前田 和寛

株式会社ミガロ.

システム事業部 システム2課

カスタマイズコンポーネント入門 ―開発効率向上テクニック

コンポーネントのカスタマイズ（継承、機能追加）を用いて、
Delphi/400 の開発効率向上ノウハウを解説する。

略歴
1983 年 11 月 21 日生まれ
2006 年大阪工業大学情報科学部卒
2006 年 04 月株式会社ミガロ. 入社
2006 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、システムの受託開発および保守対応などを担当している。

1. オブジェクト指向プログラミング

システムの構築・運用において、開発効率の向上を図ることは非常に重要である。プログラミング／メンテナンスの効率化を目的として、Delphi 言語を選択する開発者も多いのではないだろうか。Delphi 言語の特徴はコンポーネントパレット、オブジェクトインスペクタを中心としたビジュアル開発ができることだが、本質はオブジェクト指向を取り入れている点にある。

オブジェクト指向プログラミングとは、オブジェクトを中心に考えていくプログラミング手法のことだ。オブジェクト指向プログラミング言語では、その言語が備えるクラスと継承の仕組みを利用すると、開発効率が格段に向上する。

Delphi/400 ももちろんクラスと継承の仕組みを備えており、その代表的なクラス継承は以下である。

・コンポーネントの継承（以下、カスタマイズとする）
・フォームの継承

今回は、上記のうち、コンポーネントの継承、カスタマイズを行うと、どのように開発効率が向上するのか考えてみよう。

通常、システムを運用・稼働していると、エンドユーザーから変更要望が出てくる。例えば、「照会画面でデータの並び順を変更したい」という要望があがったとする。この場合、対象画面に「データの並び順を変更する」処理を追加すればよい。しかし、対象が 1 画面から 10 画面に増えると、それに伴い開発工数も増えてしまう。一方、コンポーネントをカスタマイズする場合、対象画面が増えても開発工数は画面数に依存しない。また、コンポーネントで処理記述を一元管理できるため、画面ごとの動作検証も必要最低限で済む。

本稿では、コンポーネントのカスタマイズにより、Delphi/400 の開発効率を向上させる方法を紹介したい。

なお、フォームの継承方法については『ミガロ. テクニカルレポート№1 2008 秋』のレポート「フォーム継承による効率向上開発手法」で分かりやすく紹介しているので、ぜひ参考にしていただきたい。

2.コンポーネントのカスタマイズ

コンポーネントの継承の概念について、Delphi/400 の開発でよく利用する TEdit と TMaskEdit を例に考えてみよう。

TEdit と TMaskEdit はクラス定義が異なるが、コンポーネントの機能・性質についてはそれほど大きく変わらない。その違いは TMaskEdit のクラス名が示すとおり、EditMask プロパティが定義されているくらいだろう。TEdit と TMaskEdit は継承元のコンポーネントが共通のため、機能・性質が非常に近いのである。これらは、コンポーネントの継承関係を図式化して確認すると非常に

図1　TEditとTMaskEditの継承関係

図2　コンポーネントの新規作成

図3　継承元コンポーネントの指定

分かりやすい。【図1】

●カスタマイズのポイント

カスタマイズを行う際には注意すべきポイントが存在するので、以下に説明する。これらのポイントに気を付ければ、コンポーネントのカスタマイズも容易にできる。

(1) コンポーネントを継承する

カスタマイズはコンポーネントを継承利用する。つまり、新しい機能・性質を追加する際、継承元のコンポーネントで定義されたプロパティ、手続き、関数を利用できる。

例えば、TMaskEditをもとに作成したカスタマイズコンポーネントをコンポーネントA、TEditをもとに作成したカスタマイズコンポーネントをコンポーネントBとする。コンポーネントAはTMaskEditを継承しているため、EditMaskプロパティを利用できる。それに対して、コンポーネントBの継承元であるTEditにはもちろんEditMaskプロパティが定義されていないため、コンポーネントBではEditMaskプロパティを利用できない。

