【セッションNo. 3】

プログラミングテクニックセッション
開発者が知りたい実践プログラミングテクニック！

株式会社ミガロ.
システム事業部　プロジェクト推進室
福井　和彦

【アジェンダ】

プログラミングテクニックセッション

1. 分かりやすいユーザーインターフェース
2. スケジュールボードの作成
3. カスタムコンポーネントの作成

1. 分かりやすいユーザーインターフェース

■ ユーザーがシステムに求めるもの

- 安定性（信頼性）
  - エラーやシステム障害が発生しない
  - セキュリティがしっかりしている
- **使い勝手**
  - 本来の業務がはかどる
    - ✓ 作業をできるだけ簡単に
    - ✓ 入力したデータの変更、修正を簡単に
    - ✓ 資料の作成を簡単に
- レスポンス
  - 処理速度、操作に対する反応が早い
- 汎用性、拡張性に優れている
  - 柔軟性があり新機能の追加や運用でカバーできる設計になっている

■ システムの評価

- 当初のシステム構築の目的を達成できたかどうか？
  - 出来上がったシステムが何を実現しているのかを事前に明確にしておくことが極めて重要
    - ➢ 非機能的側面
      - ✓ 売上がどのくらい増えたらOK
      - ✓ コストがどのくらい削減されたらOK
      - ✓ 平均リードタイムがどのくらい短縮されたらOK
      - ✓ Etc…
    - ➢ 機能的側面
      - ✓ パフォーマンスがどの程度ならOK
      - ✓ レスポンスがどの程度ならOK
      - ✓ バグの少なさ
      - ✓ **操作性の良さ**
      - ✓ Etc…

■ 操作性の良さ ⇒ 分かりやすさ

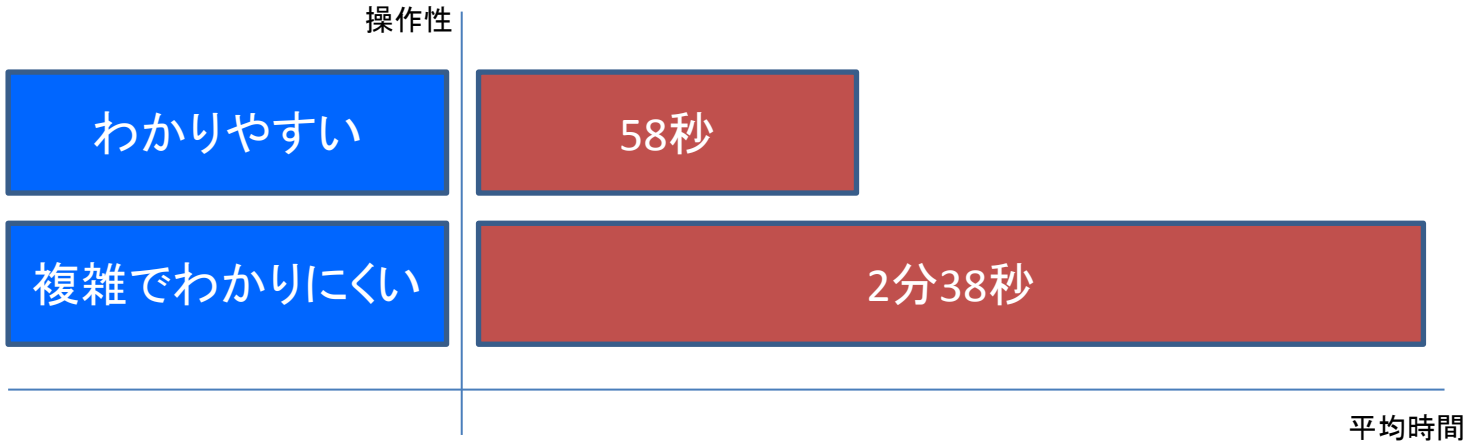
- “分かりやすさ”による効果
  - システムを使うまで
    - ✓ システムがあまりに複雑すぎるせいで、一人前のオペレータになるまでに集中的なトレーニングが必要

    →分かりやすさを重視したデザインによって、研修期間を日単位、場合によっては週単位で短縮できる
  - システムを使う
    - ✓ 求める情報を取得するのに要する時間に差異が発生
    - ✓ 誤った操作によるミスが発生

    →オペレーターの作業効率向上によりコストが削減できる

■ 操作性の良さ＝分かりやすさ

- 分かりやすさがもたらす実例
  - 社員名簿の検索



システムの使いやすさ、分かりやすさを改善することで、社員名簿から必要な情報を取得するたびに、1分40秒の節約ができる

U-site 「ニールセン博士のAlertbox」参照

■ 操作性の良さ＝分かりやすさ

- 分かりやすさがもたらす実例
  - 申請システム

	改善前	改善後
タスク達成率	33%	100%
タスク達成時間	53分	25分

- タスク達成率：ユーザーが利用して60分以内で目的を達成できた割合
- タスク達成時間：ユーザーが利用して目的を達成するまでに要した時間

分かりやすさを改善することで、利用者が短時間で目的を達成することができる

UNISYS TECHNOLOGY REVIEW　第 110 号，NOV. 2011
「Web 戦略を成功に導くためのユーザインタフェース設計プロセス」参照

■ 操作性の良さ＝分かりやすさ

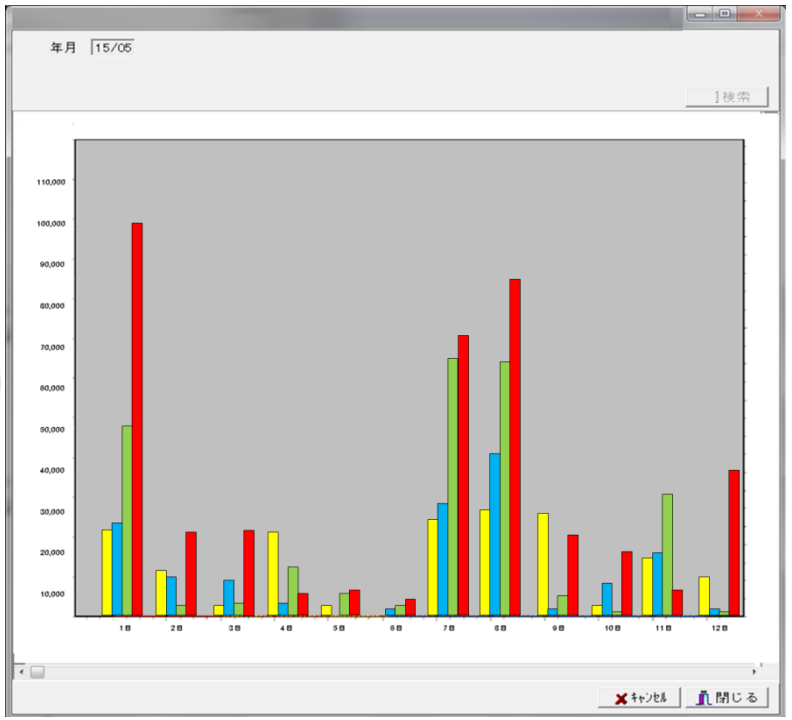
- 分かりやすさがもたらす実例
  - 分かりやすさの向上による効果例

業種	対策	効果
不動産会社	マンション販売促進サイト	閲覧者のモデルルーム予約率が3倍に上昇
新聞社	ニュースサイト改善	一人あたりのページ閲覧数が2倍に増加
携帯電話	請求書デザイン改善	コールセンターへの問い合わせ件数が半減
通信サービス	加入登録業務システム改善	登録処理業務効率が30%アップ

UNISYS TECHNOLOGY REVIEW　第 110 号，NOV. 2011
「Web 戦略を成功に導くためのユーザインタフェース設計プロセス」参照

■ 分かりやすいユーザーインターフェース

- 分かりやすいユーザーインターフェースとは？
  - ◆TDBGrid等の明細形式の照会画面をグラフで表示することで「視覚的」に変化が捉えられるような表現ができます



■ 分かりやすいユーザーインターフェース

- 分かりやすいユーザーインターフェースとは？
  - ◆スケジュール管理を行う場合、日付の範囲だけでなくバー表示を合せて表示することで、「直観的」に全体のスケジュールが把握できます

■ 分かりやすいユーザーインターフェース

- 分かりやすいユーザーインターフェースとは？
  - ◆ 入力画面等で各種チェックを行う場合、メッセージだけでなく該当項目の背景色を変更することで、「直観的」にチェック箇所が判断できます

