# 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

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。

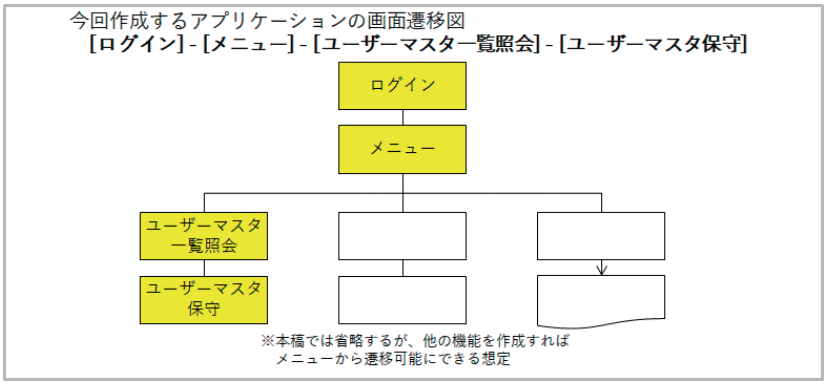
そこで本稿では、改めて2023年9月現在の最新版であるDelphi/400　11　Alexandriaを使用したアプリケーション開発手順の基本を紹介する。題材としてユーザーマスタファイルを参照、更新するアプリケーションを一から開発することを前提に具体的な手順を紹介したい。データアクセスについても最新のデータベースエンジンであるFireDAC接続を使用する。

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

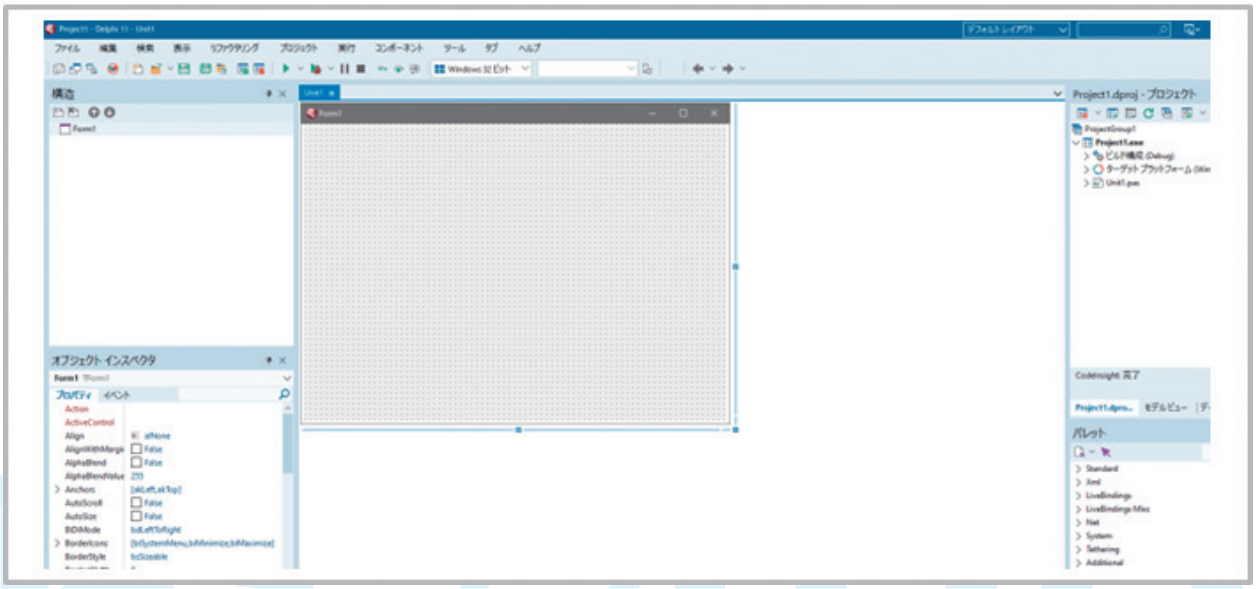
本稿でこれから作成するアプリケーションの画面遷移図を【図1】のようにまとめた。どのような画面が必要か、および画面間の遷移について事前に流れを明確にしておけば、必要な画面の過不足に後から気付くリスクを低減できる。