(2) 動的なコーディングを行う

コンポーネントは不特定多数のプロジェクトで利用するため、処理記述を動的に行う必要がある。通常、TButtonのActionプロパティにAction1が割り当てられている場合、Action1と指定すればよい。しかし、カスタマイズコンポーネントの処理記述では、同様にActionプロパティの設定内容を指定する場合、自身を基準としてSelf.Actionと動的に表現する（Selfは省略可）。

●カスタマイズの手順

ではここから、実際にDelphi/400でカスタマイズコンポーネントを作成する手順を紹介していこう。今回は、TDBGridを継承してTTRDBGridというカスタマイズコンポーネントを作成する。なお、図表の開発環境はDelphi/400 Version2010を使用している。

①コンポーネントを新規作成する

メニューの［ファイル｜新規作成｜その他］を選択する。新規作成ウィンドウの［Delphiプロジェクト｜Delphiファイル｜コンポーネント］を選択する。【図2】

②継承元コンポーネントを指定する

VCLコンポーネントの新規作成ウィンドウの第1画面で、コンポーネントのリストからTDBGridを選択し、［次へ］ボタンを押下する。【図3】

③新規コンポーネントの設定を行う

VCLコンポーネントの新規作成ウィンドウの第2画面で、「クラス名」にTTRDBGrid、「パレットページ名」にTechnicalReportを設定する。

今回のカスタマイズコンポーネントのインストール時、この「パレットページ名」の設定値が、新規追加されるコンポーネントパレットのパレットページ名となる。【図4】

④pasファイル名を指定する

VCLコンポーネントの新規作成ウィンドウの第2画面で、「ユニット名」にTechnicalReportControl.pasを指定する。作成するカスタマイズコンポーネントのpasファイル名、および、保存先を指定する。今回は、保存先にC:\Projects\TechnicalReport\Libを指定する。【図4】【図5】

手順①～④により、TDBGridを継承したTTRDBGridというカスタマイズコンポーネントを作成できた。ただし、生成されたソースを見れば分かるように、まだ個別に機能を追加していないため、TTRDBGridはTDBGridと機能的に変わらないテンプレートである点に注意してほしい。【図6】

作成したコンポーネントをDelphi/400開発環境へインストールするには、パッケージファイルが必要となる。続けて、パッケージファイルを作成する手順を紹介しよう。

⑤パッケージファイルを新規作成する

メニューの［ファイル｜新規作成｜パッケージ］を選択する。Package1.bplが新規作成されたのが確認できる。【図7】

⑥コンポーネントを追加する

メニューの［プロジェクト｜プロジェクトに追加］を選択する。プロジェクトに追加ウィンドウで、C:\Projects\TechnicalReport\LibのTechnicalReportControl.pasを指定する。

コンポーネントの追加後、メニューの［ファイル｜すべて保存］を選択し、同じフォルダにTechnicalReport.dprojとして保存する。【図8】

手順⑤⑥により、コンポーネントのインストール用パッケージファイルが作成できた。なお、これまで作成したコンポーネントやパッケージファイルは配布を行うことで、他のユーザーがDelphi/400開発環境へインストールして共有することができる。

最後に、パッケージファイルを使って、コンポーネントをインストールする手順を紹介しよう。

⑦パッケージファイルを開く

メニューの［ファイル｜開く］を選択する。プロジェクトを開くウィンドウでTechnicalReport.dprojを選択し、パッケージファイルを開く。

⑧コンポーネントをインストールする

メニューの［プロジェクト｜プロジェクト名を再構築］を選択する。再構築後、プロジェクトマネージャのTechnicalReport.bplを右クリックする。表示メニューの「インストール」を選択する。

