キー（USERID）をもとに各入力項目に値をセットした状態で画面を表示する。各入力項目は変更不可とし、更新ボタンの文言を「削除」に変更する。

ここまで実装したら、プロジェクトをコンパイルしてEXEの動作を確認してみよう。一覧照会画面から呼び出されたデータが想定通り表示されているかを確認できる**【図30】**。

**図 30　入力画面の表示例**




次は各モードの更新処理を行う。更新処理では「エラーチェック⇒更新処理」の順で処理を行うが、具体的には以下のような処理の流れになる。

-新規モード：既に同じキーのデータが存在する場合はエラーとし、問題がなければ画面の入力内容でINSERTのSQLを発行する。

-編集モード：画面の入力内容でUPDATEのSQLを発行する。

-削除モード：画面のキー（USERID）をもとに削除フラグをオンにするUPDATEのSQLを発行する。（今回は論理削除の想定だが、物理削除で良い場合はDELETEのSQLを発行する）

それぞれのモードごとに、**【ソース9】**のように更新ロジックを記述する。本稿ではサンプルのため数値・文字の桁あふれエラーについては考慮していないが、実際のプログラムにおいては全角文字の前後のシフト文字を含めて考慮が必要となる。（※桁あふれエラー対策の手順については、https://www.migaro.co.jp/tips/2910 を参照）

更新完了後には「更新しました。」というメッセージを表示し、フォームを閉じて一覧照会画面に戻るように記述している。更新によって一覧照会画面へ戻った際には**【ソース8】**の記述によって再検索が行われる。

## ソース 9　ユーザーマスタ保守画面 更新処理

```
// 更新（削除）ボタン押下時処理
procedure TfrmTRM020.btnPostClick(Sender: TObject);
begin
  inherited;
  // 処理確認メッセージ（いいえ押下時は中断）
  if (MessageBox(Handle, '更新します。よろしいですか？',
    '更新確認', MB_ICONQUESTION + MB_YESNO) = mrNo) then
    Abort;

  // 新規モードの場合
  if (FMODE = 'N') then
  begin
    // キー重複のチェック
    with qryRead do
    begin
      Close;
      SQL.Clear;
      SQL.Add(' SELECT * FROM YSADALIB/USERMASTER ');
      SQL.Add(' WHERE USERID = ' + QuotedStr(edtUSERID.Text));     // ユーザーID
      Open;
      try
        if not(Bof and Eof) then
        begin
          ShowMessage('このユーザーIDは既に存在しています。');
          edtUSERID.SetFocus;
          Abort;
        end;
      finally
        Close; // 参照に使ったQueryを閉じる
      end;
    end;
    // 更新処理
    with qryWrite do
    begin
      Close;
      SQL.Clear;
      SQL.Add(' INSERT INTO YSADALIB/USERMASTER ');
      SQL.Add(' (USERID, USERNM, USRKNM, USPASS, USBUSY, USDLFG) ');
      SQL.Add(' VALUES (');
      SQL.Add(QuotedStr(edtUSERID.Text) + ',');  // ユーザーＩＤ
      SQL.Add(QuotedStr(edtUSERNM.Text) + ',');  // ユーザー名
      SQL.Add(QuotedStr(edtUSRKNM.Text) + ',');  // ユーザー略称
      SQL.Add(QuotedStr(edtUSPASS.Text) + ',');  // パスワード
      SQL.Add(QuotedStr(edtUSBUSY.Text) + ',');  // 部署名
      SQL.Add(QuotedStr('') + ')');              // 削除フラグ＝ブランク
      ExecSQL; // SQL実行
    end;
  end
  // 編集モードの場合
  else
  if (FMODE = 'E') then
  begin
    with qryWrite do
    begin
      Close;
      SQL.Clear;
      SQL.Add(' UPDATE YSADALIB/USERMASTER SET ');
      SQL.Add(' USERNM = ' + QuotedStr(edtUSERNM.Text) + ',');  // ユーザー名
      SQL.Add(' USRKNM = ' + QuotedStr(edtUSRKNM.Text) + ',');  // ユーザー略称
      SQL.Add(' USPASS = ' + QuotedStr(edtUSPASS.Text) + ',');  // パスワード
      SQL.Add(' USBUSY = ' + QuotedStr(edtUSBUSY.Text));        // 部署名
      SQL.Add(' WHERE USERID = ' + QuotedStr(FUSERID));         // ユーザーID
      ExecSQL; // SQL実行
    end;
  end
  // 削除モードの場合
  else
  if (FMODE = 'D') then
  begin
    with qryWrite do
    begin
      Close;
      SQL.Clear;
      SQL.Add(' UPDATE YSADALIB/USERMASTER SET ');
      SQL.Add(' USDLFG = ' + QuotedStr('D'));       // 削除フラグに「D」をセット
      SQL.Add(' WHERE USERID = ' + QuotedStr(FUSERID));         // ユーザーID
      ExecSQL; // SQL実行
    end;
  end;
  // 更新完了メッセージ
  MessageBox(Handle, '更新しました。', '完了', MB_ICONINFORMATION);
  // 更新完了したら画面を閉じる
  Close;
  // ModalResultの設定により、一覧照会画面側で再検索する
  ModalResult := mrOk;
end;
```

以上で、本稿で題材としたユーザーマスタファイルを参照、更新するアプリケーションの作成は完了である。本稿で紹介した「継承」を活用したアプリケーション開発は、今後のシステム開発においても色々な局面で応用が利くので、是非参考にして頂きたい。

## 6.まとめ

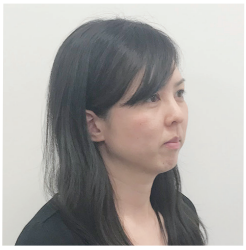
本稿では、最新のDelphi/400 11 Alexandriaを用いたアプリケーション開発方法について紹介してきた。
弊社は2021年より「Migaro. 技術Tips」と題した技術トピックを毎月掲載している。Delphi/400に関するTips記事においては、過去のメルマガに掲載した古いトピックを最新化して再掲したものや、最新のトレンドを組み入れた技術トピックまで、その内容は多岐にわたる。これら各トピックの内容を取り入れる事により、アプリケーション開発の幅は更に広がるであろう。

これからDelphi/400を使用して本格的な開発を開始するという方は、ぜひ本稿を参考にシステム開発の基盤を固め、「Migaro.技術Tips」も併せて参考にして頂ければ幸いである。

# Delphi/400

# Delphi/400アプリケーション向け開発支援ツールの作成方法

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
2. オブジェクトの自動生成（Excel→DDS変換）
3. オブジェクトのドキュメント化（DDS→Excel変換）
4. SQL文、ParamByName・FieldByNameメソッドの自動生成
5. TFDMemTableのフィールド定義の自動生成
6. さいごに

## 1.はじめに

アプリケーション開発や、関連するドキュメントの作成には、多大なリソースが必要とされる。リソース不足解消の手段として、開発支援ツールを導入している企業も多いだろう。開発支援ツールを導入することで、手軽にアプリケーション開発が可能となる。例えば、ボタンひとつで、データ取得ロジックの作成やドキュメントの作成が可能となるため、アプリケーション開発の効率化が期待される。また、プログラミングに関する知識や技術が不足しているユーザーでもアプリケーション開発に携われるため、リソース不足の解消にも繋がるだろう。

本稿では、Delphi/400アプリケーションに特化した開発支援ツールの作成例を以下の手順で紹介する。

【第2章】ExcelドキュメントからDDSソース＆オブジェクトを作成する方法
【第3章】DDSソース＆オブジェクトからExcelドキュメントを作成する方法
【第４章】ＤＤＳソース＆オブジェクトからＳＱＬ文やParamByNameメソッド、FieldByNameメソッドを作成する方法
【第5章】DDSソース＆オブジェクトからTFDMemTableのフィールド定義を作成する方法

IBMiの操作に不慣れな方や、Delphi/400の開発を始めたばかりの方でも、本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。

なお、本サンプルのＤｅｌｐｈｉ/４００バージョンは11 Alexandriaを使用する。

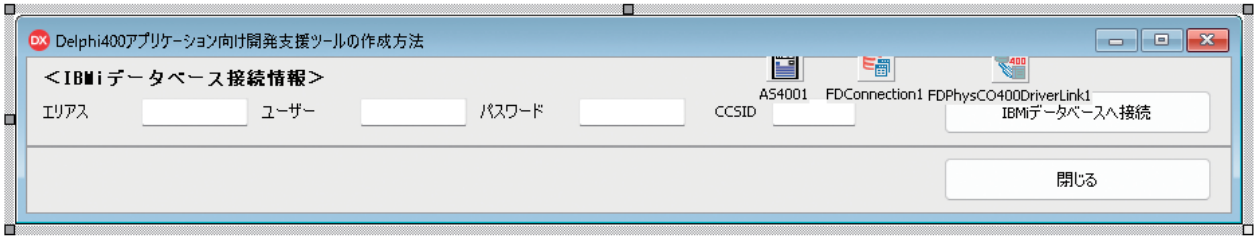
## 2.オブジェクトの自動生成（Excel→DDS変換）

本章では、Excelドキュメントで作成されたファイルレイアウト情報から、IBMiデータベース上にDDSソース並びにオブジェクトを自動生成する方法を紹介する。
IBMiデータベース上にDDSソースを使用してオブジェクトを作成する場合、①ライブラリを作成、②DDSソース格納用のオブジェクトを作成、③DDSソースを作成、④コンパイルの4ステップが必要となる。
本章で紹介する機能は、①Excelドキュメントでファイルレイアウトを作成、②Delphi/400アプリケーション上で必要情報を入力、③ボタンをクリックの3ステップで完了となる。ステップ数としては大きな違いはないが、IBMiの操作に不慣れなユーザーには、IBMi上でDDSソースを作成＆コンパイルをするより、Excelドキュメント＆Delphi/400アプリケーション上で必要な情報を入力すればオブジェクトが作成できる方が、作業の効率化に繋がるだろう。

### 2-1.IBMiデータベースへの接続

接続先情報を設定するためのTEdit、データベース接続で使用するTAS400、TFDConnection、TFDPhysCO400DriverLink、並びにTLabelやTButtonを画面に配置する【図1】。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 1　IBMi接続：コンポーネントの配置**



「IBMiデータベースへ接続」ボタンのOnClick処理で、
IBMiデータベースへ接続する処理【ソース1】を記述する。

**ソース 1**

**IBMiデータベース接続処理**

```
// AS400の接続
with AS4001 do
begin
  Active := False;
  PLUAlias     := edtAlias.Text;  // エリアス
  Userid       := edtUser.Text;   // ユーザー
  PWD          := edtPass.Text;   // パスワード
  Active := True;

  // CCSID変更
  RemoteCmd('CHGJOB CCSID(' + edtCCSID.Text + ')');
end;

// FDConnectionの接続
with FDConnection1 do
begin
  Connected := False;
  Params.Values['Database']     := edtAlias.Text;  // エリアス
  Params.Values['User_Name']    := edtUser.Text;   // ユーザー
  Params.Values['Password']     := edtPass.Text;   // パスワード
  Connected := True;
end;
```

## 2-2.Excelフォーマットの出力

本章でのオブジェクト自動生成機能は、ファイル情報がExcelドキュメントに記述される想定とする。物理ファイル用のシート、論理ファイル用のシートを定義したExcelオブジェクトの雛形ファイル【図2】を事前に作成しておく。物理ファイルは物理ファイル単位、論理ファイルは基となる物理ファイル単位でシートが作成される想定とする。

**図 2　ファイル自動生成：Excelフォーマットファイル**

物理ファイル用　※1シート目の情報

| ファイルID | ファイル名 | レコード様式 | I：ユニーク | 備考 |
|---|---|---|---|---|
| | | | | |

| No | フィールドID | フィールド名 | Key順 | A：昇順 D：降順 | 属性 | 桁数 | 小数 | 備考 |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | |

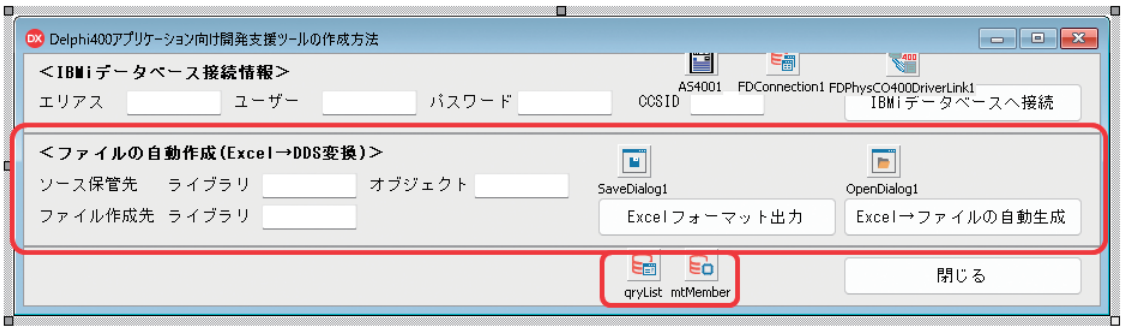
論理ファイル用　※2シート目の情報

| 物理ファイルID | 物理ファイルレコード様式 | 備考 |
|---|---|---|
| | | |

| No | 論理ファイルID | 論理ファイル名 | I：ユニーク | フィールドID | A：昇順 D：降順 | S：選択 O：除外 | AND/OR | EQ/NE/GT/GE/ LE/LT | 条件 | 備考 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | |

2-1.で作成したサンプルプログラムに、ソース保管先、オブジェクト作成先の情報を設定するためのTEdit、ファイル保存を行うためのTSaveDialog、ファイル選択を行うためのTOpenDialog、データベース接続で使用するTFDQueryとTFDMemTable、並びにTLabelやTButtonを画面に追加配置する【図3】。

**図 3　ファイル自動生成:コンポーネントの配置**




「Excelフォーマット出力」ボタンのOnClick処理で、【図2】の雛形ファイルをTSaveDialogで指定の保管先へ保存する処理を記述する【ソース2】。

**ソース 2**

**btnExcelFormatClick（「Excelフォーマット出力」ボタン押下時処理）**

```
// コピー元ファイル
sFrom := ExtractFilePath(Application.ExeName) + 'Template';
sFrom := IncludeTrailingPathDelimiter(sFrom)  + 'Template.xlsx';

// コピー先ファイル
sTo := SaveDialog1.FileName;

// フォーマットファイルを、保存ダイアログで指定した場所にコピー
bCopy := CopyFile(PChar(sFrom),   // コピー元
                  PChar(sTo),     // コピー先
                  False);         // True：同名ファイル存在時コピーしない、False：上書き

// エラー
if (not bCopy) then
begin
  ShowMessage('ファイルの保存に失敗しました。');
  Abort;
end;
```

## 2-3.オブジェクトの自動生成

「Excel→ファイルの自動生成」ボタンのOnClick処理で、オブジェクトの自動生成処理を記述する。処理の流れは、①ライブラリ作成、②DDSソース格納用のオブジェクト作成、③DDSソース作成、④コンパイルとなる。

①ライブラリ作成

ライブラリの作成には、IBMi上のコマンド「CRTLIB」を実行する必要がある。TAS400コンポーネントのRemoteCmdメソッドを利用し、コマンドを実行する。作成前にはコマンド「CHKOBJ」を実行し、ライブラリが存在しない場合のみ作成処理を行う。「CRTLIB」コマンドが実行できない場合、例外エラーが生成されるため、処理中断とする**【ソース3】**。

**ソース 3**

**CHKOBJ:存在チェック**

```
AS4001.RemoteCmd('CHKOBJ '
                 + 'OBJ(' + edtEtoD_SrcLib.Text + ') '    // ライブラリー名
                 + 'OBJTYPE(*LIB)');                      // オブジェクトタイプ：ライブラリ
```

**CRTLIB:ライブラリの作成　※例外エラー時のみ実行**

```
AS4001.RemoteCmd('CRTLIB '
                 + 'LIB(' + edtEtoD_SrcLib.Text + ') '    // ライブラリー名
                 + 'TEXT(''テストライブラリ'')' );        // テキスト
```

②DDSソース格納用のオブジェクト作成

オブジェクトの作成には、コマンド「CRTSRCPF」を実行する。作成前にはコマンド「CHKOBJ」を実行し、オブジェクトが存在しない場合のみ作成処理を行う。「CRTSRCPF」コマンドが例外エラーの場合、処理中断とする**【ソース4】**。オブジェクトのメンバー一覧(ファイル一覧)を、コマンド「DSPFD」を実行してQTEMPに出力し、TFDMemTableに内部保持しておく**【ソース5】**。

**ソース 4**

**CHKOBJ:存在チェック**

```
AS4001.RemoteCmd('CHKOBJ '
                 + 'OBJ(' + sSrcObj + ') '    // DDSソース保管先
                 + 'OBJTYPE(*FILE)');         // オブジェクトタイプ：ファイル
```

**CRTSRCPF:オブジェクトの作成　※例外エラー時のみ実行**

```
AS4001.RemoteCmd('CRTSRCPF '
                 + 'FILE(' + sSrcObj + ') '          // ライブラリ名/オブジェクト名
                 + 'IGCDTA(*YES) '                   // ユーザー指定のDBCSデータ
                 + 'TEXT(''DDSソース'') '            // テキスト
                 + 'CCSID(' + edtCCSID.Text + ')');  // 文字コード
```

**ソース 5**

**オブジェクトのメンバー一覧取得**

```
// メンバー一覧の取得
AS4001.RemoteCmd('DSPFD '
                + 'FILE(' + sSrcObj + ') '      // ライブラリ名/ファイル名
                + 'TYPE(*MBRLIST) '             // 情報のタイプ：メンバー一覧
                + 'OUTPUT(*OUTFILE) '           // 出力：ファイル
                + 'OUTFILE(QTEMP/MEMBER) '      // 出力ファイル
                + 'OUTMBR(*FIRST *REPLACE)');   // 出力メンバー：FIRST&置き換え

// メンバー一覧の読み込み
with qryList do
begin
  Close;
  SQL.Clear;
  SQL.Add('SELECT MLNAME');
  SQL.Add('FROM   QTEMP/MEMBER');
end;

// データ取得
qryList.Open;

try
  // MemTableに取得したデータを追加
  mtMember.Close;
  mtMember.AppendData(qryList, False);
finally
  qryList.Close;
end;
```

③DDSソース作成

(1)TOpenDialogで指定のExcel形式のファイル情報から、DDSソースを作成する。Excel形式のファイル情報の読み込みはOLEを利用し、シート数分、オブジェクト作成の処理を繰り返す。OLEの使用方法については、過去のミガロ.テクニカルレポート「OLEを利用したExcel出力のパフォーマンス向上手法」で記載があるため、本稿での説明は割愛させて頂く。

https://www.migaro.co.jp/tr/no11/tech/11_01_01.pdf

(2)②で取得したメンバー一覧にExcelドキュメント情報で指定のファイル名が存在しない場合、コマンド「ADDPFM」を実行してメンバーを追加する。メンバーが存在する場合、既に作成済みのDDSソースとなるため、次のシートの処理へ進む**【ソース6】**。

## ソース 6

### メンバー追加

```
// メンバーが存在しない場合、メンバーを追加
if (not mtMember.Locate('MLNAME', VarArrayOf([AFileID]), [])) then
begin
  AS4001.RemoteCmd('ADDPFM '
                 + 'FILE(' + sSrcObj + ') '          // ライブラリ名/オブジェクト名
                 + 'MBR(' + AFileID + ') '           // メンバー：ファイル名
                 + 'TEXT(''' + AFileName + ''') '    // テキスト
                 + 'SRCTYPE(' + AType + ')');        // ソース仕様タイプ
end
// メンバーが存在する場合、エラー
else
begin
  AErrList.Add('・【' + ASheetName + '】シート：既に作成済みのファイルです。' + sErrFile);
end;
```

エラーは内部保持し、まとめて
ShowMessageする

(3)Excel形式のファイル情報を基に、物理ファイル・論理ファイルそれぞれの形式に合った形のDDSソースを作成し、1ファイル単位でTStringListへ内部保持する。DDSソースの内部保持が完了すれば、(2)で追加したIBMiデータベース上のソースファイルへ内部保持値を追加する**【ソース7】**。

## ソース 7

### ソースファイルへ内部保持値を追加

```
// OVRDBF
AS4001.RemoteCmd('OVRDBF '
                 + 'FILE(' + AFileID + ') '     // 一時変更中のファイル名
                 + 'TOFILE(' + sSrcObj + ') '   // ライブラリ名/オブジェクト名
                 + 'MBR(' + AFileID + ') '      // メンバー：ファイル名
                 + 'OVRSCOPE(*JOB)');           // 有効範囲：ジョブ

try
  // DDSソースファイルに、内部保持したDDSソース情報を追加
  with qryList do
  begin
    for i := 0 to ADdsSource.Count - 1 do
    begin
      Close;
      SQL.Clear;
      SQL.Add('INSERT INTO ' + AFileID + '(SRCSEQ, SRCDAT, SRCDTA) VALUES (:SRCSEQ, :SRCDAT, :SRCDTA)');
      ParamByName('SRCSEQ').AsCurrency := i + 1;          // SEQ
      ParamByName('SRCDAT').AsInteger  := iDate;          // 日付
      ParamByName('SRCDTA').AsString   := ADdsSource[i];  // ソース
      ExecSQL;
    end;
  end;
finally
  // DLTOVR
  AS4001.RemoteCmd('DLTOVR FILE(' + AFileID + ') LVL(*JOB)');
end;
```

④コンパイル

DDSソース作成完了後、コンパイルを行う。物理ファイルはコマンド「CRTPF」、論理ファイルはコマンド「CRTLF」を実行し、画面項目「ファイル作成先」で指定のライブラリへDDSソースをコンパイルする**【ソース8】**。

**ソース 8**

**コンパイル:物理ファイル　※内部保持したファイル数分コンパイルする**

```
AS4001.RemoteCmd('CRTPF '
                 + 'FILE(' + edtEtoD_Lib.Text + '/' + sFilePF + ') '   // コンパイル先
                 + 'SRCFILE(' + sSrcObj + ') '                         // ソース保管場所
                 + 'SRCMBR(' + sFilePF + ') '                          // ファイルID
                 + 'MAXMBRS(*NOMAX) '                                  // メンバーの最大数
                 + 'SIZE(*NOMAX)');                                    // 初期レコード数
```

**コンパイル:論理ファイル　※内部保持したファイル数分コンパイルする**

```
AS4001.RemoteCmd('CRTLF '
                 + 'FILE(' + edtEtoD_Lib.Text + '/' + sFilePF + ') '   // コンパイル先
                 + 'SRCFILE(' + sSrcObj + ') '                         // ソース保管場所
                 + 'SRCMBR(' + sFilePF + ') '                          // ファイルID
                 + 'MAXMBRS(*NOMAX)');                                 // メンバーの最大数
```

以上で、Excelドキュメントに入力されたファイル情報から、DDSソース、オブジェクトを自動生成するプログラムは完成である。上記プログラムを実行し、自動生成されたDDSソース、オブジェクトを確認する。取り込むExcelドキュメントは、**【図4】**とし、Excelに記載の通り、IBMiデータベース上にDDSソース、オブジェクトが自動生成されていることが分かる**【図5～6】**。

**図 4　ファイル自動生成:実行結果(ファイルレイアウト)**

プログラムで読み込むExcelファイルのファイルレイアウト：物理ファイル　※1シート目の情報

| ファイルID | ファイル名 | レコード様式 | ﾕﾆｰｸ | 備考 |
|---|---|---|---|---|
| TEST1 | テストファイル１ | RTEST1 | 1 | |

| No. | フィールドID | フィールド名 | Key順 | A：昇順 D：降順 | 属性 | 桁数 | 小数 | 備考 |
|---|---|---|---|---|---|---|---|---|
| 1 | DATA1_1 | 半角フィールド | 1 | A | A | 10 | | |
| 2 | DATA1_2 | 全角フィールド | 2 | D | O | 20 | | |
| 3 | DATA1_3 | 数値フィールド１ | | | S | 10 | 0 | |
| 4 | DATA1_4 | 数値フィールド２ | | | P | 10 | 2 | |

プログラムで読み込むExcelファイルのファイルレイアウト：論理ファイル　※2シート目の情報

| 物理ファイルID | 物理ファイルレコード様式 | 備考 |
|---|---|---|
| TEST1 | RTEST1 | テストファイル１ |

| No. | 論理ファイルID | 論理ファイル名 | ﾕﾆｰｸ | フィールドID | A：昇順 D：降順 | S：選択 O：除外 | AND/OR | EQ/NE/GT/GE/ LE/LT | 条件 | 備考 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | TEST1L01 | テストファイル１　Ｌ０１ | 1 | DATA1_3 | A | | | | | |
| | | | | DATA1_4 | D | | | | | |
| | | | | DATA1_1 | | S | | EQ | '1' | データ１="1" |

## 図 5　ファイル自動生成：実行結果（物理ファイル）

IBMiデータベース上に自動生成されたDDSソース

```
0001.00       A*************************************************************************  230831
0002.00       A*    テストファイル１    ／  TEST1                                        230831
0003.00       A*     CREATE : 2023/08/31   作成者 :  NTSUJI                             230831
0004.00       A*     UPDATE : YYYY/MM/DD   更新者 :                                     230831
0005.00       A*************************************************************************  230831
0006.00       A                                     UNIQUE                               230831
0007.00       A          R RTEST1                   TEXT(' テストファイル１ ')           230831
0008.00       A*                                                                        230831
0009.00       A            DATA1_1       10A       COLHDG(' 半角フィールド ')           230831
0010.00       A            DATA1_2       20O       COLHDG(' 全角フィールド ')           230831
0011.00       A            DATA1_3       10S 0     COLHDG(' 数値フィールド１ ')         230831
0012.00       A            DATA1_4       10P 2     COLHDG(' 数値フィールド２ ')         230831
0013.00       A*                                                                        230831
0014.00       A* キー情報                                                               230831
0015.00       A          K DATA1_1                                                      230831
0016.00       A          K DATA1_2                  DESCEND                             230831
```

IBMiデータベース上に自動生成されたファイル

| 物理ファイル | NTSUJILIB/TEST1 | 様式名 | RTEST1 | レコード長 | 46 |
|---|---|---|---|---|---|
| 様式記述 | テストファイル１ | | | | |

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 |
|---|---|---|---|---|---|---|---|
| _ | DATA1_1 | 10 | A | 1 ANN | 1 | 10 | 半角フィールド |
| _ | DATA1_2 | 20 | O | 2 DNN | 11 | 30 | 全角フィールド |
| _ | DATA1_3 | 10 0 | S | | 31 | 40 | 数値フィールド１ |
| _ | DATA1_4 | 10 2 | P | | 41 | 46 | 数値フィールド２ |

## 図 6　ファイル自動生成：実行結果（論理ファイル）

IBMiデータベース上に自動生成されたDDSソース

```
0001.00       A*************************************************************************  230831
0002.00       A*    テストファイル１   ＬＯ１ (LF)   ／   TEST1                          230831
0003.00       A*     CREATE : 2023/08/31   作成者 :  NTSUJI                             230831
0004.00       A*     UPDATE : YYYY/MM/DD   更新者 :                                     230831
0005.00       A*************************************************************************  230831
0006.00       A                                     UNIQUE                               230831
0007.00       A          R RTEST1                   PFILE(TEST1)                         230831
0008.00       A                                     TEXT(' テストファイル１   ＬＯ１ ')   230831
0009.00       A*                                                                        230831
0010.00       A* キー情報                                                               230831
0011.00       A          K DATA1_3                                                      230831
0012.00       A          K DATA1_4                  DESCEND                             230831
0013.00       A*                                                                        230831
0014.00       A* セレクト&オミット情報                                                  230831
0015.00       A          S DATA1_1                  COMP(EQ '1')                        230831
```

IBMiデータベース上に自動生成されたファイル

| 論理ファイル | NTSUJILIB/TEST1LO1 | 様式名 | RTEST1 | レコード長 | 46 |
|---|---|---|---|---|---|
| 様式記述 | テストファイル１ | | | | |

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 |
|---|---|---|---|---|---|---|---|
| _ | DATA1_1 | 10 | A | | 1 | 10 | 半角フィールド |
| _ | DATA1_2 | 20 | O | | 11 | 30 | 全角フィールド |
| _ | DATA1_3 | 10 0 | S | 1 ASN | 31 | 40 | 数値フィールド１ |
| _ | DATA1_4 | 10 2 | P | 2 DSN | 41 | 46 | 数値フィールド２ |

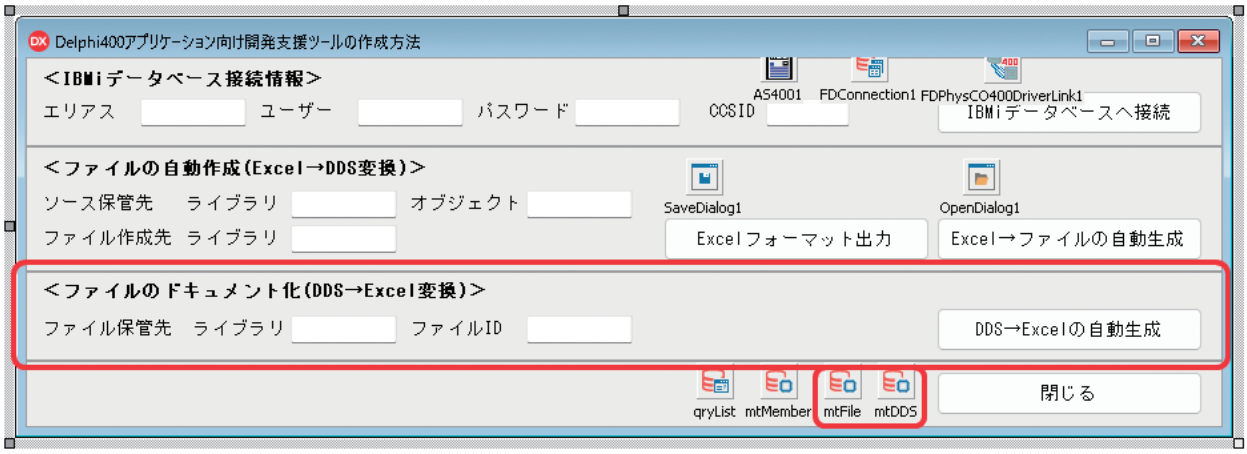
## 3.オブジェクトのドキュメント化(DDS→Excel変換)

本章では、IBMiデータベース上に作成されたDDSソース及びオブジェクトから、ファイルレイアウト(Excelドキュメント)を自動生成する方法を紹介する。
IBMiデータベース上のオブジェクトのドキュメント化が必要な場合、DDSソースを確認、もしくは、オブジェクトの情報を確認し、フィールド情報を資料に記載する必要がある。
本章で紹介する機能は、オブジェクト保管先の情報を入力すればDDS→Excelの自動変換ができるため、作業の効率化に繋がるだろう。

### 3-1.コンポーネントの配置

第2章で作成したサンプルプログラムに、ソース保管先・ファイル保管先の情報を設定するためのTEdit、データベース接続で使用するTFDMemTable、並びにTLabelやTButtonを画面に追加配置する【図7】。

**図 7 ファイルドキュメント化:コンポーネントの配置**




「DDS→Excelの自動生成」ボタンのOnClick処理で、ドキュメントの自動生成処理を記述する。自動生成するExcelドキュメントの雛形は、【図2】の雛形ファイルとする。

### 3-2.ファイル一覧の取得

コマンド「DSPOBJD」を実行し、画面項目「ファイル保管先」で指定のファイル一覧(ライブラリ内のオブジェクト一覧)および各オブジェクト情報(ファイルID、テキスト、属性、DDSソース保管先)をTFDMemTableに内部保持する【ソース9】。取得したファイル一覧の中に画面で指定のファイルIDが存在しない場合、処理中断とする。画面でファイルIDの指定がない場合、全ファイルを対象とするためチェック不要とする。

## ソース 9

### ファイル一覧の取得

```
// ファイル一覧の取得
AS4001.RemoteCmd('DSPOBJD '
                 + 'OBJ(' + edtDtoE_SrcLib.Text +'/*ALL) '  // DDSソース保管先
                 + 'OBJTYPE(*FILE) '                        // オブジェクトタイプ：ファイル
                 + 'OUTPUT(*OUTFILE) '                      // 出力：ファイル
                 + 'OUTFILE(QTEMP/FILELIST)');              // 出力ファイル

// ファイル一覧の読み込み
with qryList do
begin
  Close;
  SQL.Clear;
  SQL.Add('SELECT ODOBNM,');  // ファイルID
  SQL.Add(       'ODOBTX,');  // テキスト
  SQL.Add(       'ODOBAT,');  // 属性(PF/LF)
  SQL.Add(       'ODSRCL,');  // ソース・ファイル・ライブラリー
  SQL.Add(       'ODSRCF,');  // ソース・ファイル名
  SQL.Add(       'ODSRCM');   // ソース・ファイル・メンバー
  SQL.Add('FROM   QTEMP/FILELIST');
  SQL.Add('WHERE (ODOBAT = ''PF'' OR ODOBAT = ''LF'')');  // PF/LFのみを対象
  // ファイルID<>ブランクの場合、対象ファイルのみを取得
  if (edtDtoE_SrcFile.Text <> '') then
  begin
    SQL.Add('AND ODOBNM = :ODOBNM');
    ParamByName('ODOBNM').AsString := edtDtoE_SrcFile.Text;
  end;
end;

// データ取得
qryList.Open;

try
  // MemTableに取得したデータを追加
  mtFile.Filtered := False;
  mtFile.Close;
  mtFile.AppendData(qryList, False);
  mtFile.IndexFieldNames := 'ODOBNM';
finally
  qryList.Close;
end;
```