■ 分かりやすいユーザーインターフェース

- そこで、今回は以下の実装方法をご紹介！
  - ◆分かりやすいスケジュールボードの作成
  - ◆強調する項目反転をカスタムコンポーネントとして作成

2. スケジュールボードの作成

■ スケジュール登録の比較

- 同じ機能でも、画面によって実際の作業時間にどれだけの違いがあるでしょうか？

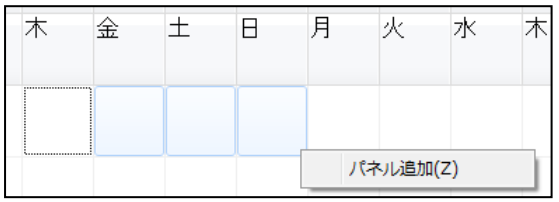
スケジュール入力

スケジュールボード

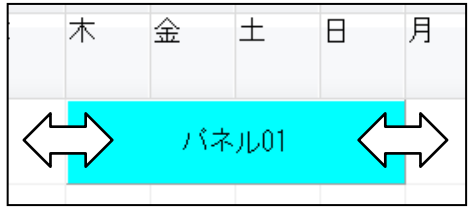
■ スケジュールボード作成の流れ

- ４つのステップ

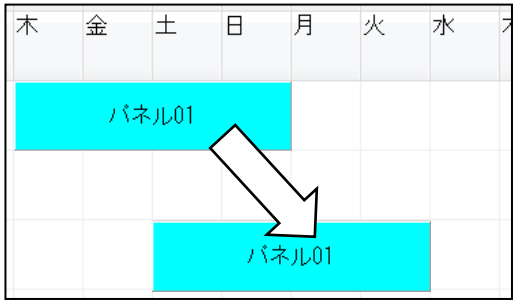
① グリッド上にパネルを表示



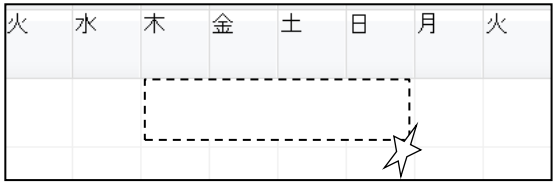
② ドラッグ＆ドロップで
パネルを伸縮



③ ドラッグ＆ドロップで
パネルを移動

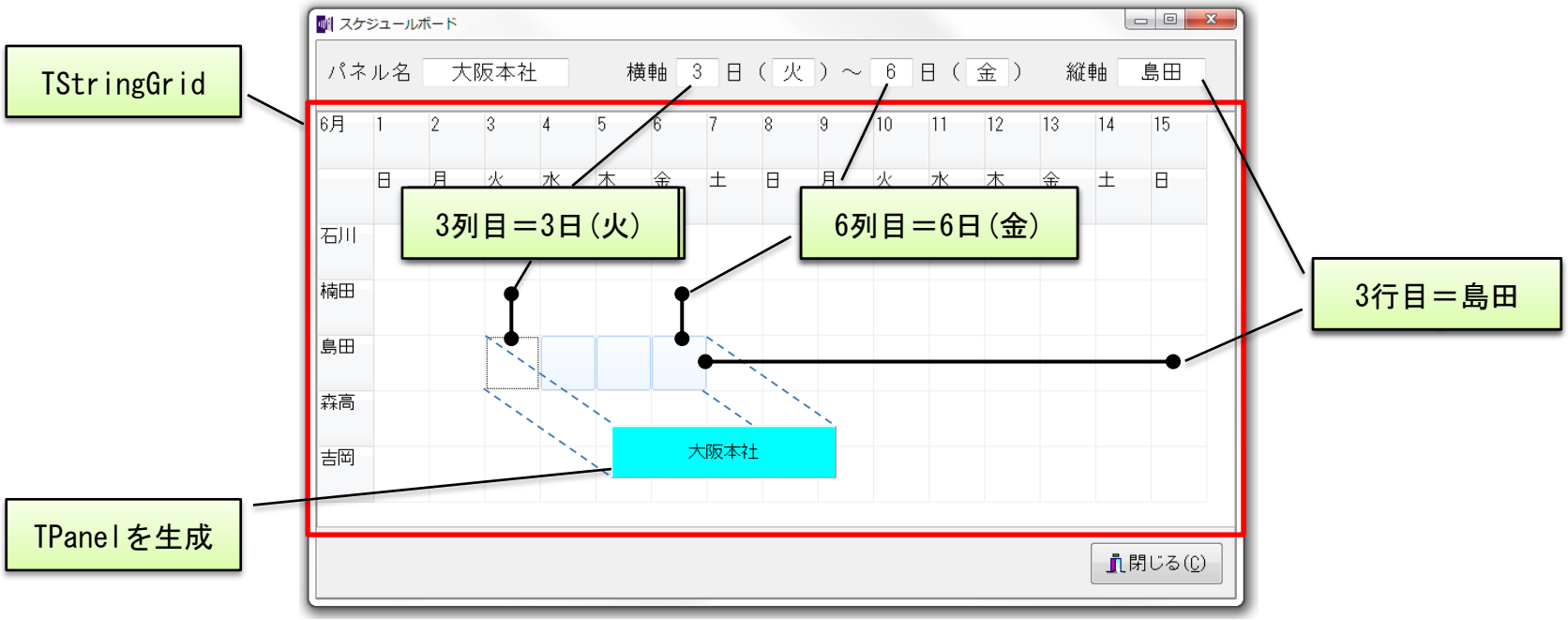


④ パネルを削除



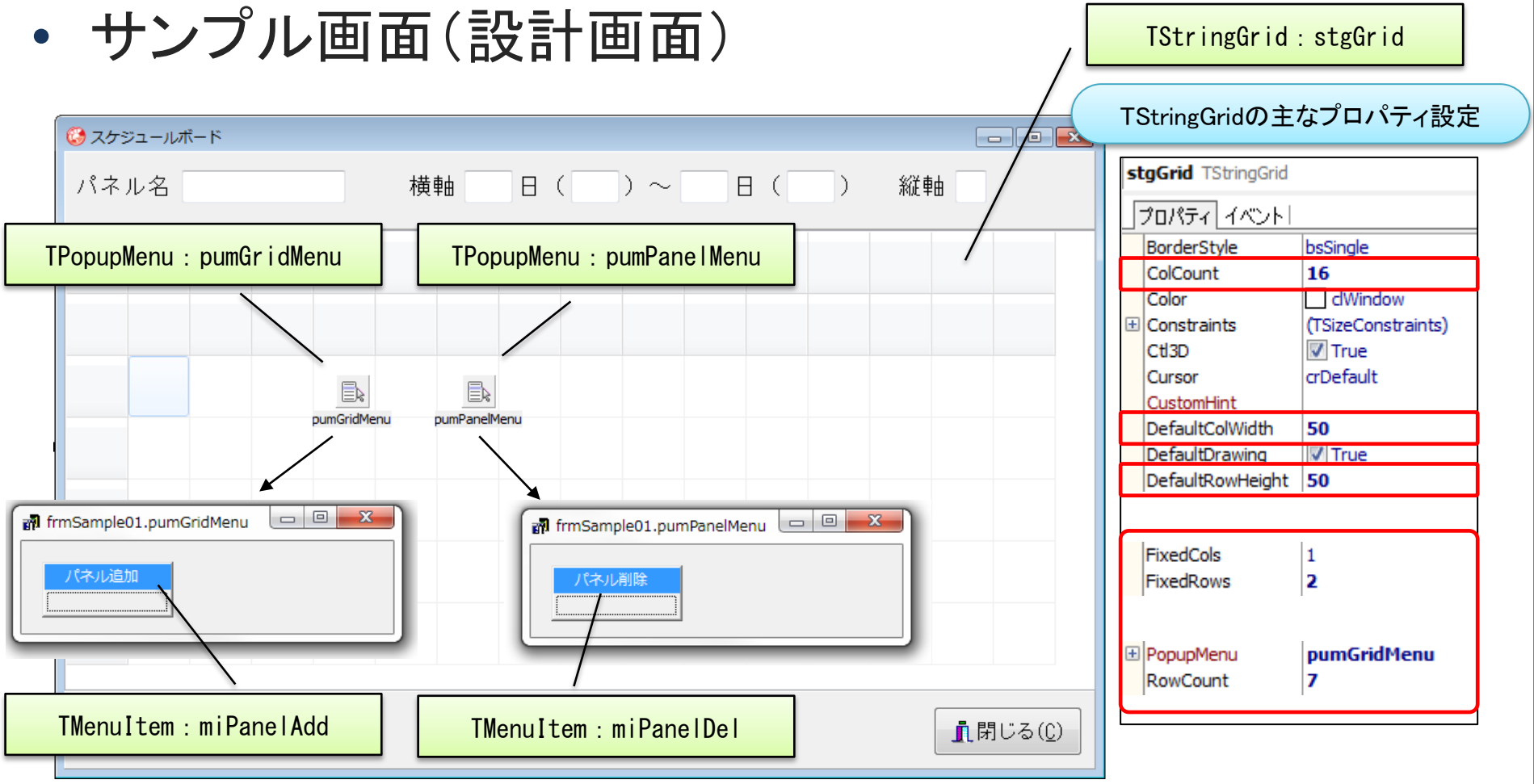
■ スケジュールボード

- 実装概要
  - カレンダー部には TStringGrid を使用する
  - TStringGrid上で選択した範囲に、動的に TPanel を生成し表示する
  - TStringGrid上に生成したTPanel の伸縮・移動・削除、そしてTPanel の位置情報より人・日付を捉えて画面の上段に表示する



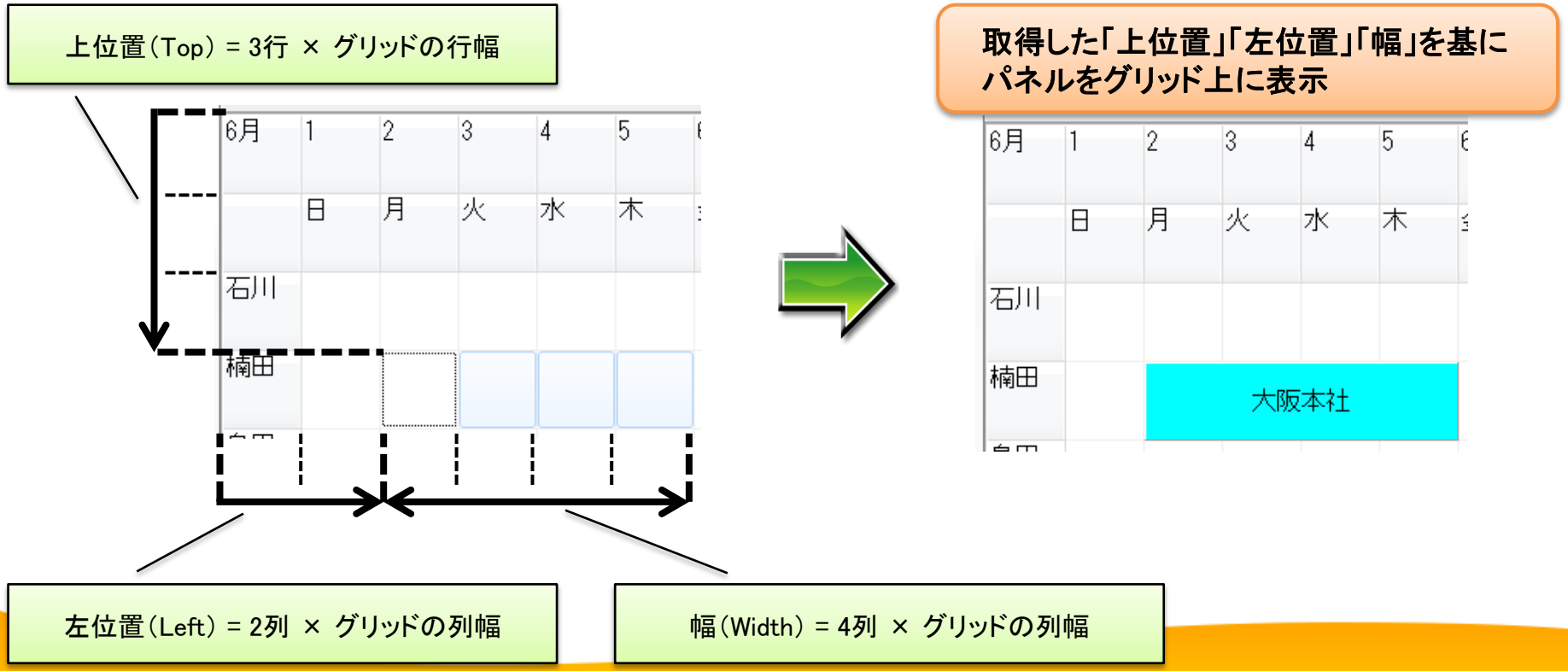
■ スケジュールボード

- サンプル画面（設計画面）



■ スケジュールボード

- ①パネルをグリッド上に表示する(1)
  - パネルをグリッド上に表示する方法
    - グリッド上で範囲選択されている「行」「選択開始列」「選択終了列」より、パネルを表示する「上位置」「左位置」「幅」を取得



■ スケジュールボード

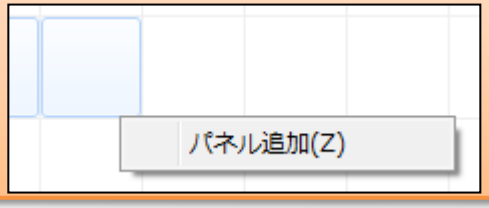
- ①パネルをグリッド上に表示する(2)

「パネル追加」メニューのonClickイベント<1>

```
  private
    FPanelID: Integer;  //パネルID
-------------------------------------------------------------------------
procedure TfrmSample01.miPanelAddClick(Sender: TObject);
var
  wkSelection: TGridRect;
  iStrCol: Integer;
  iEndCol: Integer;
  iRow: Integer;
  sPnlCap: string;
begin
  //グリッドの選択情報を取得
  wkSelection := stgGrid.Selection;

  //グリッドの選択位置を取得
  iRow    := wkSelection.Top;
  iStrCol := wkSelection.Left;
  iEndCol := wkSelection.Right;
```

グリッド上で範囲選択されている「行」「選択開始列」「選択終了列」を取得する

グリッドの右クリックメニューにある「パネル追加」メニューのonClickイベントに実装



■ スケジュールボード

- ①パネルをグリッド上に表示する(3)