インストール後は図のように、TTRDBGridを正常にインストールできたことを確認できる。【図9】

⑨Delphi/400開発環境のライブラリパスの設定を行う

メニューの［ツール｜オプション］を選択する。オプションウィンドウの［環境オプション｜Delphiオプション｜ライブラリWin32｜ディレクトリ｜ライブラリパス］の［…］ボタンを押下する。ディレクトリウィンドウでC:\Projects\TechnicalReport\Libを指定する。【図10】

インストールしたコンポーネントを開発で利用する場合、Delphi/400開発環

図4　新規コンポーネントの設定

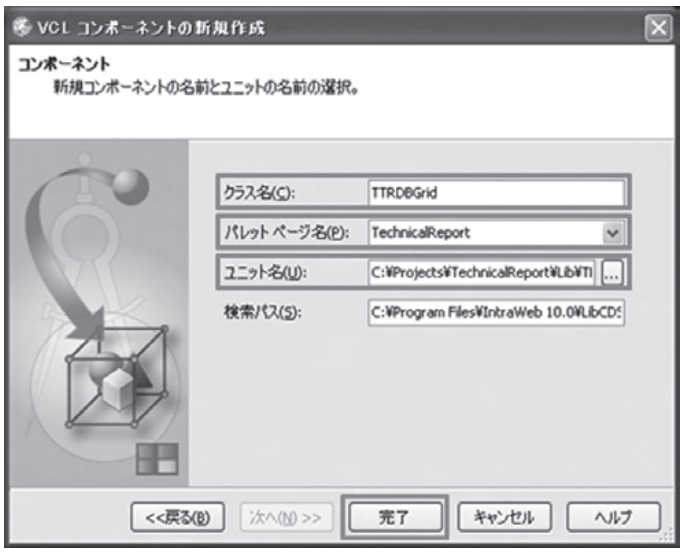


図5　pasファイルの指定

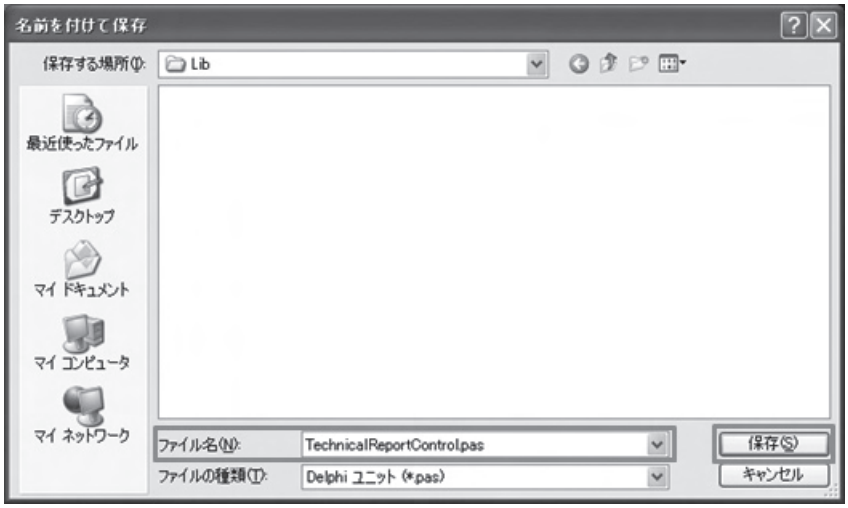
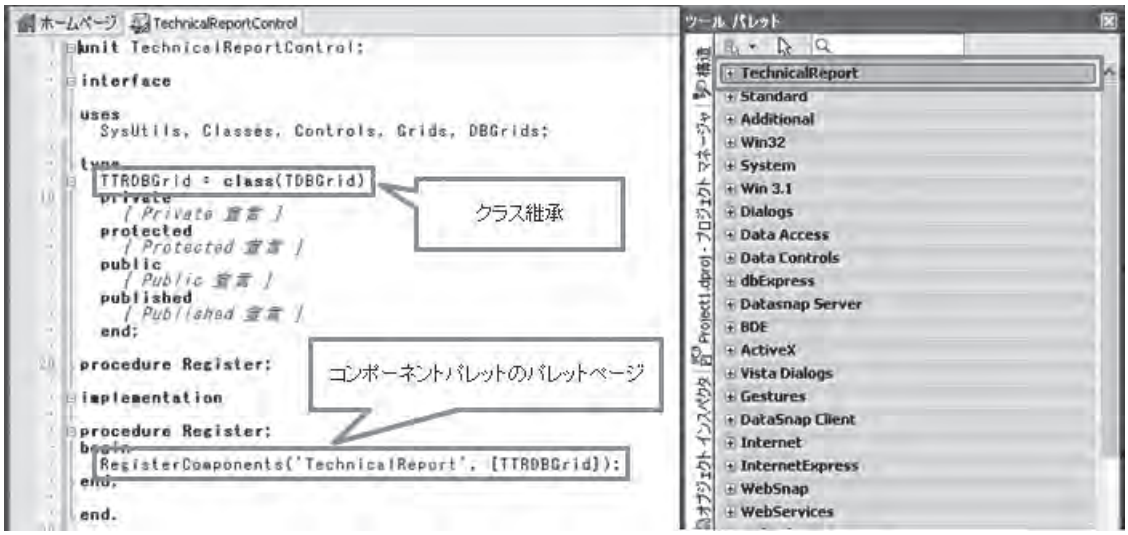