**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**

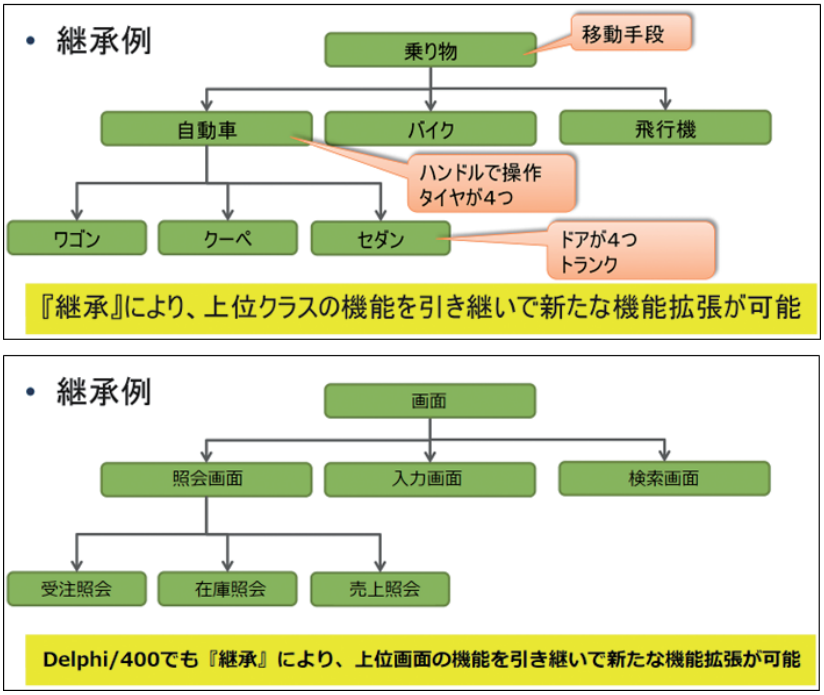


フォーム名の変更完了後、まずは先にフォームとプロジェクトを保存しておこう。保存先やファイル名は任意で指定可能であり、本稿では「BaseFrm.pas」「Project1.dproj」という名前で保存する。なお、ここで保存したプロジェクト名がコンパイルされた際の実行ファイル名（EXE名）になる。