「パネル追加」メニューのonClickイベント＜2＞

```
//パネル名入力サブ画面を開く
frmSample01Sub := TfrmSample01Sub.Create(Self);
try
  frmSample01Sub.ShowModal;
  //パネルのキャプションを取得
  sPnlCap := frmSample01Sub.PnlName;
finally
  frmSample01Sub.Release;
end;

//パネルIDの採番
Inc(FPanelID);

//新規パネル配列の追加
SetLength(FArrayPanel, FPanelID);

//新規パネル作成
SetPanel(iRow, iStrCol, iEndCol, FPanelID, sPnlCap);
```

複数パネルを作成した際、選択したパネルを
識別できるようにパネルIDを設けて採番する
→ 作成したパネルのTagプロパティで保持する
（SetPanelプロシージャにて記述）

作成するパネルは配列で管理する

グリッド上で範囲選択されている「行」「選択開始列」
「選択終了列」を基にパネルを描画する
（処理内容は次々頁以降参照）

■ スケジュールボード

- ①パネルをグリッド上に表示する(4)

「パネル追加」メニューのonClickイベント＜2＞

```
  //パネル情報表示
  mePnlName.Text   := sPnlCap;
  meDayF.Text      := stgGrid.Cells[iStrCol, 0];
  meWeekF.Text     := stgGrid.Cells[iStrCol, 1];
  meDayT.Text      := stgGrid.Cells[iEndCol, 0];
  meWeekT.Text     := stgGrid.Cells[iEndCol, 1];
  meNo.Text        := stgGrid.Cells[0, iRow];
end;
```

グリッド上で範囲選択されている「行」「選択開始列」「選択終了列」を基に追加パネルの情報を画面の上段に表示

■ スケジュールボード

- ①パネルをグリッド上に表示する(5)

SetPanelプロシージャ<1>

```
  private
    procedure SetPanel(ARow, AStrCol, AEndCol, APanelID: Integer; AJOKN: String); //パネル作成
-------------------------------------------------------------------------------

const
  cDayWidth    = 51;  // カレンダーの1日の幅(50)＋罫線の幅(1)
  cItemHeight = 51;  // カレンダーの行の幅(50)＋罫線の幅(1)
-------------------------------------------------------------------------------

procedure TfrmSample01.SetPanel(ARow, AStrCol, AEndCol, APanelID: Integer; AJOKN: String);
var
  wkPanel: TPanel;
begin
  //パネル生成
  FArrayPanel[APanelID - 1] := TPanel.Create(Self);
  wkPanel := FArrayPanel[APanelID - 1];

  //パネル表示位置・サイズ指定
  wkPanel.Left    := cDayWidth * AStrCol;
  wkPanel.Top     := cItemHeight * ARow;
  wkPanel.Height  := cItemHeight;
  wkPanel.Width   := (AEndCol - AStrCol + 1) * cDayWidth;
```

「パネル追加」メニューのonClickイベントで採番したしたパネルIDを使用して、配列でパネルを管理する

グリッド上で範囲選択されている「行」「選択開始列」「選択終了列」より、パネルを表示する「上位置」「左位置」「幅」を取得

■ スケジュールボード

- ①パネルをグリッド上に表示する(6)

SetPanelプロシージャ＜2＞

```
//表示色の指定
wkPanel.Color := clAqua;
//コンポーネント名指定
wkPanel.Name    := 'BarPanel_' + IntToStr(APanelID);
//表示文字列指定
wkPanel.Caption := AJOKN;

wkPanel.Font.Style := [];
wkPanel.BevelKind  := bkNone;
wkPanel.BevelOuter := bvRaised;

//親コンポーネントの指定
wkPanel.Parent  := stgGrid;

//TagにパネルIDを保持する
wkPanel.Tag := APanelID;

wkPanel.ParentShowHint   := False;
wkPanel.ParentColor      := False;
wkPanel.ParentBackground := False;
```

パネルの親コンポーネントをグリッドに設定する

「パネル追加」メニューのonClickイベントで採番した
したパネルIDをTagプロパティで保持する

■ スケジュールボード

- ①パネルをグリッド上に表示する(7)

SetPanelプロシージャ＜3＞

```
  //ポップアップメニューの指定
  wkPanel.PopupMenu := pumPanelMenu;

  //イベントの設定
  wkPanel.OnMouseDown  := PanelMouseDown;
  wkPanel.OnMouseMove  := PanelMouseMove;
  wkPanel.OnMouseUp    := PanelMouseUp;
  wkPanel.OnMouseLeave := PanelMouseLeave;

  wkPanel.Visible := True;
end;
```

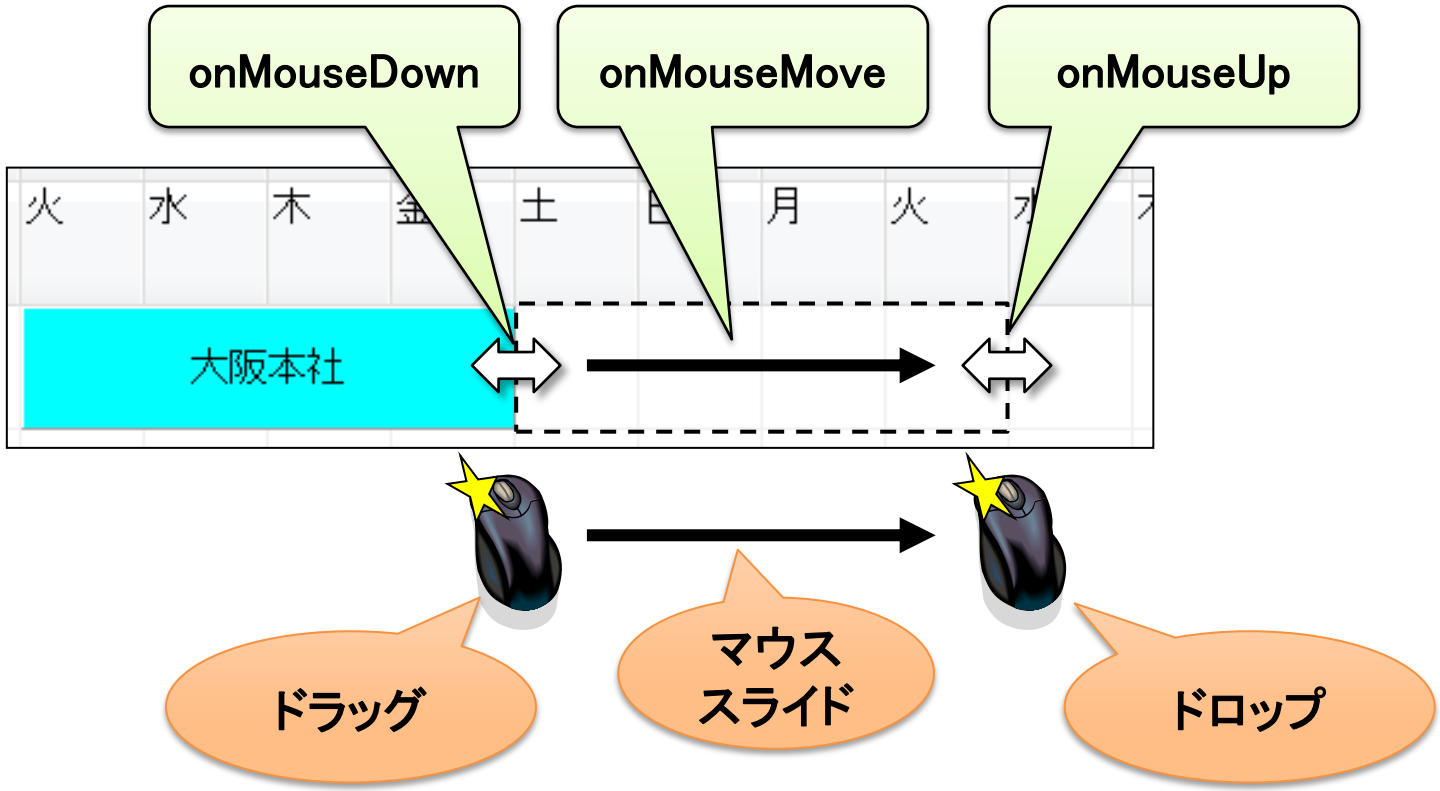
「パネル削除」のPopupMenuを指定する

パネルの伸縮、移動で使用する各イベントを指定する

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる（1）

パネルのイベントを使用してドラッグ＆ドロップで伸縮を実装！



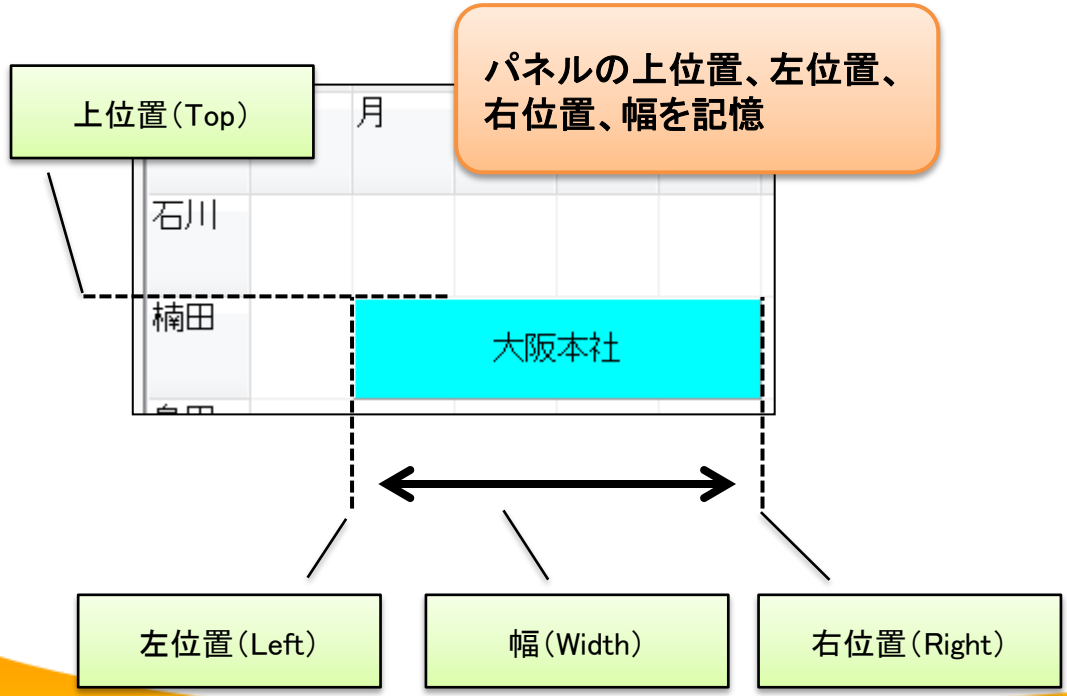
■ スケジュールボード

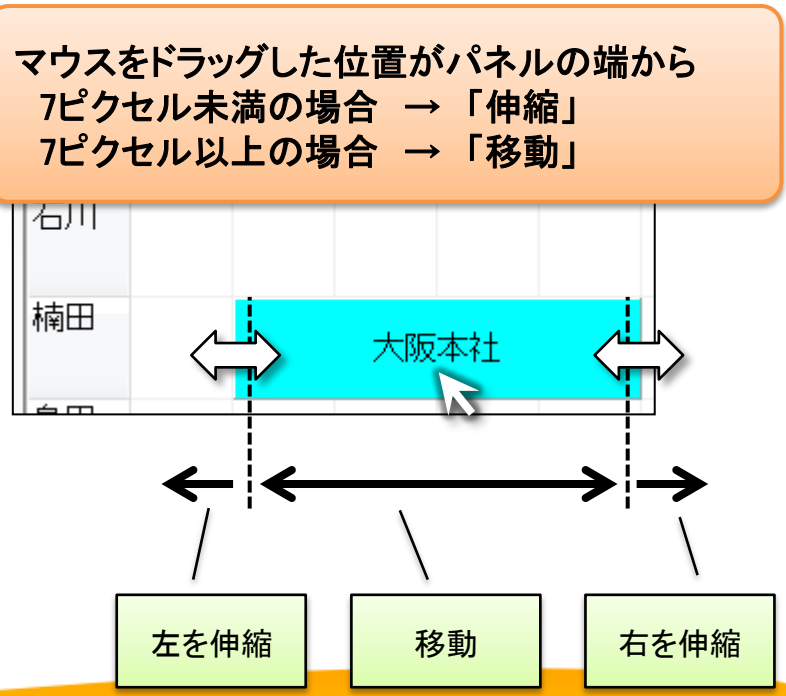
- ②パネルをドラッグ＆ドロップで伸縮させる（2）

“onMouseDown” イベントで実装すること

1）ドラッグしたパネルの位置と幅を記憶

2）パネルを伸縮するのか移動するのかを判断





■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(3)