図6　作成されたTTRDBGrid



境にパッケージファイルの格納フォルダをライブラリパスとして登録する必要がある。

以上で、カスタマイズコンポーネントを作成し、その作成したカスタマイズコンポーネントを、Delphi/400 開発環境にインストールできた。Delphi/400 の再起動後、カスタマイズコンポーネント TTRDBGrid が利用可能となる。

3.コンポーネントへの機能追加

本章では、カスタマイズコンポーネントへの機能追加を紹介したい。今回は2章で作成した TTRDBGrid に「データの並び順を変更する」機能を追加してみよう。

【図 11】のような TTRDBGrid を利用している照会画面で、明細部のタイトルをダブルクリックするごとに、選択列を基準として並び順を変更できるようにしよう。【図 11】

データの並び順は「昇順設定→降順設定→設定なし→昇順設定→…」、と設定が切り替わるようにする。また、並び順の設定が分かりやすいように、選択列のタイトルのフォント色も変更する。以下のような切り替えになる。

昇順設定：赤色
↓
降順設定：青色
↓
設定なし：黒色
↓
昇順設定：赤色
↓

●データの並び順

処理を実装する前に、データの並び順を変更する方法を説明したい。

データの並び順は、TTRDBGrid に紐付くデータセットのインデックスを利用する。照会画面の作成時、TTable や TQuery を利用することがよくある。しかし、並び順の変更はローカルキャッシュに取り込んだデータのインデックスを利用したいので、今回はデータセットに TClientDataSet を前提とする。

次に、TClientDataSet のインデックス設定について説明する。インデックス設定には、IndexName と IndexFieldNames の2種類のプロパティが存在する。前者の IndexName は、Index 定義（フィールドの優先順位、フィールドごとの昇順／降順を指定したもの）をあらかじめ準備して設定する必要がある。後者の IndexFieldNames は、フィールドの優先順位は指定できるが、降順の指定はできない。今回は昇順／降順を選択できるようにしたいので、Index 定義と IndexName を利用して並び順を変更する。なお、IndexName と IndexFieldNames は互いに排他関係にあるため、どちらか一方しか設定できない。