### 3-3.DDS→Excel変換

第2章の「Excelフォーマット出力」ボタンのOnClick処理を呼び出して雛形ファイルを保存し、DDS→Excel変換の処理を記述する。物理ファイルか論理ファイルかで処理のタイミングが少し違うが、処理の流れは、①DDSソース取得、②Excel雛形ファイルのシートコピー、③DDSソースの読み込み、④Excelドキュメント保存となる。

①DDSソース取得

コマンド「CHKOBJ」を実行してDDSソースが存在する場合、TFDQueryを利用してDDSソースを取得する**【ソース10】**。DDSソースの保管先は、3-2.で取得した値とし、ソースが存在しない場合、処理中断とする。取得したDDSソースはTFDMemTableに内部保持する。

**ソース 10**

**CHKOBJ:DDSソース存在チェック**

```
AS4001.RemoteCmd('CHKOBJ '
                 + 'OBJ(' + ASrcObj + ') '    // DDSソース保管先
                 + 'OBJTYPE(*FILE) '          // オブジェクトタイプ：ファイル
                 + 'MBR(' + ASrcMem + ')');   // メンバー：ファイル名
```

**DDSソースの取得**

```
// OVRDBF
AS4001.RemoteCmd('OVRDBF '
                 + 'FILE(' + AFileID + ') '    // 一時変更中のファイル名
                 + 'TOFILE(' + ASrcObj + ') '  // ライブラリ名/オブジェクト名
                 + 'MBR(' + AFileID + ') '     // メンバー：ファイル名
                 + 'OVRSCOPE(*JOB)');          // 有効範囲：ジョブ

try
  // DDSソースの読み込み
  with qryList do
  begin
    Close;
    SQL.Clear;
    SQL.Add('SELECT * FROM ' + AFileID);
  end;

  // データ取得
  qryList.Open;

  try
    // MemTableに取得したデータを追加
    mtDDS.Close;
    mtDDS.AppendData(qryList, False);
  finally
    qryList.Close;
  end;
finally
  // DLTOVR
  AS4001.RemoteCmd('DLTOVR FILE(' + AFileID + ') LVL(*JOB)');
end;
```

②Excel雛形ファイルのシートコピー
保存したExcelドキュメントに対してシートのコピーを行う
**【ソース11】**。

**ソース 11**

**Excelファイルのシートのコピー**

```
// シートをコピー
AWBook.Worksheets[iSheetNo].Copy(After := AWBook.Worksheets[AWBook.Sheets.Count]);
AWBook.ActiveSheet.Name := AFileID;  // シート名

// 対象シートを選択
ASheet := AWBook.Sheets[AFileID];
ASheet.Activate;

// 5～53行目を削除
ASheet.Range['A5', 'A53'].EntireRow.Delete;
```

③DDSソースの読み込み
①で内部保持したDDSソースを1行ずつ読み込み、ファイル情報とフィールド情報を内部保持する。

④Excelドキュメント保存
③で内部保持した情報をExcelドキュメントへセットする。不要になった雛形シート(1～2シート目)を削除後、上書き保存する。

以上で、DDSソース及びオブジェクトから、Excelドキュメントのファイルレイアウトを自動生成するプログラムは完成である。上記プログラムを実行し、自動生成されたExcelドキュメントを確認する。変換対象は第2章で作成したファイル**【図5～6】**とし、ファイル・フィールド情報通りにExcel変換されていることが分かる**【図8】**。

**図 8　ファイルドキュメント化:実行結果**

自動生成されたExcelファイルのファイルレイアウト：物理ファイル　※1シート目の情報

| ファイルID | ファイル名 | レコード様式 | ｜:ユニーク | 備考 |
|---|---|---|---|---|
| TEST1 | テストファイル１ | RTEST1 | 1 | |

| No. | フィールドID | フィールド名 | Key順 | A：昇順<br>D：降順 | 属性 | 桁数 | 小数 | 備考 |
|---|---|---|---|---|---|---|---|---|
| 1 | DATA1_1 | 半角フィールド | 1 | A | A | 10 | | |
| 2 | DATA1_2 | 全角フィールド | 2 | D | O | 20 | | |
| 3 | DATA1_3 | 数値フィールド１ | | | S | 10 | 0 | |
| 4 | DATA1_4 | 数値フィールド２ | | | P | 10 | 2 | |

自動生成されたExcelファイルのファイルレイアウト：論理ファイル　※2シート目の情報

| 物理ファイルID | 物理ファイルレコード様式 | 備考 |
|---|---|---|
| TEST1 | RTEST1 | |

| No. | 論理ファイルID | 論理ファイル名 | ｜:ユニーク | フィールドID | A：昇順<br>D：降順 | S：選択<br>O：除外 | AND/OR | EQ/NE/GT/GE/<br>LE/LT | 条件 | 備考 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | TEST1L01 | テストファイル１　Ｌ０１ | 1 | DATA1_3 | A | | | | | |
| | | | | DATA1_4 | D | | | | | |
| | | | | DATA1_1 | | S | | EQ | '1' | |

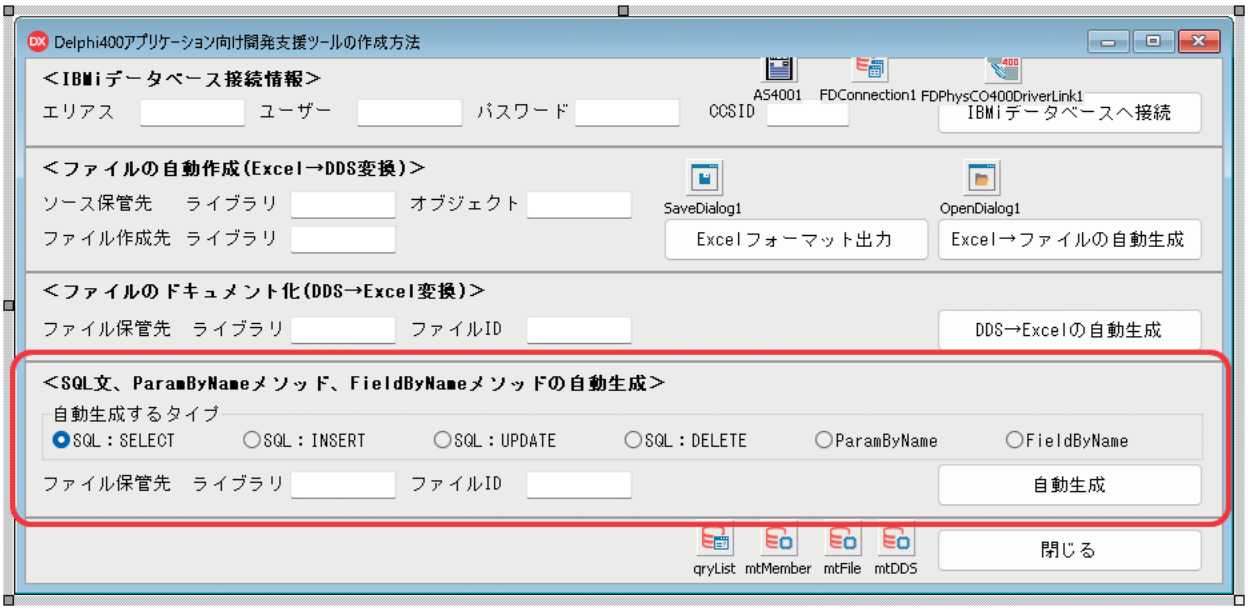
## 4.SQL文、ParamByName・FieldByNameメソッドの自動生成

本章では、IBMiデータベース上に作成されたDDSソース及びオブジェクトから、SQL文、ParamByName・FieldByNameメソッドを自動生成する方法を紹介する。SQL文の実行、Delphi/400でParamByName・FieldByNameメソッドを使用する場合、DDSソースを確認、もしくは、オブジェクトの情報を確認する必要がある。本章で紹介する機能は、オブジェクト保管先の情報を入力すれば自動生成できるため、作業の効率化に繋がるだろう。

### 4-1.コンポーネントの配置

第3章で作成したサンプルプログラムに、自動生成するタイプを選択するためのTRadioGroup、ファイル保管先の情報を設定するためのTEdit、並びにTLabelやTButtonを画面に追加配置する【図9】。

**図 9　SQL文の自動生成:コンポーネントの配置**

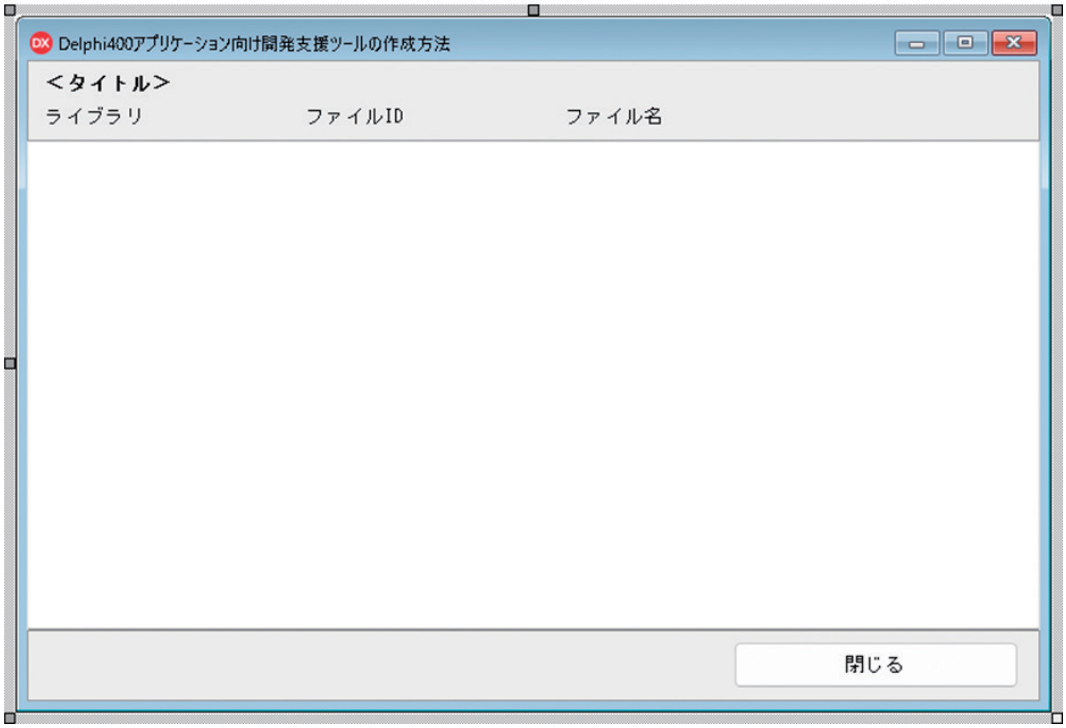


### 4-2.自動生成

「自動生成」ボタンのOnClick処理で、SQL文、ParamByName・FieldByNameメソッドを自動生成する処理を記述する。処理の流れは、①フィールド情報取得、②SQL文の自動生成、③ParamByName、FieldByNameメソッドの自動生成となる。

まず、自動生成した結果を表示するためのサブ画面を新規作成する。結果を表示するためのTMemo、並びにTLabelやTButtonを画面に配置する**【図10】**。

**図 10　SQL文の自動生成：サブ画面の作成**




### ①フィールド情報取得

コマンド「DSPFFD」を実行し、画面項目「ファイル保管先」で指定のファイル情報（ファイル名）、フィールド情報（フィールドID、フィールド名、桁数）をリストに内部保持、TFDQueryを利用してファイルを参照し、フィールドタイプをリストに内部保持する**【ソース12】**。

**ソース 12**

**フィールド情報の取得**

```
// コマンドを発行し、一時ファイルに結果出力
AS4001.RemoteCmd('DSPFFD '
                 + 'FILE(' + sLib + '/' + sFile + ') '  // ファイル
                 + 'OUTPUT(*OUTFILE) '                  // 出力：ファイル
                 + 'OUTFILE(QTEMP/' + sFile + ') '      // 出力ファイル
                 + 'OUTMBR(*FIRST *REPLACE)');          // 出力メンバー：FIRST&置き換え

// フィールド情報の読み込み
with qryList do
begin
```

```
  Close;
  SQL.Clear;
  SQL.Add('SELECT WHTEXT,');  // ファイル名
  SQL.Add(       'WHFLDI,');  // フィールドID
  SQL.Add(       'WHFTXT,');  // フィールド名
  SQL.Add(       'WHFLDB');   // 桁数
  SQL.Add('FROM   QTEMP/' + sFile);
  SQL.Add('ORDER  BY WHFOBO');
  Open;

  try
    AFileName := FieldByName('WHTEXT').AsString;  // ファイル名

    // リストに取得情報を保管
    while not Eof do
    begin
      AFieldIDList.Add(FieldByName('WHFLDI').AsString);    // フィールドID
      AFieldNameList.Add(FieldByName('WHFTXT').AsString);  // フィールド名
      AFieldLenList.Add(FieldByName('WHFLDB').AsString);   // 桁数
      Next;
    end;
  finally
    qryList.Close;
  end;
end;

// フィールドタイプの取得
with qryList do
begin
  Close;
  SQL.Clear;
  SQL.Add('SELECT * FROM ' + sLib + '/' + sFile);
  SQL.Add('FETCH FIRST 1 ROWS ONLY');
  Open;

  try
    // リストに取得情報を保管
    for i := 0 to FieldCount - 1 do
    begin
      case Fields[i].DataType of
        ftInteger : AFieldTypeList.Add('Integer');
        ftFloat   : AFieldTypeList.Add('Float');
        ftCurrency: AFieldTypeList.Add('Currency');
        else        AFieldTypeList.Add('String');
      end;
      Next;
    end;
  finally
    qryList.Close;
  end;
end;
```

②SQL文の自動生成

画面項目「自動生成するタイプ」で「SQL：SELECT」、「SQL：INSERT」、「SQL：UPDATE」、「SQL：DELETE」のいずれかを選択時、②で内部保持した情報を基にSQL文を自動生成する**【ソース13】**。WHERE句やSET句で使用する条件（右辺）、VALUES句で使用する値については、Delphi/400ソース上で使用することを想定しバインド変数とする。自動生成した内容は、①で作成したサブ画面に表示する。

③ParamByName、FieldByNameメソッドの自動生成

画面項目「自動生成するタイプ」で「ParamByName」、「FieldByName」のいずれかを選択時、②で内部保持した情報を基にParamByName、FieldByNameメソッドを自動生成する**【ソース14】**。自動生成した内容は、①で作成したサブ画面に表示する。

**ソース 13**

**SQL文の自動生成**

```
case iMode of
  1: slText.Add('SELECT');                          // SELECT
  2: slText.Add('INSERT INTO ' + sSQL_FileID + ' ('); // INSERT
  3: slText.Add('UPDATE '      + sSQL_FileID);       // UPDATE
  4: slText.Add('DELETE');                          // DELETE
end;

// UPDATE
if (iMode = 3) then
begin
  slText.Add('SET');
end;

// DELETE
if (iMode = 4) then
begin
  slText.Add('FROM ' + sSQL_FileID);
  slText.Add('');
end;

// フィールド情報(DELETE以外)
if (iMode <> 4) then
begin
  for i := 0 to slFieldID.Count - 1 do
  begin
    case iMode of
      1: sText := slFieldID[i];                       // SELECT
      2: sText := slFieldID[i];                       // INSERT
      3: sText := slFieldID[i] + ' = :' + slFieldID[i]; // UPDATE
    end;

    // 最終フィールドでない場合、カンマを付与
    if (i <> (slFieldID.Count - 1)) then
```

```
    begin
      sText := sText + ',';
    end;

    // 内部保持
    slText.Add(sText);

  end;
end;

case iMode of
  1: slText.Add(#13#10 + 'FROM ' + sSQL_FileID);  // SELECT
  2: slText.Add(#13#10 + ') VALUES (');           // INSERT
end;

// WHERE区を追加
case iMode of
  1: slText.Add(#13#10 + 'WHERE');  // SELECT
  3: slText.Add(#13#10 + 'WHERE');  // UPDATE
  4: slText.Add('WHERE');           // DELETE
end;

// フィールド情報
for i := 0 to slFieldID.Count - 1 do
begin
  case iMode of
    2:   sText := ':' + slFieldID[i];                    // INSERT
    else sText := slFieldID[i] + ' = :' + slFieldID[i];  // 上記以外
  end;

  // 最終フィールドでない場合、カンマ or " AND"を付与
  if (i <> (slFieldID.Count - 1)) then
  begin
    case iMode of
      2:   sText := sText + ',';     // INSERT
      else sText := sText + ' AND';  // 上記以外
    end;
  end;

  // 内部保持
  slText.Add(sText);
end;

// INSERTの場合、")"を追加
if (iMode = 2) then
begin
  slText.Add(')');
end;
```

## ソース 14

### ParamByName、FieldByNameの自動生成

```
for i := 0 to slFieldID.Count - 1 do
begin
  // フィールド情報
  case iMode of
    5: sText := 'ParamByName';
    6: sText := 'FieldByName';
  end;
  sText := sText + '('''' + slFieldID[i] + ''').As' + slFieldType[i] + ' := ';

  // フィールドタイプ
  // 文字
  if (slFieldType[i] = 'String') then
  begin
    sText := sText + '''';';
  end
  // 数値
  else
  begin
    sText := sText + '0;';
  end;

  // 内部保持
  slText.Add(sText);
end;
```

以上で、DDSソース及びオブジェクトから、SQL文、ParamByName・FieldByNameメソッドを自動生成するプログラムは完成である。上記プログラムを実行し、サブ画面を確認する。変換対象は第2章で作成したファイル【図5】とし、ファイル・フィールド情報通りの内容がサブ画面にセットされていることが分かる【図11～16】。

**図 11　SQL文の自動生成:実行結果(SQL文:SELECT)**

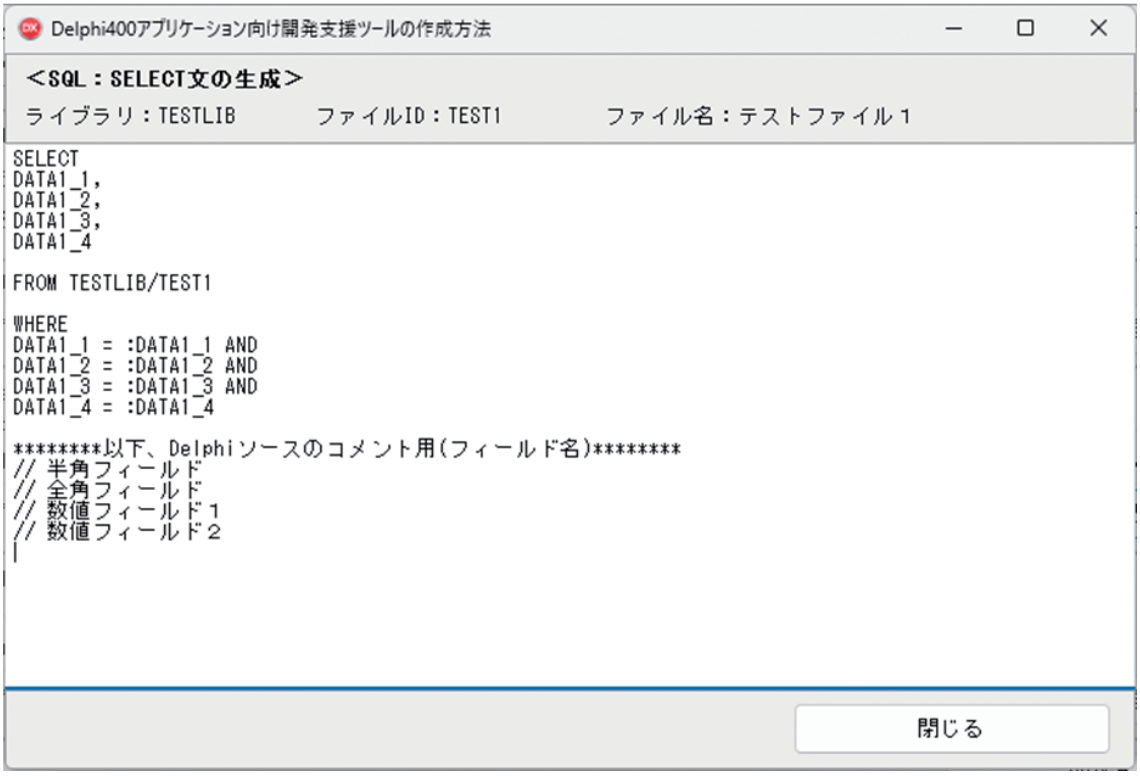
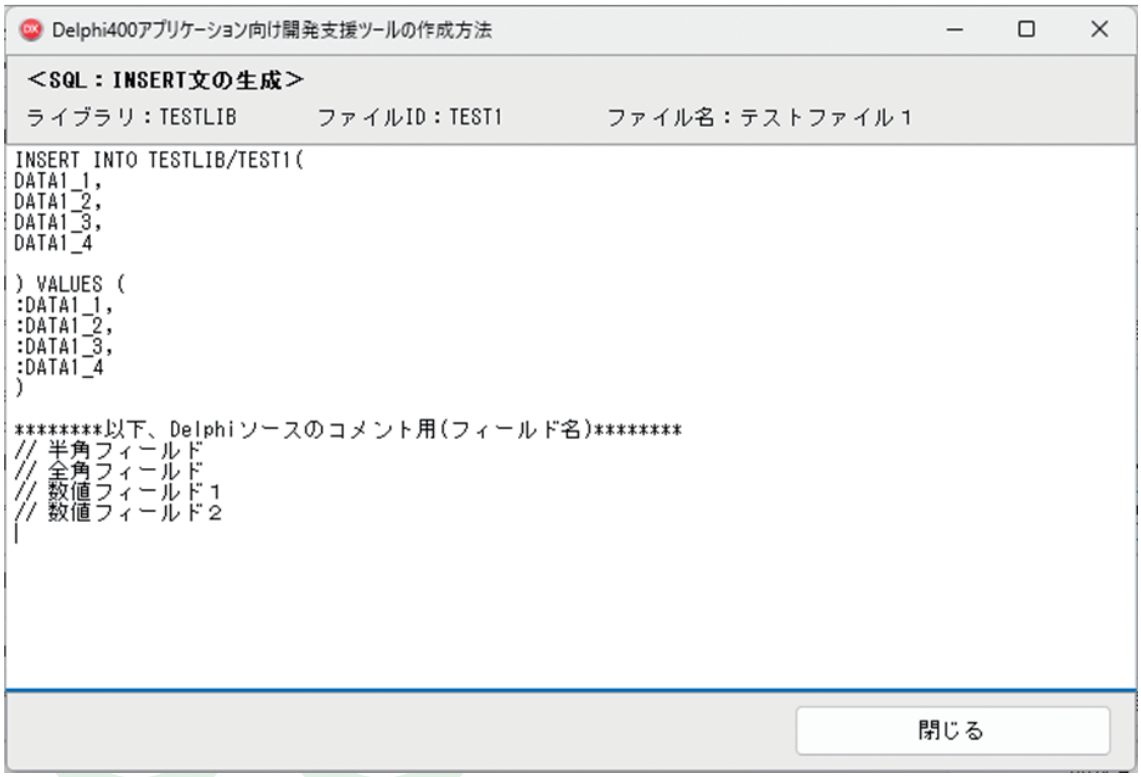
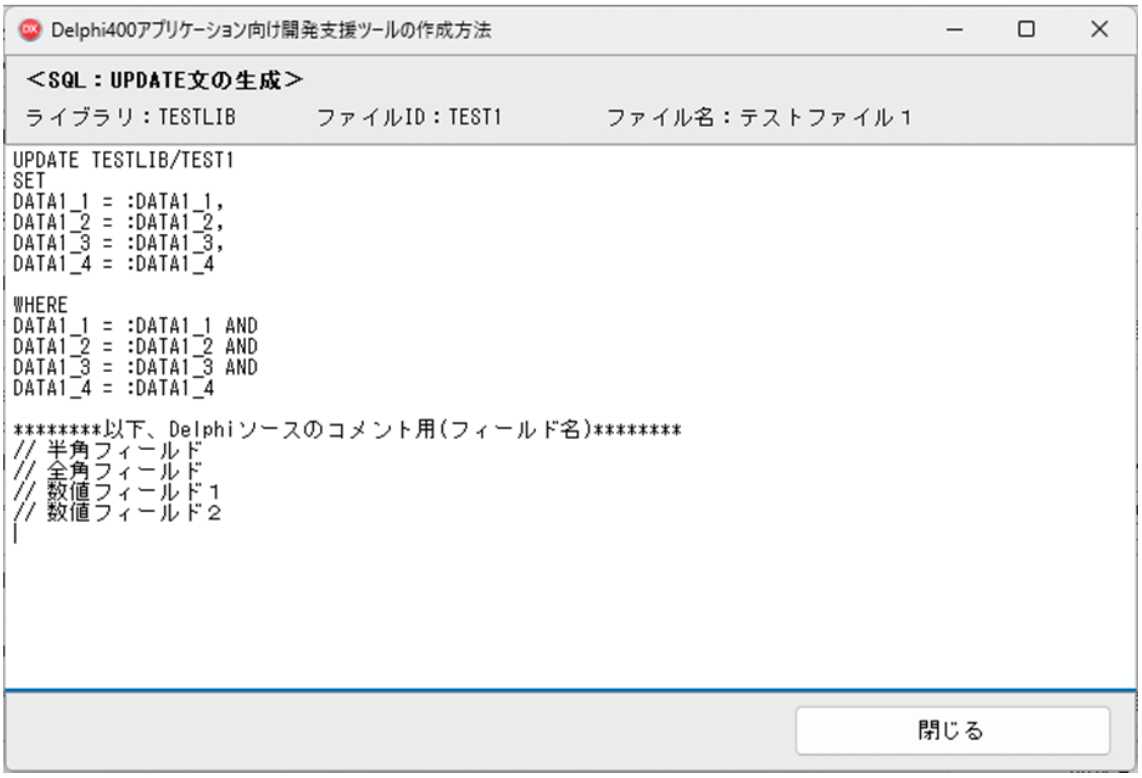



**図 12　SQL文の自動生成:実行結果(SQL文:INSERT)**



## 図 13 SQL文の自動生成:実行結果(SQL文:UPDATE)




## 図 14 SQL文の自動生成:実行結果(SQL文:DELETE)

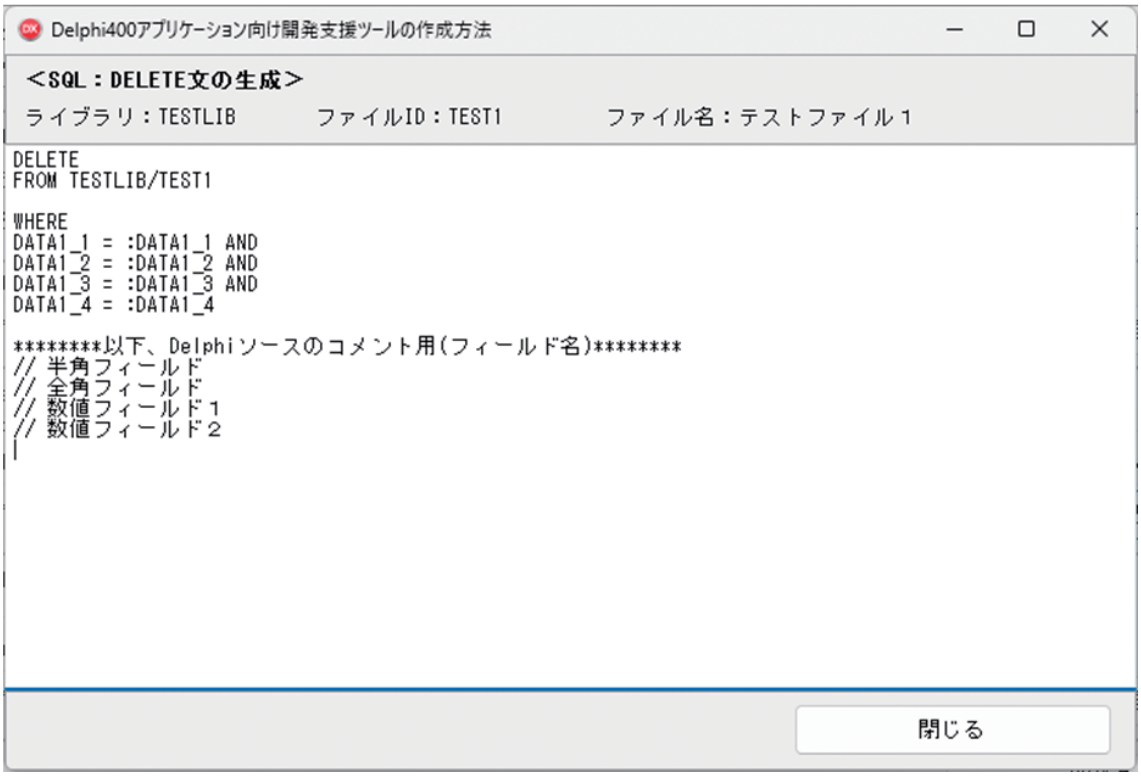


**図 15　SQL文の自動生成:実行結果(ParamByName)**

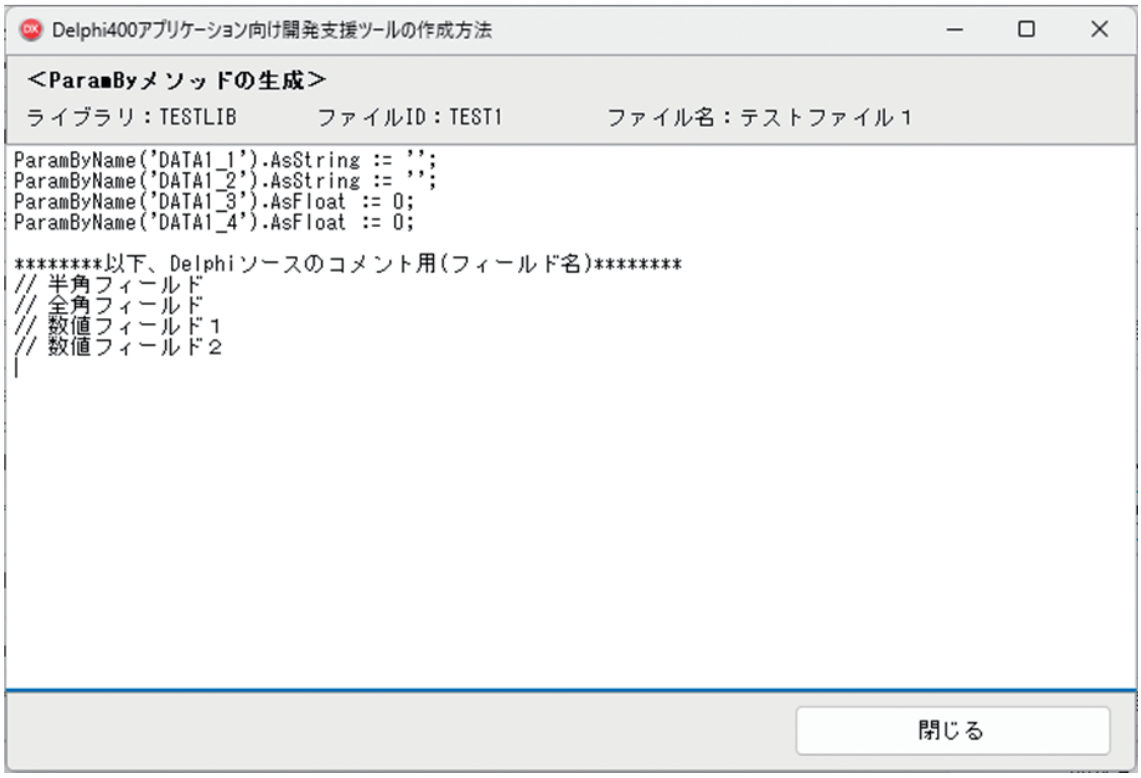
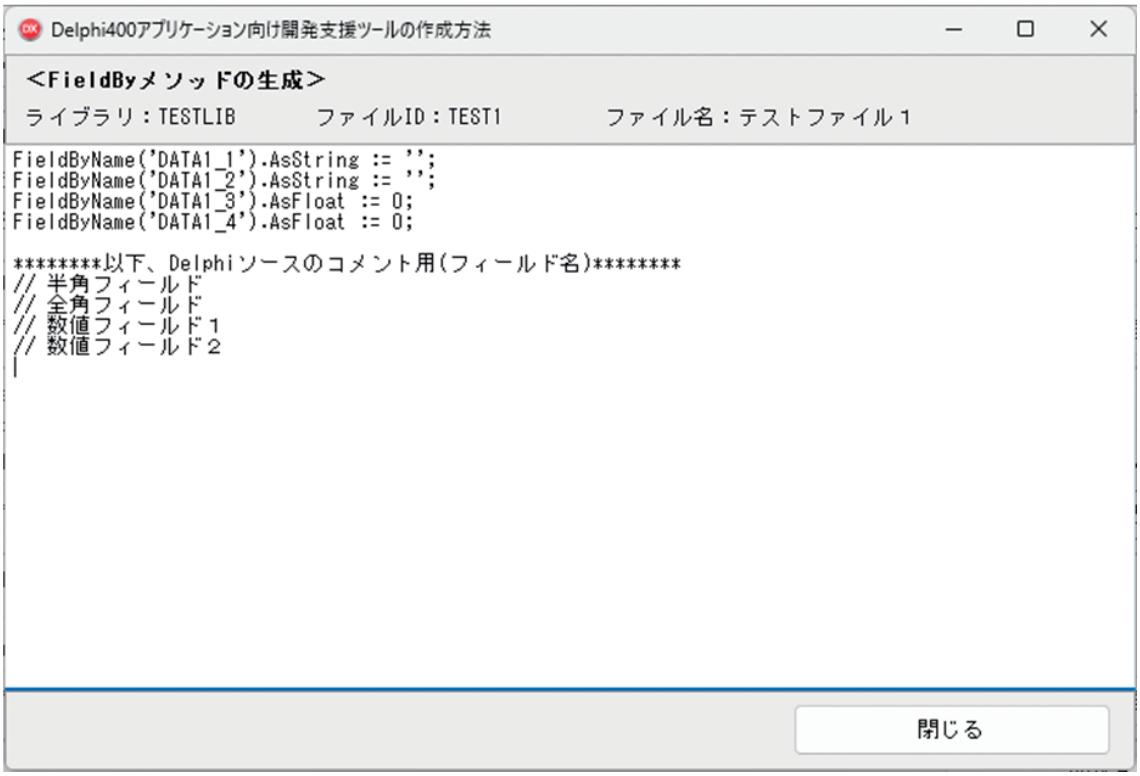