パネルのonMouseDownイベント<1>

```
  private
    procedure PanelMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
      X, Y: Integer);
-------------------------------------------------------------------------------
procedure TfrmSample01.PanelMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
  X, Y: Integer);
var
  Coord: TGridCoord;
begin
  //マウスダウン処理を実行しているときは、別の処理はしない
  if FMouseDownFLG or FMouseLeftDownFLG or FMouseRightDownFLG then Abort;

  //選択したパネルのIDを変数に保持
  FSelectPanelID := (TPanel(Sender).Tag);

        ～ 省略

 if Button = mbLeft then
  begin
    FPanelX := X;    //パネルのｘ座標を，panelxに保存する
```

FSelectPanelIDはパネルを削除する際に使用

マウスの左ボタンが押された場合のみ以下の処理を行う

マウスのドラッグ位置を記憶（onMouseMoveで使用）

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(4)

パネルのonMouseDownイベント＜2＞

```
FPanelY := Y;    //パネルのｙ座標を，panelyに保存する
```

マウスのドラッグ位置を記憶（ onMouseMoveで使用）

```
FOldPanelTop     := TPanel(Sender).Top;
FOldPanelLeft    := TPanel(Sender).Left;
FOldPanelWidth   := TPanel(Sender).Width;
FOldPanelRight   := FOldPanelLeft + FOldPanelWidth;
FOldPanelCaption := TPanel(Sender).Caption;
```

パネルの上位置、左位置、右位置、幅を記憶

```
//押下された位置がパネルに左端から7ピクセル未満の場合、スケジュール変更とする
if (X < 7) then
begin
  Screen.Cursor := crSizeWE;
  FMouseLeftDownFLG := True;
end
//押下された位置がパネルの右端から7ピクセル未満の場合、スケジュール変更とする
else if (X > (FOldPanelWidth - 7)) then
begin
  Screen.Cursor := crSizeWE;
  FMouseRightDownFLG := True;
end
```

マウスをドラッグした位置がパネルの端から
7ピクセル未満の場合　→「伸縮」

左側伸縮 ： FMouseLeftDownFLG
右側伸縮 ： FMouseRightDownFLG

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(5)

パネルのonMouseDownイベント＜3＞

```
    //その他の場合、スケジュール変更とする
    else
    begin
      Screen.Cursor := crDefault;
      FMouseDownFLG := True;
    end;
  end;
end;
```

マウスをドラッグした位置がパネルの端から
7ピクセル以上の場合　→「移動」

移動：FMouseDownFLG

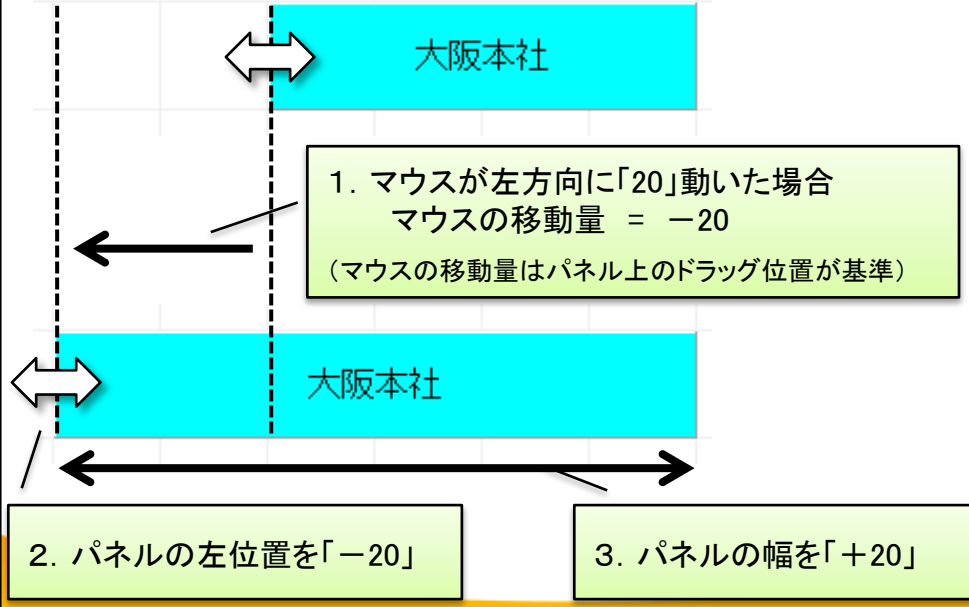
■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる（6）

“onMouseMove” イベントで実装すること

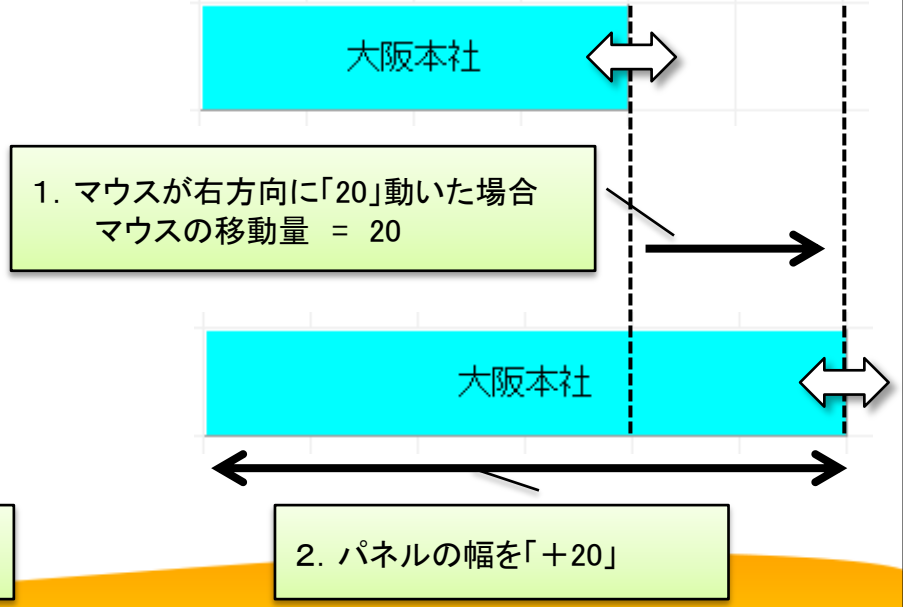
1）パネルの左側を伸縮

マウスの移動量分パネルの左位置をスライドし
マウスの移動量分パネルの幅を調整する



2）パネルの右側を伸縮

マウスの移動量分パネルの幅を調整する



■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(7)

パネルのonMouseMoveイベント＜1＞

```
  private
    procedure PanelMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
-------------------------------------------------------------------------------
procedure TfrmSample01.PanelMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
    ｛ 「③パネルをドラッグ＆ドロップで移動させる」の処理

  //パネルの左側が押下された場合、以下のプログラムを実行する
  if FMouseLeftDownFLG then
  begin
    //ここでパネルの位置を移動させる
    TPanel(Sender).Left := TPanel(Sender).Left + X - FPanelX;

    //ここでパネルの幅を調整する
    TPanel(Sender).Width := FOldPanelWidth + FOldPanelLeft - TPanel(Sender).Left;

    FMouseLeftMoveFLG := True;
  end;
```

＜パネルの左側を伸縮＞
マウスの移動量分パネルの左位置をスライドし
マウスの移動量分パネルの幅を調整する

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(8)

パネルのonMouseMoveイベント＜2＞

```
  //パネルの右側が押下された場合、以下のプログラムを実行する
  if FMouseRightDownFLG then
  begin
    //ここでパネルの幅を調整する
    TPanel(Sender).Width := FOldPanelWidth + X - FPanelX;
    FMouseRightMoveFLG := True;
  end;

  //パネルの左側または右側にマウスカーソルが有る場合、マウスイメージを変更する
  if (X < 7) or (X > (TPanel(Sender).Width - 7)) then
  begin
    Screen.Cursor := crSizeWE;
    Exit;
  end;

  Screen.Cursor := crDefault;
end;
```

＜パネルの右側を伸縮＞
マウスの移動量分パネルの幅を調整する

■ スケジュールボード

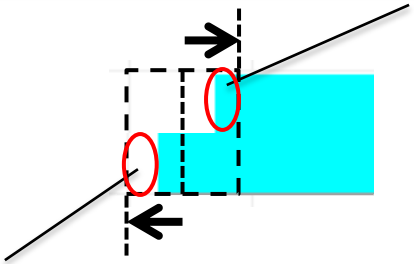
- ②パネルをドラッグ＆ドロップで伸縮させる（9）

“onMouseUp” イベントで実装すること

1）マウスのドロップ位置による調整

列幅に対するドロップ位置によって1列分プラスするかどうかを判断（左右同様）

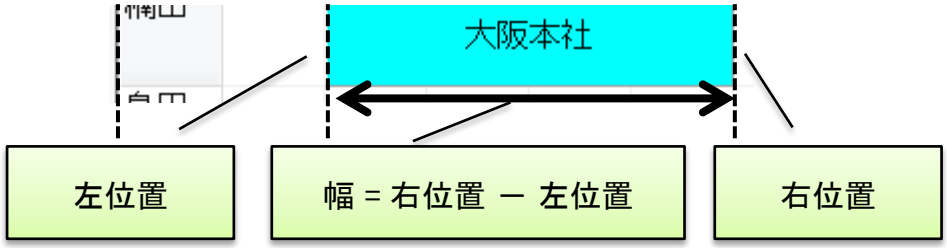
マウスをドロップした位置が列幅の半分未満の場合は1列分プラスしないため余計に伸びた分をマイナス



マウスをドロップした位置が列幅の半分以上左の場合は1列分プラス迄の残りをプラス

2）パネルの幅を調整（パネルの左側を伸縮）

1で左位置を決定した後に幅を決定する



3）パネルの幅を調整（パネルの右側を伸縮）

1で幅が決定する

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(10)

パネルのonMouseUpイベント＜1＞

```
  private
    procedure PanelMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
      X, Y: Integer);
-------------------------------------------------------------------------------
procedure TfrmSample01.PanelMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
  X, Y: Integer);
var
  bLeftRightMode: Boolean;              //スケジュール左右調整の場合True
  Coord: TGridCoord;
begin
  //初期化
  bLeftRightMode := False;
  Screen.Cursor := crDefault;

  try
    //パネルの左側が押下された場合、以下のプログラムを実行する
    if FMouseLeftDownFLG then
    begin
      FMouseLeftDownFLG := False;
      Screen.Cursor := crDefault;
```