## 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**

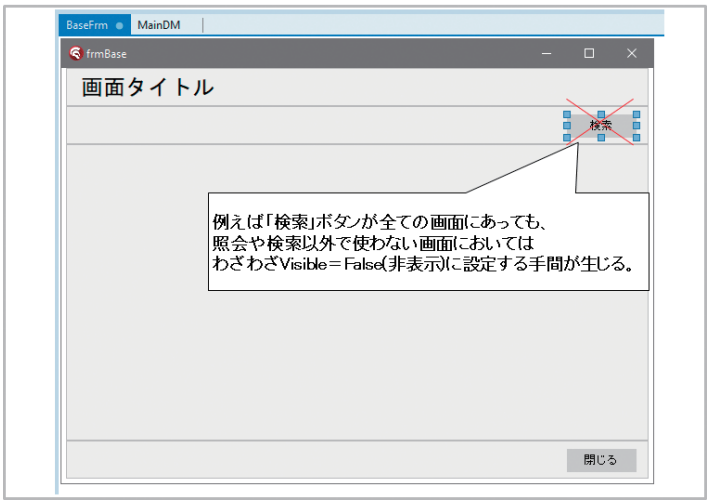
```
// 画面表示時処理
procedure TfrmBase.FormShow(Sender: TObject);
begin
  // 画面のタイトルを表示
  Self.Caption := lblTITLE.Caption;
end;

// 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。

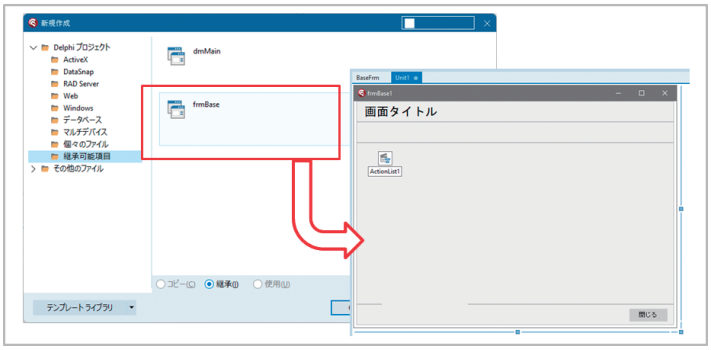
継承元フォーム作成時の注意点としては、よく使用するコンポーネントやロジックのみを継承元でセットする点である。継承元フォームで配置したコンポーネントは、継承先フォームでは削除することができない。例えば、使用頻度の低い項目・ロジックを継承元にセットすると、各画面で不要な項目の非表示や、不要なロジックを無効化する余分な手間が発生する**【図5】**。継承は便利である一方、継承先への制約となることも意識する必要がある。

**図 5　継承元には必要な項目だけを配置する**



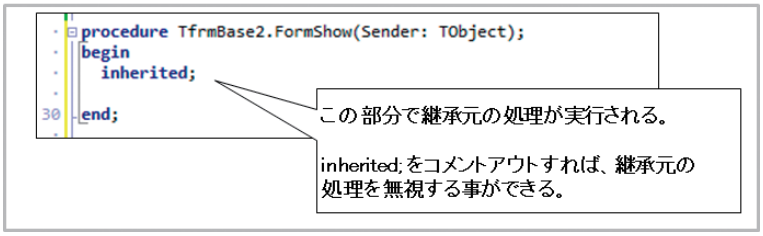

次に、継承先のフォームを作成する。[ファイル]メニューの「新規作成」から「その他」をクリックした後、表示される新規作成ダイアログの中から「継承可能項目」を選択する。すると、現在のプロジェクト内のフォームが一覧で表示されるので、この中から継承元にしたいフォーム(今回はfrmBase)を選択する。これにより、選択したフォームを継承した新しいフォームが作成される**【図6】**。

**図 6　継承先フォームの作成**



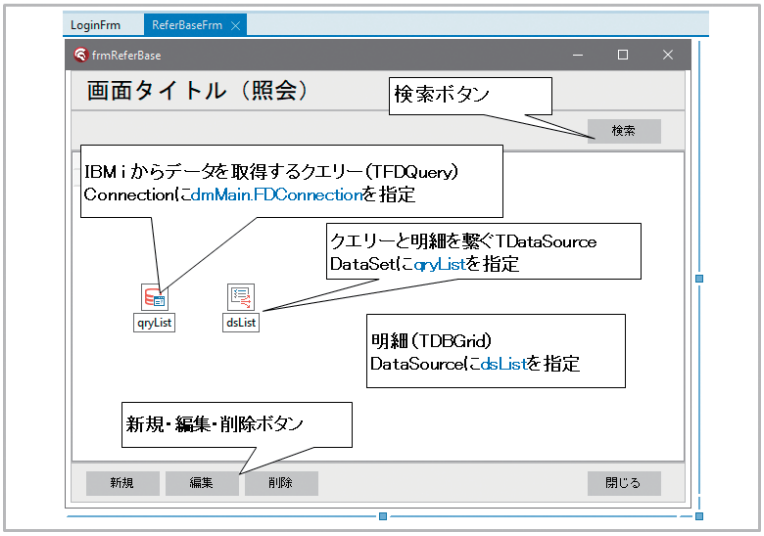