**図 16　SQL文の自動生成:実行結果(FieldByName)**



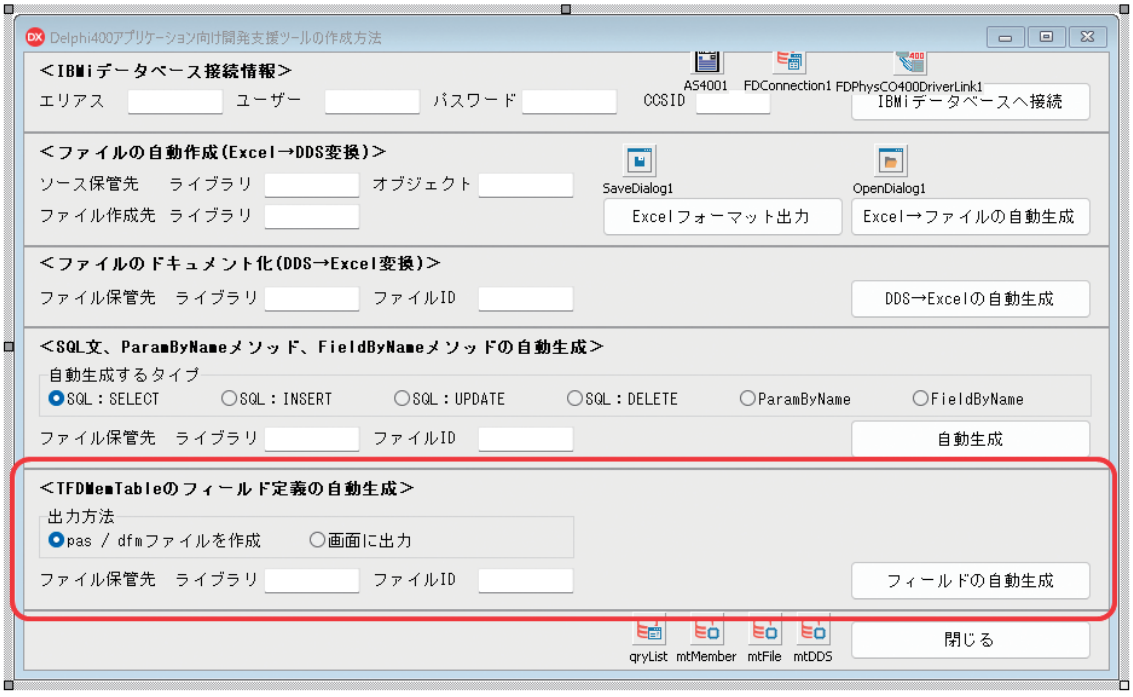
## 5.TFDMemTableのフィールド定義の自動生成

本章では、IBMiデータベース上に作成されたDDSソース及びオブジェクトから、TFDMemTableで使用するフィールド定義を自動生成する方法を紹介する。
TFDMemTableのフィールド定義の生成方法は、自動取込と手動作成の2パターンがある。自動取込の場合、①必要なフィールドを取得するSQL文を記述したTFDQueryを配置する。②TFDQueryをOpenし、フィールドエディタですべてのフィールドを取り込む。③TFDMemTable上で右クリック→メニューの「データセット割り当て」を選択→①のTFDQueryを選択する。④TFDMemTable上で右クリック→メニューの「フィールドエディタ」を選択→TFDQueryを選択する。⑤「フィールドエディタ」上で右クリック→メニューの「すべてのフィールドを追加」を選択し、フィールド定義を作成する。手動作成の場合、DDSソースを確認、もしくは、オブジェクトの情報を確認し、フィールド定義を作成する。
本章で紹介する機能は、オブジェクト保管先の情報を入力すれば自動生成できるため、作業の効率化に繋がるだろう。

### 5-1.コンポーネントの配置

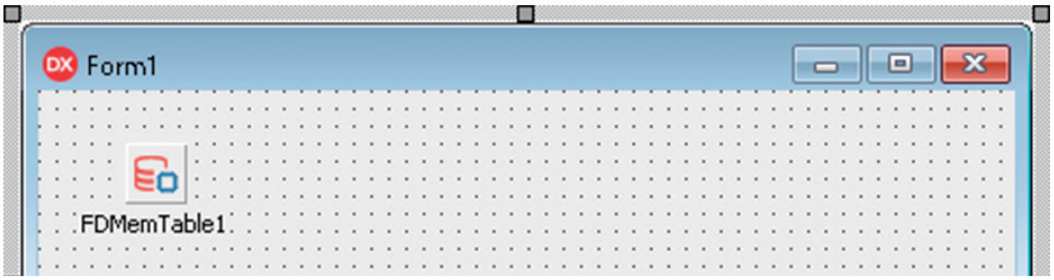
第4章で作成したサンプルプログラムに、出力方法を選択するためのTRadioGroup、ファイル保管先の情報を設定するためのTEdit、並びにTLabelやTButtonを画面に追加配置する【図17】。

**図 17　フィールド定義自動生成:コンポーネントの配置**



「フィールドの自動生成」ボタンのOnClick処理で、TFDMemTableのフィールド定義の自動生成処理を記述する。pas/dfmファイルを作成時用に、TFDMemTableのみを配置した雛形フォームを新規作成しておく【図18】。

**図 18　フィールド定義自動生成：雛形となるDelphi/400画面**




### 5-2.雛形フォームの保存

画面項目「出力方法」で「pas/dfmファイルを作成」を選択時、5-1.で作成した雛形フォームをTSaveDialogで指定の保管先へ保存する処理を記述する【ソース15】。

**ソース 15**

**雛形フォームの保存**

```
// 保存名の初期値
SaveDialog1.FileName   := 'Unit1';
SaveDialog1.DefaultExt := '.pas';
SaveDialog1.Filter     := 'Delphi ユニット (*.pas)|*.pas';

// 保存ダイアログの実行
if (not SaveDialog1.Execute) then
begin
  Abort;
end;
```

### 5-3.フィールド定義の生成

第4章の4-2.①【ソース12】の方法で、画面項目「ファイル保管先」で指定のファイルのフィールド情報をリストに内部保持する。内部保持した情報を基に、pasファイル用、dfmファイル及び画面出力用のTFDMemTableのフィールド定義を内部保持する処理を記述する【ソース16】。

**ソース 16**

**フィールド定義を内部保持する処理**

```
// dfmのスペース
case rgMemTableOut.ItemIndex of
  0:    sSpace := '    ';  // pas / dfmファイルを作成
  else sSpace := '';       // 画面に出力
end;

// フィールド定義の生成
for i := 0 to slFieldID.Count - 1 do
begin
  sName      := 'FDMemTable1' + slFieldID[i];    // 名前
  sFieldType := 'T' + slFieldType[i] + 'Field';  // フィールドタイプ

  // pasファイル
  slText_pas.Add('    ' + sName + ': ' + sFieldType + ';');

  // dfmファイル
  slText_dfm.Add(sSpace + 'object ' + sName + ': ' + sFieldType);          // 名前&フィールドタイプ
  slText_dfm.Add(sSpace + '  DisplayLabel = ''' + slFieldName[i] + '''');  // 表示ラベル
  slText_dfm.Add(sSpace + '  FieldName = ''' + slFieldID[i] + '''');       // フィールド名
  // 桁数(文字のみ)
  if (slFieldType[i] = 'String') then
  begin
    slText_dfm.Add(sSpace + '  Size = ' + slFieldLen[i] + '');
  end;
  slText_dfm.Add(sSpace + 'end');
end;
```

### 5-4.自動生成した内容を保存

画面項目「出力方法」で「pas/dfmファイルを作成」を選択時、5-2.で保存したpasファイル、dfmファイルに、5-3.で内部保持した情報を追記し、上書き保存する【ソース17】。画面項目「出力方法」で「画面に出力」を選択時、第4章の4-2.で作成したサブ画面【図10】に表示する。

**ソース 17**

**フィールド定義を保存：pas/dfmファイルを作成**

```
// コピー元ファイルの保管先
sFrom := ExtractFilePath(Application.ExeName) + 'Template';
sFrom := IncludeTrailingPathDelimiter(sFrom)  + 'Unit1.pas';

// 保存ファイル名（拡張子なし）
sSaveFileName := ExtractFileName(SaveDialog1.FileName);
sSaveFileName := StringReplace(sSaveFileName, '.pas', '', [rfReplaceAll]);

// 【pasファイル】
// コピー元ファイルの読み込み
slPas.LoadFromFile(sFrom);

// TFDMemTableの宣言部へ移動し、その後ろにフィールド情報を追加
iIndex := slPas.IndexOf('    FDMemTable1: TFDMemTable;');
for i := 0 to slText_pas.Count - 1 do
begin
  slPas.Insert(iIndex + i + 1, slText_pas[i]);
end;

// Unit1→ファイル保存ダイアログで指定の名前に変更
slPas.CommaText := StringReplace(slPas.CommaText, 'Unit1', sSaveFileName, [rfReplaceAll]);

// 保存
slPas.SaveToFile(SaveDialog1.FileName);

// 【dfmファイル】
// 拡張子を「dfm」に変更
SaveDialog1.FileName := System.SysUtils.ChangeFileExt(SaveDialog1.FileName, '.dfm');
sFrom                := System.SysUtils.ChangeFileExt(sFrom, '.dfm');

// コピー元ファイルの読み込み
slPas.LoadFromFile(sFrom);

// 末尾2行目にフィールド情報を追加
for i := 0 to slText_dfm.Count - 1 do
begin
  slPas.Insert(slPas.Count - 2, slText_dfm[i]);
end;

// 保存
slPas.SaveToFile(SaveDialog1.FileName);
```

以上で、DDSソース及びオブジェクトから、フィールド定義を自動生成するプログラムは完成である。上記プログラムを実行し、生成されたpas/dfmファイル及びサブ画面を確認する。変換対象は第2章で作成したファイル【図5】とし、ファイル・フィールド情報通りの内容がファイル保存、もしくはサブ画面にセットされていることが分かる【図19〜23】。

**図 19　フィールド定義自動生成：実行結果（pasファイル）**

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, FireDAC.Stan.Intf, FireDAC.Stan.Option,
  FireDAC.Stan.Param, FireDAC.Stan.Error, FireDAC.DatS, FireDAC.Phys.Intf,
  FireDAC.DApt.Intf, Data.DB, FireDAC.Comp.DataSet, FireDAC.Comp.Client;

type
  TForm1 = class(TForm)
    FDMemTable1: TFDMemTable;
    FDMemTable1DATA1_1: TStringField;
    FDMemTable1DATA1_2: TStringField;
    FDMemTable1DATA1_3: TFloatField;
    FDMemTable1DATA1_4: TFloatField;
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

自動生成されたフィールド情報

**図 20　フィールド定義自動生成：実行結果（dfmファイル）**

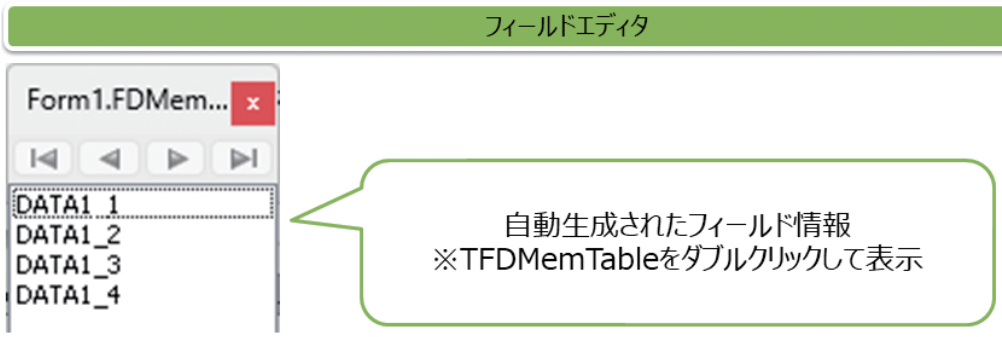
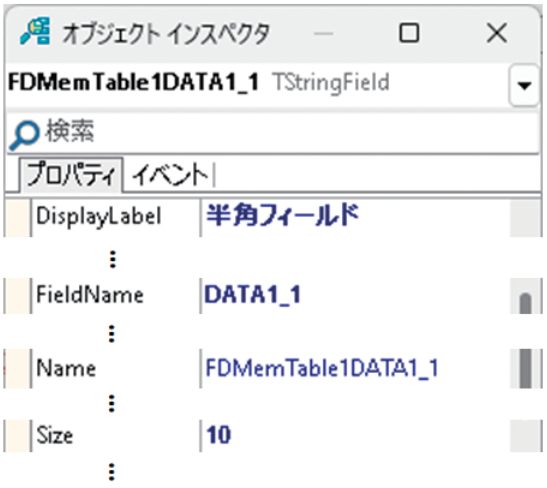


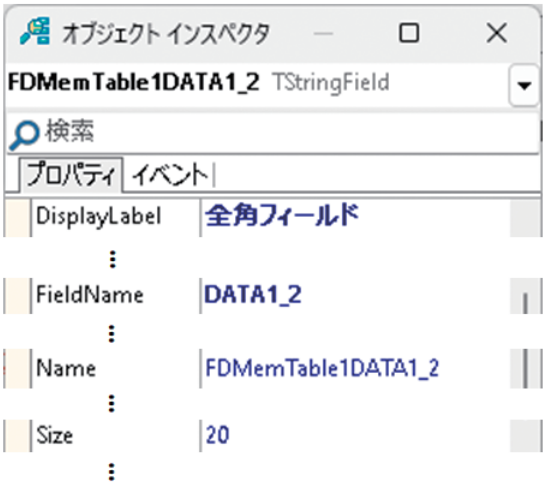
**図 21　フィールド定義自動生成：実行結果（dfmファイル）**

オブジェクトインスペクタ　※サンプルプログラムから設定したプロパティのみピックアップ

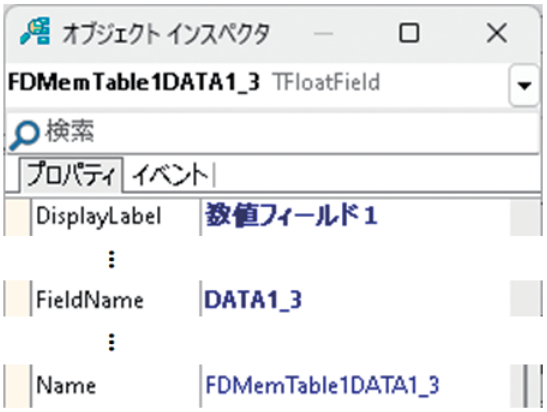
フィールド：DATA1_1



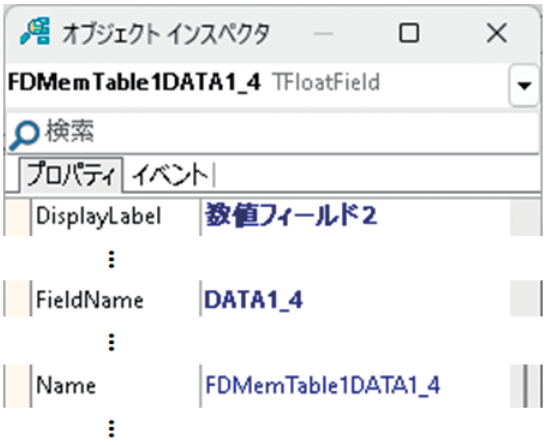

フィールド：DATA1_2




フィールド：DATA1_3




フィールド：DATA1_4



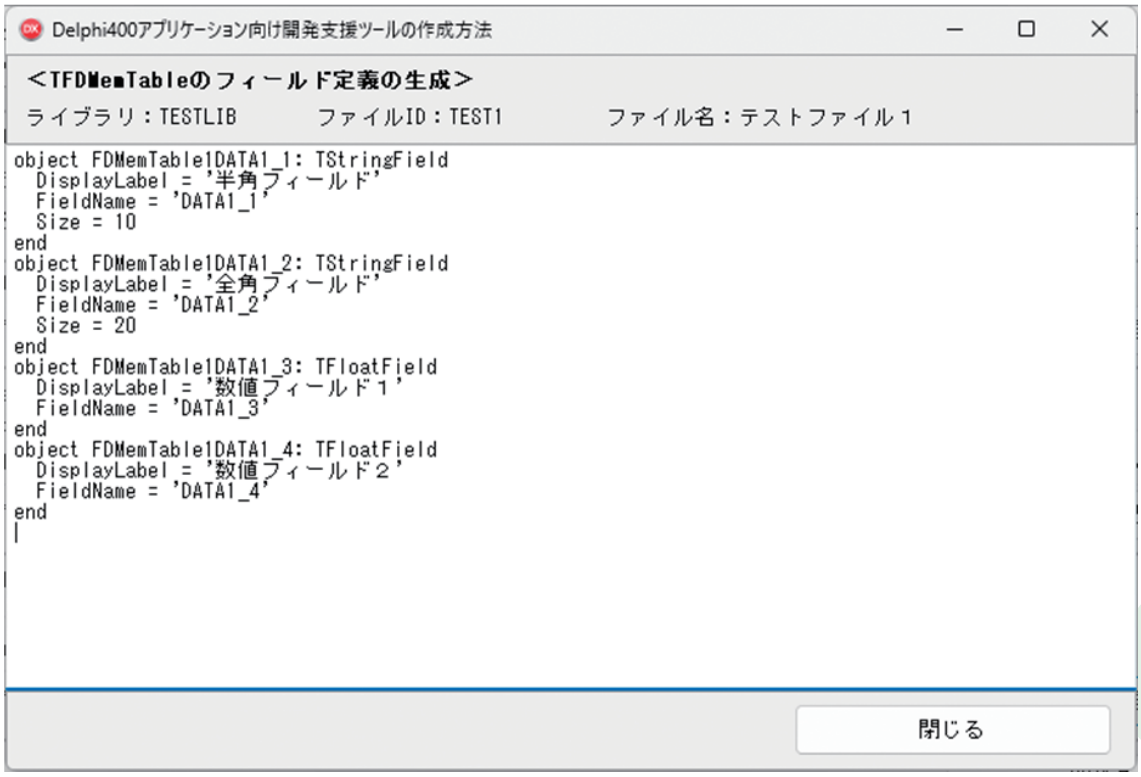
## 図 22 フィールド定義自動生成：実行結果（dfmファイル）

フィールドエディタ

```
object Form1: TForm1
  Left = 0
  Top = 0
      :
    object FDMemTable1DATA1_1: TStringField
      DisplayLabel = #21322#35282#12501#12451#12540#12523#12489
      FieldName = 'DATA1_1'
      Size = 10
    end
    object FDMemTable1DATA1_2: TStringField
      DisplayLabel = #20840#35282#12501#12451#12540#12523#12489
      FieldName = 'DATA1_2'
    end
    object FDMemTable1DATA1_3: TFloatField
      DisplayLabel = #25968#20516#12501#12451#12540#12523#12489#65297
      FieldName = 'DATA1_3'
    end
    object FDMemTable1DATA1_4: TFloatField
      DisplayLabel = #25968#20516#12501#12451#12540#12523#12489#65298
      FieldName = 'DATA1_4'
    end
  end
end
```

自動生成されたフィールド情報

## 図 23 フィールド定義自動生成：実行結果（画面出力）



## 6.さいごに

本稿では、Delphi/400アプリケーション向けの開発支援ツールの作成例を紹介した。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の効率化に役立てて頂ければ幸いである。

なお、今回紹介した開発支援ツール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/ntsuji2023.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

# 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

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**
**2. コミュニケーションツールの特徴**
**3. Indyの利用**
**4. メール送信機能の実装**
**5. チャット送信機能の実装**
**6. おわりに**

## 1.はじめに

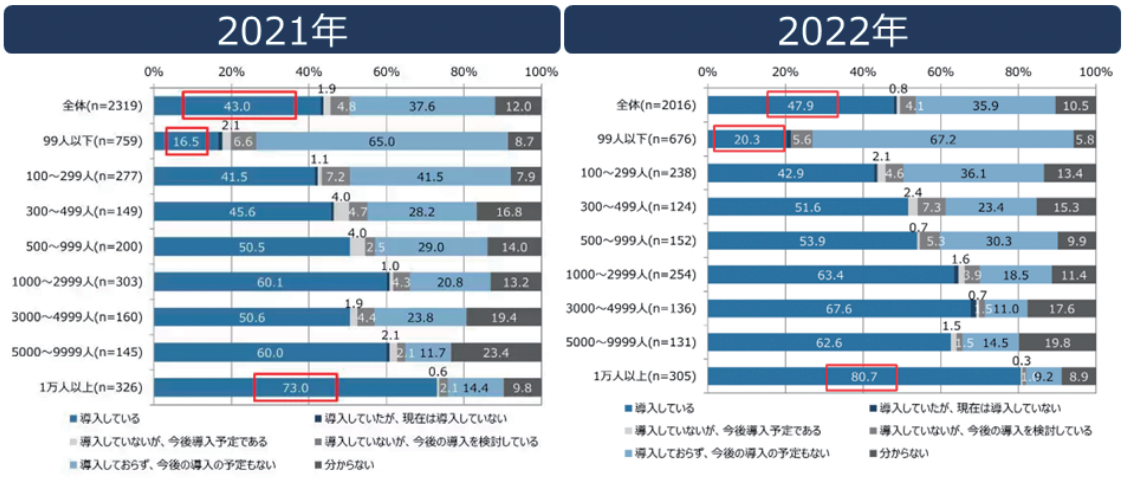
コミュニケーションツールの代表的なものとしてメールがある。使用しているサービスは企業により様々であるが、恐らくほとんどの企業が導入・活用しているであろう。

近年では新たなコミュニケーションツールとしてチャットを導入する企業が増えている。きっかけは、テレワークの実施が大きな要因であったかもしれない。

しかし近年、テレワークは実施する企業が徐々に減少している。一方で企業のチャットサービスの導入率は【図1】のように増加している。つまり、チャットサービスはテレワークの有無に関わらず、コミュニケーションツールのひとつとして認識され、活用されていることが分かる。

そこで本稿では、業務アプリケーションとメール・チャットサービスとの連携方法について紹介する。連携によるメリットは次章にて記載する。

**図 1　ビジネスチャット企業導入率**




出典：Biz Clip（https://www.bizclip.ntt-west.co.jp/articles/bcl00014-027.html）

## 2.コミュニケーションツールの特徴

メール・チャットにはそれぞれ異なる特徴がある。チャットはリアルタイムでメッセージの同期が可能である。簡単な連絡やグループでの共有、即座に確認すべきことがある場合に使用が適している。一方でメールは、重要な連絡や長文でメッセージを送る場合、記録として残す必要がある場合に適している。

上記の特徴により、社外への連絡はメールを使用し、社内への連絡はチャットを利用するなど、状況によってコミュニケーションツールを使い分けている企業も多いのではないだろうか。

しかし、日常の業務において使用するツールが増えることは、複数サービス(アプリ)を切り替えて使い分けなければいけないというデメリットも生まれる。

例えば、業務アプリケーションでデータを照会し、照会結果を基にメール内容を入力するシーンでは、業務アプリケーションの起動及びメールサービスの起動が必要となる。さらに社内への連絡が追加で必要となれば、チャットサービスの起動も必要となる。PCの限られた画面スペースの中で複数サービスを立ち上げることは業務効率の低下にも繋がりかねない。

本稿で紹介する業務アプリケーションとの連携を行うと、メールやチャットサービスの立ち上げが不要となる。よって、円滑に業務を進めることが可能となる。

## 3.Indyの利用

業務アプリケーションと各サービスの連携にはDelphiに標準付属しているIndyを利用する。Indyとは、オープンソースのネットワーク関連コンポーネントである。

本稿の例では業務アプリケーションで見積書を出力し、先方にメール送付するシーンを想定する。また、添付する見積書は圧縮+パスワード付与を行い、メール送信後は社内にチャットで送信完了を報告する業務フローとする【図2】。

Indyを利用し、業務アプリケーションとの連携を実現すると【図3】のようなフローとなる。では、次章より具体的な実装方法について紹介していく。

本サンプルでは、Delphi/400 11 Alexandriaを使用し、メールサービスはOffice365(Outlook)、チャットサービスはChatworkを使用する。

**図 2　メール・チャットサービス連携前業務フロー**

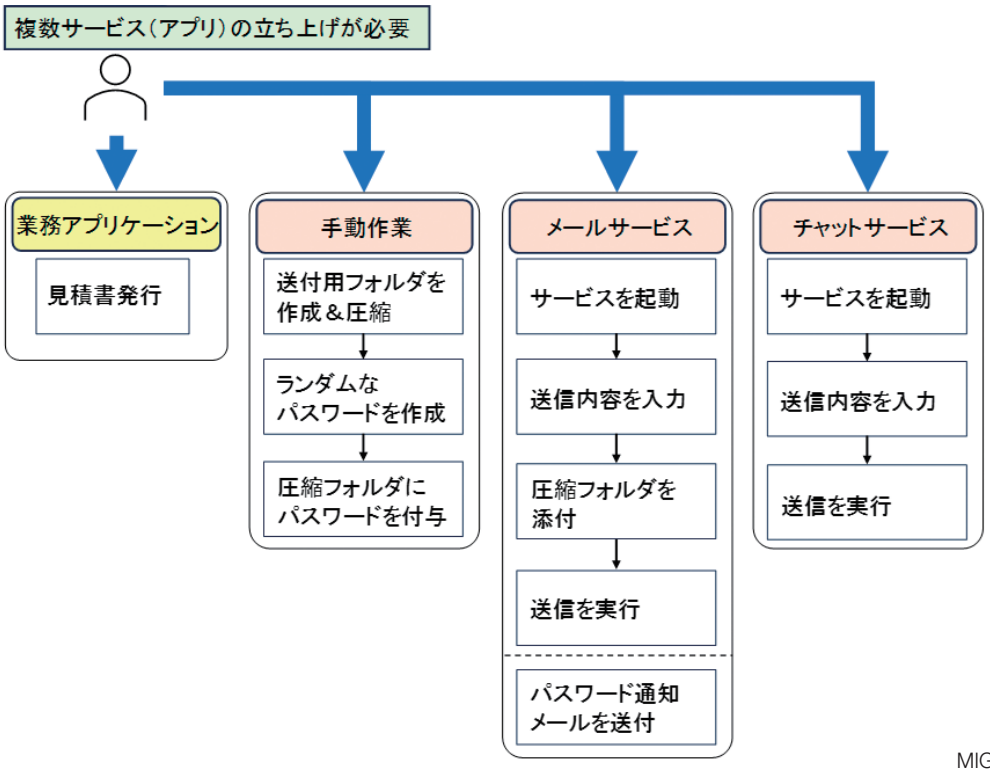
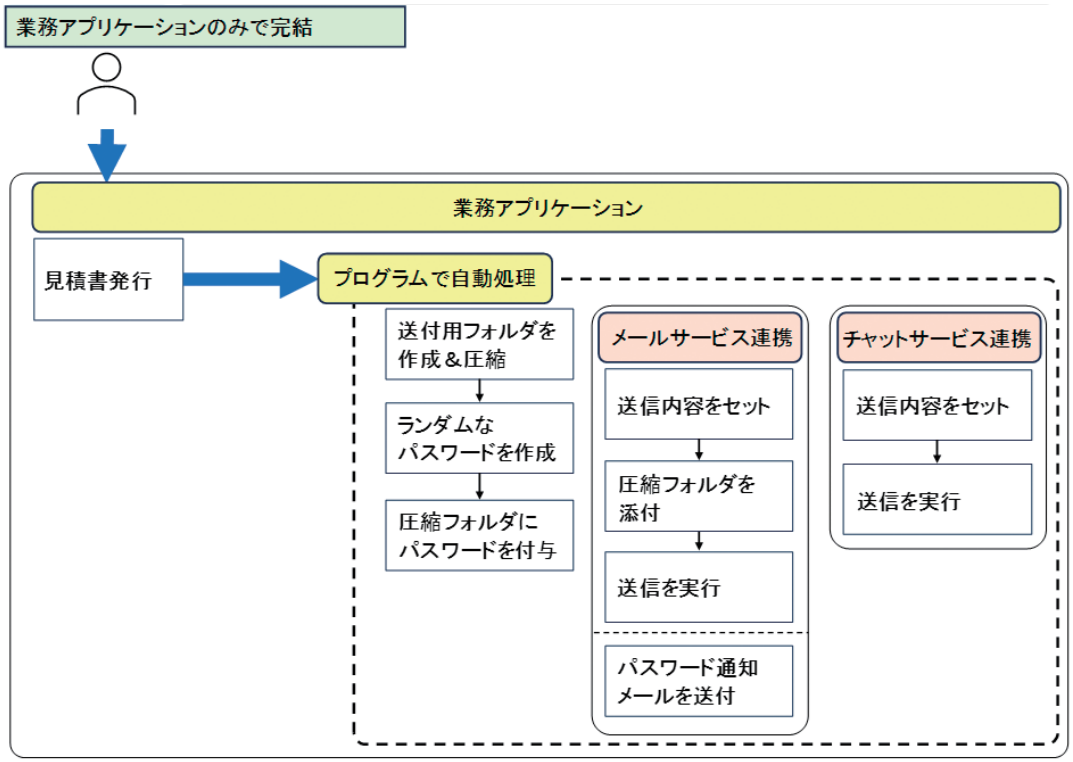


**図 3　メール・チャットサービス連携後業務フロー**

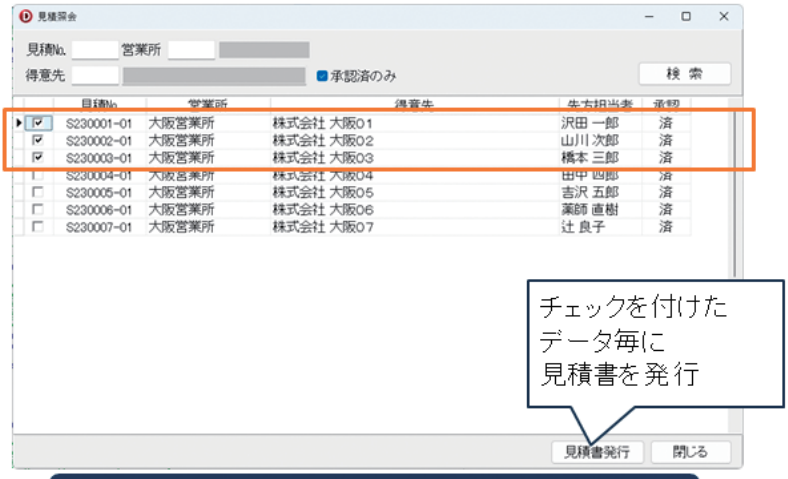



## 4.メール送信機能の実装

本稿では、業務アプリケーションの仕様を以下と仮定する。

見積照会で作成済みの見積データを検索し、チェックを付けたデータ毎に見積書を発行可能とする。見積照会では、見積書の送付先である先方担当者の名前を表示しており、メールアドレスなどの紐づく担当者情報をマスタにて保管している【図4】。

**図 4　業務アプリケーション（改修前）**




マスタにメールアドレス等を保管

```
*****************************************************
   FILE-ID    :    MTKTNF
   FUNCTION   :    得意先担当者マスタ
*****************************************************
                              UNIQUE
     R MTKTNR                 TEXT(' 得意先担当者マスタ ')
       TTTKCD        5A       COLHDG(' 得意先コード ')
       TTTNCD        3A       COLHDG(' 得意先担当者コード ')
       TTTNNM      220        COLHDG(' 担当者名 ')
       TTTNML     1000        COLHDG(' メールアドレス ')
       TTTNMF       1A        COLHDG(' メール送信対象 FLG ')
     K TTTKCD
     K TTTNCD
```