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(11)

パネルのonMouseUpイベント＜2＞

＜パネルの左側を伸縮＞
マウスのドロップ位置による調整でパネルの左位置を決定

```
//左側が移動された場合、処理を行う
if FMouseLeftMoveFLG then
begin
  FMouseLeftMoveFLG := False;

  if  TPanel(Sender).Left mod cDayWidth > (cDayWidth div 2) then
  begin
    TPanel(Sender).Left:= TPanel(Sender).Left
                        + (cDayWidth - (TPanel(Sender).Left mod cDayWidth));
  end
  else
  begin
    TPanel(Sender).Left:= TPanel(Sender).Left - (TPanel(Sender).Left mod cDayWidth);
  end;

  //1列分以下の場合、1列分の幅とする
  if FOldPanelRight - TPanel(Sender).Left < cDayWidth then
  begin
    TPanel(Sender).Left := FOldPanelRight - cDayWidth;
  end;
```

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(12)

パネルのonMouseUpイベント＜3＞

```
      TPanel(Sender).Width := FOldPanelRight - TPanel(Sender).Left;

      bLeftRightMode := True;
    end;
  end;
  //パネルの右側が押下された場合、以下のプログラムを実行する
  if FMouseRightDownFLG then
  begin
    FMouseRightDownFLG := False;
    Screen.Cursor := crDefault;
    //右側が移動された場合、処理を行う
    if FMouseRightMoveFLG then
    begin
      FMouseRightMoveFLG := False;

      if  TPanel(Sender).Width mod cDayWidth > (cDayWidth div 2) then
      begin
        TPanel(Sender).Width:= TPanel(Sender).Width
                             + (cDayWidth - (TPanel(Sender).Width mod cDayWidth));
      end
```

＜パネルの左側を伸縮＞
マウスのドロップ位置による調整でパネルの左位置を決定した後、パネルの幅を決定

＜パネルの右側を伸縮＞
マウスのドロップ位置による調整でパネルの幅を決定

■ スケジュールボード

- ②パネルをドラッグ＆ドロップで伸縮させる(13)

パネルのonMouseUpイベント＜4＞

```
      else
      begin
        TPanel(Sender).Width := TPanel(Sender).Width - (TPanel(Sender).Width mod cDayWidth);
      end;

      //1列分以下の場合、1列分の幅とする
      if TPanel(Sender).Width < cDayWidth then
      begin
        TPanel(Sender).Width := cDayWidth;
      end;

      bLeftRightMode := True;
    end;
  end;
```

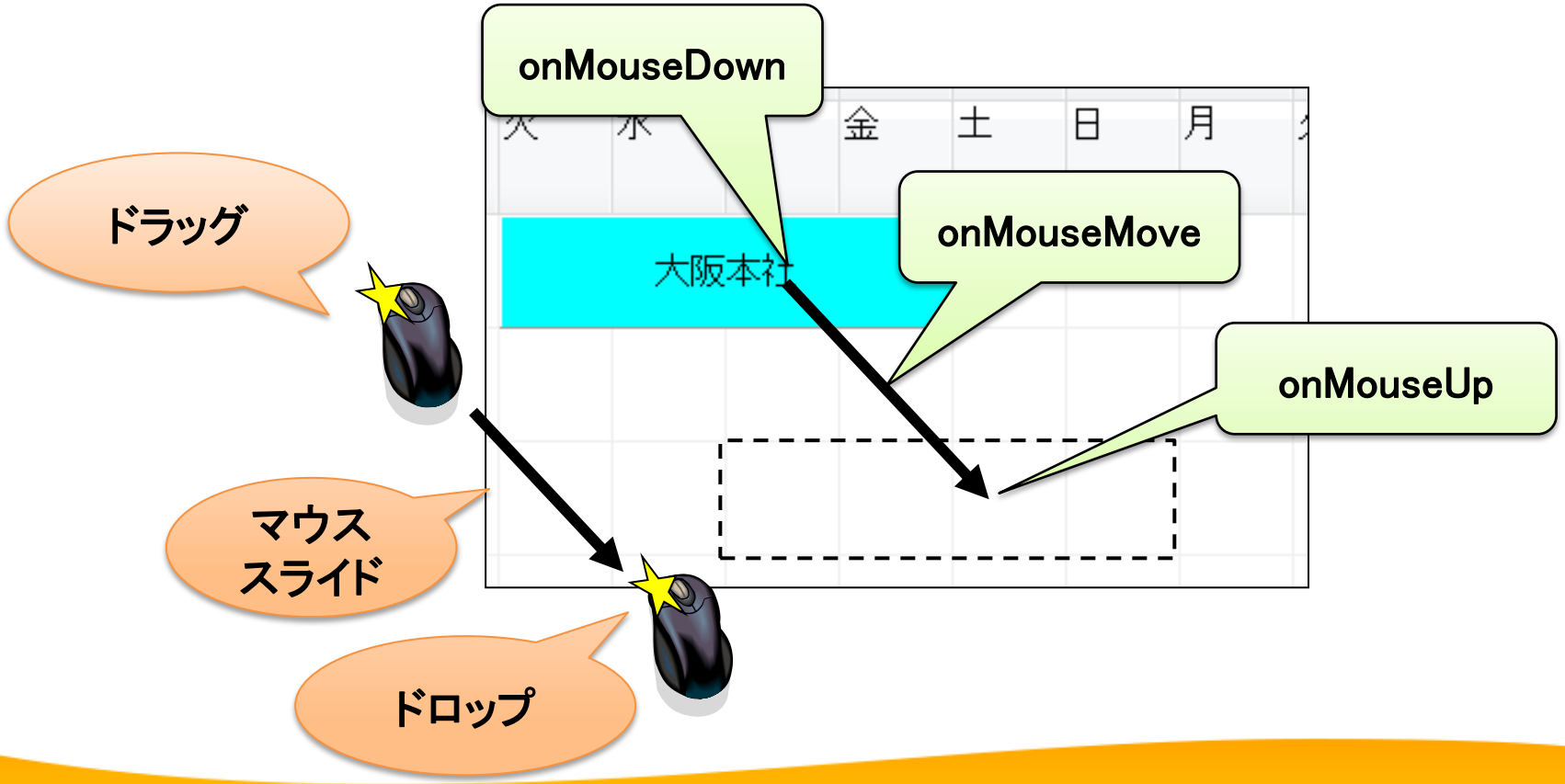
＜パネルの右側を伸縮＞
マウスのドロップ位置による調整でパネルの幅を決定

省略

■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる（1）

パネルのイベントを使用してドラッグ＆ドロップで移動を実装！



■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる（2）

“onMouseDown” イベントで実装すること

実装内容は「②パネルをドラッグ＆ドロップで伸縮させる」と同じ

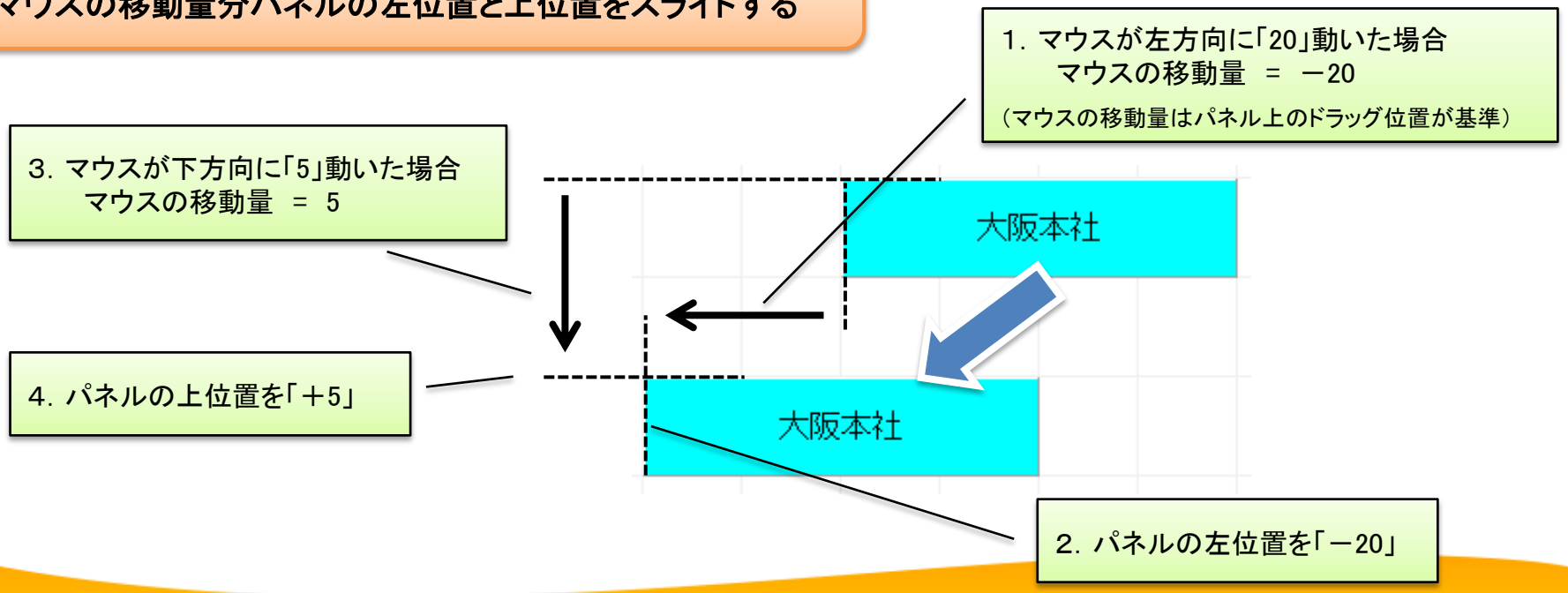
■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる（3）

“onMouseMove” イベントで実装すること

1）パネルの上位置と左位置を調整

マウスの移動量分パネルの左位置と上位置をスライドする



■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる(4)

パネルのonMouseMoveイベント＜1＞

```
procedure TfrmSample01.PanelMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  //マウスが押されていれば以下のプログラムを実行する
  if FMouseDownFLG then
  begin
    //ここでパネルの位置を移動させる
    TPanel(Sender).Left := TPanel(Sender).Left + X - FPanelX;
    TPanel(Sender).Top  := TPanel(Sender).Top  + Y - FPanelY;

    FMouseMoveFLG := True;
    Screen.Cursor := crDefault;
    Exit;
  end;

  ｛ 「②パネルをドラッグ＆ドロップで伸縮させる」の処理

end;
```

マウスの移動量分パネルの左位置と上位置をスライドする

■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる（5）

“onMouseUp” イベントで実装すること

1）マウスのドロップ位置による調整

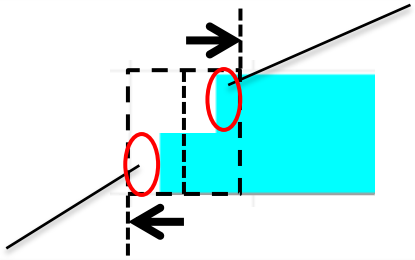
2）パネルの位置を調整

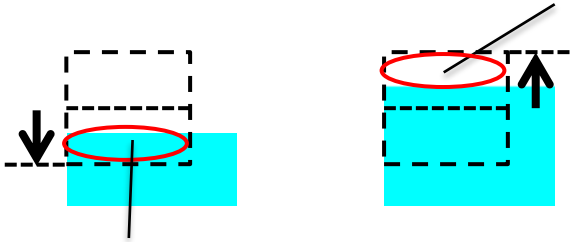
列幅、行の高さに対するドロップ位置によって1列分、
1行分プラスするかどうかを判断（上位置、左位置同様）