画面項目だけでなくロジックも継承されている。例えば今回は継承元フォームにて**【ソース1】**のロジックを記述したが、継承先フォームでは何も記述しなくても継承元と同じ処理が行われてフォーム表示時にタイトルがセットされる。
継承先ではフォーム表示時処理のロジックが**【図7】**のようになっており、『inherited;』と記述された部分で継承元の処理が呼び出されている。従って、この行をコメントアウトすると継承元の処理を実施しない。

**図 7　継承先のイベント**



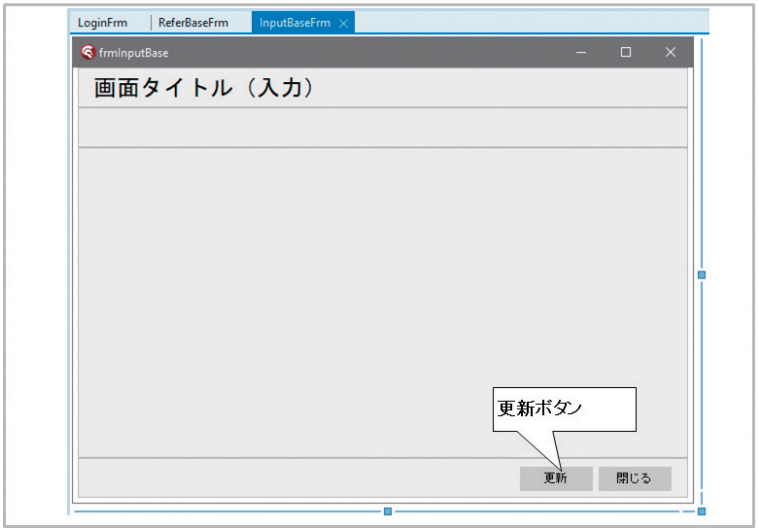

**【図6】**で作成したフォームは、照会画面の継承元として使用する。名前（Ｎａｍｅ）を「frmReferBase」に変更し、「ReferBaseFrm.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。続けて、照会画面で共通して使用するコンポーネントを**【図8】**のように配置する。

**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。

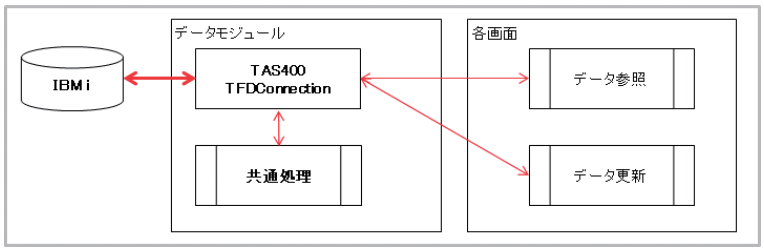
**図 9　入力画面の継承元にセットするコンポーネント**



## 4.データモジュールを活用したIBM iとの接続処理

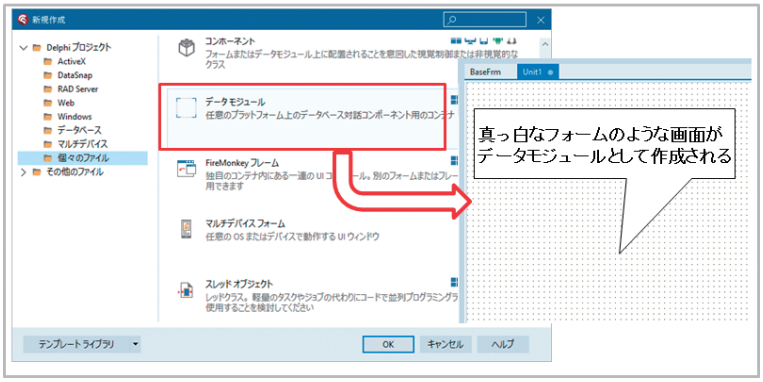
ここからはデータモジュールの作成方法を説明する。通常のフォームではなくデータモジュールを使用することで、データベースへのアクセスやデータ処理に関するロジックを専用の独立したモジュールに分離し、コーディングの簡略化や保守性の向上を図ることができる。また、同じデータモジュールを複数のフォームやユニットで共有し、一か所で管理することによって、コードの重複を避けて保守性や拡張性を高めることができる【図10】。