### 4-1.コンポーネントの貼り付け

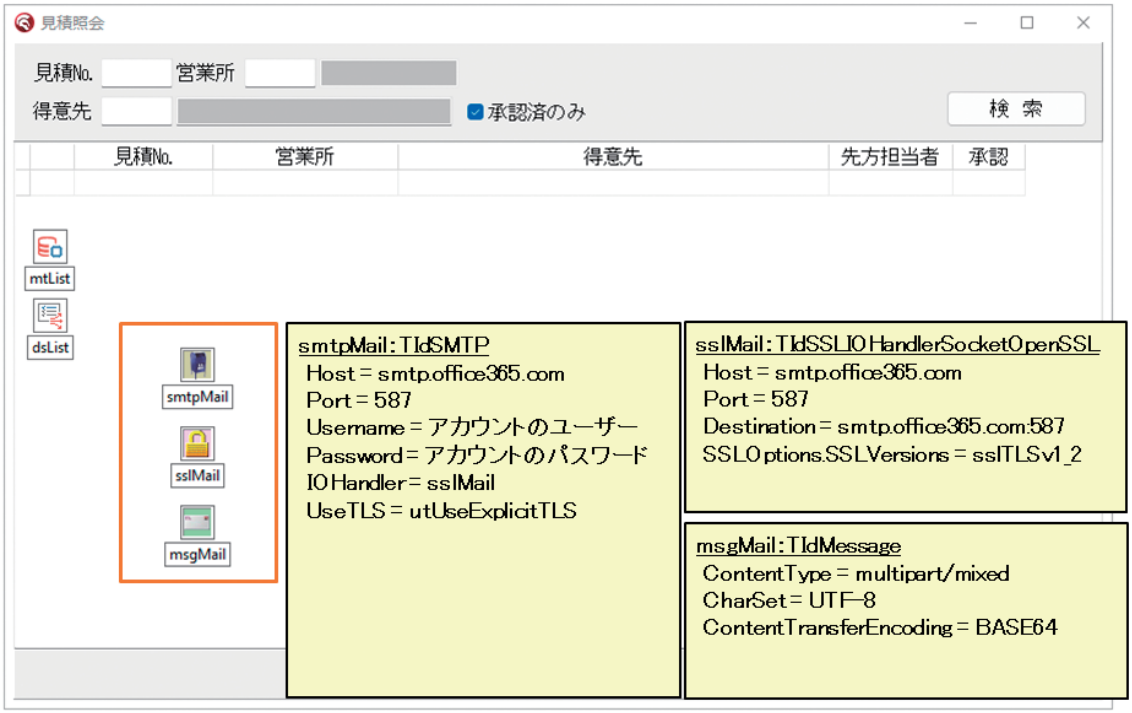
まずは、メール送信処理を実装するために必要なコンポーネントの貼り付けを行う。メール送信機能を実装したい画面に、TIdSMTPとTIdSSLIOHandlerSocketOpenSSL、TIdMessageを配置し、各プロパティ設定を行う【図5】。配置した各コンポーネントの役割は以下の通りである。

smtpMail:TIdSMTP
メールサーバーとの接続及び送信処理の実施

sslMail:TIdSSLIOHandlerSocketOpenSSL
TLS/SSLを使用した暗号化通信を可能にする

msgMail:TIdMessage
送信先アドレスやメッセージ内容の設定

**図5 メール送信用コンポーネントの配置**




各コンポーネントに対するプロパティ設定について詳しく解説していく。

smtpMailでは、メールサーバーへの接続情報について設定している。本稿ではoffice365を使用するため、Hostは「smtp.office365.com」、Portは「587」にて設定する。設定値については使用するメールサービスによって異なるため、各メールサービスのヘルプなどでご確認いただきたい。また、使用するメールサービスによっては2段階認証の設定をメールサービス側で行い、外部アプリケーションで使用するためのパスワードを生成する必要がある。その際は、生成パスワードをPasswordプロパティに設定しなければならない。

sslMailでは、暗号化通信を行うための設定を行う。本稿ではSSL/TLS(STARTTLS)を使用して通信を行うためsmtpMailと同様にsslMailにHostやPortの設定を行う。SSLVersionsプロパティにはTIdSSLIOHandlerSocketOpenSSLで指定可能な最新バージョンである「sslTLSv1_2」を設定する。

smtpMailのIOHandlerプロパティには「sslMail」を指定し、UseTLSプロパティには「utUseExplicitTLS」を指定する。しかし、UseTLSプロパティは暗号化方式などにより、設定値が異なる為、各メールサービスに合わせて設定いただきたい。

msgMailでは、送信するメッセージの文字タイプや文字コードについて設定している。ContentTypeプロパティには主に以下のような形式を指定できるが、本稿では「multipart/mixed」を指定する。

text/plain：文字テキストのみのメール形式
text/html：htmlと同様に文字に装飾が可能な形式
multipart/mixed：複数のメール形式が混在可能な形式

CharSetプロパティでは文字コードを指定する。Unicode形式に対応するため「UTF-8」を設定する。ContentTransferEncodingプロパティではメールの変換方式について指定できる。本稿ではメールにファイル添付を行う想定であるため、エンコード方式は「BASE64」を設定する。

### 4-2.メール送信ロジックの実装

コンポーネントの貼り付け完了後、次は送信ロジックの実装を行う。実装は「見積書出力ボタン」押下時のタイミングで行う。**【ソース１】**が送信ロジック実装前の処理である。本稿では、自動送信とするため、件名及び本文を予めAS/400にデータ登録しておく**【図６】**。追加のロジックとしては、**【ソース２】【ソース３】**の通り

①メール内容のデータを取得
②SMTPサーバーへの接続
③メールの送信元と送信先のセット
④メールの件名と本文のセット
⑤送信の実行
⑥SMTPサーバーの接続解除

である。また②の実施後は⑥を必ず実行する必要があるためtry～finally～endにて処理している。また、メール件名の文字化けを防ぐために、msgMailのOnInitializeISOイベントに処理を追加する**【ソース４】**。メール送信のロジックは以上で完了である。しかし、実行すると**【図７】**のようなエラーが発生する。

**ソース１**

**メール送信ロジック実装前**

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
begin
  mtList.DisableControls;
  mtList.First;
  try
    while not mtList.Eof do
    begin
      if mtList.FieldByName('CHEK').AsInteger = 1 then
      begin
        // 見積書出力(見積No.のフォルダ)
        OutPutMTMR;
      end;

      mtList.Next;
    end;
  finally
    mtList.EnableControls;
  end;
end;
```

## 図 6 登録メッセージ

**ファイルレイアウト**

```
A*       FILE-ID    :    MMESGF
A*       FUNCTION   :    メッセージ登録マスタ
A                                UNIQUE
A         R MMESGR               TEXT(' メッセージ登録マスタ ')
A           MSSKEY       10A     COLHDG(' メッセージ識別キー ')
A           MSMSRW        3S 0   COLHDG(' メッセージ行 ')
A           MSMESG       100     COLHDG(' メッセージ ')
A         K MSSKEY
A         K MSMSRW
```

**メール件名**

| | メッセージ識別キー | メッセージ行 | メッセージ |
|---|---|---|---|
| 000001 | MTMRMAILK | 1 | 御見積書の送付 |

**メール本文**

データ毎にセットする値を変更するものは仮の値とし、ロジックにて変換する

| | | | |
|---|---|---|---|
| 000001 | MTMRMAILH | 1 | [CANPANY] |
| 000002 | MTMRMAILH | 2 | [TANTNM] 様 |
| 000003 | MTMRMAILH | 3 | いつもお世話になっております。株式会社ミガロ．の [MYNAME] です。 |
| 000004 | MTMRMAILH | 4 | |
| 000005 | MTMRMAILH | 5 | この度は見積もりのご依頼をいただき、誠にありがとうございます。 |
| 000006 | MTMRMAILH | 6 | 御見積書を添付ファイルにてお送りします。 |
| 000007 | MTMRMAILH | 7 | 内容にご不明な点、ご要望がございましたらお申し付けください。 |
| 000008 | MTMRMAILH | 8 | |
| 000009 | MTMRMAILH | 9 | ═══════════ |
| 000010 | MTMRMAILH | 10 | 株式会社ミガロ． |
| 000011 | MTMRMAILH | 11 | 〒 556-0017 |
| 000012 | MTMRMAILH | 12 | 大阪府大阪市浪速区湊町 2-1-57 |
| 000013 | MTMRMAILH | 13 | 難波サンケイビル 13F |
| 000014 | MTMRMAILH | 14 | TEL:06-6631-8601　FAX:06-6631-8603 |
| 000015 | MTMRMAILH | 15 | ═══════════ |

## ソース 2

### メール送信処理の実装(メイン処理)

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
var
  sSubject, sGetBody, sBody: String;
  sConpany: String;
  sTANTNM: String;
  sMYNAME: String;
begin
  // メール内容取得
  sSubject := GetMSG('MTMRMAILK'); // 件名
  sGetBody := GetMSG('MTMRMAILH'); // 本文

  mtList.DisableControls;
  mtList.First;
  try
    // SMTPサーバーへの接続
    smtpMail.Connect;
    try
      while not mtList.Eof do
      begin
        if mtList.FieldByName('CHEK').AsInteger = 1 then
        begin
          // 見積書出力(見積No.のフォルダ)
          OutPutMTMR;
```

① GetMSG関数はソース3で記述

②

```
          try
            // メールFrom - To
            msgMail.From.Address := 'mgrtechnical@outlook.jp'; // メールFrom
            msgMail.Recipients.Clear;                          // クリア
            msgMail.Recipients.EMailAddresses :=
              mtList.FieldByName('TTTNML').AsString;           // メールTo

            msgMail.Subject := sSubject; // 件名セット

            // メール本文
            // 動的な値をStringReplaceで変換
            sConpany  := mtList.FieldByName('TKNM').AsString; // 得意先名
            sTANTNM   := mtList.FieldByName('STNM').AsString; // 先方担当者
            sMYNAME   := '前坂'; // 送信者名

            msgMail.Body.Clear;
            sBody := StringReplace(sGetBody, '[CANPANY]', sConpany, [rfReplaceAll]);
            sBody := StringReplace(sBody, '[TANTNM]', sTANTNM, [rfReplaceAll]);
            sBody := StringReplace(sBody, '[MYNAME]', sMYNAME, [rfReplaceAll]);
            msgMail.Body.Text := sBody; // 本文セット

            // 送信処理
            smtpMail.Send(msgMail);
          except
            on E: Exception do
            begin
              raise;
            end;
          end;
        end;

        mtList.Next;
      end;
    finally
      // SMTPサーバー接続解除
      smtpMail.Disconnect;
    end;
  finally
    mtList.EnableControls;
  end;
end;
```

③ ④ ⑤ ⑥

## ソース 3

### メール送信処理の実装(メール内容の取得)

```
{*******************************************************************************
 目的: メッセージマスタ取得処理
 引数:
 戻値:
*******************************************************************************}
function TfrmReview.GetMSG(AGetKEY: String): String;
begin
  qryMSG.Close;
  qryMSG.SQL.Text :=
    ' SELECT MSMESG FROM MMESGF WHERE MSSKEY = :MSSKEY ORDER BY MSMSRW ';
  qryMSG.ParamByName('MSSKEY').AsString := AGetKEY;
  qryMSG.Open;
  try
    while not qryMSG.Eof do
    begin
```

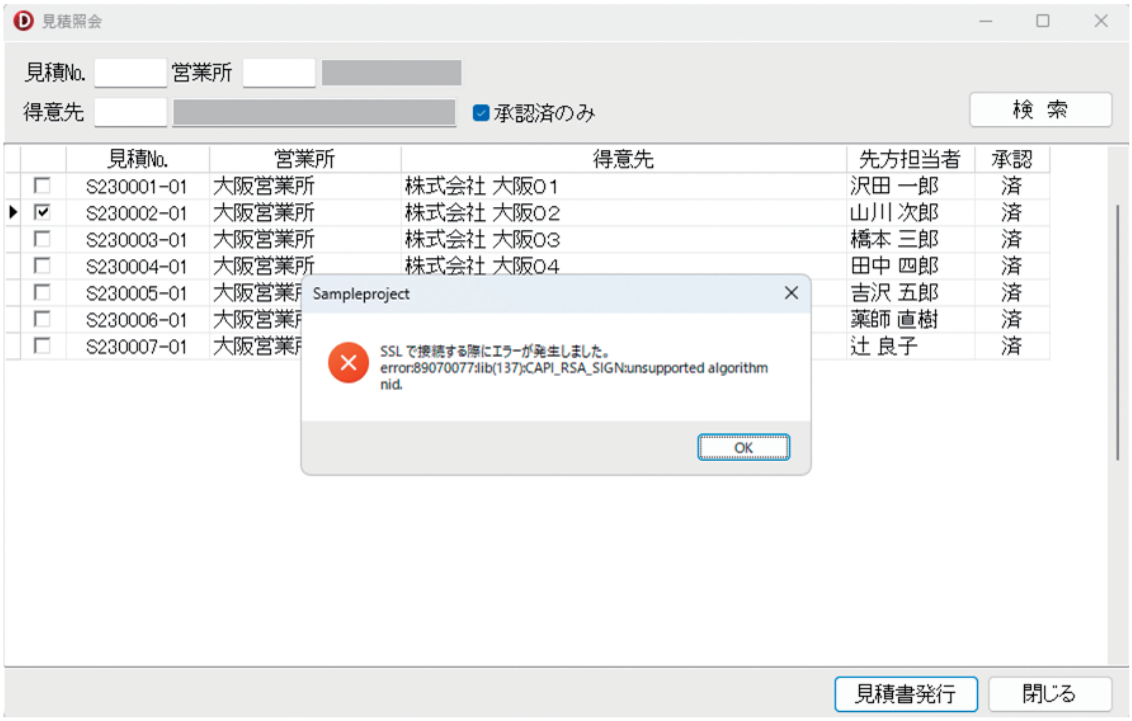
```
        if Result <> '' then
        begin
          // 改行
          Result := Result + #13#10;
        end;
        Result := Result + qryMSG.FieldByName('MSMESG').AsString;
        qryMSG.Next;
      end;
    finally
      qryMSG.Close;
    end;
  end;
```

## ソース 4

### メール件名の文字化け防止

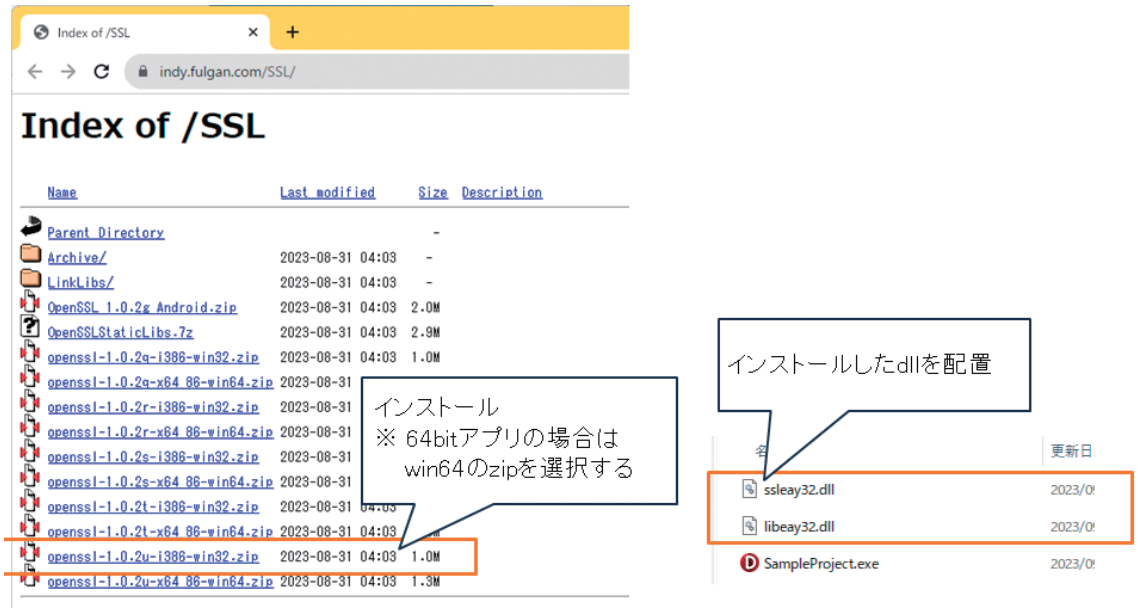
```
{*******************************************************************************
 目的: TIdMessage Initialize処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.msgMailInitializeISO(var VHeaderEncoding: Char;
  var VCharSet: string);
begin
  VHeaderEncoding := 'B';
  // メールヘッダーのCharSetはここで設定する(メール件名に影響)
  VCharSet := 'UTF-8';
end;
```

## 図 7 メール送信エラー

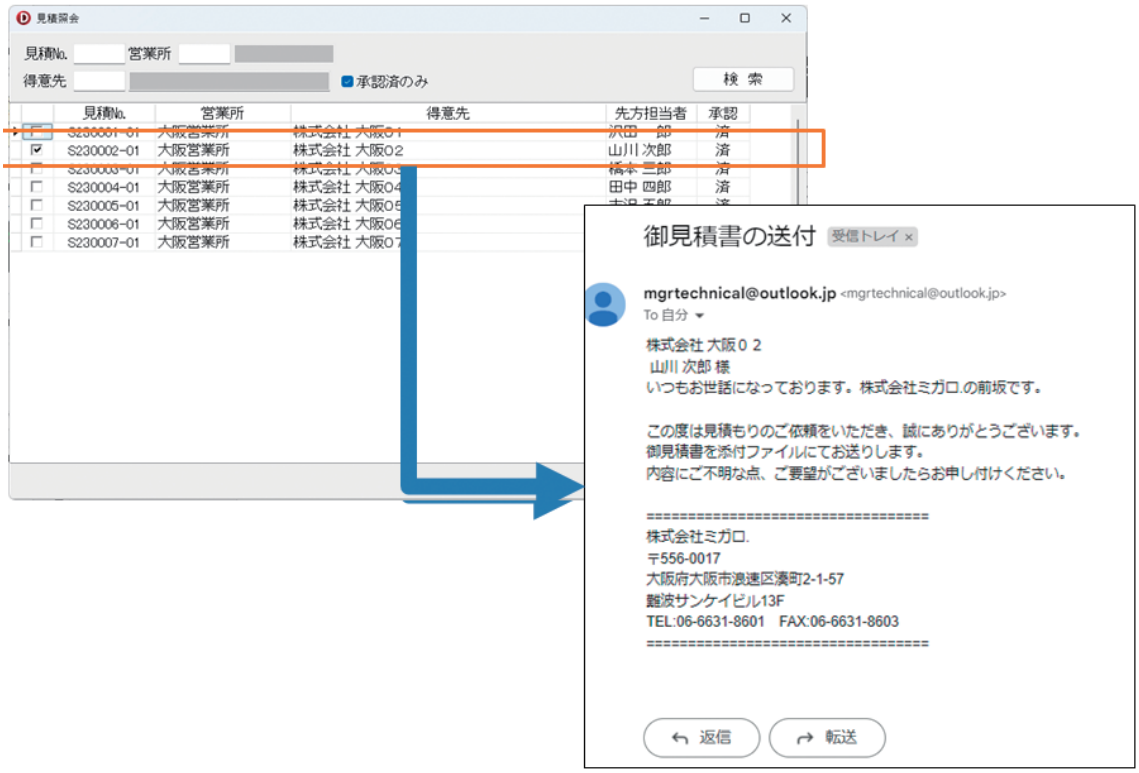


【図7】のエラーを解決するためには https://indy.fulgan.com/SSL/ より「libeay32.dll」「ssleay32.dll」を取得し、実行モジュールと同階層に配置する必要がある【図8】。各dllを配置した後、再度モジュールを実行するとメール送信がエラーなく実行される【図9】。

## 図8 dllのインストール




## 図9 メール送信実行結果



### 4-3.出力フォルダの圧縮とパスワード付与

本稿では、出力した見積書は特定のフォルダに保管される仕組みとする。メールには該当フォルダを圧縮後、パスワードを付与したものを添付する。

フォルダの圧縮には「TAbZipper」コンポーネントを使用する。「TAbZipper」は「GetItパッケージマネージャー」から「Abbrevia　2021.11」を選択しインストールする。インストール方法は**【図 10】**のとおりである。インストールが完了したら、画面に「TAbZipper」コンポーネントを貼り付けてみよう**【図 11】**。

**図 10　コンポーネントのインストール**

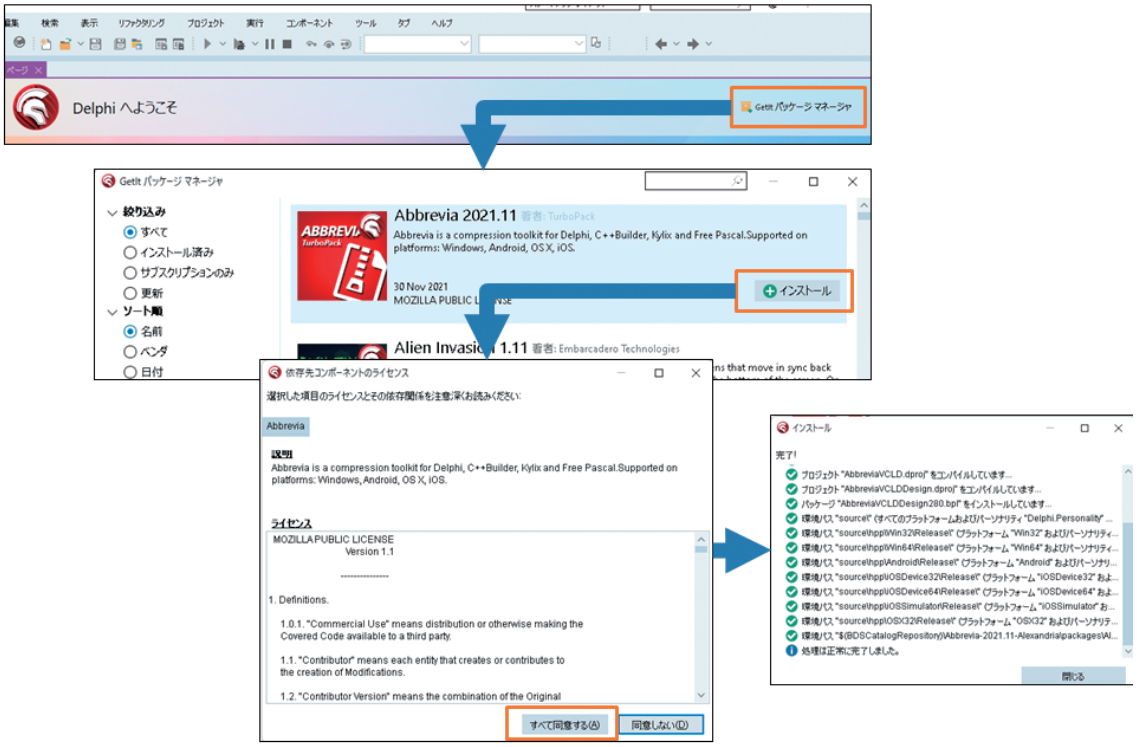
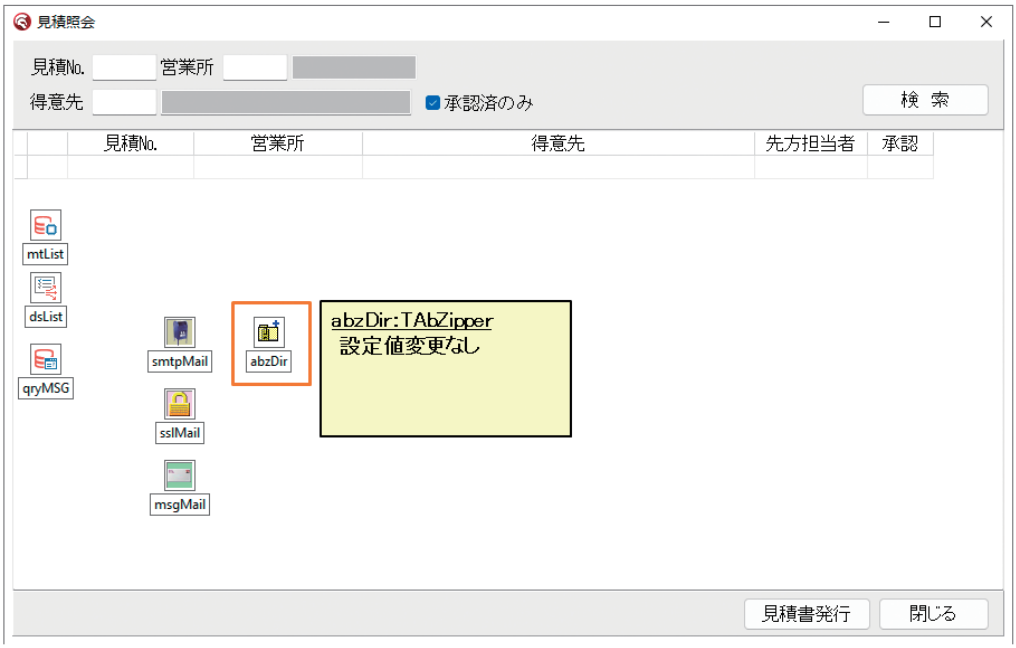



**図 11　ファイル圧縮用コンポーネントの貼り付け**



コンポーネントの貼り付けが完了したら、まずはランダムパスワードを作成する関数を実装する。ランダムな文字列を設定するにはRandom関数を使用する。Random関数の呼び出し前には、Randomize手続きにより乱数生成関数を初期化しておく【ソース5】。ランダムパスワードの実装が完了したら、次は圧縮化の処理を実装する。圧縮化の処理はFileNameに圧縮後のファイル名（フルパス）を指定し、BaseDirectoryに圧縮対象のフォルダ（フルパス）を指定する。その後、AddFilesにて圧縮化するファイルを指定し、SaveにてFileNameで指定した場所に保管される。本稿のようにPassWord設定を行いたい場合は、PassWordに文字列をセットする【ソース6】。この時に併せて、設定パスワードとZipファイル名を戻り値にしておく（メールへの添付処理で使用）。

## ソース 5

### ランダムパスワード作成処理

```
{*************************************************************************
 目的: ランダムパスワード作成処理
 引数:
 戻値: 生成パスワード
*************************************************************************}
function TfrmReview.GetRandomPass: String;
var
  sRandPassStr: String;
begin
  Randomize;
  sRandPassStr := 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890';
  repeat
    Result := Result + sRandPassStr[Random(Length(sRandPassStr)) + 1];
  until (Length(Result) = 10);
end;
```

## ソース 6

### フォルダ圧縮処理

```
{*************************************************************************
 目的: フォルダ圧縮処理
 引数: AZipPassWord - 設定パスワード　戻り値
 戻値: 圧縮ファイル名（フルパス）
*************************************************************************}
function TfrmReview.ZipFileCreate(var AZipPassWord: String): String;
var
  LFileName: TFileName;
  sZipFileName: String;
begin
  abzDir.FileName:= 'C:¥Document¥見積書¥'+ mtList.FieldByName('MTNO').AsString+ '.zip';
  abzDir.BaseDirectory:= 'C:¥Document¥見積書¥'+ mtList.FieldByName('MTNO').AsString;
```

```
  // abzDir.BaseDirectory直下のファイルを全て圧縮
  for LFileName in TDirectory.GetFiles(
    abzDir.BaseDirectory, '*.*', TSearchOption.soAllDirectories) do
  begin
    sZipFileName :=
      StringReplace(LFileName,
        IncludeTrailingPathDelimiter(abzDir.BaseDirectory), '', [rfIgnoreCase]);
    abzDir.AddFiles(sZipFileName, 0);
  end;

  abzDir.Password := GetRandomPass; // ランダムパスワード設定
  AZipPassWord := abzDir.Password;
  abzDir.Save;

  Result := abzDir.FileName;
  abzDir.CloseArchive;
end;
```

### 4-4.圧縮ファイルの添付

圧縮ファイル作成の実装が完了したら、メールへの添付処理を実装する。メールへの添付はUses節にIdAttachmentFileを追加し、TIdAttachmentFileを生成する。後は、圧縮ファイル名及び各プロパティを設定するだけで実装完了である【ソース7①】。また、パスワードメールも送付するよう併せて実装する【ソース7②】。ファイル添付処理を実装した結果が【図12】である。

**ソース7**

**ファイル添付処理**

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
var
  sSubject, sGetBody, sBody: String;
  sConpany: String;
  sTANTNM: String;
  sMYNAME: String;
  sZipFileName, sZipPassWord: String;
begin
  // メール内容取得
  sSubject := GetMSG('MTMRMAILK'); // 件名
  sGetBody := GetMSG('MTMRMAILH'); // 本文

  mtList.DisableControls;
  mtList.First;
```

```
  try
    // SMTPサーバーへの接続
    smtpMail.Connect;
    try
      while not mtList.Eof do
      begin
        if mtList.FieldByName('CHEK').AsInteger = 1 then
        begin
          // 見積書出力(見積No.＋システム日時のフォルダ)
          OutPutMTMR;

          // Zip化 (Zipファイル名を返却)
          sZipFileName := ZipFileCreate(sZipPassWord);
          try
```

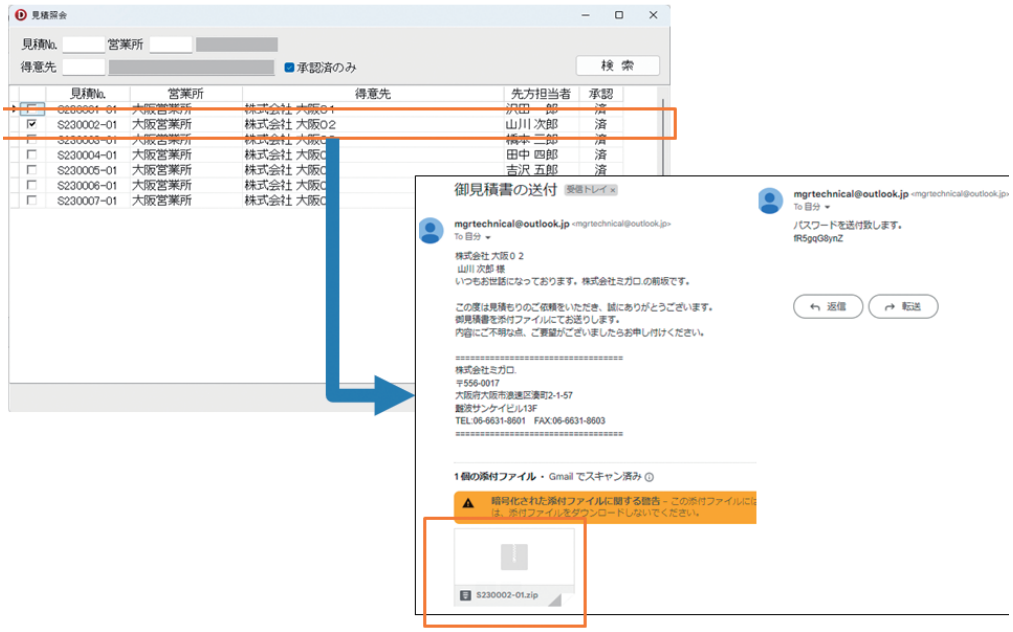
～ メール本文設定処理 省略 ～

```
            // 添付ファイルの設定
            msgMail.MessageParts.Clear;
            with TIdAttachmentFile.Create(msgMail.MessageParts, sZipFileName) do
            begin
              FileName := sZipFileName;
              ContentType := 'application/octet-stream';
              ContentTransfer := 'base64';
            end;

            // メール本文送信処理
            smtpMail.Send(msgMail);

            // パスワードメールの送付
            msgMail.Subject := sSubject; // 件名セット
            msgMail.Body.Clear;
            msgMail.Body.Text := 'パスワードを送付致します。' + #13#10 + sZipPassWord; // 本文セット
            msgMail.MessageParts.Clear;  // 添付ファイルクリア