●機能追加

では、機能追加に必要な処理記述を以下に洗い出す。

・タイトルをダブルクリックするごとに、並び順を変更する。
・変更前のインデックス設定を退避／復元する。
・選択列のフォント色を変更する。

ここでは、メインの処理である「タイトルをダブルクリックするごとに、並び順を変更する」を中心に説明したい。

まず、タイトルのダブルクリックにより、並び順を変更するため、選択列を特定する必要がある。WMLButtonDown イベントで、マウスカーソル位置から選択列を取得し、WMLButtonDblClk イベントにて、インデックスの設定処理を呼び出す。【ソース 1】【ソース 2】

次に、特定した選択列をもとに、インデックスを設定する。ここで「動的なコーディングを行う」というポイントを思い出してほしい。

コンポーネントの処理記述時、TTRDBGrid に紐付くデータセットの Name や選択列に設定されるフィールドの FieldName は不明である。そのため、自身を基準として、紐付くデータセットは DataSource.DataSet、選択列に設定されるフィールドは Columns.Items[FCoord.X-1].FieldName と表現する。

また、今回の Index 定義は昇順／降順を問わず、GRID_IDX と命名する。TClientDataSet の AddIndex を利用し、選択列のフィールドと昇順／降順を設定する。昇順の場合、[ixCaseInsensitive]、降順の場合、[ixDescending] を指定する。

後は IndexName に GRID_IDX を設定することで、並び順の変更ができる。【ソース 3】

なお、その他の処理の「変更前のインデックス設定を退避／復元する」「選択列のフォント色を変更する」については、末尾の参考ソースを参照していただきたい。【ソース 4】【ソース 5】【ソース 6】【ソース 7】

では、コンポーネントへの機能追加ができたので、コンポーネントの再インストール後に照会画面の動作を確認してみよう。明細部のタイトルをダブルクリックするごとに、昇順や降順に明細データの並び順が切り替わることが確認できるだろう。また、同様に TTRDBGrid を利用している画面が他にもあれば、すべての画面で並び順の変更が可能になる。【図 12】【図 13】

4.まとめ

コンポーネントのカスタマイズと聞くと難しいイメージを抱くかもしれないが、処理手順を見ていただいたとおり、カスタマイズコンポーネントのテンプレートは簡単な操作で作成できる。また、コンポーネントへの機能追加についてもポイントを押さえれば、比較的簡単に実現できる。

一度、カスタマイズコンポーネントを作成・機能追加すれば、そのコンポーネントを利用しているすべての画面で、追加した機能が利用できるようになる。また、作成したコンポーネントは、他のユーザーもインストールして利用が可能だ。このようにコンポーネントのカスタマイズにより、Delphi/400 の開発効率を向上できることが理解いただけただろう。

本稿で紹介した TTRDBGrid の「データの並び順を変更する」機能を参考にして、コンポーネントのカスタマイズ、および独自の機能追加にぜひ挑戦してほしい。これらの技術情報がシステム開発の助けとなれば幸いである。