**図 10　データモジュールの役割**



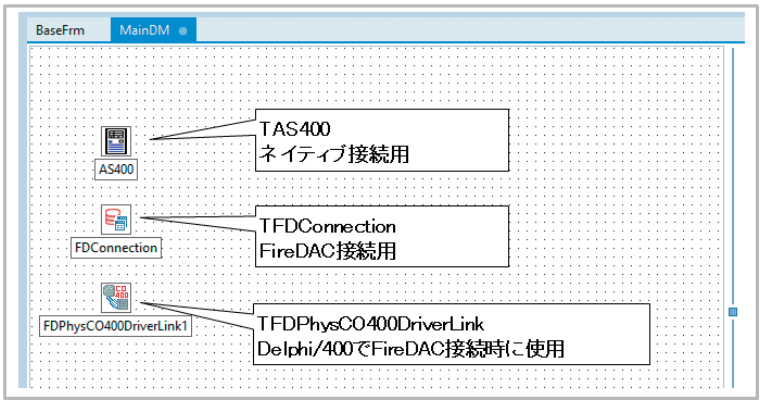

データモジュールを新規作成するには、[ファイル]メニューの「新規作成」から「その他」を選択し、新規作成の設定ダイアログから「個々のファイル」を選び、「データモジュール」を指定する【図11】。

**図 11　データモジュールの新規作成**



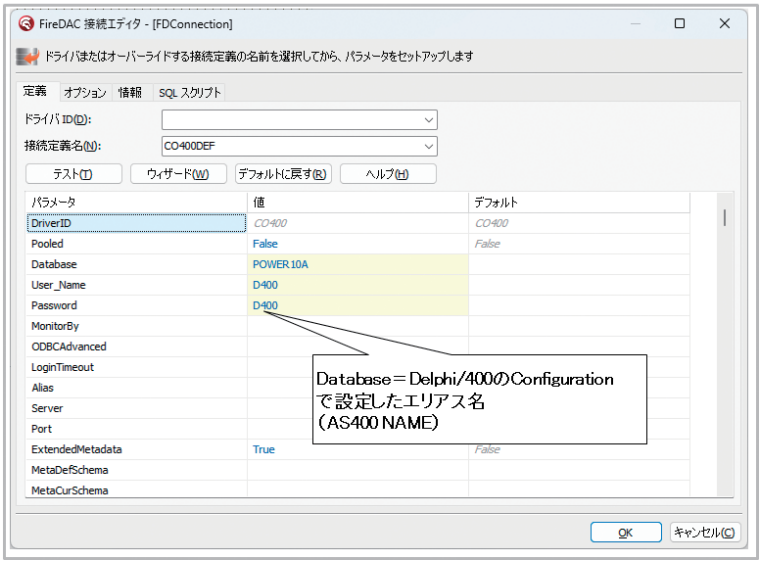

作成したデータモジュールの名前（Ｎａｍｅ）を「ｄｍＭａｉｎ」とし、「MainDM.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。dmMainにはデータベース接続に使用する各コンポーネントを配置していく。今回はFireDAC接続用コンポーネントとしてTFDConnection・TFDPhysCO400DriverLinkを、ネイティブ接続コンポーネントとしてTAS400をフォームに配置する【図12】。

**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。

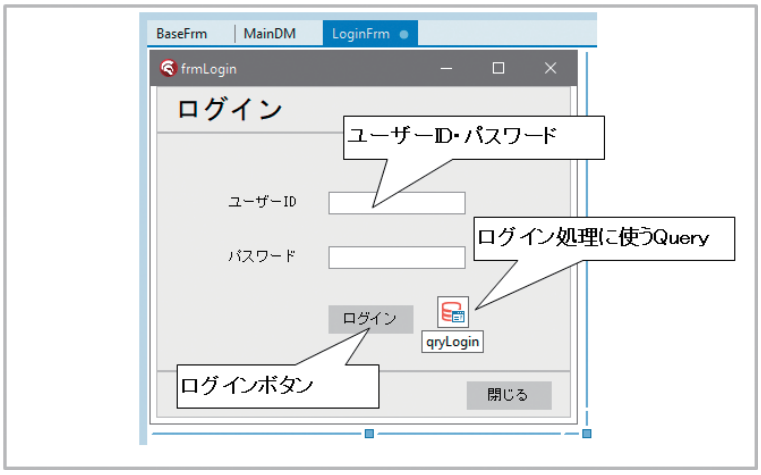
### 5-1. ログイン・メニュー画面

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。

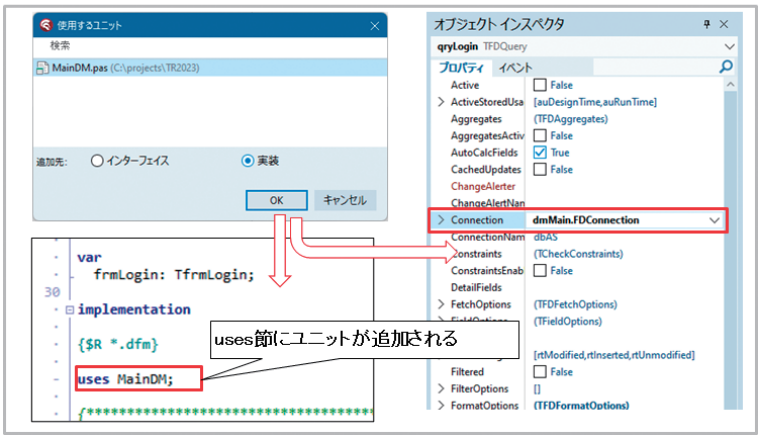
**図 14 ログイン画面の項目**




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;
```