            // パスワードメール送信処理
            smtpMail.Send(msgMail);
          except
            on E: Exception do
            begin
              raise;
            end;
          end;
        end;

        mtList.Next
      end;
    finally
      // SMTPサーバー接続解除
      smtpMail.Disconnect;
    end;
  finally
    mtList.EnableControls;
end;
end;
```

①

②

**図 12　ファイル添付処理実行結果**




## 5.チャット送信機能の実装

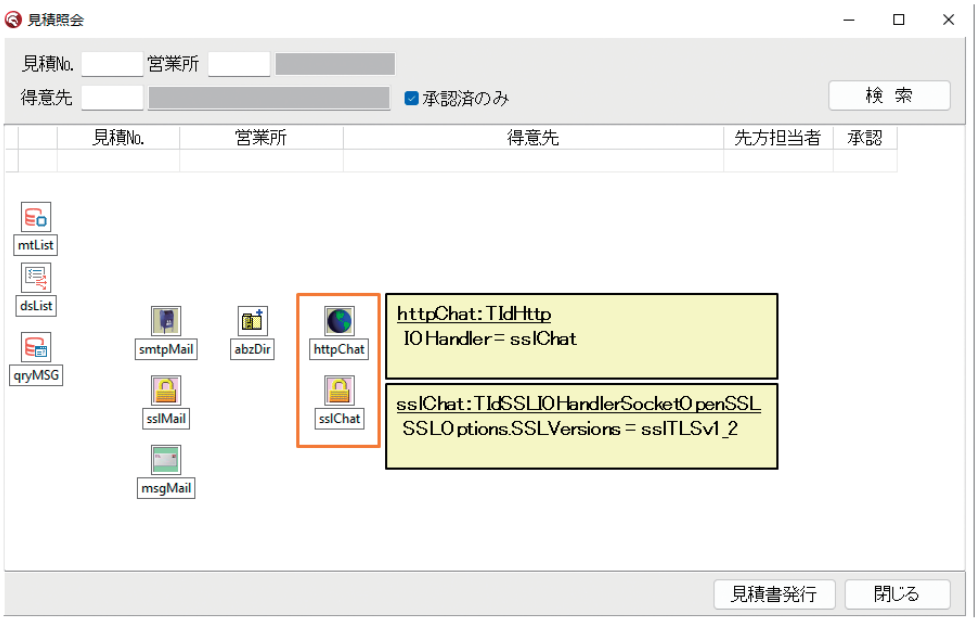
本章では、チャット送信機能の実装方法について紹介していく。

### 5-1.コンポーネントの貼り付け

まずは、チャット送信処理を実装するために必要なコンポーネントの貼り付けを行う。TIdHTTPとTIdSSLIOHandlerSocketOpenSSLを配置し、各プロパティ設定を行う【図 13】。配置した各コンポーネントの役割は以下の通りである。

httpChat:TIdHTTP
Postメソッドを使用し、指定URLにデータ送信を実施
sslChat:TIdSSLIOHandlerSocketOpenSSL
TLS/SSLを使用した暗号化通信を可能にする

**図 13　チャット送信用コンポーネントの配置**



### 5-2.APIエンドポイントの確認

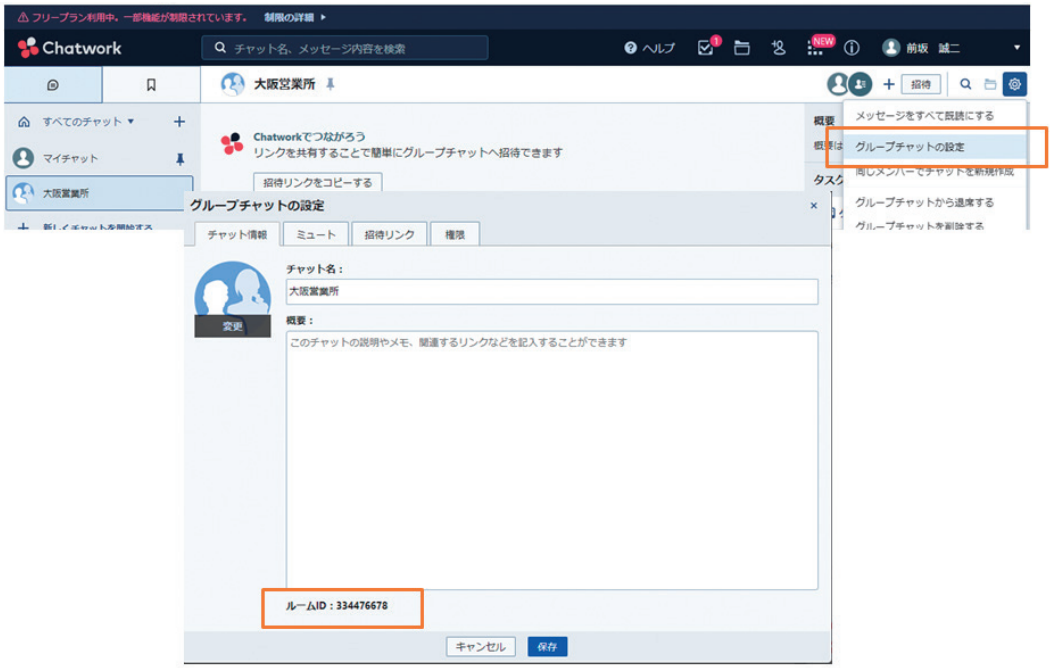
APIエンドポイントとは、Webサービスやアプリケーションに外部クライアントが通信するためのURLである。つまり、本稿で使用するChatworkにおいても、業務アプリケーションが通信するためのURLを確認しなければならない。Chatworkでは、https://developer.chatwork.com/docs/endpointsのページにて、設定するURLやパラメータに関する記載がある。また、チャット送信を行うためには併せてトークン、チャットのルームIDを確認する必要がある。

トークンとはAPIの認証を提供するセキュリティの仕組みであり、APIを使用するためのパスワードのようなものである。トークンの取得はサービスによって異なるが、Chatworkの場合は【図 14】のように、サービス連携のリンクから確認することで取得可能である。

ルームIDはチャットを行うグループの識別IDであり、ルームIDの指定により、指定グループへのチャット送信が可能となる、ルームIDの確認方法は【図 15】である。

**図 14　トークンの確認方法**


**図 15　ルームIDの確認方法**



### 5-3.チャット送信ロジックの実装

チャット送信ロジックの実装は先ほどのAPIエンドポイントを確認したページを参考にする。リクエストに必要なContentTypeプロパティとCustomHeadersプロパティに値を設定する。後は、送信先URLと送信するメッセージ内容を引数にPostメソッドを実行すると、実装は完了である**【ソース8】【ソース9】**。実行結果は**【図16】**となる。

**ソース 8**

**チャット送信処理(呼び出し元)**

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
var
  sSubject, sGetBody, sBody: String;
  sConpany: String;
  sTANTNM: String;
  sMYNAME: String;
  sZipFileName, sZipPassWord: String;
  sSendMTNO: String;
begin
  // メール内容取得
  sSubject := GetMSG('MTMRMAILK'); // 件名
  sGetBody := GetMSG('MTMRMAILH'); // 本文

  mtList.DisableControls;
  mtList.First;
  try
    // SMTPサーバーへの接続
    smtpMail.Connect;
    try
      while not mtList.Eof do
      begin
        if mtList.FieldByName('CHEK').AsInteger = 1 then
        begin
          // 見積書出力(見積No.+システム日時のフォルダ)
          OutPutMTMR;

          ～ メール送信処理 省略 ～

          // 送付見積No.
          if sSendMTNO <> '' then
          begin
            sSendMTNO := sSendMTNO + #13#10;
          end;
          sSendMTNO := sSendMTNO + mtList.FieldByName('MTNO').AsString

        end;

        mtList.Next;
      end;
    finally
      // SMTPサーバー接続解除
      smtpMail.Disconnect;
    end;
  finally
    mtList.EnableControls;
  end;

  // チャット送信処理
  SendChat(sSendMTNO);
end;
```

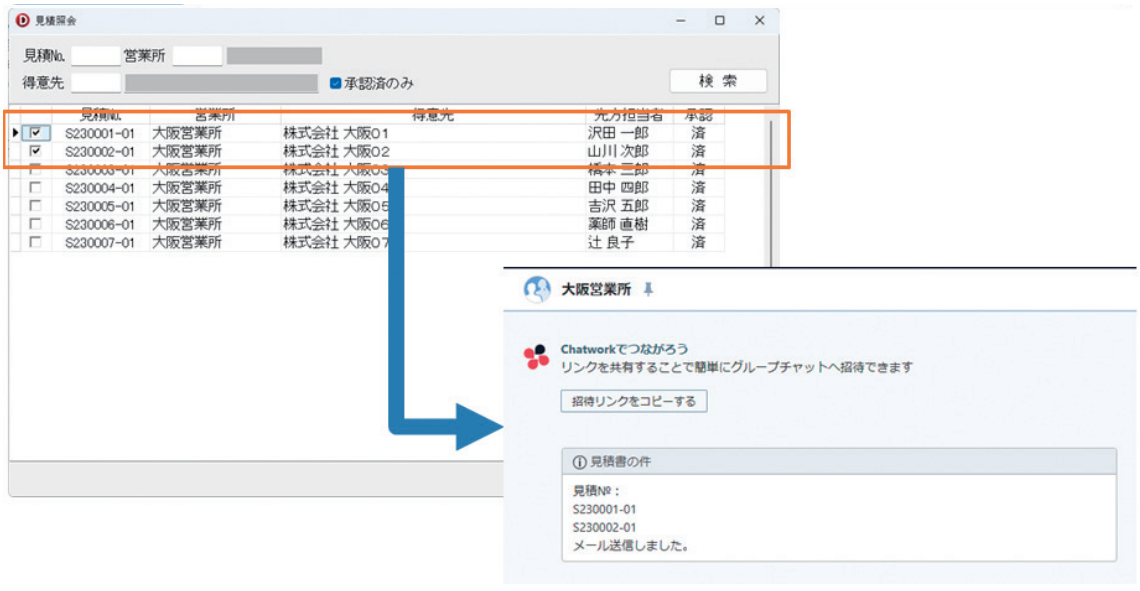
SendChat処理は
ソース9で記述

## ソース 9

### チャット送信処理(メイン処理)

```
{*******************************************************************************
 目的: チャット送信処理
 引数: AMTNO : 送付見積No. ※複数可能性あり
 戻値:
*******************************************************************************}
procedure TfrmReview.SendChat(AMTNO: String);
var
  slBody: TStringList;
  sSetInfo: String;
  sURL, sTOKN, sRMID: String;
begin

  sURL  := 'https://api.chatwork.com/v2/rooms/';  // URL
  sTOKN := '7fe7a578371e92cdaa609e34b0ae43a2';     // トークン
  sRMID := '334476678';                           // ルームID

  slBody := TStringList.Create;
  try
    sSetInfo := 'body=[info]';
    sSetInfo := sSetInfo + '[title]見積書の件[/title]';

    sSetInfo := sSetInfo + '見積No.:' + #13#10 + AMTNO + #13#10 + 'メール送信しました。';
    sSetInfo := sSetInfo + '[/info]';
    slBody.Add(sSetInfo);

    httpChat.Request.ContentType := 'application/x-www-form-urlencoded';
    httpChat.Request.CustomHeaders.AddValue('X-ChatWorkToken', sTOKN);
    try
      httpChat.Post(sURL + sRMID + '/messages', slBody);
    except
      raise;
    end;
  finally
    FreeAndNil(slBody);
  end;

end;
```

## 図 16 チャット送信処理実行結果



## 6.おわりに

各サービスのメッセージ送信画面を立ち上げることなく、業務アプリケーション内で処理が完結することをおわかりいただけただろうか。また、各サービスとの連携方法についても、Indyを利用すると簡単に実装が可能であることを実感いただけたと思う。

本稿は、送信の自動化による例でご紹介したが、もちろん自身でメッセージ送信用の入力画面を作成いただくことも可能である。また、コンポーネントを変更すれば受信機能についても容易に実装可能である。

現在は、メールサービス、チャットサービスの種類が豊富で、どれを導入すべきか判断が難しい。そこで、現在使用している業務アプリケーションと連携可能かといった観点をひとつの判断基準としてもよいのかもしれない。

本稿でご紹介した実装方法であれば、プロパティの設定値を変更するだけで他のサービスへの連携も可能である。もし、現在ご検討されているメールサービスやチャットサービスがあれば、本稿を参考に連携可能かをお試しいただけると幸いである。

# SmartPad4i

# SmartPad4i ExcelJSで簡単！ Excel活用テクニック

株式会社ミガロ.
プロダクト事業部　技術支援課
**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。

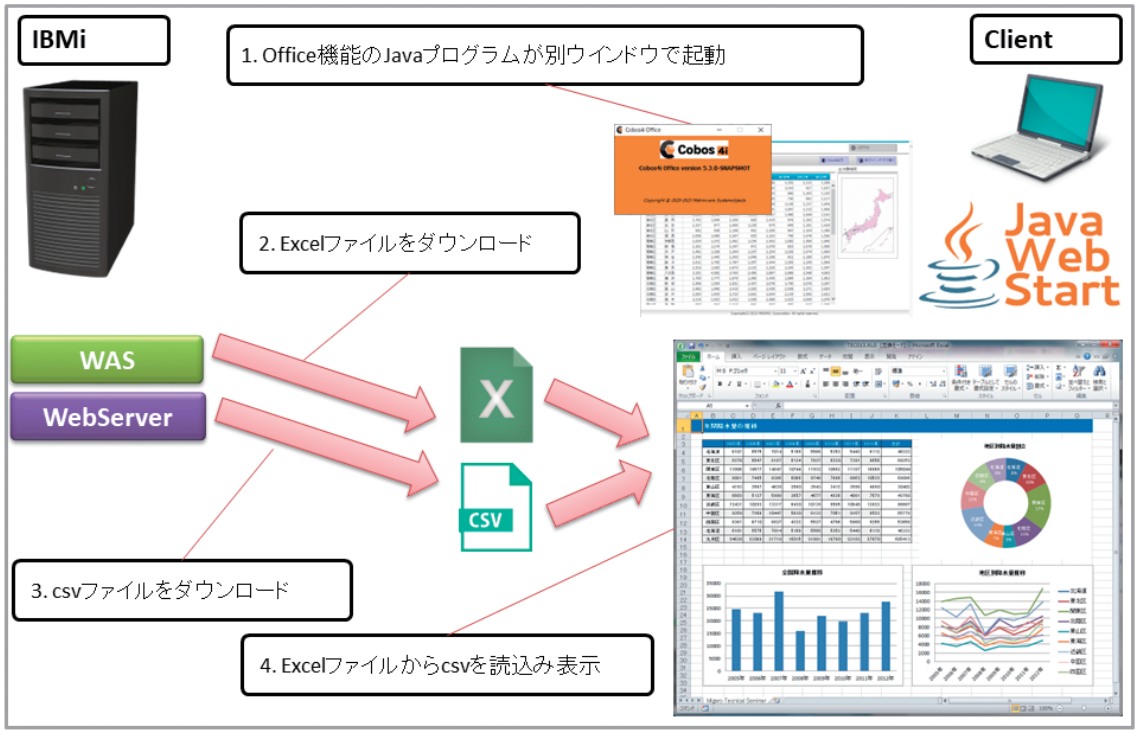
1. はじめに
2. JavaScriptでExcelファイルを作成する方法
3. テンプレートを読込んで売上伝票を作成する方法
4. おわりに

## 1.はじめに

SmartPad4i(Cobos4i)には、IBMiからExcelファイルをダウンロード後、CSVファイルと連携させてExcel/Wordを表示できるOffice機能が存在する。**【図1】**

Office機能はRPGを記述することでクライアント端末にExcelファイルを作成できる便利な機能である。

**図 1　SmartPad4i(Cobos4i)　Office機能**



Office機能はSmartPad4i(Cobos4i)の最新版では、Java Web Startで実装されている。
Java Web StartはJava11以降で廃止となったため、Office機能を使用するにはJava8(※1)をクライアント端末にインストールする必要がある。

(※1)2023年9月現在、Javaはバージョン21まで存在しており、Java8,Java11,Java17,Java21のみサポートが継続されている。

Java8は2030 年 12 月までサポートされるため、Office機能は使用し続けて頂けるが、クライアント端末にJava8を導入する必要があるため、動作環境が制限される。
また、Office機能はcsvファイルを表データとして取込むため、セル単位のレベルでデータをExcelファイルへ出力できない。

Webアプリケーションで Excelファイルを出力する場合、以前はサーバーサイドのプログラム開発(Java,PHP ,Perl等)が必要であった。多言語でのプログラム開発の知識や、サーバー環境の構築が必要なため、実装にはハードルが高い。
しかし、OSS(オープンソースソフトウェア)のExcelJSを使用すると簡単にExcelファイルが作成できる。
JavaScriptの実装は必要であるが、誰でも簡単に利用可能だ。
JavaScriptのみでExcelファイルが作成できれば、サーバー環境の構築や、多言語でのプログラム開発の必要はない。クライアント環境には、JavaScriptが動作するブラウザだけで済む。
また、ExcelJSの場合、セル単位でデータの出力が制御できるため、任意のレイアウトでExcelファイルを作成することが可能だ。さらに、JavaScriptはHTMLに追加するだけで動作するため、SmartPad4i(Cobos4i)に組込むことも容易である。

そこで、本稿ではExcelJSでExcelファイルを作成する方法について紹介する。

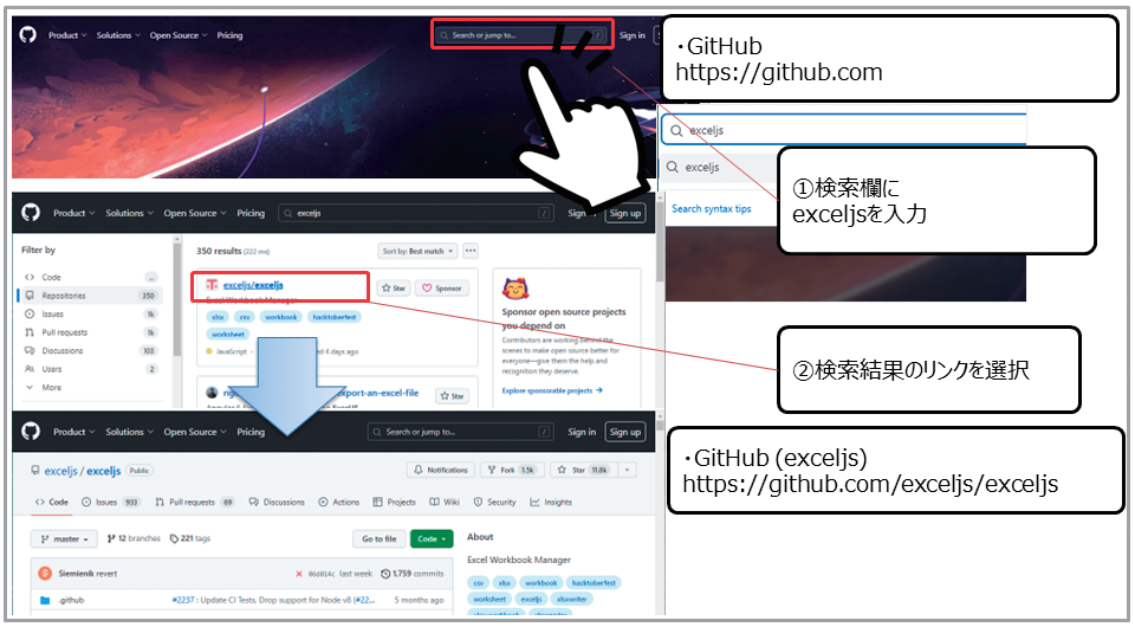
## 2.JavaScriptでExcelファイルを作成する方法

はじめに、JavaScriptでExcelファイルを作成できるOSS(オープンソースソフトウェア)のExcelJSについて説明する。

### 2-1. ExcelJSとは

ExcelJSはJavaScriptを使用してExcelファイルを操作するライブラリで、Node.js環境(サーバーサイド)で動作し、Excelファイルの生成/編集が可能である。【図2】

図 2 ExcelJS



● ExcelJS
https://github.com
検索欄にexceljsを入力して検索
または、https://github.com/exceljs/exceljs

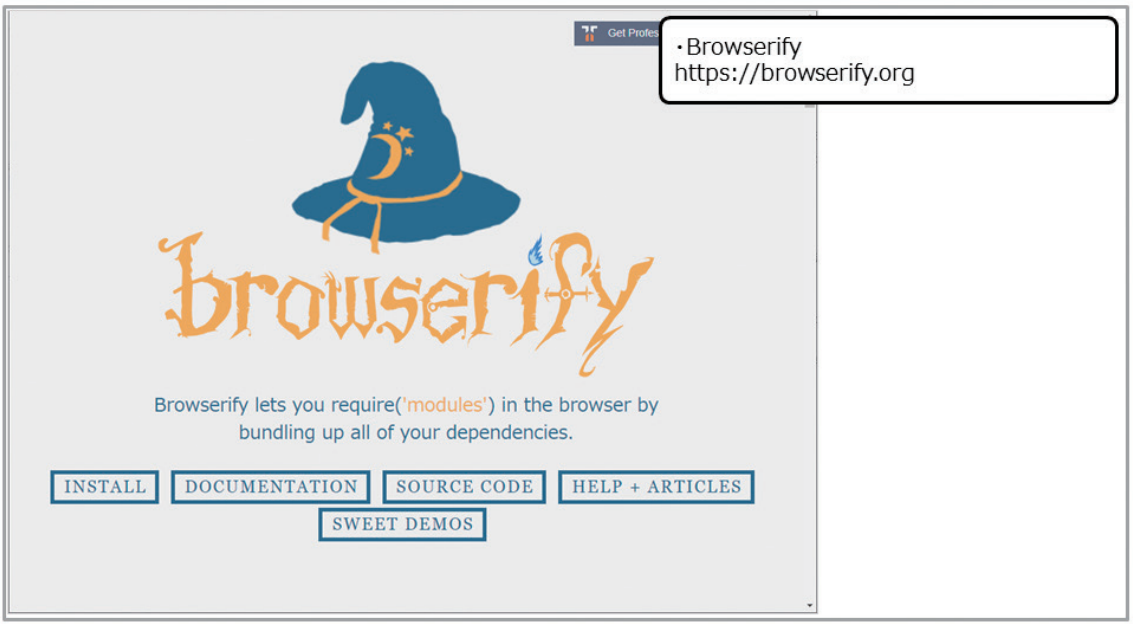
ExcelJSは通常、サーバーサイドで使用するライブラリであるが、Browserifyで変換されたJavaScriptファイルを使用することで、クライアントサイドのJavaScriptでも利用可能である。

### 2-1-1. Browserifyとは

BrowserifyはJavaScriptで開発をする際に使用するツールの一つで、Node.jsのモジュールをブラウザ上で実行できる形式に変換するためのツールである。**【図3】**

● Browserify
https://browserify.org

**図 3 Browserify**




Node.jsはサーバーサイドのJavaScript実行環境であるため、同じJavaScriptをブラウザ上で実行しても動作しない。Browserifyはこの制約を解消し、Node.jsで作成されたモジュールをブラウザ上で動作するものに変換することができる。

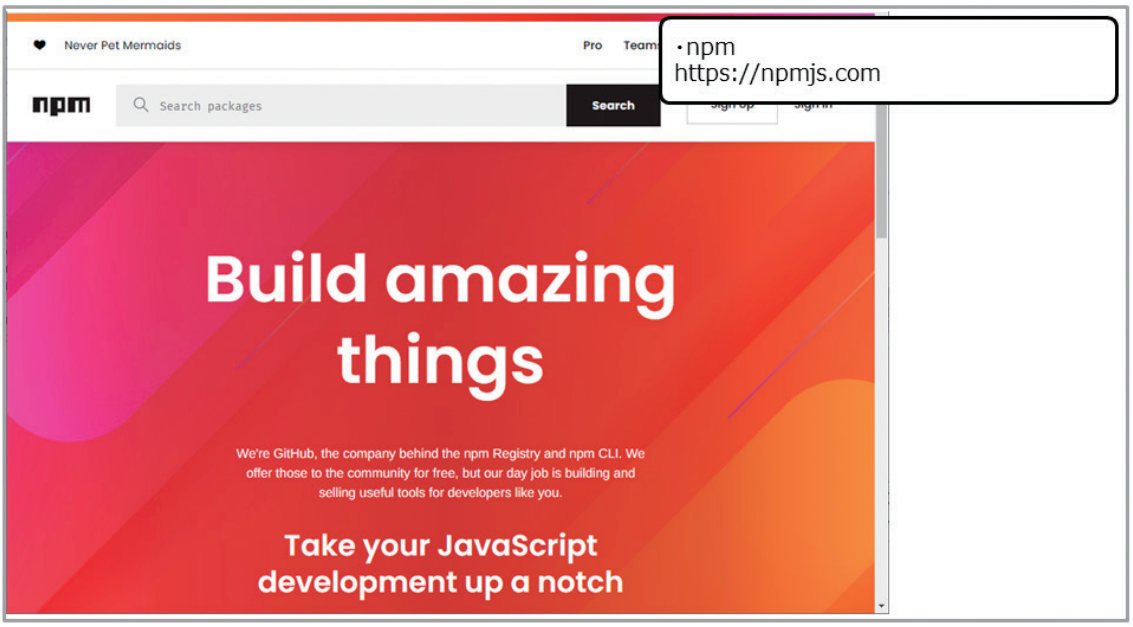
### 2-2. ExcelJSの導入方法

ExcelJSはnpm（Node Package Manager）と呼ばれる、パッケージ管理システムを利用してダウンロードする。npmは、Node.jsプロジェクトで使用されるさまざまなパッケージ（ライブラリやモジュール）を管理し、インストール、アップデート、削除などの操作を簡単に行えるようにするツールだ。

npmはNode.jsに含まれているため、Node.jsをインストールするとnpmもインストールされる。**【図4】**

● npm
https://npmjs.com

**図 4 npm**

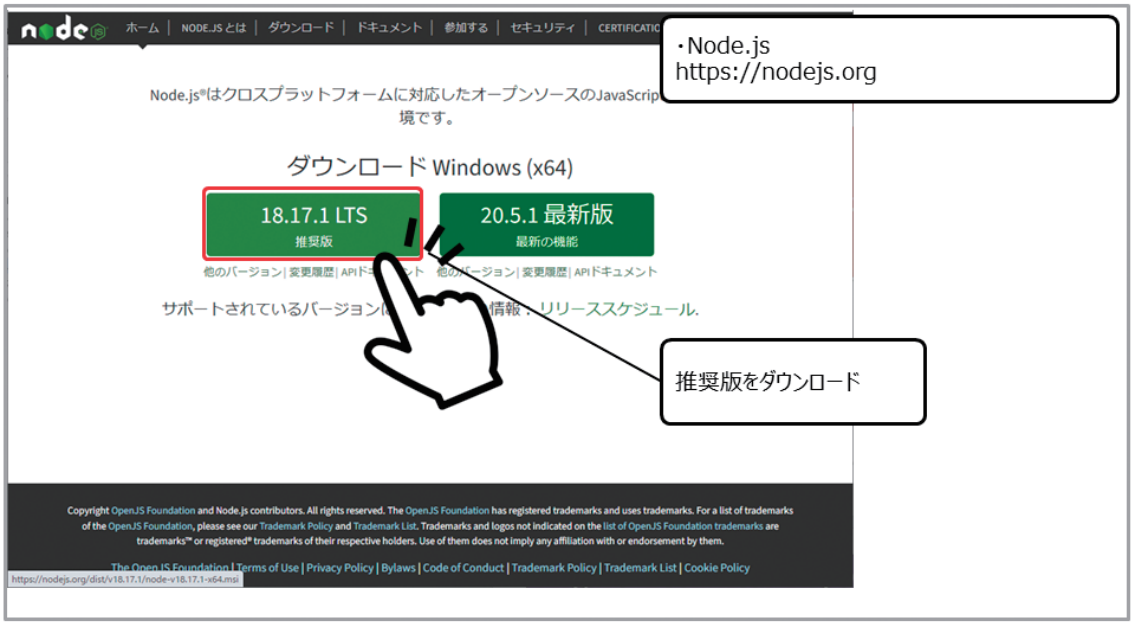


### 2-2-1. Node.jsの導入

Node.jsは、公式サイトからダウンロードしたインストーラーを実行することで導入可能である。**【図5】**

● Node.js
https://nodejs.org

**図 5　Node.js**




では、次の項で、npmを使用してExcelJSのファイルを取得する手順を説明する。

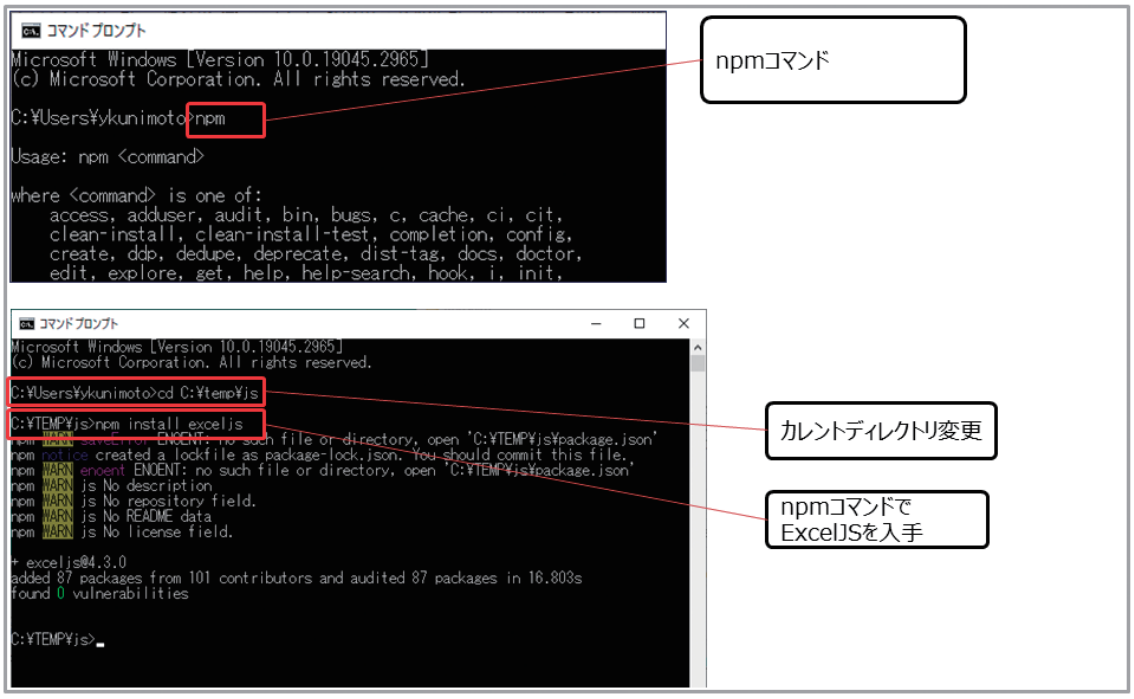
### 2-2-2. ExcelJSパッケージの導入

Node.jsのインストール後、コマンドプロンプトから、'npm'コマンドを入力することで、npmの導入を確認できる。

npm導入を確認した後、'cd'コマンドを使用して、ExcelJSパッケージをダウンロードするフォルダにカレントディレクトリを移動する。

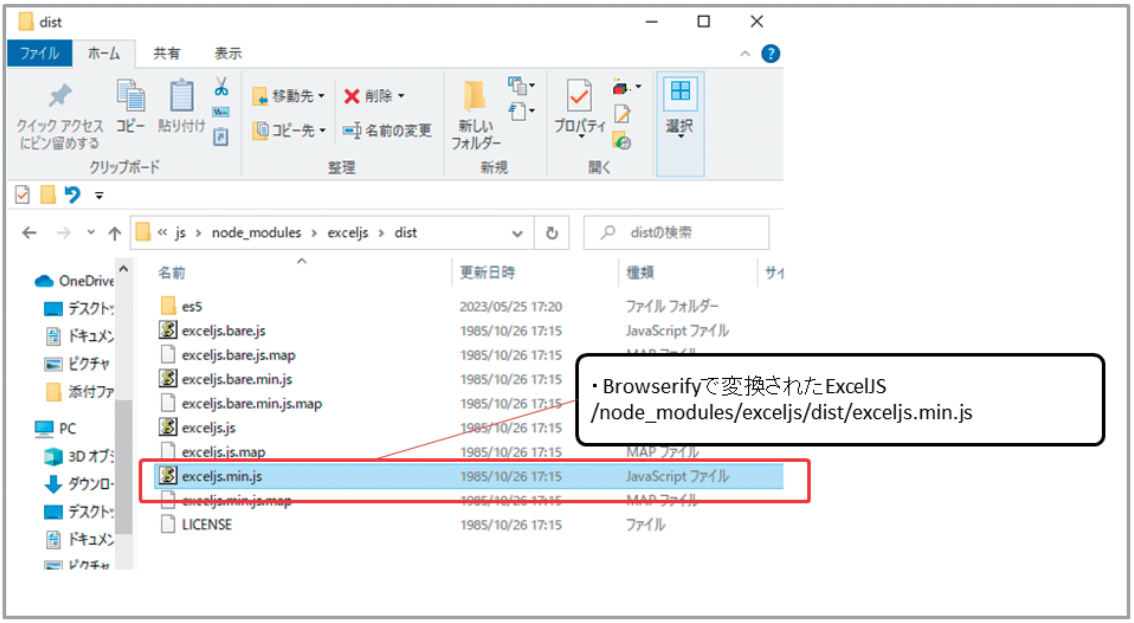
例えば、ExcelJSをダウンロードしたいフォルダがC:\temp\jsである場合、「cd　C:\temp\js」のコマンドを実行する。カレントディレクトリを変更後、次のコマンド'npm install　exceljs'を実行することで、ExcelJSパッケージがダウンロードされる。**【図6】**

## 図 6 npmコマンド




カレントディレクトリには、ExcelJSパッケージに関連する多くのファイルがダウンロードされる。ExcelJSパッケージには、あらかじめbrowserifyでまとめられたJavaScriptファイルが含まれており、クライアントサイドでExcelJSを使用する場合には、JavaScriptファイル(/node_modules/exceljs/dist/exceljs.min.js)を使用する【図7】。

## 図 7 Browserifyで変換されたExcelJS

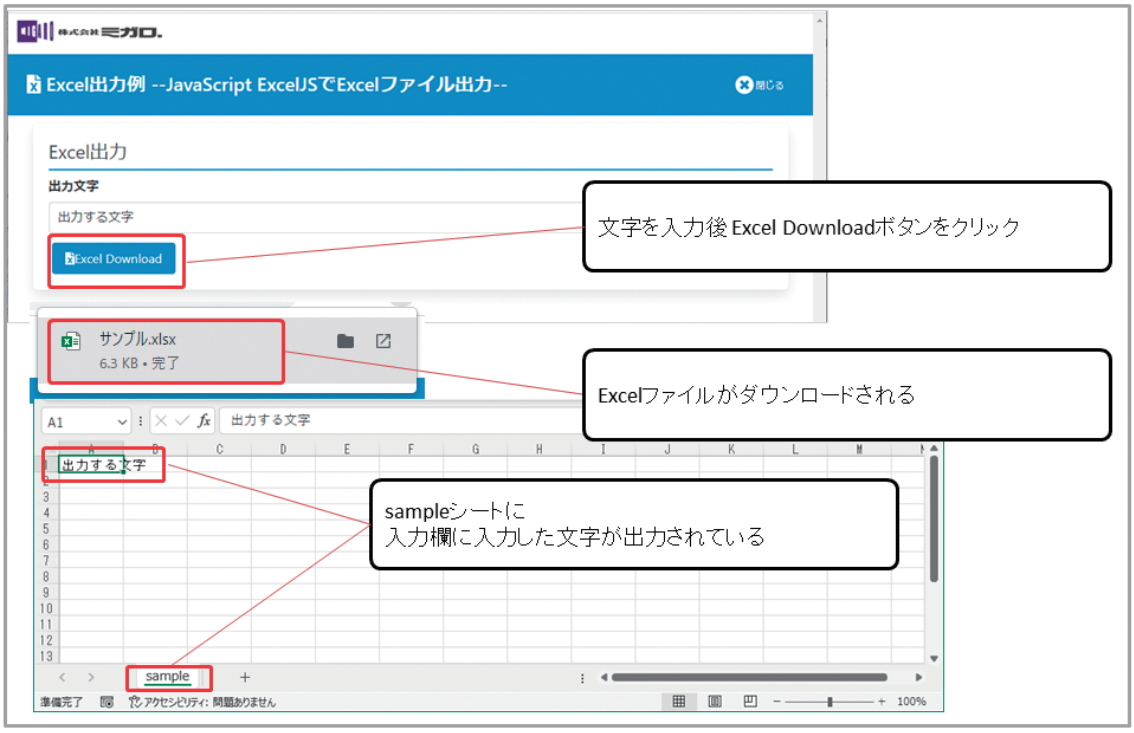



このJavaScriptファイルをブラウザ上で読込むことで、ExcelJSの機能をクライアントサイドで利用できるようになる。

### 2-3. ExcelJSの利用方法

ExcelJSの利用方法を簡単なサンプルアプリSMP010を元に説明する。SMP010では、HTML（SMP010.html）に配置したボタンをクリック時に、入力欄の文字列をExcelファイルに書込み、Excelファイルを作成してダウンロードする。**【図8】**