1）で左位置と上位置を決定する

マウスをドロップした位置が列幅の
半分未満の場合は1列分プラスしない
ため余計に伸びた分をマイナス

マウスをドロップした位置が行の高さの
半分以上上の場合は1行分プラス迄
の残りをプラス





マウスをドロップした位置が列幅の
半分以上左の場合は1列分プラス迄
の残りをプラス

マウスをドロップした位置が行の高さの
半分未満の場合は1行分プラスしない
ため余計に移動した分をマイナス

■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる(6)

パネルのonMouseUpイベント＜1＞

```
procedure TfrmSample01.PanelMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
  X, Y: Integer);
var
  bLeftRightMode: Boolean;                //スケジュール左右調整の場合True
  Coord: TGridCoord;
begin

  ｝「②パネルをドラッグ＆ドロップで伸縮させる」の処理


    //マウスが押されていれば以下のプログラムを実行する
    if FMouseDownFLG then
    begin
      FMouseDownFLG := False;

      //場所が移動された場合、処理を行う
      if FMouseMoveFLG then
      begin
        FMouseMoveFLG := False;
```

■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる(7)

パネルのonMouseUpイベント＜2＞

マウスのドロップ位置による調整でパネルの上位置を決定

```
//移動場所のパネル座標を確定する
if  TPanel(Sender).Top mod (cItemHeight) > (cItemHeight div 2) then
begin
  TPanel(Sender).Top := TPanel(Sender).Top
                        + (cItemHeight - (TPanel(Sender).Top mod (cItemHeight)));
end
else
begin
  TPanel(Sender).Top := TPanel(Sender).Top - (TPanel(Sender).Top mod (cItemHeight));
end;
```

マウスのドロップ位置による調整でパネルの左位置を決定

```
if  TPanel(Sender).Left mod cDayWidth > (cDayWidth div 2) then
begin
  TPanel(Sender).Left:= TPanel(Sender).Left
                        + (cDayWidth - (TPanel(Sender).Left mod cDayWidth));
end
else
begin
  TPanel(Sender).Left:= TPanel(Sender).Left - (TPanel(Sender).Left mod cDayWidth);
end;
```

■ スケジュールボード

- ③パネルをドラッグ＆ドロップで移動させる(8)

パネルのonMouseUpイベント＜3＞

```
    }  省略

  finally
    //パネル情報表示
    mePnlName.Text := TPanel(Sender).Caption; // パネル名

    //パネル座標のグリッド行列位置を取得
    Coord := stgGrid.MouseCoord(TPanel(Sender).Left, TPanel(Sender).Top);

    meNo.Text     := stgGrid.Cells[0, Coord.Y];
    meDayF.Text   := stgGrid.Cells[Coord.X, 0];
    meWeekF.Text  := stgGrid.Cells[Coord.X, 1];

    //パネル座標のグリッド行列位置を取得
    Coord := stgGrid.MouseCoord(TPanel(Sender).Left + TPanel(Sender).Width - 1, TPanel(Sender).Top);

    meDayT.Text   := stgGrid.Cells[Coord.X, 0];
    meWeekT.Text  := stgGrid.Cells[Coord.X, 1];
  end;
end;
```

＜パネルの情報を表示(伸縮・移動共通)＞
グリッドに対するパネルの位置からグリッドの行列を取得し、グリッドより行列に該当する担当者、日付(開始)、曜日(開始)を表示

＜パネルの情報を表示(伸縮・移動共通)＞
グリッドに対するパネルの位置からグリッドの行列を取得し、グリッドより行列に該当する日付(終了)、曜日(終了)を表示

■ スケジュールボード

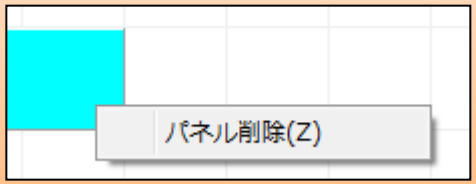
- ④パネルを削除する

「パネル削除」メニューのonClickイベント＜1＞

```
procedure TfrmSample01.miPanelDelClick(Sender: TObject);
begin
  //選択パネルを破棄する
  FreeAndNil(FArrayPanel[FSelectPanelID - 1]);
end;
```

onMouseDownで記憶しておいた
FSelectPanelIDを使用して削除

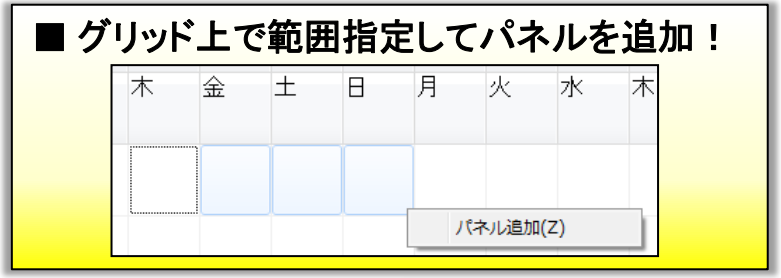
パネルの右クリックメニューにある
「パネル削除」メニューのonClick
イベントに実装

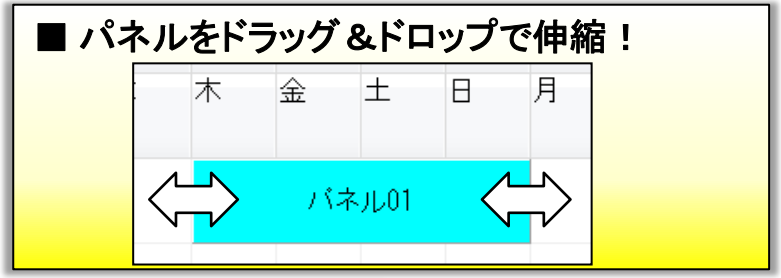


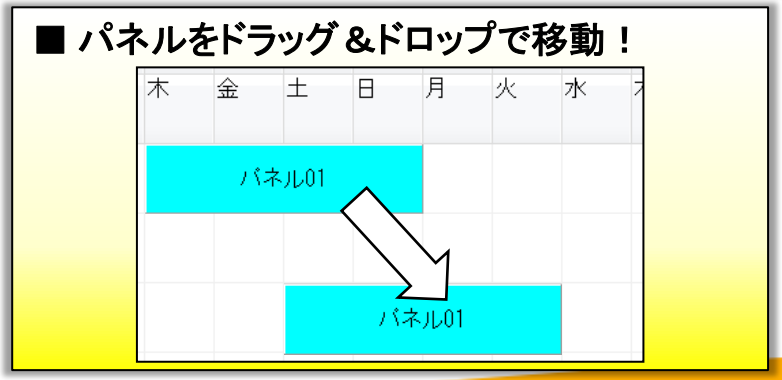
■ スケジュールボードのメリット

- 誰でも簡単に操作できる
  - 全体の予定を把握しやすい
  - 表示内容を切り替えずに、全員の予定を一度に入力・編集が可能
  - 入力、編集をマウス操作でできるので簡単