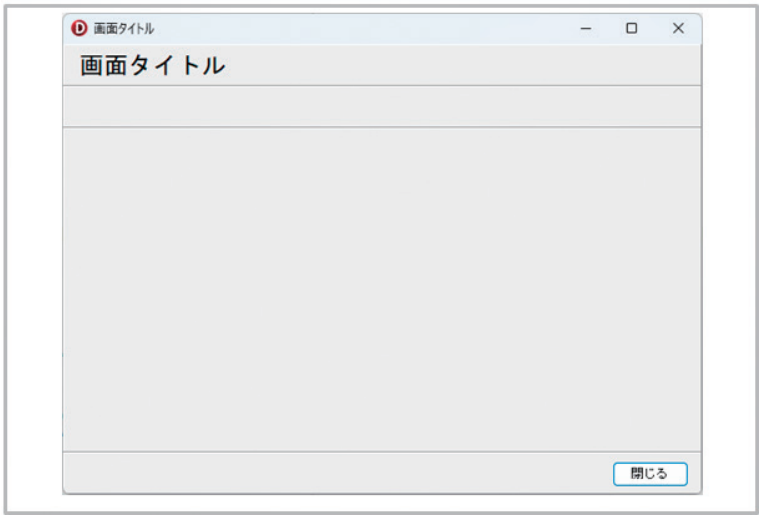
①

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**

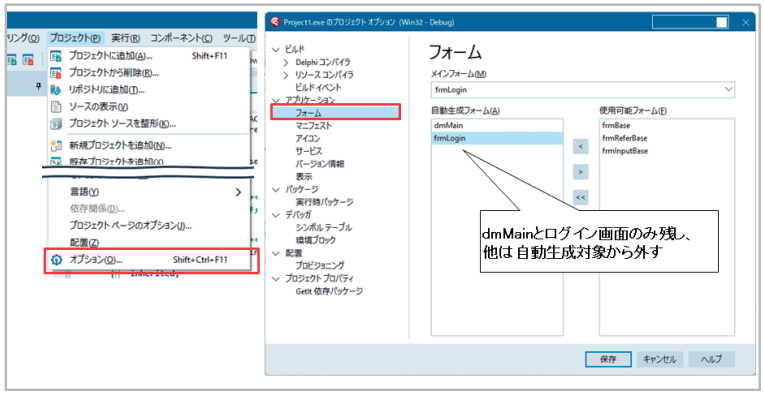
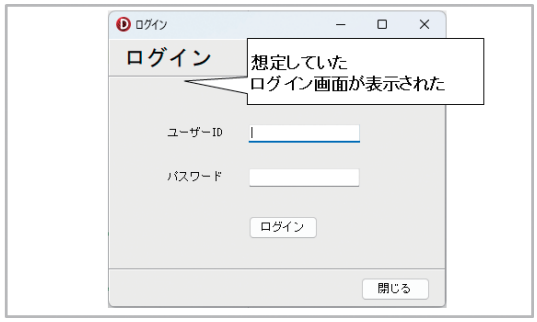



**図 19 改めてEXE起動**

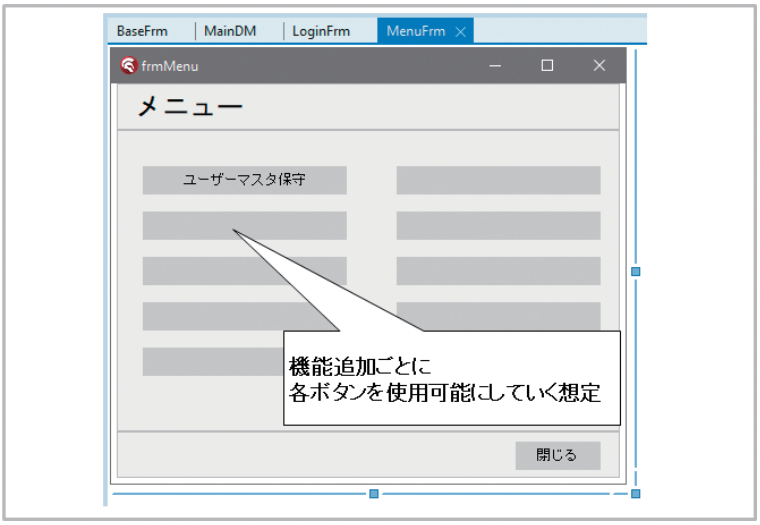


次にメニュー画面を作成する。先ほどのログイン画面と同様に基底フォーム(frmBase)を継承して新規作成し、名前(Ｎａｍｅ)を「frmMenu」とし、「MenuFrm.pas」というファイル名で他のファイルと同じディレクトリ内に保存する。また、プロジェクトオプションの「自動生成フォーム」に追加されている「frmMenu」を除外する。