**図 8　ExcelJSを使用した簡単な例 SMP010**




#### 2-3-1. exceljs.min.jsの配置と設定

前項でnpmから取得した'exceljs.min.js'をSmartPad4i（Cobos4i）環境に配置する。本稿では、サンプルのアプリを'/smartpad4i/html/SP4IREP23'ディレクトリに配置していると仮定し、同ディレクトリに新しい'js'フォルダを作成後'exceljs.min.js'を配置する。

次に、配置した'exceljs.min.js'を外部参照する設定をHTMLに追加する。**【ソース1】**

**ソース 1**

**exceljs.min.jsの読込み（SMP010.HTML HTML 外部JavaScript）**

```
～省略～
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
polyfill/6.26.0/polyfill.js"></script>
  <script src="../../../smartpad4i/html/SP4IREP23/js/exceljs.min.js"></script>
  </body>
</html>
```

外部読込みのJavaScriptはHTML内のどこに追加しても動作するが、画面表示速度を向上させるために、通常は'</body>'タグの前に記述することを推奨している。

また、ExcelJSを読込む前に、CDNから'polyfill.js'を読込む。'polyfill.js'は、新しい機能やAPIを古いブラウザ環境でも動作させるためのコードであり、実行に必要な機能を持たない環境下では、不足機能を模倣したコードを実行することにより、ブラウザの互換性を保つ役割を果たす。

### 2-3-2.ボタンの配置と実行処理

Excelファイルをダウンロードするためのボタンを配置する。ボタンはアンカータグで設定し、ボタンの要素を識別するために、タグへカスタムデータ属性'data-exceldownload="true"'を設定する。**【ソース2】**

**ソース 2**

**ボタンの配置（SMP010.HTML HTML タグ追加）**

```
<a class="button is-info load-b ml-1" data-exceldownload="true" >
  <i class="fa fa-file-excel"></i>Excel Download
</a>
```

次に、ボタンをクリックした際の処理をJavaScriptで追加する。**【ソース3】**

**ソース 3**

**ボタンクリック時の処理（SMP010.HTML内 JavaScript）**

```
<script>
  /**
   * SmartPad4i(Cobos4i) 初期処理用のinitpage関数
   */
  function initpage(){
    // ダウンロード用ボタンの取得
    var excelButton = document.querySelector('[data-exceldownload]');
    /**
     * ダウンロードボタン クリックイベント
     */
    excelButton.addEventListener( "click" , function () {
      createExcel();
    } , false );

    /**
     *  xcelファイルを作成する非同期関数
     */
    async function createExcel(){
      // 新しいworkbookを作成
      const workbook = new ExcelJS.Workbook();
      // シート名 sampleのworksheetを追加
      workbook.addWorksheet("sample");
      // シート名 sampleのworksheetの取得
      const worksheet = workbook.getWorksheet("sample");
      // 1行目の1列目に入力欄(id="INP01")の文字を設定
      worksheet.getCell(1, 1).value =
         SP4i.getElementById('INP01').value;
      // 書込みした内容をwriteBufferメソッドで出力
      const uint8Array = await workbook.xlsx.writeBuffer();
      // Blobオブジェクトの作成
      const blob = new Blob([uint8Array],
        {type: 'application/octet-binary'});
      // ダウンロードするためにblobのURL作成
      const url = window.URL.createObjectURL(blob);
      // アンカーを作成してURL設定
      const a = document.createElement('a');
      a.href = url;
      // ダウンロードするファイル名を指定してダウンロード
      a.download = 'サンプル.xlsx';
    } a.click();
    }   a.remove();
  </
</script>
body>
```

①
②
③
④

■【ソース3】①

まず、【ソース3】①の処理でカスタムデータ属性'data-exceldownload'が設定された要素を取得する。'document.querySelector'はCSSセレクタに一致する最初の要素を取得するメソッドだ。このメソッドでボタンの要素を取得している。

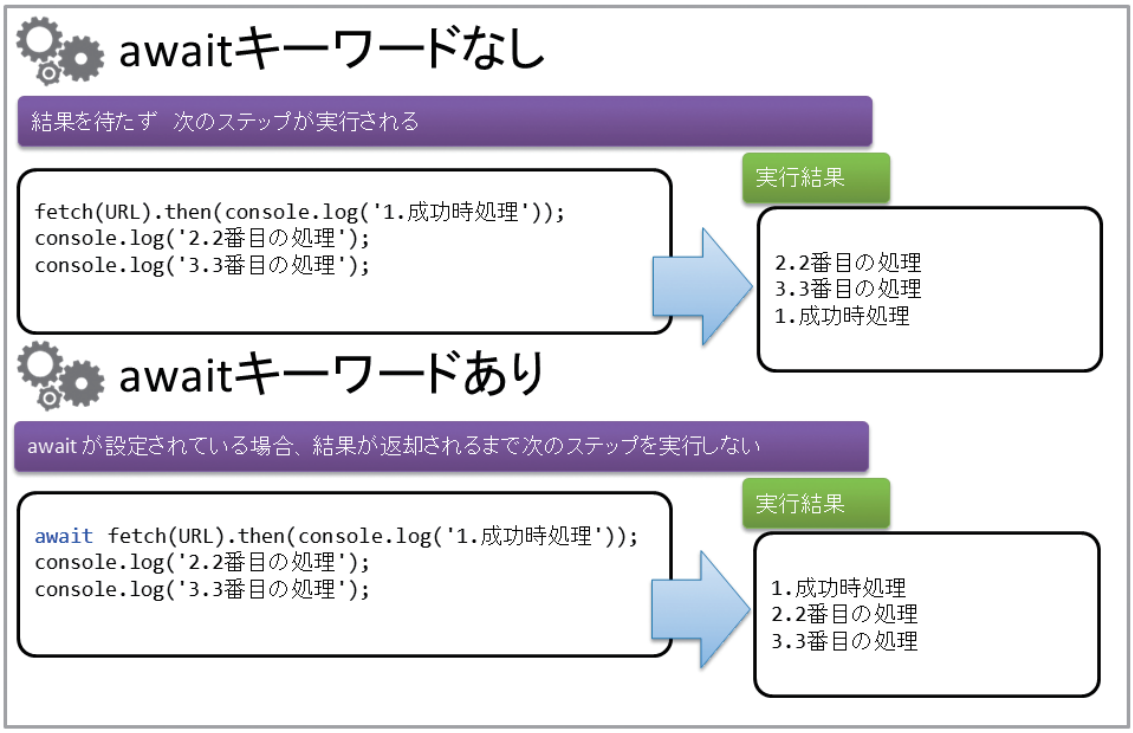
次に、取得したボタンの'addEventListener'メソッドを使用してイベントリスナー(イベントハンドラ)を追加する。ボタンがクリックされた際に非同期関数'createExcel'を実行するように設定する。

■【ソース3】②

'createExcel'は非同期関数として定義している。非同期関数の定義方法は、functionの前に'async'を記述するだけだ。非同期関数内で処理の終了を待機する'await'キーワードを利用するために使用している。'await'キーワードを利用することで、処理の終了を待機(同期的に処理)することが可能となり、非同期処理をより効果的に扱うことができる。

例えば、fetch APIは通信を実行する処理で、HTTPリクエストを行うXMLHttpRequestをより柔軟に使いやすくした機能である。fetchはPromiseオブジェクトを返却する非同期処理のため、'await'キーワードがない場合、fetchを実行後、処理結果が返却される前に次のステップへ進んでしまう。'await'キーワードがある場合、fetchの処理が完了するまで待機するため、fetchの処理結果を利用して後続処理を記述することが可能になる。【図9】

**図 9　awaitキーワードでの同期処理**




ExcelJSパッケージのExcelJSオブジェクトの'Workbook'メソッドを使用して、ワークブックを作成後、ワークブックオブジェクトの'addWorksheet'メソッドでワークブックに新しいシートを追加する。追加したシートオブジェクトはワークブックオブジェクトの'getWorksheet'メソッドを使用して取得する。

**■【ソース3】③**

追加したシートオブジェクトの'getCell'メソッドを使用して、シートの特定のセルにデータを設定する。'getCell'メソッドの第1引数は行番号、第2引数は列番号である。ワークブックオブジェクトに含まれるxlsxオブジェクトの'writeBuffer'メソッドを実行することで、オブジェクトに追加した内容をUnit8Array形式のデータとして取得している。

なお、'writeBuffer'メソッドは非同期処理されるため、'await'キーワードを設定して、同期的に処理している。

Unit8Array形式のデータは、JavaScriptのTypedArrayの一種で、8ビット符号なし整数（バイト）の配列を表現するために使用されるデータ型で、この型は主にバイナリデータやバイトストリームを操作するために使用される。

**■【ソース3】④**

次に、Unit8Array形式のデータをBlobに変換する。BlobはJavaScriptでバイナリデータや大きなデータの断片を表現するためのオブジェクトであり、Blobに変換することで、後続のダウンロード処理が可能となる。

ファイルのダウンロードを開始するために、URLオブジェクトの'createObjectURL'メソッドを使用して、URLを含む文字列を生成する。

この生成したURL文字列を動的に作成したアンカータグのhrefプロパティに設定し、クリック処理を実行すると、ExcelJSで生成したExcelファイルがブラウザを介してダウンロードされる。

この手順で、Excelファイルの生成とダウンロードが行える。

## 3.テンプレートを読込んで売上伝票を作成する方法

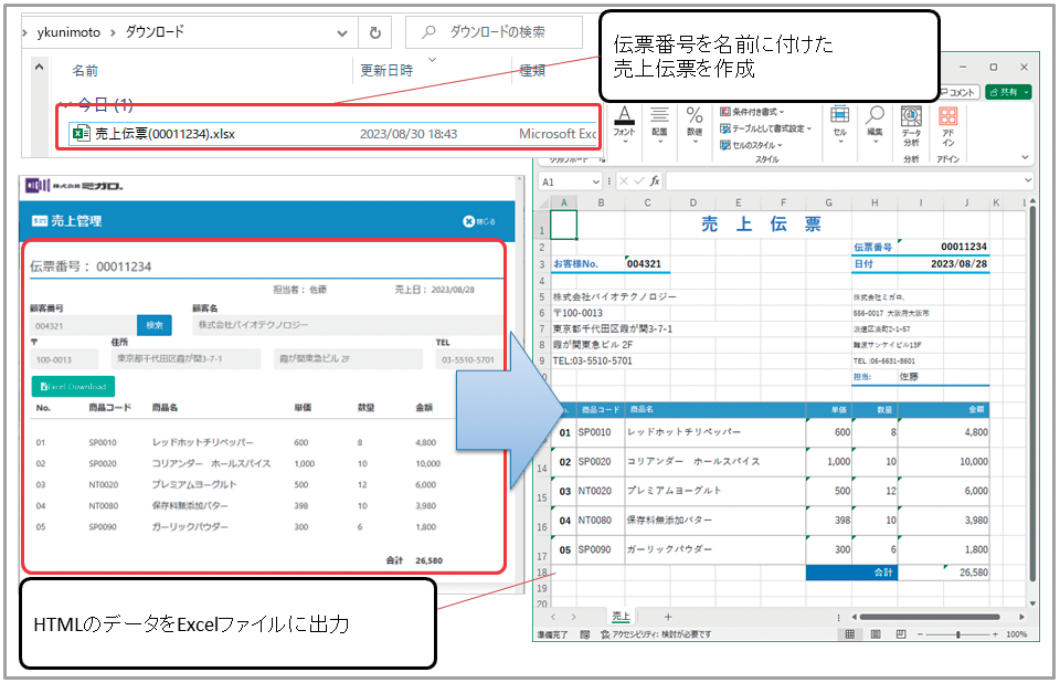
次は応用編として、Excelファイルのテンプレートを事前にサーバー上に配置しておき、ブラウザからテンプレートを読込み後、Excelファイルに売上情報を書込み、ダウンロードする方法を別のサンプルアプリSMP020で説明する。

サンプルアプリSMP020の画面は**【図10】**だ。

**図 10　サンプルアプリ SMP020 画面①**

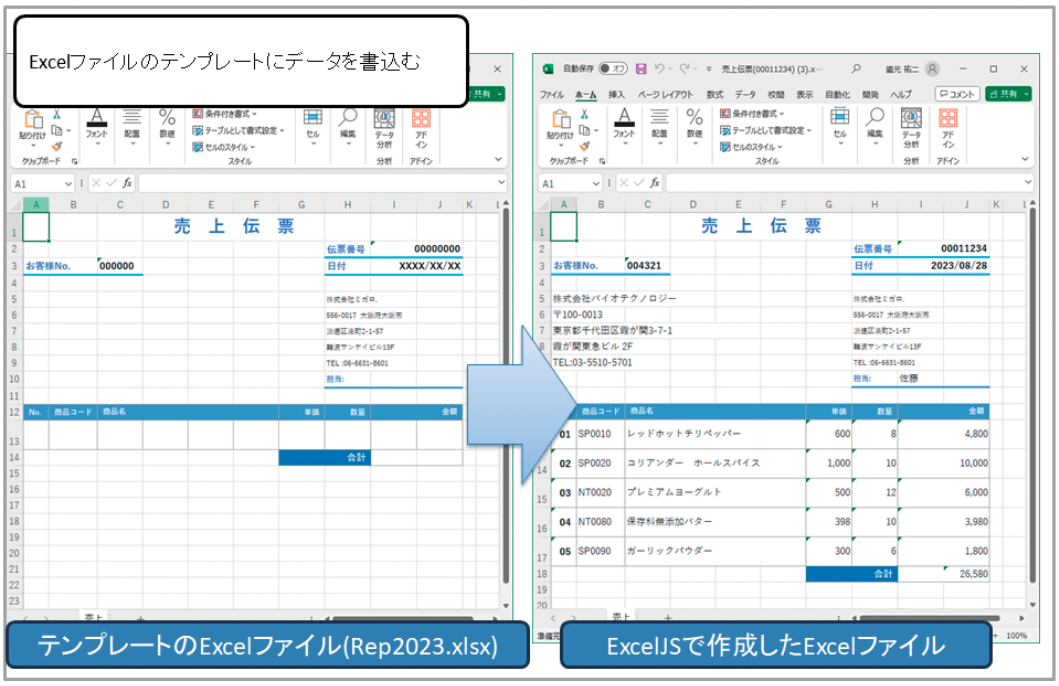
【図10】画面で「Excel　Download」のボタンをクリックすると、HTML上に出力されている情報を元に書込まれた売上伝票のExcelファイルが出力される。【図11】

## 図 11　サンプルアプリ SMP020 画面②




作成された売上伝票のExcelファイルはサーバー上に事前に作成したテンプレートのExcelファイル（Rep2023.xlsx）を元に作成する。【図12】

## 図 12　テンプレートのExcelファイルを元に作成



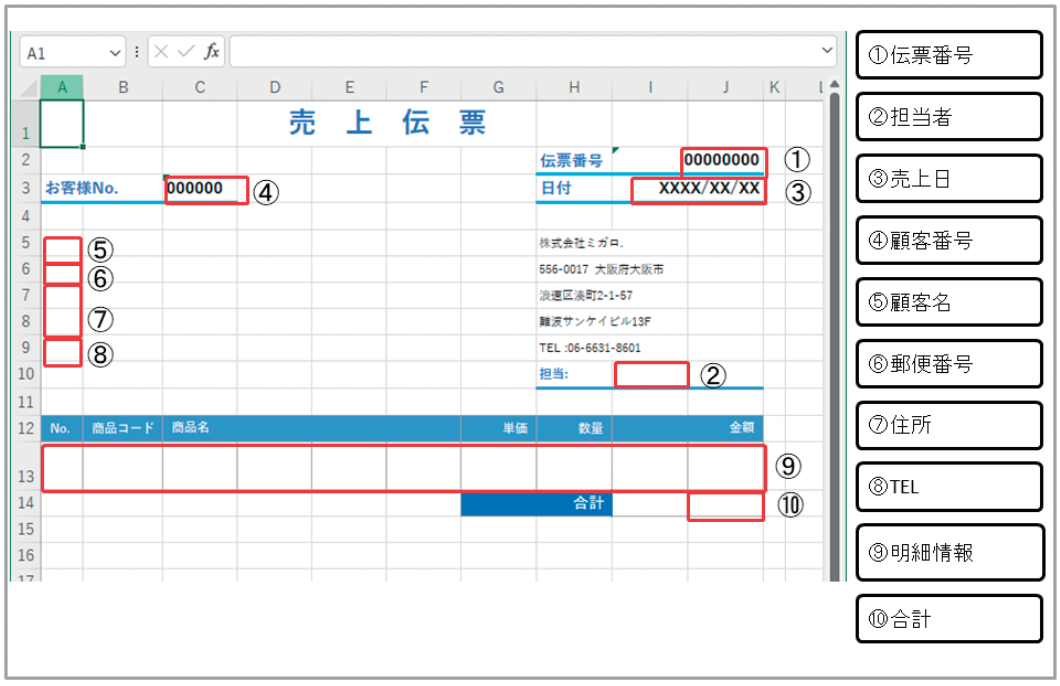
### 3-1. テンプレートの読込み方法

では、JavaScriptで読込むテンプレート（Rep2023.xlsx）を
Excelで作成する。HTMLから取得する情報は【図13】だ。

**図 13　HTMLから取得する情報**


【図13】の①～⑩の項目の情報をHTMLから取得する。取得した情報は【図14】テンプレートのExcelファイル（Rep2023.xlsx）の①～⑩の箇所に出力する。

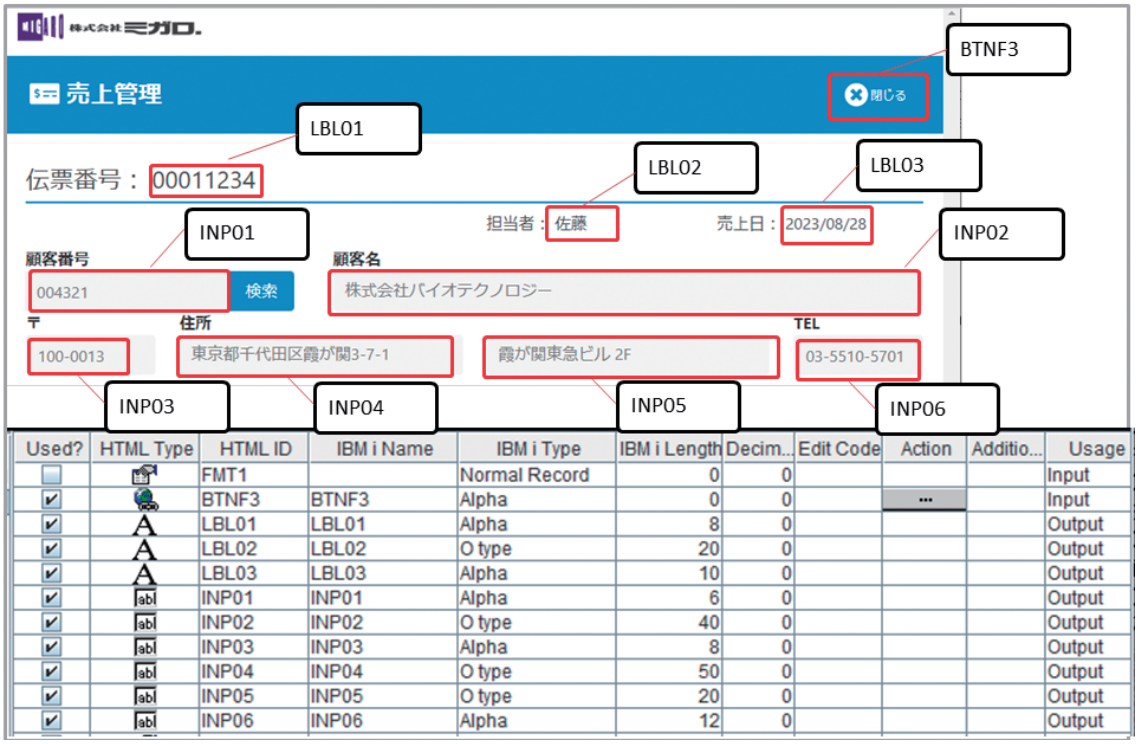
**図 14　テンプレートのExcelファイル（Rep2023.xlsx）**



**【図14】**⑨明細情報の箇所は、JavaScriptで行をコピーして挿入するため、書式設定などは1行のみ定義する。
作成したテンプレートのExcelファイル（Rep2023.xlsx）はJavaScriptで動的に読込むため、本稿ではHTMLファイルと同階層に配置する。

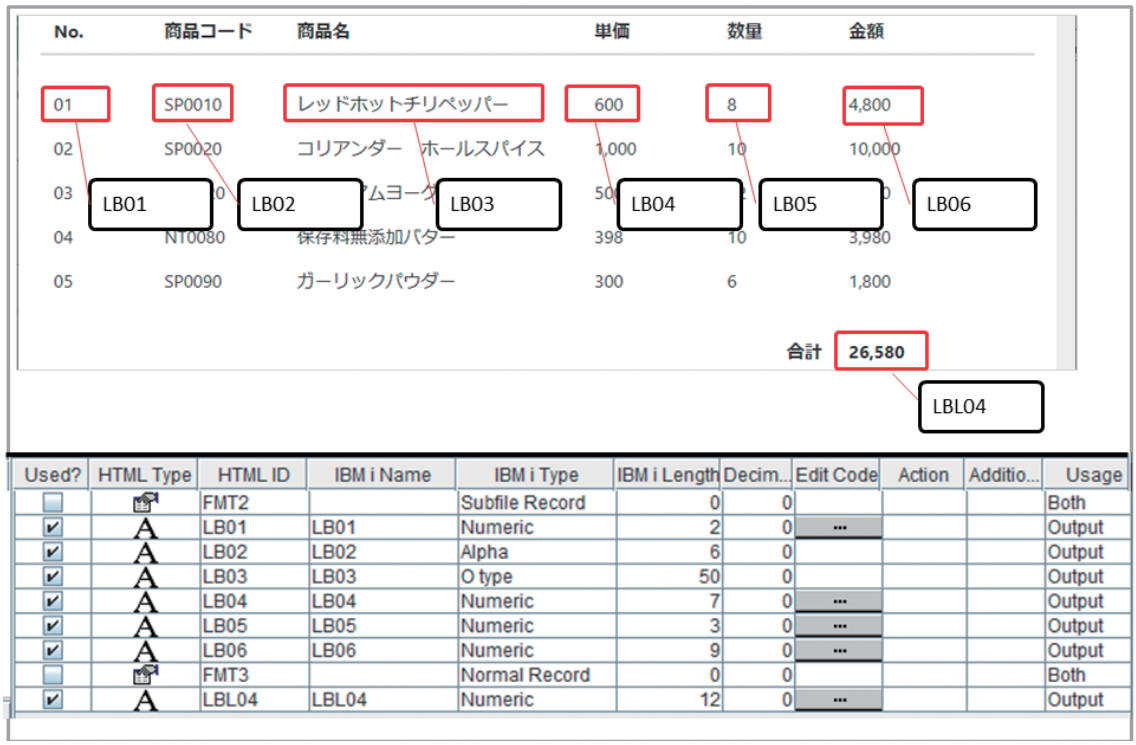
では、サンプルアプリSMP020のHTMLについて説明する。
サンプルアプリSMP020のDesignerの設定は**【図15】**
**【図16】**だ。

**図 15　サンプルアプリSMP020のDesigner定義①**




| Used? | HTML Type | HTML ID | IBM i Name | IBM i Type | IBM i Length | Decim... | Edit Code | Action | Additio... | Usage |
|---|---|---|---|---|---|---|---|---|---|---|
| ☐ | | FMT1 | | Normal Record | 0 | 0 | | | | Input |
| ☑ | | BTNF3 | BTNF3 | Alpha | 0 | 0 | | ... | | Input |
| ☑ | | LBL01 | LBL01 | Alpha | 8 | 0 | | | | Output |
| ☑ | | LBL02 | LBL02 | O type | 20 | 0 | | | | Output |
| ☑ | | LBL03 | LBL03 | Alpha | 10 | 0 | | | | Output |
| ☑ | | INP01 | INP01 | Alpha | 6 | 0 | | | | Output |
| ☑ | | INP02 | INP02 | O type | 40 | 0 | | | | Output |
| ☑ | | INP03 | INP03 | Alpha | 8 | 0 | | | | Output |
| ☑ | | INP04 | INP04 | O type | 50 | 0 | | | | Output |
| ☑ | | INP05 | INP05 | O type | 20 | 0 | | | | Output |
| ☑ | | INP06 | INP06 | Alpha | 12 | 0 | | | | Output |

## 図 16　サンプルアプリSMP020のDesigner定義②




| Used? | HTML Type | HTML ID | IBM i Name | IBM i Type | IBM i Length | Decim... | Edit Code | Action | Additio... | Usage |
|---|---|---|---|---|---|---|---|---|---|---|
| ☐ | | FMT2 | | Subfile Record | 0 | 0 | | | | Both |
| ☑ | A | LB01 | LB01 | Numeric | 2 | 0 | ... | | | Output |
| ☑ | A | LB02 | LB02 | Alpha | 6 | 0 | | | | Output |
| ☑ | A | LB03 | LB03 | O type | 50 | 0 | | | | Output |
| ☑ | A | LB04 | LB04 | Numeric | 7 | 0 | ... | | | Output |
| ☑ | A | LB05 | LB05 | Numeric | 3 | 0 | ... | | | Output |
| ☑ | A | LB06 | LB06 | Numeric | 9 | 0 | ... | | | Output |
| ☐ | | FMT3 | | Normal Record | 0 | 0 | | | | Both |
| ☑ | A | LBL04 | LBL04 | Numeric | 12 | 0 | ... | | | Output |

設定した、項目にIBMiプログラム（RPG）から値を出力している。
HTML側の記述としては、ExcelJSを使用するために**【ソース1】**の外部読込みと、ボタンクリック時にファイルをダウンロードするため**【ソース2】**と同じようにボタンを追加している。
また、明細の各項目にJavaScriptでアクセスできるようにするため、カスタムデータ属性を設定した。**【ソース4】**

## ソース 4

### 明細の定義　（SMP020.HTML HTML カスタム属性定義）

```
<table class="table is-fullwidth is-hoverable" id="SFL01">
  <tbody>
    <tr>
      <td id="LB01" data-dwn="saleno" style="width:100px">01</td>
      <td id="LB02" data-dwn="salecd" style="width:120px" > </td>
      <td id="LB03" data-dwn="salename"  > </td>
      <td id="LB04" data-dwn="saleprice" style="width:120px"> </td>
      <td id="LB05" data-dwn="saleunit" style="width:110px"> </td>
      <td id="LB06" data-dwn="saleamount" style="width:180px"> </td>
    </tr>
  </tbody>
</table>
```

'data-dwn="フィールド項目"'のように定義しており、カスタムデータ属性を設定することで、JavaScriptの'document.querySelectorAll'メソッドから要素にアクセスすることが可能となる。

サンプルアプリSMP020では【図10】のようにダウンロード用ボタンを配置している。これは前項のサンプルアプリSMP010と同じ仕組みで、ボタンがクリックされた際に、Excelファイルを出力する非同期関数を呼出すように設定した。クリック時の処理は【ソース5】だ。

## ソース 5

### ボタンクリック時の処理① (SMP020.HTML内 JavaScript)

```
<script>
  /**
   * SmartPad4i(Cobos4i) 初期処理用のinitpage関数
   */
  function initpage(){
    // ダウンロード用ボタンの取得
    var excelButton = document.querySelector('[data-exceldownload]');

    /**
     * ダウンロードボタン クリックイベント
     */
    excelButton.addEventListener( "click" , function () {
      createExcel();
    } , false );

    /**
     * Excelファイルを作成する非同期関数
     */
    async function createExcel(){
      // HTMLから情報取得
      let d_No     = SP4i.getElementById('LBL01').textContent; // ①伝票番号
      let d_Tanto  = SP4i.getElementById('LBL02').textContent; // ②担当者名
      let d_Date   = SP4i.getElementById('LBL03').textContent; // ③売上日
      let d_CCd    = SP4i.getElementById('INP01').value;       // ④顧客番号
      let d_CName  = SP4i.getElementById('INP02').value;       // ⑤顧客名
      let d_CZip   = '〒' + SP4i.getElementById('INP03').value;// ⑥郵便番号
      let d_CAddr1 = SP4i.getElementById('INP04').value;       // ⑦住所1
      let d_CAddr2 = SP4i.getElementById('INP05').value;       // ⑦住所2
      let d_CTel   = 'TEL:'+SP4i.getElementById('INP06').value;// ⑧TEL
      let d_All    = SP4i.getElementById('LBL04').textContent; // ⑩売上合計

      // テンプレートのExcelファイルパス
      const EXCEL_URL = '/smartpad4i/html/SP4IREP23/Rep2023.xlsx';
      // テンプレートのExcelファイルを読込み、バイト配列で取得
      const existingExcelBytes = await fetch(EXCEL_URL).then(res =>
         res.arrayBuffer());
      // バイト配列で取得したExcelファイルを8ビット符号なし整数値の配列に変換
      const exceldata = new Uint8Array(existingExcelBytes);
      // workbookを作成
      const workbook = new ExcelJS.Workbook();
      // テンプレートのExcelファイルのデータを読込み
      await workbook.xlsx.load(exceldata);
      // テンプレートで設定したシート取得
      const worksheet = workbook.getWorksheet('売上');
      // シートの用紙サイズ、縦横設定
      worksheet.pageSetup = { paperSize: 9, orientation:'portrait'};
      // ページ 余白設定 (単位inch)
      worksheet.pageSetup.margins = {
        left: 0.23, right: 0.23,
        top: 0.75 , bottom: 0.75 ,
        header: 0.31, footer: 0.31
      };

      // ソース6に続く
```

**■【ソース5】①**

SMP010サンプルアプリと同じ仕組みでボタンクリック時にExcelファイルを作成する非同期処理関数を呼び出している。

**■【ソース5】②**

Excelファイルに書込む内容を取得している。SmartPad4iのメソッド'SP4i.getElementById'を使用して、各id属性の要素のテキストや値を取得する処理だ。

spanタグの場合は'textContent'プロパティで出力されているテキストを取得し、inputタグの場合は、'value'プロパティで入力値を取得する。

**■【ソース5】③**

テンプレートのExcelファイルを読込む処理になる。
テンプレートの読込みは、JavaScriptのfetch　APIを使用する。
HTTPリクエストの処理はサーバーへのリクエスト処理後、HTTPレスポンスを取得するまでに時間が掛かる場合もあるため、'await'キーワードを使用して呼出すことで、HTTPレスポンスが返却されるまでの処理を同期的に実行している。
fetchの第1引数には取得したいリソースを指定する。テンプレートのExcelファイルをHTTP経由で取得するため、URLを指定する。fetchの処理はPromiseオブジェクトを返却する。

Promiseオブジェクトは非同期処理を簡潔かつ効果的に扱うための仕組みである。fetchのチェーンメソッドとして'then'を指定すると、処理が成功した場合（Promiseオブジェクトに処理成功のステータスが設定された場合）に'then'メソッド内の処理が実行される。
thenメソッド内ではレスポンスの結果を'arrayBuffer'メソッドでバイト配列として'existingExcelBytes'変数に取得している。
取得した、'existingExcelBytes'変数のデータをUnit8Arrayで変換後、ワークブックオブジェクトに含まれるxlsxオブジェクトの'load'メソッドを使用して、テンプレートのExcelファイルを読込む。

**■【ソース5】④**

テンプレートのExcelファイルに定義されている売上シートをワークブックオブジェクトの'getWorksheet'メソッドを使用して取得している。
シートを取得後、シートオブジェクトの'pageSetup'メソッドで、用紙サイズ（'paperSize'）と縦横（'orientation'）を指定後、さらに'worksheet.pageSetup.margins'オブジェクトを指定することで、ページの余白設定を変更する。Excelの余白表示はcm単位であるのに対して、'margins'に設定する単位はインチ（inch）単位であるため注意が必要だ。