    →処理時間の短縮





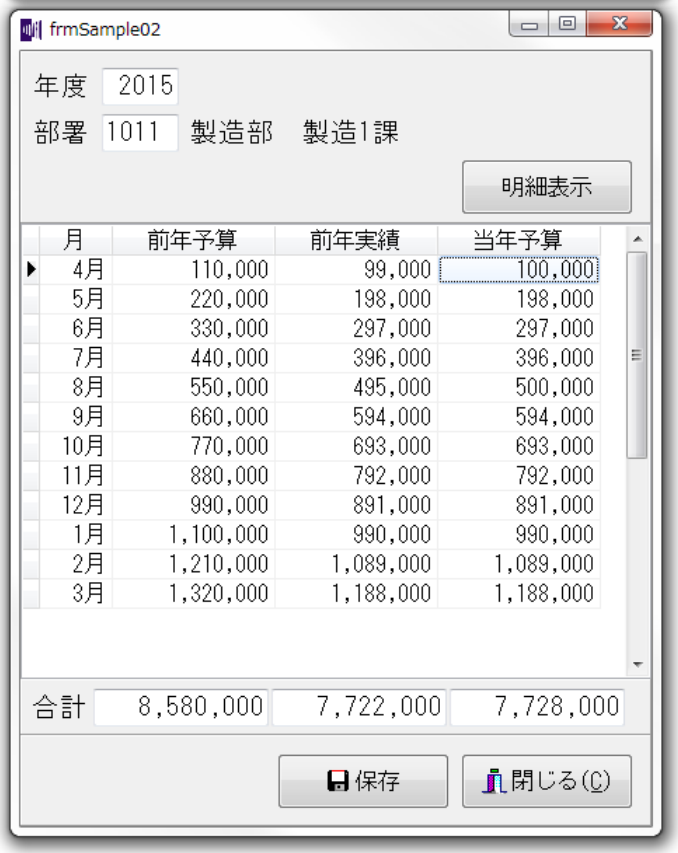


3. カスタムコンポーネントの作成

■ グリッドを分かりやすく表現する

- グリッドの見た目を変えて分かりやすくする

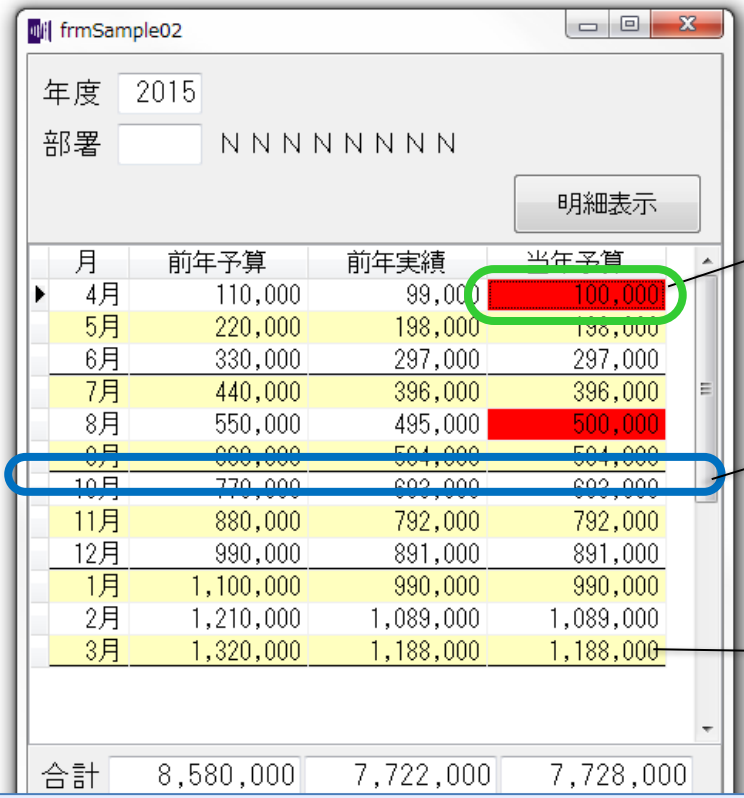
カスタマイズ前



カスタマイズ後

■ グリッドを分かりやすく表現する

- TDBGridをカスタマイズして実装



前年実績より予算が多い場合、
セルの背景色を変更する

四半期毎に区切り線を表示する

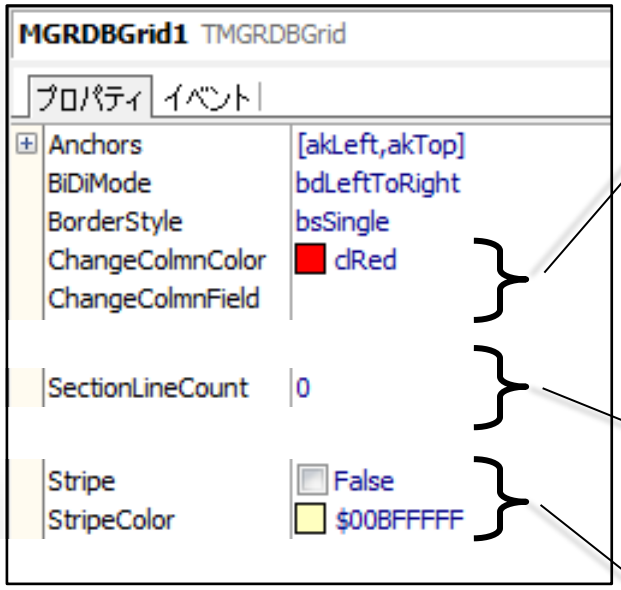
1行毎に背景色を変更して明細を
分かりやすく表示する

機能ごとに個別に制御を実装できるが、コンポーネント化することで、
個別に実装・テストを行う手間を省け、開発効率や品質向上が期待できる

■ カスタムコンポーネント

- TMGRDBGridにそれぞれのプロパティを実装

追加プロパティ



■特定のセルのみ背景色を変更する

ChangeColmnFieldプロパティには、連結するDataSetの項目を指定する

指定した項目には、背景色を変更する列を数値として保管する

変更する背景色はChangeColmnColorプロパティに指定する

区分をChangeColmnFieldプロパティに指定
区分に保管されている値と同じ列の背景色を変更
-1を指定した場合は全ての列を対象にする

■特定の間隔毎に区切り線を表示する

SectionLineCountプロパティに指定した行数毎に区切り線を表示する

■ 1行毎に背景色を変更して分かりやすく明細を表示する

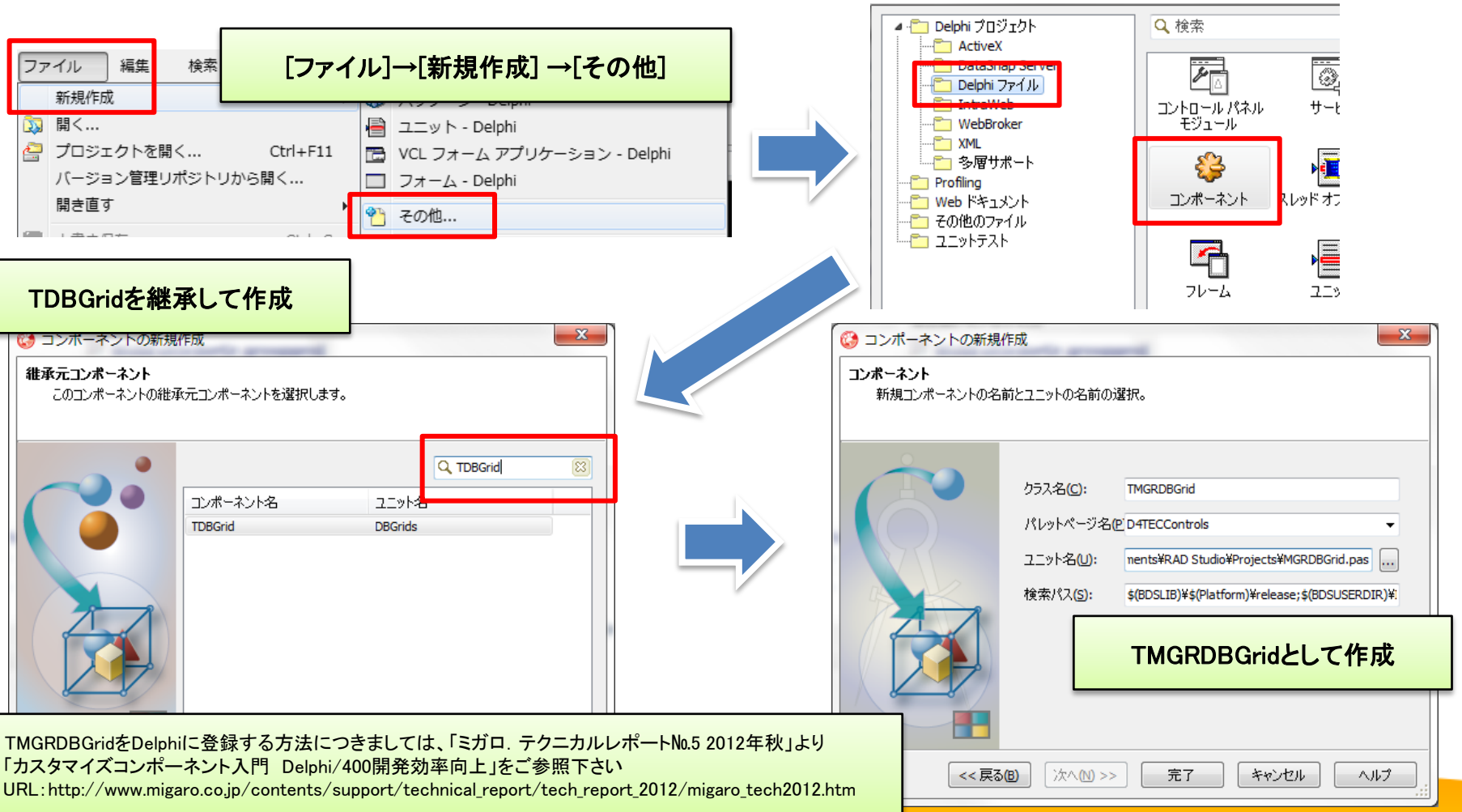
StripeプロパティをTrueにすると1行毎に背景色を変更する

変更する背景色はStripeColorプロパティに指定する

※TClientDatasetの使用が前提

■ グリッドを分かりやすく表現する

- カスムコンポーネント（TMGRDBGrid）を作成する

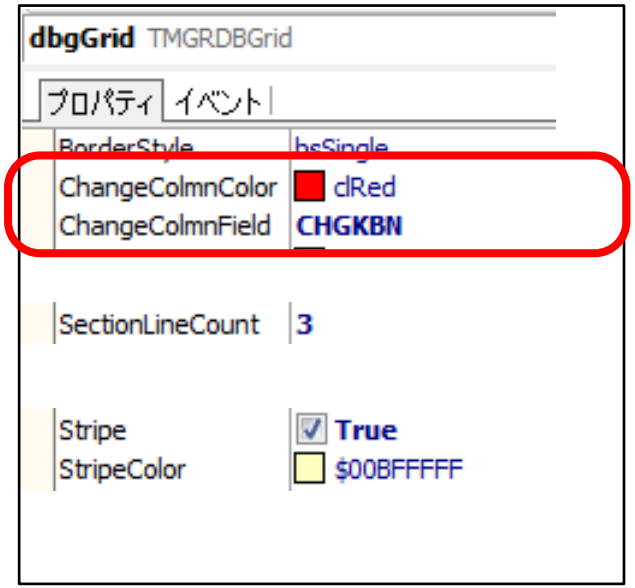


※ TMGRDBGridをDelphiに登録する方法につきましては、「ミガロ. テクニカルレポートNo.5 2012年秋」より
「カスタマイズコンポーネント入門　Delphi/400開発効率向上」をご参照下さい
URL：http://www.migaro.co.jp/contents/support/technical_report/tech_report_2012/migaro_tech2012.htm

■ カスタムコンポーネント

- 特定セルの背景色変更機能を実装(1)

追加プロパティ



実装イメージ

月	前年予算	前年実績	当年予算
4月	110,000	99,000	100,000
5月	220,000	198,000	198,000
6月	330,000	297,000	297,000
7月	440,000	396,000	396,000
8月	550,000	495,000	500,000
9月	660,000	594,000	594,000
10月	770,000	693,000	693,000
11月	880,000	792,000	792,000
12月	990,000	891,000	891,000
1月	1,100,000	990,000	990,000
2月	1,210,000	1,089,000	1,089,000
3月	1,320,000	1,188,000	1,188,000

■ カスタムコンポーネント

- 特定セルの背景色変更機能を実装(2)

追加するプロパティの設定

```
  private
    FChangeColmnColor: TColor;
    FChangeColmnField: string;
    procedure SetChangeColmnColor(const Value: TColor);
    procedure SetChangeColmnField(const Value: string);
  published
    property ChangeColmnColor: TColor read FChangeColmnColor write SetChangeColmnColor
                                                                        default clRed;
    property ChangeColmnField: string read FChangeColmnField write SetChangeColmnField;
-------------------------------------------------------------------------------------
procedure TMGRDBGrid.SetChangeColmnColor(const Value: TColor);
begin
  FChangeColmnColor := Value;
  Invalidate;
end;
-------------------------------------------------------------------------------------
procedure TMGRDBGrid.SetChangeColmnField(const Value: string);
begin
  FChangeColmnField := Value;
  Invalidate;
end;
```

■ カスタムコンポーネント

- 特定セルの背景色変更機能を実装(3)

描画処理<1>

```
  protected
    procedure DrawColumnCell(const Rect: TRect; DataCol: Integer; Column: TColumn;
      State: TGridDrawState); override;
-------------------------------------------------------------------------------------
procedure TMGRDBGrid.DrawColumnCell(const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var
  bDraw: Boolean; //描画実行FLG
begin
  bDraw := False; //描画実行FLG

  ～ 「1行毎に背景色を変更する機能」の処理

  //特定セルの背景色描画
  if (FChangeColmnField <> '') then
  begin
    if StrToIntDef(DataSource.DataSet.FieldByName(FChangeColmnField).AsString, 0) = -1 then
    begin
      //背景色の変更
      Canvas.Brush.Color := FChangeColmnColor;
      bDraw := True;  //描画実行FLG
```

ChangeColmnFieldプロパティに設定されている項目の値が「－1」の場合、
行単位で背景色をChangeColmnColorプロパティに設定されている色にする

■ カスタムコンポーネント

- 特定セルの背景色変更機能を実装(4)

描画処理＜2＞

```
    end
    else
    begin
      if StrToIntDef(DataSource.DataSet.FieldByName(FChangeColmnField).AsString, 0) - 1
                                                                    = Column.Index then
      begin
        //背景色の変更
        Canvas.Brush.Color := FChangeColmnColor;
        bDraw := True;  //描画実行FLG
      end;
    end;
  end;
  //描画処理実行
  if bDraw then
  begin
    Canvas.FillRect(Rect);
    DefaultDrawColumnCell(Rect, DataCol, Column, State);
  end;
```

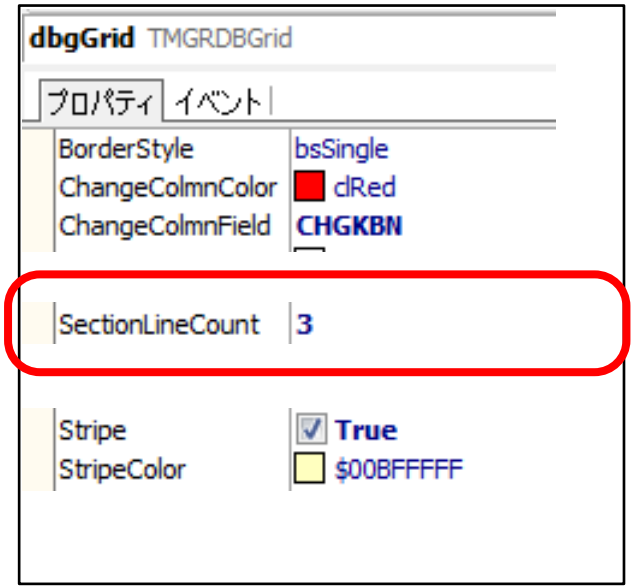
ChangeColmnFieldプロパティに設定されている項目の値に該当する、
列の背景色をChangeColmnColorプロパティに設定されている色に変更