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**

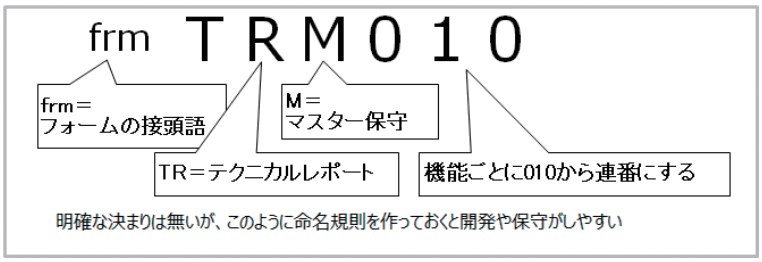
```
// ユーザーマスタ保守ボタン押下時処理
procedure TfrmMenu.Button1Click(Sender: TObject);
begin
  inherited;

  // 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**

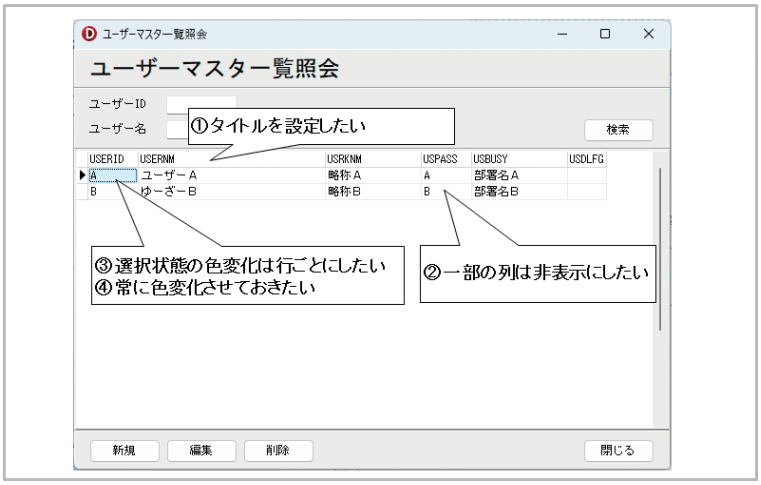
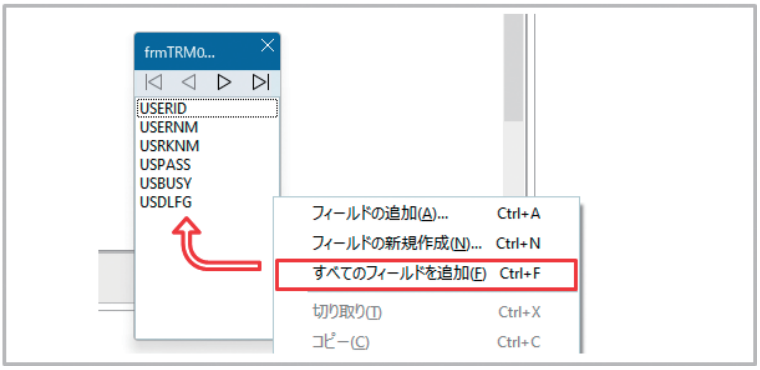


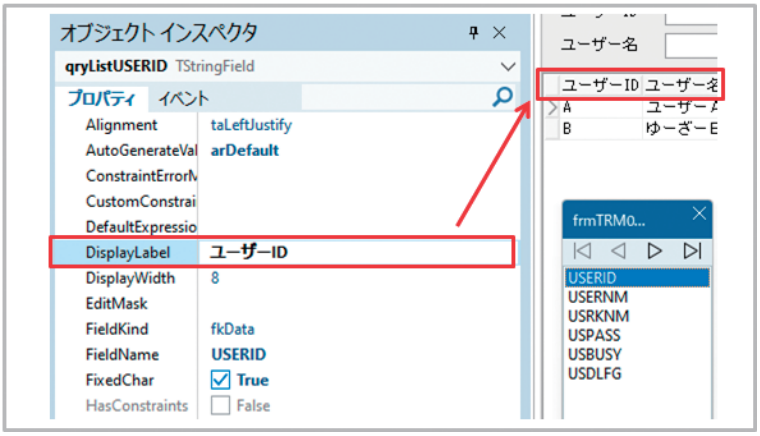

**図 24　設計画面上で明細を開く**


**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。

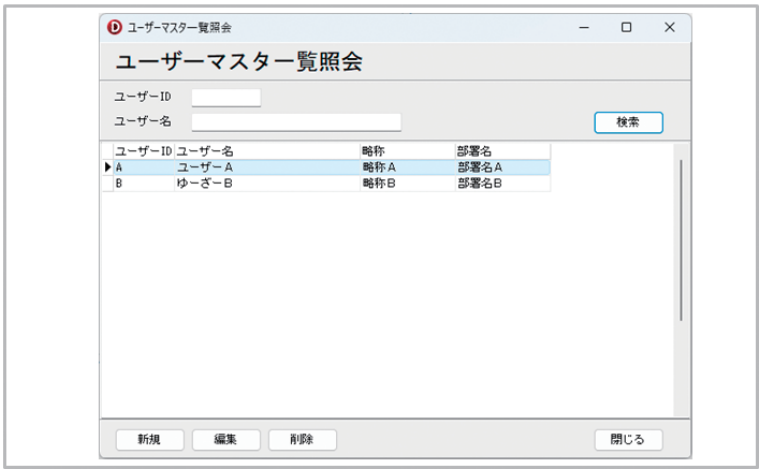