### 3-2. 帳票出力方法

取得したデータをExcelファイルに書込む処理が**【ソース6】**だ。

**ソース 6**

**ボタンクリック時の処理② (SMP020.HTML内 JavaScript)**

```
<script>
      ～省略～
  /**
   * Excelファイルを作成する非同期関数
   */
  async function createExcel(){
        ～省略～
    worksheet.getCell(2 , 9).value = d_No;          // ①伝票番号
    worksheet.getCell(10, 9).value = d_Tanto;       // ②担当者名
    worksheet.getCell(3 , 9).value = d_Date;        // ③売上日
    worksheet.getCell(3 , 3).value = d_CCd;         // ④顧客番号
    worksheet.getCell(5 , 1).value = d_CName;       // ⑤顧客名
    worksheet.getCell(6 , 1).value = d_CZip;        // ⑥郵便番号
    worksheet.getCell(7 , 1).value = d_CAddr1;      // ⑦住所1
    worksheet.getCell(8 , 1).value = d_CAddr2;      // ⑦住所2
    worksheet.getCell(9 , 1).value = d_CTel;        // ⑧TEL
    worksheet.getCell(14 , 10).value = d_All;       // ⑩売上合計

    // ⑨明細情報 **********************

    let no_arr=document.querySelectorAll('[data-dwn="saleno"]');          //明細No.
    let cd_arr=document.querySelectorAll('[data-dwn="salecd"]');          //商品コード
    let name_arr=document.querySelectorAll('[data-dwn="salename"]');      //商品名
    let price_arr=document.querySelectorAll('[data-dwn="saleprice"]');    //価格
    let unit_arr=document.querySelectorAll('[data-dwn="saleunit"]');      //数量
    let amount_arr=document.querySelectorAll('[data-dwn="saleamount"]');//金額

    // データを書込む起点設定
    const startRow = 13; // 明細開始行番号
    const startCol =  1; // 明細開始列番号
    let row         = {};
    let rownum      =  0;
    // Excelの開始行をデータの数コピー
    worksheet.duplicateRow(startRow , no_arr.length - 1 , true);

    // シートに書き込み（明細行数でループ処理）
    for (var i=0;i < no_arr.length; i++) {
      let pos = startRow + i;
      row = worksheet.getRow(pos);
      // セルのマージ 商品名 3列目～6列目
      worksheet.mergeCells(pos , 3 , pos , 6);
      // セルのマージ 金額   9列目～10列目
      worksheet.mergeCells(pos , 9 , pos , 10);
      row.getCell(startCol).value     = no_arr[i].textContent;    //明細No.
      row.getCell(startCol + 1).value = cd_arr[i].textContent;    //商品コード
      row.getCell(startCol + 2).value = name_arr[i].textContent;  //商品名
      row.getCell(startCol + 6).value = price_arr[i].textContent; //価格
      row.getCell(startCol + 7).value = unit_arr[i].textContent;  //数量
      row.getCell(startCol + 8).value = amount_arr[i].textContent;//金額
    }

    // 書込みした内容をwriteBufferメソッドで出力
    const uint8Array = await workbook.xlsx.writeBuffer();
    // Blobオブジェクトの作成
    const blob = new Blob([uint8Array],
      {type: 'application/octet-binary'});
    // ダウンロードするためにblobのURL作成
    const url = window.URL.createObjectURL(blob);
    // アンカーを作成してURL設定
    const a = document.createElement('a');
    a.href = url;
    // ダウンロードするファイル名に伝票番号を設定してダウンロード
    a.download = '売上伝票(' + d_No  + ').xlsx';
    a.click();
    a.remove();
  }
}
</script>
```

**■【ソース6】①**
ワークシートオブジェクトの'getCell'メソッドを使用して、対象のセルに書込み処理を実行している。

**■【ソース6】②**
明細部分のデータを'document.querySelectorAll'メソッドで取得している。'document.querySelectorAll'はCSSセレクタを指定することで、対象のノードリストを取得することができる。

**■【ソース6】③**
明細データを書込む開始行や開始列を定義している。最後の処理では、ワークシートオブジェクトの'duplicateRow'メソッドを使用して、Excelの行をコピーしている。'duplicateRow'メソッドの第1引数は、コピーを開始する対象行番号、第2引数は、コピーする回数、第3引数は行のコピー時に新しい行を挿入するか、置換するかのフラグを指定する。サンプルでは、コピーした行を挿入したいため、第3引数に'true'を設定する。

**■【ソース6】④**
明細行を書込んでいる。ワークシートオブジェクトの'getRow'メソッドを使用して、指定した行番号の行オブジェクトを取得する。取得した行オブジェクトはrow変数に格納される。取得したrowオブジェクトのセルは全て1列ずつの状態のため、ワークシートオブジェクトの'mergeCells'メソッドでセルをマージ(結合)する。
'mergeCells'メソッドの第1引数は結合する開始セルの行番号、第2引数は、列番号、第3引数には結合する終了セルの行番号、第4引数には列番号を指定する。
この方法で、商品名の3列目～6列目と金額の9列目～10列目のセルが結合される。

データの書込みは、rowオブジェクトの'getCell'メソッドに列番号を指定することで対象のセルを取得することができるため、valueプロパティに各ノードリスト項目の'textContent'(テキスト)を設定することで、データを書込む。

**■【ソース6】⑤**
サンプルアプリSMP010と同様に、書込んだワークブックオブジェクトを配列データに変換後、ダウンロードする処理を記述している。また、サンプルアプリSMP020では、ダウンロードするファイル名に伝票番号を設定する。
このようにして、Excelファイルのテンプレートを活用し、データを書込み、ダウンロードする処理が実行できる。

## 4.おわりに

JavaScriptは進化し続けており、ブラウザで実現できる範囲が増えてきている。
本稿では、Excelファイルを出力するOSSを紹介したが、インターネット上には、クライアントサイドのJavaScriptのみでMicrosoft Wordファイルや、PDFファイルを出力できるOSSのライブラリも存在する。
クライアントサイドのJavaScriptのみで実装する場合、既存のSmartPad4i(Cobos4i)アプリにも簡単に組込むことができるため、OSSを活用して、システム開発に役立てて頂ければ幸いである。

Valence

# [Edit Grid]ウィジェット活用術

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**
**2. [Edit Grid]ウィジェットの基本**
**3. RPG連携テンプレート"EXNABVAL"の使用方法**
**4. [Edit Grid]ウィジェット活用術**
**5. 複数行明細入力フォーム作成テクニック**
**6. さいごに**

## 1.はじめに

IBM i(AS/400)環境に特化したモダナイゼーションツールであるValenceには、ローコード開発機能である「Nitro App Builder」(以下「App Builder」と記載)が搭載されている。
「App Builder」は、[①データソースの作成]-[②ウィジェットの定義]-[③アプリケーション作成]という3ステップで容易にアプリケーションを作成できるのが特徴である。その中でも、UI(ユーザーインターフェース)の肝となるが、[②ウィジェットの定義]である。Valence6.2には、全16種類のウィジェット(部品)が用意されており、データソースと関連付けてウィジェットを定義する事で、表現力の高いGUI画面が実装できる。

参照/照会系のWebアプリ開発では、一覧形式(Grid)でデータ出力する[Grid]ウィジェットや、クロス集計表を実現する[Pivot Grid]ウィジェットを使用するのが一般的である。
また、登録/更新系のWebアプリ開発では、[Grid]ウィジェットの機能にデータの更新機能を追加した[Edit Grid]ウィジェットを使用する事ができる。
この[Edit Grid]ウィジェットについて、「App Builder」が登場した2018年当時は、シンプルなデータ編集機能しか実装されていなかった。その為、このウィジェットは、簡単なマスタメンテナンスアプリ位でしか利用できないと認識の方もいるだろう。しかし、Valenceの進化と共に[Edit Grid]ウィジェットの機能も大幅に進化しており、現在は多彩な編集処理が可能である。
ただ、これまで[Edit Grid]ウィジェットの機能を体系的に纏めた事が無かった為、今回改めて本稿を執筆する事にした。

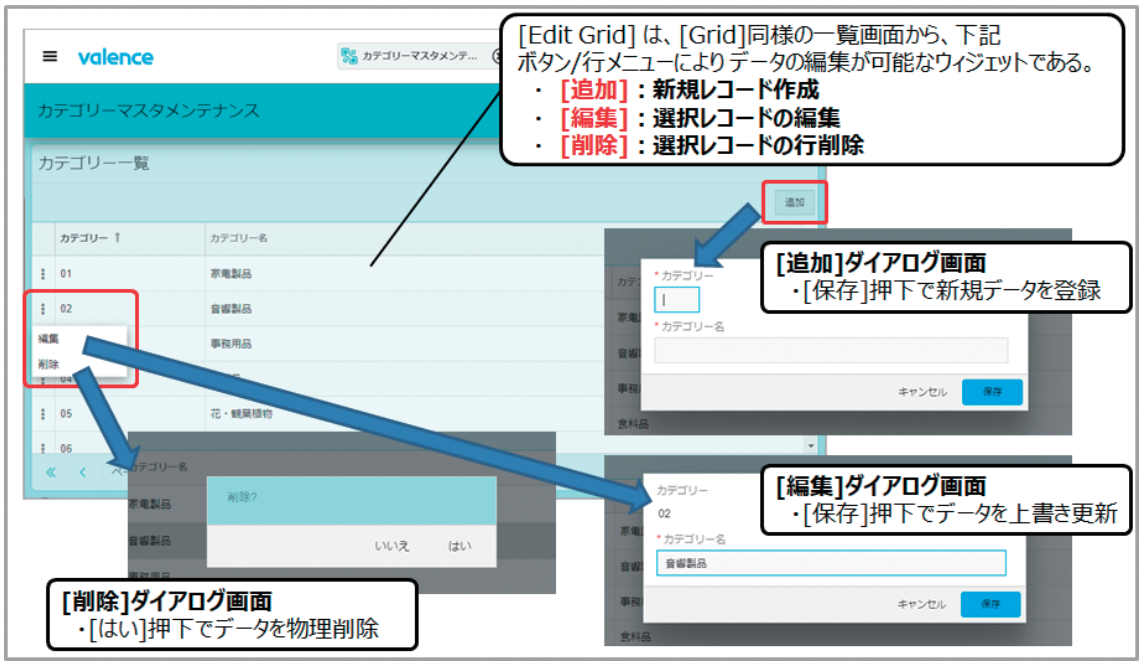
本稿では、登録/更新系アプリ開発には欠かせない[Edit Grid]ウィジェットに焦点を絞り、基本的な使用方法から、応用的な活用術までを具体例を交えて紹介したい。なお、本稿は2023年8月現在の最新版Valence6.2(Build 6.2.20230808.0)を前提とした内容となっている。旧バージョンを利用している場合、無償で製品バージョンアップが可能である為、最新版を導入の上で試してほしい。
なお、本稿で提示している各サンプルアプリは、以下より、全てダウンロードできるようにしている。ぜひサンプルアプリを導入して実際の動きを確認する事をお勧めしたい。

【サンプルアプリダウンロードURL】
https://www.migaro.co.jp/valsample/kozaki2023.zip
※サンプルダウンロードには、Valenceメンテナンスサイトへのログインユーザー・パスワードが必要

## 2.[Edit Grid]ウィジェットの基本

「App Builder」における[Edit Grid]ウィジェットは、データを一覧形式で表示可能な[Grid]ウィジェットの機能に、選択行のデータに対する編集/削除の機能や新規レコード追加の機能が付加されたデータ編集用ウィジェットである。例えば、**【図1】**は、シンプルにコードと名称のみを持つ『カテゴリーマスタ』を元に[Edit Grid]ウィジェットを使用して作成したカテゴリーマスタメンテナンスアプリの実行画面である。

**図 1　カテゴリーマスタメンテナンスアプリ実装例**




ウィジェットの外観は、[Grid]と同様だが、画面右上部には[追加]ボタンが、各行の左側には、行メニュー([⋮])が追加されている事が分かる。[追加]をクリックすると、ポップアップで空のダイアログ画面が開き新規レコードを登録できる。また、行メニューから[編集]をクリックすると、選択行の編集用ダイアログ画面からデータの編集が行える。[削除]をクリックすると、確認ダイアログが表示され、[はい]を選択すると、対象レコードを削除できる。これが[Edit Grid]ウィジェットの基本動作である。

このサンプルアプリの[Edit Grid]ウィジェット設計画面を開くと、[Grid]ウィジェット同様の[カラム]/[設定]/[フィルタ]タブに加え、[編集]タブが用意されている事がわかる。**【図2】**

## 図 2　Edit Grid 設計画面 – [編集]タブ


[編集]タブには、[追加時]及び[編集時]の入力対象フィールドが選択できるようになっている。**【図2】**の場合、[追加時]は、[カテゴリーコード]と[カテゴリー名]の2つが入力項目、[編集時]は、[カテゴリーコード]は読取り専用(表示のみ)とし、[カテゴリー名]のみ入力項目にするという定義となっている。なお、[削除時]の設定は、画面右側にある[設定]欄に含まれる[削除を許可]にチェックを付ければよい。

次に[Edit Grid]ウィジェットを使用した商品マスタメンテナンスアプリを紹介する。ここで使用するのは、**【図3】**のようなファイルレイアウトを持つ商品マスタである

## 図 3　商品マスタ　ファイルレイアウト

DSPFMT　　レ コ ー ド 設 計 書　　日付 23/08/28　時刻 19:45:21

物理ファイル TECREPLIB/MPRODP　様式名 MPRODR　レコード長 111
様式記述 商品マスタ
5= 詳細

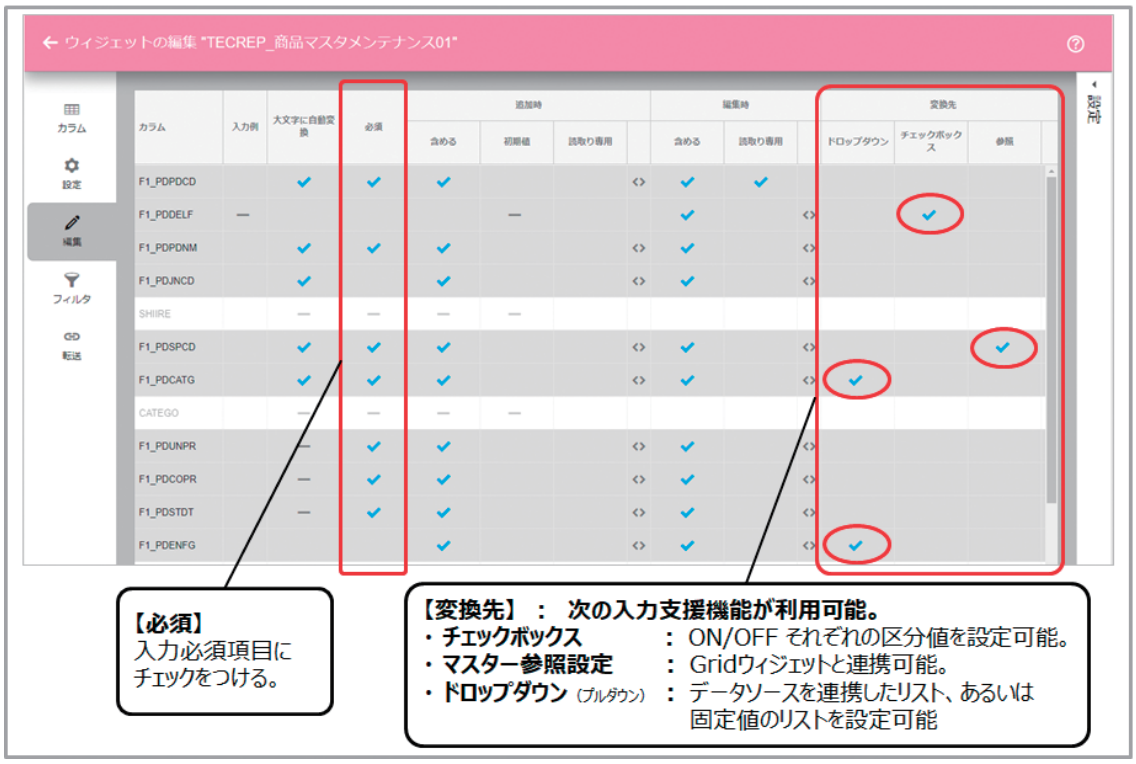
| 選択 | 項目名 | 桁数 | | 属性 | キー順 | 開始 | 終了 | テキスト記述／欄見出し |
|---|---|---|---|---|---|---|---|---|
| _ | PDDELF | 1 | | A | | 1 | 1 | 削除フラグ |
| _ | PDPDCD | 10 | | A | 1 ANN | 2 | 11 | 商品コード |
| _ | PDPDNM | 42 | | O | | 12 | 53 | 商品名 |
| _ | PDJNCD | 13 | | A | | 54 | 66 | ＪＡＮコード |
| _ | PDSPCD | 6 | | A | | 67 | 72 | 仕入先コード |
| _ | PDCATG | 2 | | A | | 73 | 74 | カテゴリー |
| _ | PDUNPR | 9 | 0 | P | | 75 | 79 | 販売単価 |
| _ | PDCOPR | 9 | 0 | P | | 80 | 84 | 仕入単価 |
| _ | PDSTDT | 8 | 0 | S | | 85 | 92 | 発売日 |
| _ | PDENFG | 1 | | A | | 93 | 93 | 取扱終了フラグ |
| _ | PDUPDT | 8 | 0 | S | | 94 | 101 | 最終更新日 |
| _ | PDUPUS | 10 | | A | | 102 | 111 | 最終更新者 |

**仕入先コード** ： 仕入先マスタ（MSUPLP）に登録された仕入先ＣＤとする。
**カテゴリー** ： カテゴリマスタ（MCATGP）に登録されたカテゴリＣＤとする。
**取扱終了フラグ** ： ブランク、"1"(生産終了)、"2"(在庫切れ)のいずれかとする。
**販売単価、仕入単価** ： **販売単価＜仕入単価**は**エラー**となるよう**入力値チェック**を行う。
**最終更新日** ： データ登録/変更時に、システム日付値数値8桁をセットする。
**最終更新者** ： データ登録/変更時に、ログオンしているユーザーIDをセットする。

商品マスタの項目に含まれる[PDCATG](カテゴリー)は、先に紹介したカテゴリーマスタと関連付けする。同様に、[PDSPCD](仕入先コード)は仕入先マスタと関連付ける。[PDENFG](取扱終了フラグ)の値は、ブランクか“1”(生産終了)、“2”(在庫切れ)のいずれかとする。また、[PDUNPR](販売単価)と[PDCOPR](仕入単価)は、登録・更新時に大小チェックを行い、販売単価<仕入単価の場合はエラーとする入力チェックを設けたい。さらに画面上の表示項目とはしないが、データ登録・更新時には、[PDUPDT](最終更新日)、[PDUPUS](最終更新者)にも自動的に値をセットしたい。

商品マスタメンテナンスで使用する[Edit Grid]ウィジェットの設計画面は、**【図4】**である。

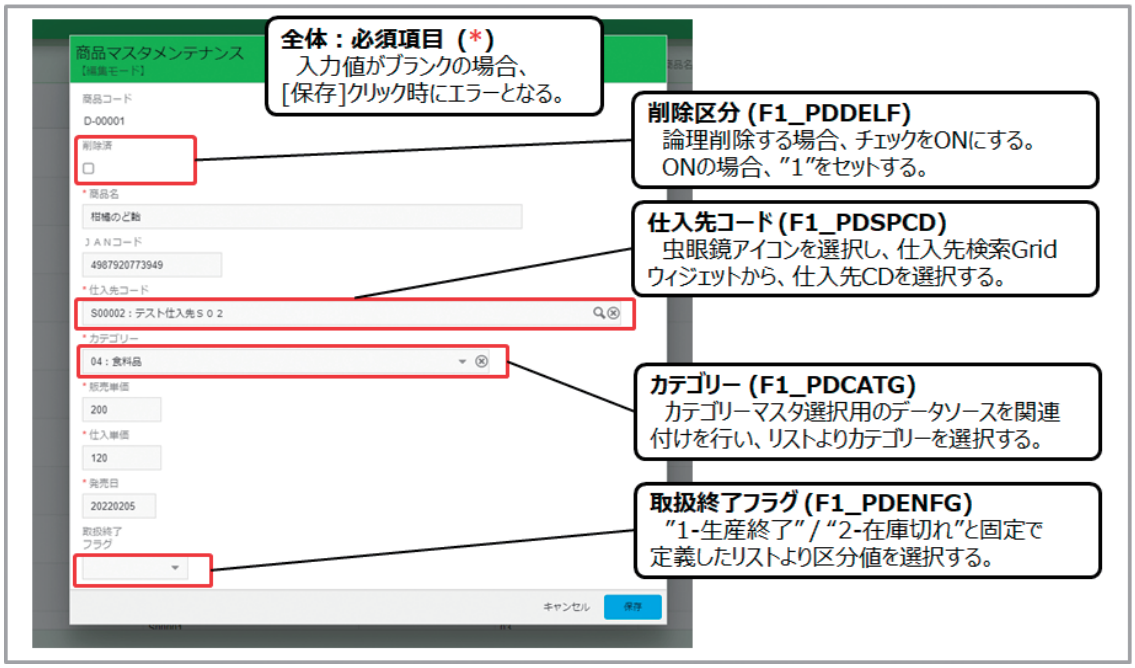
**図 4　商品マスタメンテナンス　Edit Grid設計画面 – [編集]**




入力必須項目は、単に[必須]チェックを付ければよい。また入力項目には、入力支援機能としてドロップダウン(プルダウン)、チェックボックス、あるいは[Grid]ウィジェットを関連付けしたマスタ検索画面の呼出しが定義可能である。ドロップダウンについては、リスト項目を別途定義したデータソースから作成する事や固定値の選択肢を直接定義する事ができる。詳細な定義内容は、実際のサンプルアプリの設計画面で確認してほしい。

作成したアプリケーションの実行画面は、**【図5】**である。設計画面で定義したとおりに、入力項目に入力支援機能が付加されている事がわかる。

**図 5　商品マスタメンテナンス実行画面 - 編集ダイアログ**




これで[Edit Grid]ウィジェットによる編集画面の定義は完了であるが、現状では、商品マスタメンテナンスアプリとして、以下の処理の実装が不足している。

- 販売単価と仕入単価の入力値妥当性チェック
- 新規登録時の商品コードキー重複チェック
- 非表示項目（最終更新日、最終更新者）の書き込み

このように、[Edit Grid]ウィジェットの設計画面では定義できないものは、どのように対処すればよいだろうか？「App Builder」の設計画面で定義できない独自のビジネスロジックや内部処理等の追加には、RPGプログラムを使用する。次章では、[Edit Grid]ウィジェットにおけるRPG追加手順を紹介する。

## 3. RPG連携テンプレート"EXNABVAL"の使用方法

「App Builder」におけるRPGプログラムは、1から全てを新規に作成する事も可能であるが、専用のAPIを利用する為に必要な各種定義を都度記述するのは面倒である。「App Builder」には、RPG連携の種類毎に合わせたテンプレートソースが用意されており、連携に必要な宣言等が予め定義されている為、このテンプレートソースを自身のソースライブラリにコピーして利用するのが一般的である。2023年8月現在の最新版であるValence6.2には、全部で8種類のテンプレートプログラムが用意されている。**【図6】**

## 図 6 Valence App Builder 連携RPGテンプレート

| ソースファイル名 | 機能 | 概要 |
|---|---|---|
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |

- **VALENCE6/QRPGLESRC**内に収録
  ※Valence6.2から完全なフリーフォーム形式で収録されている
  ※以前の形式のテンプレートは VALENCE6/QRRGLESRC2内に収録されている
- EXNABVAL以外は、全て**"process"**サブルーチンの中にユーザープログラムを記述すればよい。

本章でメインに紹介する"EXNABVAL"以外のテンプレートには、"process"サブルーチンが定義されている為、独自のユーザーロジックは、この中に記述すればよい。
なおRPG連携プログラムの基本的な作成方法やソースのコンパイル手順については、2019年度テクニカルレポート「Valence App Builder RPG連携テクニック」（https://www.migaro.co.jp/tr/no12/tech/12_01_06.pdf）で紹介しているのでそちらも参照してほしい。

では、ここから本題となる[Edit Grid]ウィジェットにおけるRPG追加手順を紹介する。ここで使用するテンプレートソースは、"EXNABVAL"である。このテンプレートは、[Edit Grid]ウィジェットで行追加/行編集/行削除の各種更新処理を実行する際に呼び出されるものである。
始めに「App Builder」上の設定箇所を見ておこう。[Edit Grid]ウィジェットの[編集]タブの中にある[設定]を開くと、オプション設定画面が開く。この中にある[検証]欄に呼出ししたいRPGのプログラムIDを記述すればよい。**【図7】**

## 図 7 Edit Grid – RPGプログラム設定方法

次にRPGロジックだが、この“EXNABVAL”は、追加/編集/削除の状態により、[更新前]、[更新後]、[エラー時]に処理を行う複数のサブルーチンが用意されている。**【図8】**

**図 8　EXNABVAL サブルーチン一覧**

| サブルーチン名 | 概要 |
|---|---|
| ProcessAdd | レコード登録前に　エラーチェックや 非入力項目へ値を設定するサブルーチン |
| ProcessDelete | レコード削除前に　エラーチェックを行うサブルーチン |
| ProcessEdit | レコード更新前に　エラーチェックや 非入力項目へ値を設定するサブルーチン |
| ProcessPostAdd | レコード登録後に、独自の追加ロジックを実行したい場合に使用するサブルーチン |
| ProcessPostDelete | レコード削除後に、独自の追加ロジックを実行したい場合に使用するサブルーチン |
| ProcessPostEdit | レコード更新後に、独自の追加ロジックを実行したい場合に使用するサブルーチン |
| ProcessErrAdd | レコード登録時に、エラーが発生した場合に独自のエラー処理を記述するサブルーチン |
| ProcessErrDelete | レコード削除時に、エラーが発生した場合に独自のエラー処理を記述するサブルーチン |
| ProcessErrEdit | レコード更新時に、エラーが発生した場合に独自のセラー処理を記述するサブルーチン |

今回のプログラムでは、行追加/行編集の更新前に行うエラーチェックと非表示項目への値の書き込みが目的となる為、“ProcessAdd”と“ProcessEdit”の各サブルーチンに処理を記述すればよい。例えば、“ProcessAdd”サブルーチンの実装例は、**【ソース1】**である。

**ソース 1　REP010:商品マスタメンテナンス更新チェック(一部抜粋)**

```
** ------------------------------------------------------------
p ProcessAdd      b
d                 pi
D*-----変数宣言
D INPDCD          S             10A
D RECCNT          S             10S 0
D INUNPR          S              9S 0
D INCOPR          S              9S 0
D SYSDAT          S               D   DATFMT(*ISO) INZ(*SYS)   1-①
D*
 /free
   //-----画面入力値を取得
   INPDCD = GetValue('MPRODP':'PDPDCD');            //商品ＣＤ
   INUNPR = %DEC(GetValue('MPRODP':'PDUNPR'):9:0);  //販売単価
   INCOPR = %DEC(GetValue('MPRODP':'PDCOPR'):9:0);  //仕入単価   1-②

   //-----商品ＣＤのキー重複エラーチェック
   exec sql  SELECT COUNT(*) INTO :RECCNT FROM MPRODP
             WHERE PDPDCD = :INPDCD;
   if RECCNT > 0;
     SendError('キーが重複しています':'F1_PDPDCD');
     return;
   endif;                                           1-③

   //-----販売単価と仕入単価の相関チェック
   if INUNPR < INCOPR;
     SendError('販売単価が仕入単価未満です':'F1_PDUNPR');
     return;
   endif;                                           1-④

   //-----非入力項目の値をセット
   SetValue('MPRODP':'PDUPDT':%CHAR(%DEC(SYSDAT:*ISO)));  //最終更新日
   SetValue('MPRODP':'PDUPUS':GetAppVar('nabUser'));      //最終更新者

 /end-free                                          1-⑤
p                 e
** ------------------------------------------------------------
```

ソースの先頭部分はサブルーチン内で使用する変数宣言部である。今回システム日付値を最終更新日にセットする為に、1-①のようなシステム日付値(日付型)を宣言した。
1-②は、登録/編集画面で入力された値を取得する処理である。GetValueメソッドを使用すればよい。このメソッドは取得対象となるファイル名とフィールド名を指定すれば、画面入力値が文字列として取得できる。従って、数値型の変数へ代入する場合には型変換(%DEC)が必要となる。
1-③は、キー項目の重複チェック処理である。ここではSQLを使用して1-②で取得した商品CDを持つデータのレコード件数を取得し、1件以上存在する場合はエラーとなる処理とした。SendErrorメソッドを使用すれば、ブラウザ側にエラーメッセージを渡して更新処理を中断する事ができる。
1-④は販売単価と仕入単価との大小エラーチェックである。
1-⑤は、非入力項目へ値をセットする処理である。SetValueメソッドを使用すればよい。このメソッドは、ファイル名とフィールド名そしてセットしたい値をパラメータに指定すればよい。値は、フィールドの属性に関わらず文字列としてセットする点に注意してほしい。
例えば、[PDUPDT](最終更新日)は数値8桁の定義だが、"%CHAR(%DEC(SYSDAT:*ISO))"として、日付型のシステム日付値を一旦8桁の数値に変換したものを更に文字列に変換した値をセットしている。また、[PDUPUS](最終更新者)には、「App Builder」に用意されたデフォルトのアプリ変数"nabUser"の取得値をセットしている。"nabUser"は、ログインしているValenceユーザーIDを表している。
このように[Edit Grid]ウィジェットの定義だけで実現できない部分は、RPGロジックを追加する事で制御できるので覚えておこう。

## 4. [Edit Grid]ウィジェット活用術

前章では、[Edit Grid]ウィジェットの基本的な使用方法を紹介したが、現在の[Edit Grid]は、更なる便利な入力機能が実現できる。今回は、「登録/編集用の独自フォームの設定方法」、「インライン編集機能」そして、「ローカル編集機能」の3つを紹介したい。

### 4-1. 登録/編集フォームの独自設定

前章で紹介した商品マスタメンテナンスアプリだが、デフォルトの追加/編集用ダイアログ画面は、IBM i標準のDFU(Data File Utility)と同様に、編集画面のレイアウトを一切変更する事ができない。単に[Edit Grid]ウィジェットの[編集]タブの中に定義された入力可能項目を上から順番に表示するのみである。(順番自体は[Edit Grid]設計画面で変更が可能。)また、入力項目の細かな制御を行う事も現状難しい。
この為、[Edit Grid]ウィジェットを本格な入力系アプリに適用するのが以前は難しいと感じていたが、最新のValenceでは、この編集画面を独自に[Form]ウィジェットへ差し替える事ができるようになった。[Form]ウィジェットであれば、画面レイアウトを自由に設計でき、またFormヘルパー機能を組み合わせる事により、入力項目の細かな制御も可能になる。

本節では、前章の商品マスタメンテナンスに登録/編集用の独自フォームを設定する方法を紹介する。
まず、[Edit Grid]ウィジェットの関連元となるデータソースに対し、新たに[Form]ウィジェットを追加する。
次に[Edit Grid]ウィジェットの[編集]タブの中にある[設定]を開く。オプション設定の中にある[フォーム]欄に作成した[Form]ウィジェットを割り当てればよい。**【図9】**

## 図 9　Edit Grid – 編集用フォームの割り当て


[Form]ウィジェットを割り当てると、[追加時][編集時]の設定欄が、割り当てた[Form]ウィジェット上の項目表示に切り替わる為、ここでは、それぞれの場合に入力項目とするか、読取り専用にするかといった定義のみ行えばよい。

ここまでで、基本的な設定は完了である。アプリケーションを実行すると、編集画面が独自フォームに変更された事が確認できる。**【図10】**

## 図 10　商品マスタメンテナンス実行画面 –独自フォーム表示

更に[Form]ウィジェットでは、各入力項目に対し、入力制約が設定可能である。[Form]ウィジェットの設計画面にある[フィールド]タブで、各項目の定義を行うのだが、[フォーマッティング]欄の歯車アイコンを選択すると【図11】のような設定画面が使用できる。

**図 11　Form – 入力項目の書式設定**


従来から、[必須]項目や[大文字に自動変換]は設定可能であったが、Valence6.2では、[形式チェック]において、regexと呼ばれる正規表現設定が可能になった。これにより、入力項目の内容にあった文字列や数値以外エラーとするようなチェックが可能となっている。実際に今回の入力フォームでは、[PDJNCD](JANコード)は、数値13桁、[PDSTDT](発売日)は、数値8桁でかつ、日付値として有効な値のみといった設定を行っている。【ソース2】は、正規表現の定義例である。いろいろな場面で応用できると思われるので参考にしてほしい。

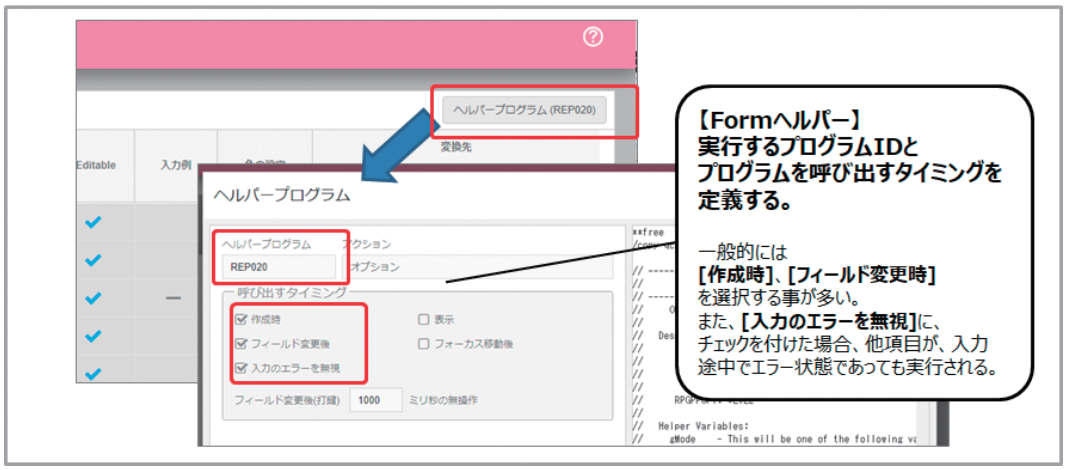
**ソース 2　Formウィジェット［形式チェック］：正規表現設定例**

```
//-----JANコード（13桁数値）
[0-9]{13}

//-----郵便番号（[3桁数値]-[4桁数値]）
^[0-9]{3}-[0-9]{4}$

//-----電話番号（[2-4桁数値]-[2-4桁数値]-[4桁数値]）
¥d{2,4}-¥d{2,4}-¥d{4}

//-----数値8桁の日付値(yyyymmdd:実在する日付以外はエラー)
^(?!([02468][1235679]|[13579][01345789])000229)(([0-
9]{4}(01|03|05|07|08|10|12)(0[1-9]|[12][0-9]|3[01]))|([0-
9]{4}(04|06|09|11)(0[1-9]|[12][0-9]|30))|([0-9]{4}02(0[1-9]|1[0-9]|2[0-
8]))|([0-9]{2}([02468][048]|[13579][26])0229))$
```

また、[Form]ウィジェットには、フォームを初期表示した際の初期値セットや、入力項目の値を変更した場合にインタラクティブなエラーチェックを行う為のFormヘルパー機能が実装できるようになっており、RPGプログラムで記述できる。Formヘルパー機能で使用するRPGテンプレートは、“EXNABHLP”である。この機能は、[Edit Grid]ウィジェットの編集用に設定した独自フォームからも使用する事ができる。