M

図7　パッケージファイルの新規作成

図8　コンポーネントの追加と保存

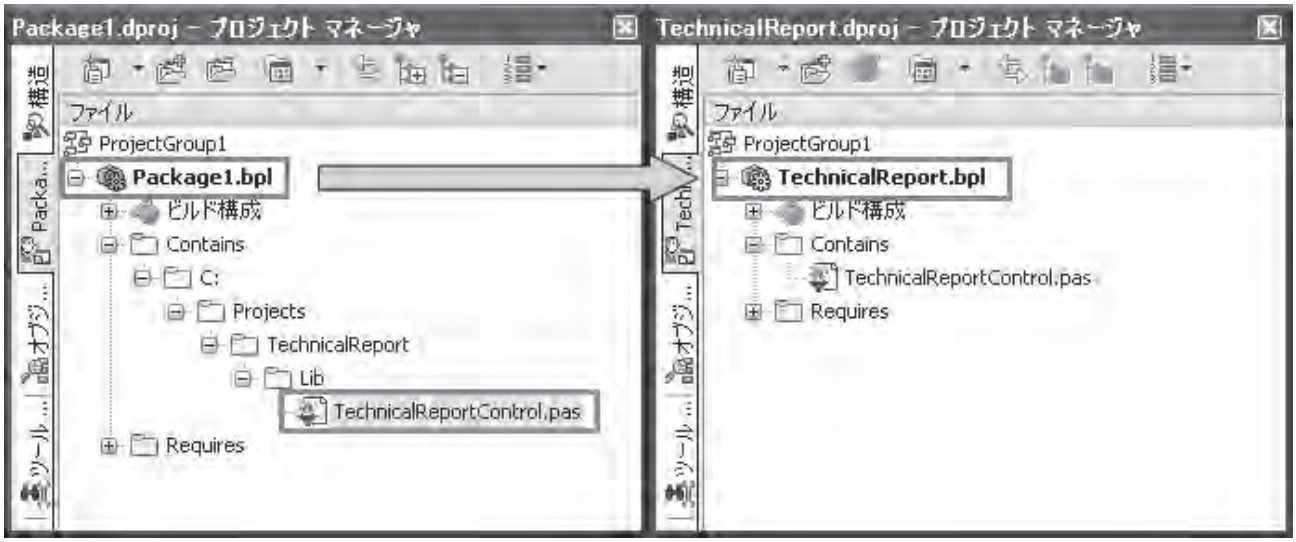
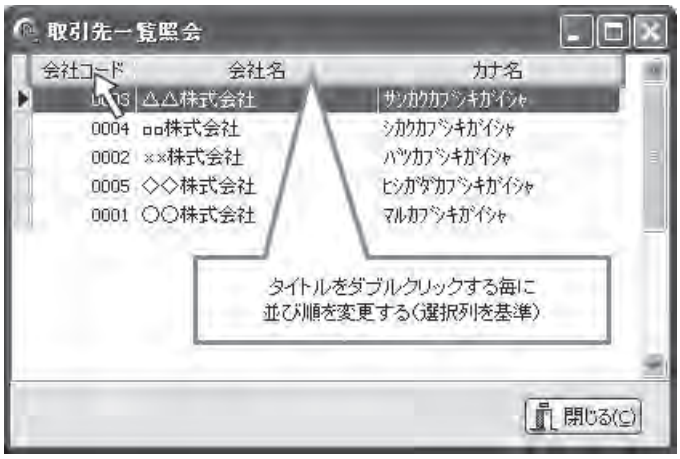


図9　コンポーネントのインストール

図10　ライブラリパスの設定

図11　照会画面と追加機能



ソース1　WMLButtonDownイベント

```
{*************************************************************************************
 目的：マウス左ボタンのダウン時処理
 引数：
 戻値：
*************************************************************************************}
procedure TTRDBGrid.WMLButtonDown(var Msg: TWMLButtonDown);
begin
  inherited;

  // セル位置を取得する
  FCoord := MouseCoord(Msg.XPos, Msg.YPos);
end;
```

選択列の特定

ソース2　WMLButtonDblClkイベント

```
{*************************************************************************
 目的：マウス左ボタンのダブルクリック時処理
 引数：
 戻値：
*************************************************************************}
procedure TTRDBGrid.WMLButtonDblClk(var Msg: TWMLButtonDblClk);
begin
  // タイトル行の場合、ソート処理を実施する
  if (FCoord.Y = 0) then
  begin
    // 1:昇順
    if (FsNowIndexField <> Columns.Items[FCoord.X-1].FieldName) or
       (FiNowSortKind = 0) then
      Sorting(1)

    // 2:降順
    else if (FiNowSortKind = 1) then
      Sorting(2)

    // 0:解除
    else if (FiNowSortKind = 2) then
      Sorting(0);
  end

  // タイトル行でない場合、既定処理を実施する
  else
  begin
    inherited;
  end;
end;
```