**図 26　明細タイトルの設定**




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**

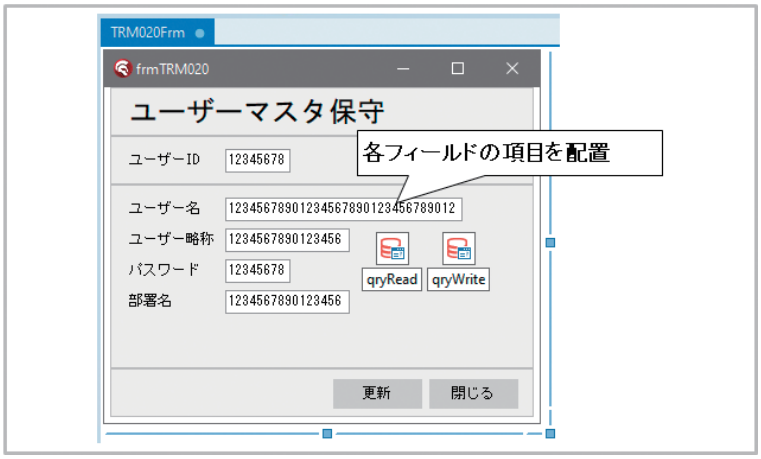
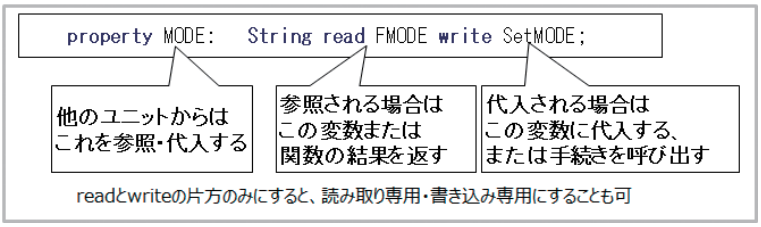



**図 29　プロパティ変数のイメージ**




**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。

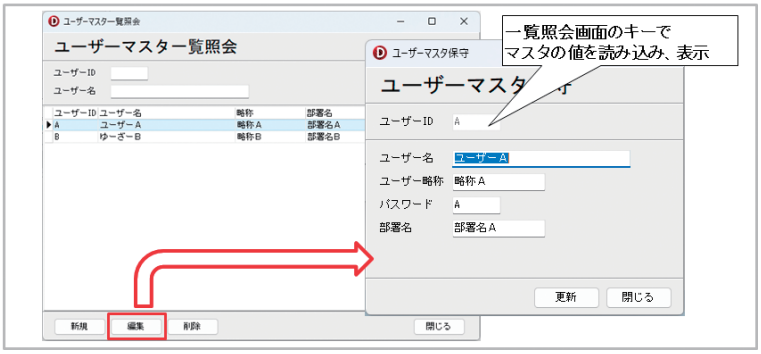
-新規モード（MODE＝"N"）：各入力項目をクリアした状態で画面を表示する。

-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。