[Form]ウィジェット上の設定方法は、[フィールド]タブ内にある[ヘルパープログラム]ボタンをクリックし、実行したいRPGプログラムや、Formヘルパープログラムを呼び出すタイミングを指定する。**【図12】**

**図 12　Form – Formヘルパー設定**




Formヘルパーに独自処理を追加した実装例は、**【ソース3】**である。Formヘルパーへ呼び出されるタイミングが変数gModeに自動的にセットされるようになっており、また値変更時には、変更したフィールド名が変数gFieldにセットされるようになっている。**【図12】**の定義どおり、今回はフォーム生成時とフィールド変更時に呼び出す設定となっている為、3-①のような条件分岐を組み込めばよい。

プログラムの詳細だが、3-②は、新規レコード追加時に[PDSTDT](発売日)に初期値としてシステム日付をセットする処理である。

3-③は、[PDSPCD](仕入先コード)に値を入力した際に、SQLを使用して仕入先マスタを検索し、仕入先名をセットする処理である。SQL文を実行した結果対象データが存在しない場合は、SQLSTATEに“02000”がセットされる為、エラーを返す事ができる。

3-④は、[PDUNPR](販売単価)と[PDCOPR](仕入単価)との大小チェックである。

Formヘルパーを使用して、画面上の項目に値をセットする場合、SetValueメソッドを使用し、エラーを渡す場合、SetErrorメソッドを使用するという事を覚えておこう。

**ソース 3** REP020:商品マスタ入力フォーム:入力チェック(一部抜粋)

```
** ------------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D INSTDT           S               8S 0
D SYSDAT           S                D   DATFMT(*ISO) INZ(*SYS)
D INUNPR           S               9S 0
D INCOPR           S               9S 0
D INSPCD           S               6A
D SQSPNM           S              42A
D*
 /free
   if gMode = 'formRender';                          3-①
     //フォーム生成時処理
     //-----発売日が０の場合システム日付を初期セット
     INSTDT = vvIn_num('F1_PDSTDT');                 3-②
     if INSTDT = *ZERO;
       SetValue('F1_PDSTDT':%CHAR(%DEC(SYSDAT:*ISO)));
     endif;
   elseif gMode = 'formShow';                        3-①
     //フォーム表示時処理
     //
   else;
     //項目変更時処理
     //-----仕入コード変更時
     if gField = 'F1_PDSPCD';                        3-①
       //入力値を取得
       INSPCD = vvIn_char('F1_PDSPCD');              3-③
       if INSPCD <> *blank;
         //仕入先マスタを検索
         exec sql SELECT SUSPNM INTO :SQSPNM FROM MSUPLP
                  WHERE SUSPCD = :INSPCD;
         //仕入先ＣＤが存在しない場合エラー
         if SQLSTATE = '02000';
           SetError(gField:'仕入先ＣＤが正しくありません');
           return;
         else;
           //仕入先名欄に取得値をセット
           SetValue('F2_SUSPNM':SQSPNM);
         endif;
       else;
         //仕入先名欄にブランクをセット
         SetValue('F2_SUSPNM':'');
       endif;
     endif;
     //-----販売単価ＯＲ仕入単価変更時
     if gField = 'F1_PDUNPR' or gField = 'F1_PDCOPR';  3-①
       //入力値を取得
       INUNPR = vvIn_num('F1_PDUNPR');               3-④
       INCOPR = vvIn_num('F1_PDCOPR');
       //-----販売単価と仕入単価の相関チェック
       if INUNPR <> *ZERO and INCOPR <> *ZERO;
         if INUNPR < INCOPR;
           SetError(gField:'販売単価が仕入単価未満です');
           return;
         endif;
       endif;
     endif;
   endif;
 /end-free
p                  e
```

このようにFormヘルパープログラムを組み込めば、編集フォーム上で値入力時のリアルタイムなエラーチェックやマスタから取得した名称の自動セットが行える。その為、入力系アプリとしての使い勝手を大幅に向上させる事ができる。

### 4-2. インライン編集機能

次に紹介するのは、[Edit Grid]ウィジェットにおけるインライン編集機能である。第2章で紹介した通り、[Edit Grid]ウィジェットは、Grid上で行メニューから[編集]を選択、あるいは行をダブルクリックし、編集用ダイアログ画面からデータをメンテナンスできる。しかし、インライン編集機能を有効にすると、Grid上のセルに値を直接入力する事や、直接行編集を行う事が可能となる。**【図13】**

**図 13　Edit Grid - インライン編集機能**


本節では、このインライン編集機能を使用して作成した商品マスタの価格一括入力プログラムを紹介する。設定は至ってシンプルである。[Edit Grid]ウィジェット設計画面の[編集]タブにある[設定]の[インライン]欄を「セル編集」に設定すれば良い。後は、[カラム]タブで設定したGridの表示項目の中から、入力対象となる項目を[編集時]欄に追加すればよい。

なお、インライン編集機能は現在のところ、[編集時]のみに対応している。[追加時]に入力項目を設定した場合は、通常の新規入力用ダイアログ画面が表示されるので注意してほしい。

インライン編集を有効にした場合も、“EXNABVAL”テンプレートを使用すれば、入力値のエラーチェックや非表示項目への値の書き込みが可能である。インライン編集時のプログラム実装例は、**【ソース4】**である。

基本的な考え方は、第3章で紹介した**【ソース1】**と同様だが、インライン編集の場合、エラー時の制御方法が異なる。通常、エラーが発生した場合は、SendErrorメソッドを使用すればよいが、インライン編集の場合はこのメソッドは利用できない。代わりに4-①のように、SetResponseメソッドを使用してレスポンスメッセージを送る必要がある。ただ、このままではエラーと判断されても更新処理が中断されない為、outStopProcess=*on; を記述して、後続の更新処理を明示的に中断させる必要があるので注意してほしい。

**ソース 4** REP030:商品マスター括入力行更新チェック(一部抜粋)

```
p ProcessEdit     b
d                 pi
D*-----変数宣言
D INUNPR          S              9S 0
D INCOPR          S              9S 0
D SYSDAT          S               D   DATFMT(*ISO) INZ(*SYS)
D*
 /free
   //-----画面入力値を取得
   INUNPR = %DEC(GetValue('MPRODP':'PDUNPR'):9:0); //販売単価
   INCOPR = %DEC(GetValue('MPRODP':'PDCOPR'):9:0); //仕入単価

   //-----販売単価と仕入単価の相関チェック
   if INUNPR < INCOPR;
     //※インライン編集の場合SendErrorは使用不可
     //SendError('販売単価が仕入単価未満です':'F1_PDUNPR');
     SetResponse('success':'false');
     SetResponse('msg':'販売単価が仕入単価未満です');
     outStopProcess = *on;
     return;
   endif;

   //-----非入力項目の値をセット
   SetValue('MPRODP':'PDUPDT':%CHAR(%DEC(SYSDAT:*ISO))); //最終更新日
   SetValue('MPRODP':'PDUPUS':GetAppVar('nabUser'));      //最終更新者
 /end-free
p                 e
```

4-①

以上で、インライン編集機能の実装は完了である。インライン編集機能を使用すれば、ユーザーは画面を切り替える事なくデータを一括入力できる為、操作性は大きく向上するだろう。

### 4-3. ローカル編集機能

前節では、インライン編集機能を紹介したが、あくまで[Edit Grid]ウィジェットの基本動作は、選択した行に対するデータの更新である。複数明細が一括で入力可能なインライン編集機能だが、実際には行を選択して入力を行い、確定する毎に、都度データベースへの更新が行われる。しかし、一括入力を行う画面の場合、複数明細を画面入力後に、[保存]等のボタンをクリックして、複数データを一括でデータベースに反映するような画面構成を求められる事が多いだろう。最新の「App Builer」の[Edit Grid]ウィジェットは、この要望にも対応できる。それがローカル編集機能である。この機能は、[Edit Grid]ウィジェットによって表示されたデータについて、画面操作中は、データベースへの更新は行わずに、画面(ブラウザ)の中の値のみが更新される仕組みである。

実装方法だが、まず、[Edit Grid]ウィジェットの[設定]タブを開き、[ページング]カテゴリー内の[アクティブ]を無効にする。次に[データ]カテゴリーを開き、[ローカル]を有効にすればよい。

以上の定義でローカル編集機能が有効となる。この状態でアプリケーションを実行すると、インライン編集で画面上のセル入力で値を変更しても、データベースへは反映されなくなる。実際にデータベースに書き込む処理は、RPGプログラムで実装すればよい。

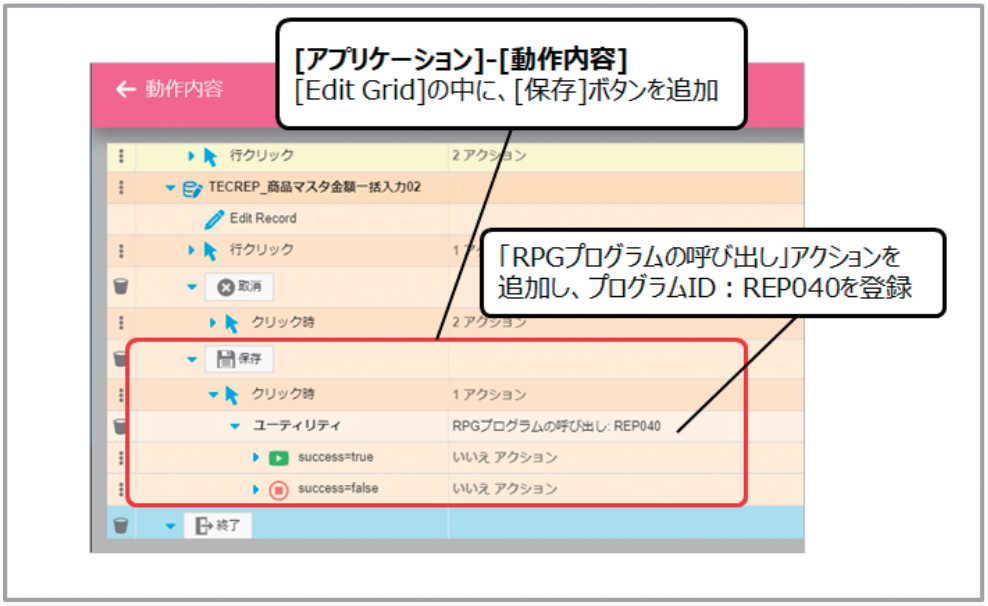
ローカル編集機能の実装例を紹介する。**【図14】**は、完成したアプリケーションの実行例である。

## 図 14 商品マスタ金額一括更新アプリ実装例


画面左部でカテゴリーを選択すると、該当するカテゴリーの商品が右側の[Edit Grid]ウィジェットに一覧表示される。販売単価と仕入単価がそれぞれセル編集となっている為、値を編集できるが、変更を行ってもローカル編集機能が有効な為、データベースには反映されなくなっている。最後にウィジェット下部にある[保存]ボタンをクリックした時に、一括でエラーチェックおよびデータベースへの反映を行うという流れである。

アプリケーション設計画面をみてみよう。[Edit Grid]ウィジェット上に追加した[保存]ボタンのイベントとして、[RPGプログラムの呼び出し]を追加し、REP040を登録している。**【図15】**

## 図 15 商品マスタ金額一括更新アプリ設計画面



ボタンクリック時に呼び出されるプログラムのRPGテンプレートは、“EXNABBTN”である。このテンプレートを元に作成したプログラム（REP040）が、**【ソース5】**である。

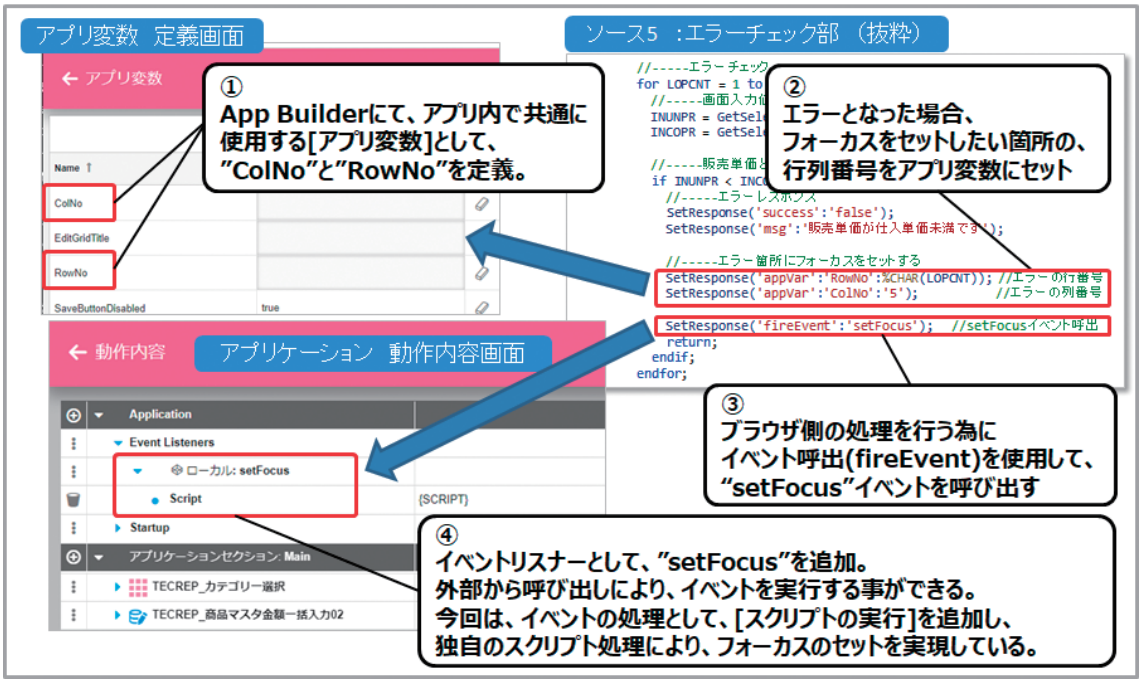
**ソース 5　REP040：商品マスター括更新処理（一部抜粋）**

```
** ----------------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D LOPCNT           S              9S 0
D INPDCD           S             10A
D INUNPR           S              9S 0
D INCOPR           S              9S 0
D SYSDAT           S               D   DATFMT(*ISO) INZ(*SYS)
D INUPDT           S              8S 0
D INUPUS           S             10A
D*
 /free
   //-----エラーチェック
   for LOPCNT = 1 to gSelectionCnt;                    5-①
     //-----画面入力値を取得
     INUNPR = GetSelectionNum(LOPCNT:'F1_PDUNPR');  //販売単価
     INCOPR = GetSelectionNum(LOPCNT:'F1_PDCOPR');  //仕入単価   5-①

     //-----販売単価と仕入単価の相関チェック
     if INUNPR < INCOPR;                                  5-②
       //-----エラーレスポンス
       SetResponse('success':'false');
       SetResponse('msg':'販売単価が仕入単価未満です');

       //-----エラー箇所にフォーカスをセットする
       SetResponse('appVar':'RowNo':%CHAR(LOPCNT)); //エラーの行番号
       SetResponse('appVar':'ColNo':'5');           //エラーの列番号
       SetResponse('fireEvent':'setFocus');   //setFocusイベント呼出
       return;
     endif;                                               5-④
   endfor;

   //-----更新処理
   //-----共通項目値を取得
   INUPDT = %DEC(SYSDAT:*ISO);           //最終更新日
   INUPUS = %Trim(GetAppVar('nabUser')); //最終更新者

   for LOPCNT = 1 to gSelectionCnt;                    5-①
     //-----画面入力値を取得
     INPDCD = GetSelectionChar(LOPCNT:'F1_PDPDCD'); //商品ＣＤ
     INUNPR = GetSelectionNum(LOPCNT:'F1_PDUNPR');  //販売単価
     INCOPR = GetSelectionNum(LOPCNT:'F1_PDCOPR');  //仕入単価   5-①

     //-----データの更新処理
     exec sql  UPDATE MPRODP SET                          5-③
               PDUNPR = :INUNPR,
               PDCOPR = :INCOPR,
               PDUPDT = :INUPDT,
               PDUPUS = :INUPUS
               WHERE PDPDCD = :INPDCD;
   endfor;

   //-----正常終了レスポンス
   SetResponse('success':'true');
   SetResponse('info':'更新が終了しました');
 /end-free
p                  e
```

このテンプレートでは、[Grid]や[Edit Grid]の各要素にアクセスする為のAPIが使用できるようになっている。5-①の部分に着目してほしい。gSelectionCntには、Gridの全行数が格納される。GetSelectionChar(文字項目)及びGetSelectionNum(数値項目)が、指定した行番号の要素にアクセスして、Grid上の内容を取得するメソッドである。
今回のプログラムでは、大きく分けてエラーチェック処理とデータ更新処理の2段階となっており、それぞれfor–endforを使用したループ処理としている。Grid上のデータを上から順番に取得しながら、5-②では入力値に対するエラーチェックを、5-③ではSQLを使用したデータの更新処理を行っている。

大きな流れは以上だが、5-②のエラーチェックでは、エラー発生時にSetResponseメソッドを使用して、レスポンスメッセージを作成している事がわかる。単にエラーメッセージダイアログをブラウザに表示するだけであれば、処理はシンプルなのだが、残念ながら現在の「App Builder」では、Grid上の項目でエラーが発生した際に、エラー項目にフォーカスをセットする事ができない。今回のアプリケーションでは、フォーカスをセットする為にもう一工夫を行っているので、その部分の実装方法を紹介する。
**【図16】**がエラー発生時にフォーカスをセットする処理の流れである。

**図 16　エラー時に、対象フィールドにフォーカスセットする仕組み**




フォーカス制御を行う為に、アプリケーション内に独自のアプリ変数"ColNo"(列番号)と"RowNo"(行番号)を定義している。**【ソース5】**の5-④の部分で、SetResponseメソッドのパラメータ"AppVar"を使用して、エラーが発生した行列番号をアプリ変数にセットしている。このようにRPG側からアプリ変数へ値をセットする事ができる。さらにSetResponseメソッドのパラメータ"fireEvent"を使用すれば、「App Builder」に定義した任意のイベントを呼び出す事もできる。ここでは、フォーカスをセットする為のイベントとして、"setFocus"イベントを呼び出している。
「App Builder」側の[アプリケーション]→[動作内容]画面では、独自のイベントを追加する[Event Listener]が用意されているので、ローカル新規イベントとして、"setFocus"を定義している。そして追加した"setFocus"イベントに、[スクリプトの実行]を追加すれば、独自のスクリプト処理を実行する事ができ、今回はスクリプトを使用してフォーカスをセットする処理を実行している。

スクリプトイベントの実装例は、**【ソース6】**である。少し複雑な処理の為、詳細は割愛するが、JavaScriptは原則非同期実行となる為、単純にスクリプトを記述するだけでは、実行時に表示されるエラーメッセージダイアログが表示されている間にスクリプトが終了してしまい、フォーカスが制御できない。従って、6-①のように、メッセージダイアログのオブジェクトを取得しておき、そのオブジェクトの要素が画面から削除された事（つまりメッセージダイアログが閉じられたという状態）を検知して、その後スクリプト処理を実行するような仕組みとしている。

6-②がアプリ変数の値を取得し、対象となる［Edit Grid］ウィジェットを取得する処理である。なお、ソース中の“PRODGrid”は、アプリケーション画面で、ウィジェットに付けた名前である。

6-③が、ウィジェット内のGrid上の対象となる行列番号の要素を取得する処理で、6-④がフォーカスをセットする処理である。実際には、対象となるセルを疑似的にダブルクリックする事で、フォーカスがセットされるという仕掛けとなっている。

**ソース 6**　**setFocusイベント　：　【スクリプトの実行】**

```
//メッセージBOX要素を取得
var Box = document.querySelector('.vv-common-dialog');  6-①

//要素が変化した場合、処理を実行
let observer = new MutationObserver(mutationRecords => {
    //フォーカスをセットする行列番号を取得
    var rowNo = getAppVar('RowNo');
    var colNo = getAppVar('ColNo');
    //対象となるEditGridウィジェットを取得
    var gridWidght = getWidget('PRODGrid'); //----【EditGridに設定した名称を記述】  6-②
    //ウィジェット内のgridを取得
    var grid = gridWidght.down('grid');
    //フォーカスをセットするセル要素を取得
    const el = '[data-recordindex="' +  String(Number(rowNo) - 1) + '"]';
    var Td = grid.getEl().dom.querySelector(el).querySelectorAll('td')[Number(colNo) - 1];  6-③

    //ダブルクリックイベントの定義
    const dblclick =  new MouseEvent("dblclick", {
      bubbles: true,
      cancelable: true,
      view: window,
    });

    //セル要素ダブルクリックイベントの呼出し
    function settd(){
        Td.dispatchEvent(dblclick);
    }

    //500ミリ秒経過後にダブルクリックイベントを実行
    setTimeout(settd,500);
});  6-④

// 対象となる要素の追加・削除を監視
observer.observe(Box, {
  childList: true
});
```

この仕組みは若干複雑だが、他のアプリケーションでも応用できるよう汎用的な仕組みとして検討しているので、是非試してみてほしい。

## 5.複数行明細入力フォーム作成テクニック

ここまで主に商品マスタを使用して、[Edit Grid]ウィジェットの活用方法を紹介してきたが、最後に応用例として受注入力を行うアプリケーションの実装例を紹介する。
**【図17】**が、完成したアプリケーションの実行画面である。
アプリケーションを起動すると、新規受注を登録する為の画面として、左側に[受注見出]用の[Form]ウィジェットが、右側に[受注明細]用の[Edit Grid]ウィジェットが表示される。受注明細は、10明細分の登録が可能な構成となっている。画面上部の[保存]ボタンをクリックする事で、受注ファイルおよび受注明細ファイルに新規レコードを登録するといった処理である。

**図 17　受注入力(新規登録)画面実装例**


このアプリケーションのポイントとなるのが、[Edit Grid]に関連付けられる受注明細情報である。前章までに紹介した商品マスタとは異なり、今回のアプリは、受注明細情報の新規登録であり、元となるデータが存在しない為、そのまま受注明細ファイルをデータソースとして紐づける事はできない。そこで今回は、明細入力用に**【図18】**のようなレイアウトをもつ受注明細ワークファイルを定義した。

**図 18　受注明細ワークファイル　ファイルレイアウト**

DSPFMT　　レ コ ー ド 設 計 書　　日付 23/09/01　時刻 19:30:36

| 物理ファイル | TECREPLIB/WODRDP | 様式名 | WODRDR | レコード長 | 101 |
|---|---|---|---|---|---|
| 様式記述 | 受注明細ワーク | | | | |

5= 詳細

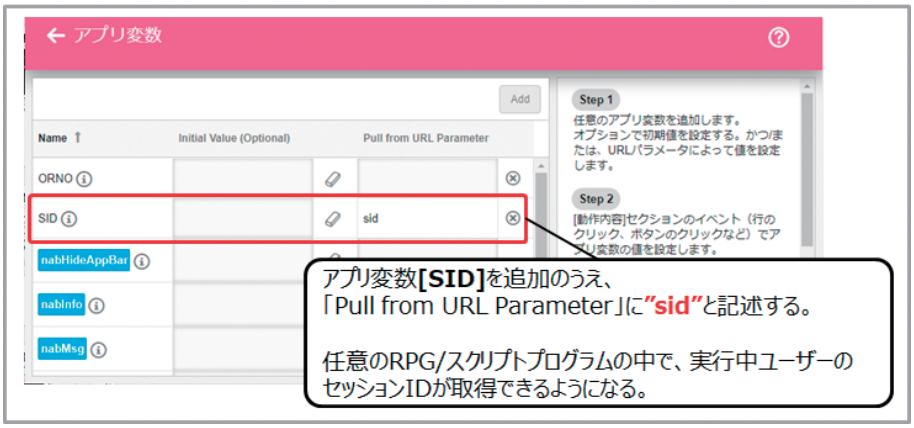
| 選択 | 項目名 | 桁数 | 属性 | キー順 | 開始 | 終了 | テキスト記述／欄見出し |
|---|---|---|---|---|---|---|---|
| _ | WDSID | 64 | A | 1 ANN | 1 | 64 | セッションＩＤ |
| _ | WDLNNO | 2 0 | S | 2 ASN | 65 | 66 | 行ＮＯ |
| _ | WDPDCD | 10 | A | | 67 | 76 | 商品ＣＤ |
| _ | WDQTY | 6 0 | S | | 77 | 82 | 数量 |
| _ | WDUNIT | 9 0 | S | | 83 | 91 | 単価 |
| _ | WDPRCE | 10 0 | S | | 92 | 101 | 金額 |

**セッションID**　：　実行しているブラウザ（セッション）を識別するID。
**行NO**　：　10明細分（1～10）の行番号

一般的にPC5250アプリをRPG等で開発する場合、ワークファイルを使用する際に、QTEMPライブラリを使用する事が多い。実行中一つのジョブを専有する形で処理される5250セッションであればこの方法は有効だが、残念ながらValenceは、ブラウザ上の複数セッションでジョブを共有利用する仕組みの為、QTEMPライブラリを使用する事は現実的ではない。そこで、Valenceで実行するユーザーのブラウザ毎に独自のワークファイルを作成する際に役立つのが、[セッションID]である。「Valence Portal」に含まれる「活動セッション」アプリを実行するとわかるが、Valenceを実行中のユーザーには、一意となる64桁の[セッションID]が自動的に付与されるようになっている。この情報をワークファイルのキー項目にする事で、ユーザー毎に独立したワークファイルを処理する事が可能となる。

「App Builder」で作成するアプリケーションにおいて、[セッションID]を使用する方法を紹介する。アプリ変数の設定画面にて変数“SID”を追加する。そして、オプションパラメータの[Pull from URL Parameter]に小文字で“sid”を記述すれば良い。これで、任意のRPGプログラムやブラウザ側のスクリプト処理のなかで[セッションID]が利用できるようになる。**【図19】**

**図 19　アプリ変数へセッションID追加**

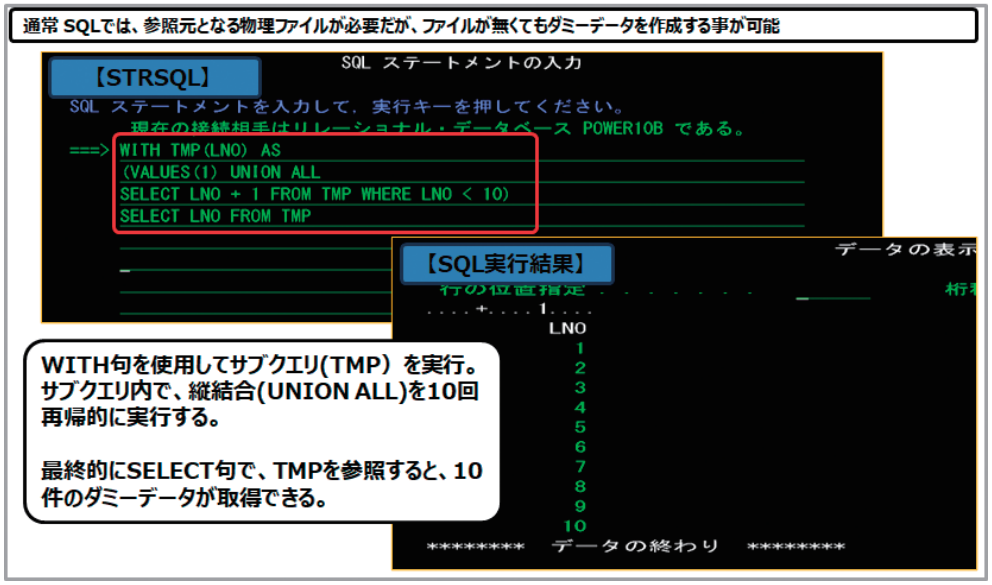



次に、この[セッションID]を使用して受注明細ワークファイルを作成する方法を紹介する。「App Builder」には、アプリケーションの起動時及び終了時に呼び出し可能なRPGプログラムが設定できるようになっている。[動作内容]画面の中にある[起動時/閉じる]欄である。この中に、起動時/終了時に実行したいプログラムを定義すれば良い。**【図20】**

**図 20　アプリケーション – 動作内容　：　起動時 / 閉じる**

最後にワークファイルを作成するテクニックを紹介する。今回はアプリケーション起動時にセッションIDをキー項目とした空のワークファイルを10明細分作成する必要がある。もちろんRPG処理において、ファイルを10件WRITEすれば良いのだが、SQLを使用すれば1回のSQLで任意の件数の空データを作成できる。肝となるのが、**【図21】**のようなSQLである。

**図 21　複数レコードのダミーデータを作成するSQL**




通常SQLを使用する場合、参照元のファイル(テーブル)をFROM句に指定する必要があるが、実はファイルを使用しないSQLも作成できる。今回のSQLでは、数値1つを取得するサブクエリを10回再帰的に実行する事で、1から10の10明細分の結果セットを取得する事ができる。この仕組みを使用して取得した複数レコードのダミーテーブルをワークファイルに挿入(INSERT)すれば良い。

ソース記述例を紹介する。まず、起動時に実行するRPGプログラムは、**【ソース7】**である。このプログラムは、RPGテンプレート“EXNABSTART”を元に作成すれば良い。

**ソース 7**

**REP050:アプリケーション起動時処理(一部抜粋)**

```
** ------------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D LNCNT            S              2S 0
D SID              S             64A
D*
 /free
   //-----ワークファイル作成に必要な設定値
   LNCNT = 10;                 //ワークファイルを作成する行数
   SID = GetAppVar('SID');     //セッションＩＤ

   //-----同じセッションＩＤのレコードを全て削除
   exec sql DELETE FROM WODRDP WHERE WDSID = :SID;

   //-----指定した明細行分、ダミーレコードを作成する
   //SQLを使用して、明細用ワークファイルにレコードを追加
   exec sql INSERT INTO WODRDP (WDSID, WDLNNO)
            WITH TMP(LNO) AS (VALUES(1) UNION ALL
            SELECT LNO + 1 FROM TMP WHERE LNO < :LNCNT)
            SELECT :SID, LNO FROM TMP;
 /end-free
p                  e
```

7-① 7-②

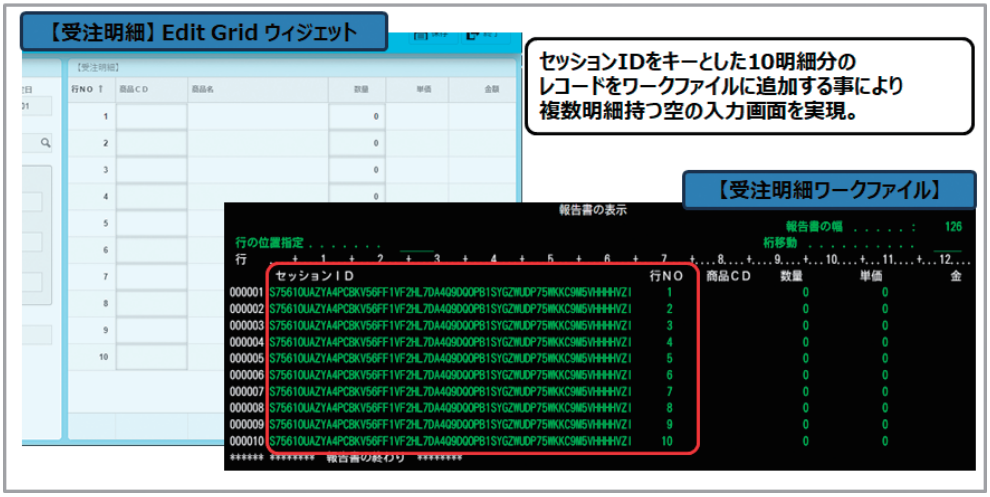
7-①にてセッションIDを取得している。7-②のようなSQLを実行すれば、[セッションID]をキーとした複数件の明細をもつワークファイルを一度のSQL処理で作成できる。
なお、ワークファイルは、アプリケーション終了時に削除する必要もある。削除時のRPGプログラム実装例は**【ソース8】**である。終了時も起動時と同様に、8-①にてセッションIDを取得し、8-②のようなSQLを使用してワークファイルを削除している。なお、アプリケーション終了時プログラムは、“EXNABCLOSE”テンプレートから作成する。
以上で、受注明細ワークファイルを使用した複数明細を持つ入力用[Edit Grid]ウィジェットの作成は完了である。完成したアプリケーションの実行結果は、**【図22】**である。

**ソース 8**

**REP060:アプリケーション終了時処理(一部抜粋)**

```
** ----------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D SID              S              64A
D*s
 /free
   //-----セッションＩＤ取得
   SID = GetAppVar('SID');                    8-①

   //-----ワークファイルの削除
   //使用したセッションＩＤのレコードを削除
   exec sql DELETE FROM WODRDP WHERE WDSID = :SID;   8-②
 /end-free
p                  e
```

**図 22　初期プログラム(REP050)実行結果**




このように複数明細行を持つ入力画面を[Edit Grid]ウィジェットを使用して作成したい場合、今回紹介した仕組みが活用できるので、ぜひ参考にしてほしい。

## 6.さいごに

本稿では、「App Builder」における『[Edit Grid]ウィジェット活用術』として、[Edit Grid]ウィジェットの基本から機能強化された各種活用法についてサンプルを交えて紹介してきた。冒頭にも案内したとおり、紙面だけでは伝わりにくい部分がある為、今回Valence6.2上で動作するサンプルアプリを用意している。是非、サンプルアプリをインポートの上、実際の動作や、定義内容を確認してほしい。[Edit Grid]ウィジェットを使用したアプリ開発におけるヒントが見つかるであろう。皆様も[Edit Grid]ウィジェットの活用に色々挑戦してみてほしい。

# 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

# MIGARO.
# Technical Report 2023

**No.16 2023年**

**2023年12月25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