並び順の変更処理を呼出す

ソース3　Sorting手続き

```
{*************************************************************************
 目的　ソート処理
 引数：ソート種別(1:昇順、2:降順、0:解除)
 戻値：
*************************************************************************}
procedure TTRDBGrid.Sorting(ASortKind: Integer);
var
  ADataSet: TClientDataSet;
begin
  inherited;

  // TClientDataSetが設定されている、かつ、
  // 指定列のフィールドが計算項目の場合、
  if (Assigned(DataSource)) and (Assigned(DataSource.DataSet)) and
     (DataSource.DataSet.Active) and
     (DataSource.DataSet is TClientDataSet) and
     (FCoord.X >= 1) and (FCoord.X <= Columns.Count) and
     (Columns.Items[FCoord.X-1].Field.FieldKind <> fkCalculated) then
  begin
    ADataSet := (DataSource.DataSet as TClientDataSet);

    // 本来の設定の退避
    if (FiNowSortKind = 0) then
    begin
      FsBaseIndexName       := ADataSet.IndexName;
      FsBaseIndexFieldNames := ADataSet.IndexFieldNames;
    end;

    // Indexの初期化
    try
      ADataSet.DeleteIndex(GRID_IDX);
    except
      //
    end;

    // 1:昇順
    if (ASortKind = 1) then
    begin
      ADataSet.AddIndex(GRID_IDX, Columns.Items[FCoord.X-1].FieldName, [ixCaseInsensitive]);
      ADataSet.IndexName := GRID_IDX;

      FiNowSortKind   := 1;
      FsNowIndexField := Columns.Items[FCoord.X-1].FieldName;
    end

    // 2:降順
    else if (ASortKind = 2) then
    begin
      ADataSet.AddIndex(GRID_IDX, Columns.Items[FCoord.X-1].FieldName, [ixDescending]);
      ADataSet.IndexName := GRID_IDX;

      FiNowSortKind   := 2;
      FsNowIndexField := Columns.Items[FCoord.X-1].FieldName;
    end

    // 0:解除
    else if (ASortKind = 0) then
    begin
      ADataSet.IndexName       := FsBaseIndexName;
      ADataSet.IndexFieldNames := FsBaseIndexFieldNames;

      FiNowSortKind   := 0;
      FsNowIndexField := '';
    end;

    ADataSet.First;
  end;
end;
```

TTRDBGridを基点とした動的表現

変更前インデックスの退避

TTRDBGridを基点とした動的表現

参考ソース4　TTRDBGridの宣言部

```
ページ  TechnicalReportControl
unit TechnicalReportControl;

interface

uses
  SysUtils, Classes, Controls, Grids, DBGrids,
  DB, DBClient, Menus, ActnList, Messages, Windows, Graphics;

type
  TTRDBGrid = class(TDBGrid)
  private
    { Private 宣言 }
    // 本来の設定
    FsBaseIndexName: String;          // IndexName
    FsBaseIndexFieldNames: String;    // IndexFieldNames

    // 現在の設定
    FiNowSortKind: Integer;           // ソート種別(1:昇順, 2:降順, 0:設定なし)
    FsNowIndexField: String;          // ソート対象フィールド

    // 列の位置を保持
    FCoord: TGridCoord;

    procedure WMLButtonDown(var Msg: TWMLButtonDown); message WM_LBUTTONDown;
    procedure WMLButtonDblClk(var Msg: TWMLButtonDblClk); message WM_LBUTTONDBLCLK;
    procedure Sorting(ASortKind: Integer);
    procedure BeforeClose(DataSet: TDataSet);
  protected
    { Protected 宣言 }
    procedure Loaded; override;
    procedure DrawCell(ACol, ARow: Longint; ARect: TRect;
      AState: TGridDrawState); override;
  public
    { Public 宣言 }
  published
    { Published 宣言 }
  end;

procedure Register;

implementation

const
  GRID_IDX: string = 'GRID_IDX';

procedure Register;
begin
  RegisterComponents('TechnicalReport', [TTRDBGrid]);
end;
```

参考ソース5　BeforeClose手続き

```
{*******************************************************************************
 目的：データセットClose前処理
 引数：
 戻値：
*******************************************************************************}
procedure TTRDBGrid.BeforeClose(DataSet: TDataSet);
var
  ADataSet: TClientDataSet;
begin
  inherited;

  // ソート設定を行っている場合、
  if ((FiNowSortKind = 1) or (FiNowSortKind = 2)) then
  begin
    ADataSet := (DataSource.DataSet as TClientDataSet);

    ADataSet.DeleteIndex(GRID_IDX);
    FiNowSortKind   := 0;
    FsNowIndexField := '';

    ADataSet.IndexName       := FsBaseIndexName;
    ADataSet.IndexFieldNames := FsBaseIndexFieldNames;
  end;
end;
```

変更前インデックスの復元

参考ソース6　Loadedイベント

```
{*******************************************************************************
 目的：ロード時処理
 引数：
 戻値：
*******************************************************************************}
procedure TTRDBGrid.Loaded;
begin
  inherited;

  // TClientDataSetが設定されている場合、
  // BeforeClose時にソート解除処理を行うように設定
  if not(csDesigning in ComponentState) and
     (Assigned(DataSource)) and (Assigned(DataSource.DataSet)) and
     (DataSource.DataSet is TClientDataSet) then
  begin
    (DataSource.DataSet as TClientDataSet).BeforeClose := BeforeClose;
  end;
end;
```

参考ソース7　DrawCellイベント

```
{*******************************************************************************
 目的：セル描画処理
 引数：
 戻値：
*******************************************************************************}
procedure TTRDBGrid.DrawCell(ACol, ARow: Integer; ARect: TRect;
  AState: TGridDrawState);
const
  // 常にセンター表示とする
  AlignFlags : array [TAlignment] of Integer =
    ( DT_CENTER or DT_WORDBREAK or DT_EXPANDTABS or DT_NOPREFIX,
      DT_CENTER or DT_WORDBREAK or DT_EXPANDTABS or DT_NOPREFIX,
      DT_CENTER or DT_WORDBREAK or DT_EXPANDTABS or DT_NOPREFIX );
var
  sText: String;
  Flags: LongInt;

  C: Integer;
  DrawFig: longint;
  Column: TColumn;
begin
  inherited DrawCell(ACol, ARow, ARect, AState);

  // タイトルの表示
  if (ARow = 0) and (ACol <> 0) then
  begin
    // 背景色の塗り潰し
    InflateRect(ARect, -1, -1);
    Canvas.Brush.Color := FixedColor;
    Canvas.FillRect(ARect);
    Canvas.Font.Assign(TitleFont);

    // フォント色の設定
    if (Assigned(DataSource)) and
       (Assigned(DataSource.DataSet)) and
       (DataSource.DataSet is TClientDataSet) then
    begin
      // ソート設定を行っている場合、
      if ((FiNowSortKind = 1) or (FiNowSortKind = 2)) and
         (Columns.Items[ACol-1].FieldName = FsNowIndexField) then
      begin
        //昇順ソート
        if (FiNowSortKind = 1) then
        begin
          Canvas.Font.Color := clRed;
        end

        //降順ソート
        else if (FiNowSortKind = 2) then
        begin
          Canvas.Font.Color := clBlue;
        end;
      end;
    end;

    // 文字の再描画
    sText := Self.Columns[ACol-1].Title.Caption;
    Flags := DT_SINGLELINE or DT_VCENTER or AlignFlags[Columns[ACol-1].Title.Alignment];
    DrawText(Canvas.Handle, PChar(sText), Length(sText), ARect, Flags);
  end;
end;
```

選択列のフォント色を設定

図12　照会画面･会社コードの昇順設定

図13　照会画面･会社コードの降順設定