「区切り線を表示する機能」の処理

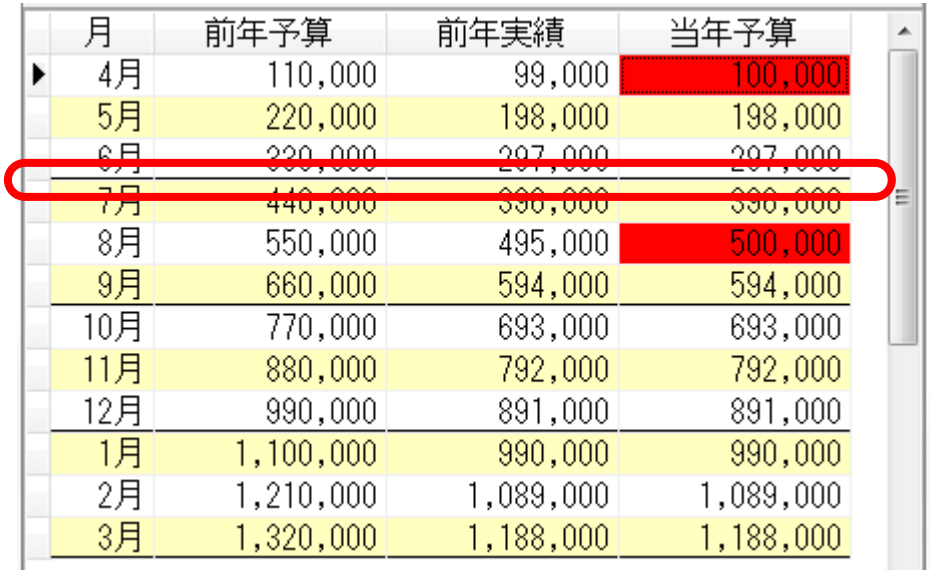
■ カスタムコンポーネント

- 区切り線を表示する機能を実装(1)

追加プロパティ



実装イメージ



月	前年予算	前年実績	当年予算
4月	110,000	99,000	100,000
5月	220,000	198,000	198,000
6月	330,000	297,000	297,000
7月	440,000	396,000	396,000
8月	550,000	495,000	500,000
9月	660,000	594,000	594,000
10月	770,000	693,000	693,000
11月	880,000	792,000	792,000
12月	990,000	891,000	891,000
1月	1,100,000	990,000	990,000
2月	1,210,000	1,089,000	1,089,000
3月	1,320,000	1,188,000	1,188,000

■ カスタムコンポーネント

- 区切り線を表示する機能を実装(2)

追加するプロパティの設定

```
  private
    FSectionLineCount: Integer;

    procedure SetSectionLineCount(const Value: Integer);
  published
    property SectionLineCount: Integer read FSectionLineCount write SetSectionLineCount default 0;
-------------------------------------------------------------------------------------------------
procedure TMGRDBGrid.SetSectionLineCount(const Value: Integer);
begin
  FSectionLineCount := Value;
  Invalidate;
end;
```

■ カスタムコンポーネント

- 区切り線を表示する機能を実装(3)

描画処理

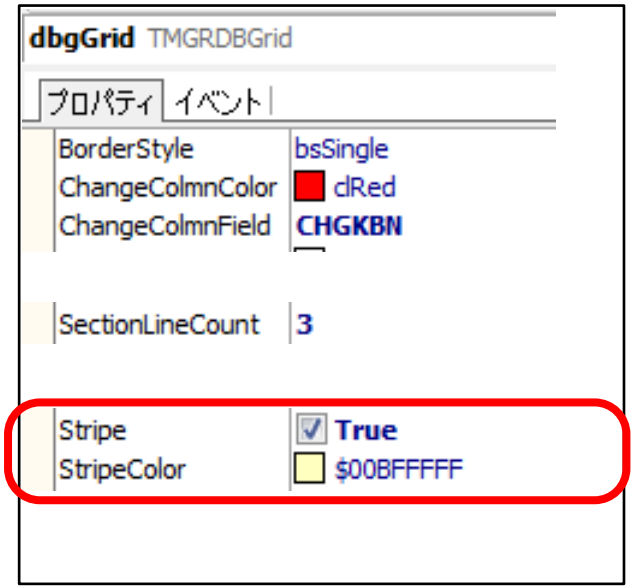
```
procedure TMGRDBGrid.DrawColumnCell(const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var
  bDraw: Boolean; //描画実行FLG
begin
  ｛ 「特定セルの背景色変更機能」「1行毎に背景色を変更する機能」の処理

  //区切り線描画処理
  if (FSectionLineCount > 0) then
  begin
    if (DataSource.DataSet as TClientDataSet).RecNo mod FSectionLineCount = 0 then
    begin
      Canvas.Pen.Color := clBlack;
      Canvas.Pen.Width := 1;
      Canvas.MoveTo(Rect.Left , Rect.Bottom - 1);
      Canvas.LineTo(Rect.Right, Rect.Bottom - 1);
    end;
  end;
  inherited;
end;
```

SectionLineCountプロパティに設定されているレコード数毎に区切り線を表示
レコード数の判断はClientDatasetの「RecNo」を使用する

■ カスタムコンポーネント

- 1行毎に背景色を変更する機能を実装(1)

追加プロパティ



実装イメージ

月	前年予算	前年実績	当年予算
4月	110,000	99,000	100,000
5月	220,000	198,000	198,000
6月	330,000	297,000	297,000
7月	440,000	396,000	396,000
8月	550,000	495,000	500,000
9月	660,000	594,000	594,000
10月	770,000	693,000	693,000
11月	880,000	792,000	792,000
12月	990,000	891,000	891,000
1月	1,100,000	990,000	990,000
2月	1,210,000	1,089,000	1,089,000
3月	1,320,000	1,188,000	1,188,000

■ カスタムコンポーネント

- 1行毎に背景色を変更する機能を実装(2)

追加するプロパティの設定

```
  private
    FStripe: Boolean;
    FStripeColor: TColor;
    procedure SetStripe(const Value: Boolean);
    procedure SetStripeColor(const Value: TColor);
  published
    property Stripe: Boolean     read FStripe      write SetStripe default False;
    property StripeColor: TColor read FStripeColor write SetStripeColor default $00BFFFFF;
-------------------------------------------------------------------------------------------
procedure TMGRDBGrid.SetStripe(const Value: Boolean);
begin
  FStripe := Value;
  Invalidate;
end;
-------------------------------------------------------------------------------------------
procedure TMGRDBGrid.SetStripeColor(const Value: TColor);
begin
  FStripeColor := Value;
  Invalidate;
end;
```

■ カスタムコンポーネント

- 1行毎に背景色を変更する機能を実装(3)

描画処理<1>

```
procedure TMGRDBGrid.DrawColumnCell(const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var
  bDraw: Boolean; //描画実行FLG
begin
  bDraw := False; //描画実行FLG

  //ストライプ描画処理
  if FStripe and (DataSource.DataSet is TClientDataSet) then
  begin
    if (DataSource.DataSet as TClientDataSet).RecNo mod 2 = 0 then
    begin
      if not (gdSelected in State) then
      begin
        Canvas.Brush.Color := FStripeColor;
        bDraw := True;  //描画実行FLG
      end;
    end;
  end;
```

Stripeプロパティが「True」の場合、レコード数が偶数の行の背景色を
StripeColorプロパティに設定されている色に変更
レコード数の判断はClientDatasetの「RecNo」を使用する

■ カスタムコンポーネント

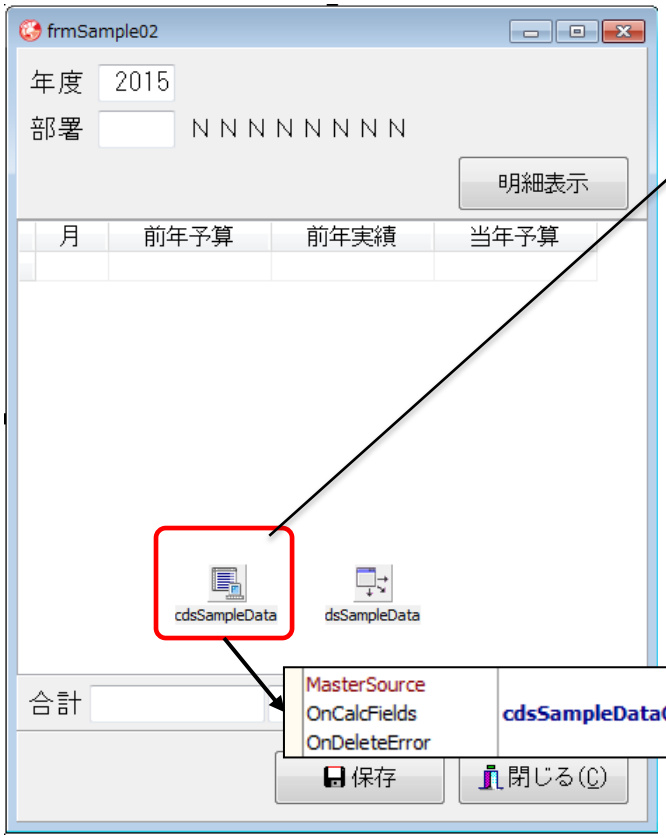
- 1行毎に背景色を変更する機能を実装(4)

描画処理＜2＞

```
  ∫ 「特定セルの背景色変更機能」の処理

  //描画処理実行
  if bDraw then
  begin
    Canvas.FillRect(Rect);
    DefaultDrawColumnCell(Rect, DataCol, Column, State);
  end;

  ∫ 「区切り線を表示する機能」の処理

  inherited;
end;
```

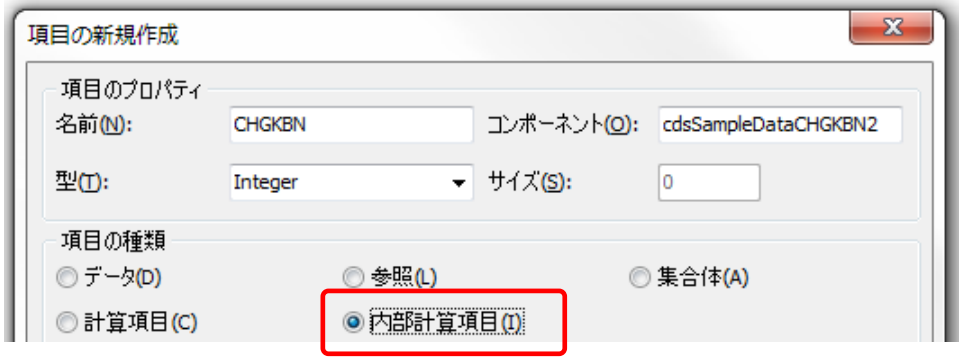
■ カスタムコンポーネント

- カスタムコンポーネントの使用方法（1）



TClientDatasetをフォームに貼り付け、色を変更する列数を保管する項目として内部計算項目「CHGKBN」を追加する

※ 今回は内部計算項目を設定しDelphiで更新を行いますが、ファイルに項目を設定し、RPG等で更新する方法もあります。



TClientDatasetのonCalcFieldsイベントで、背景色を変更する条件に従って内部計算項目「CHGKBN」に値をセットする（処理内容は次頁参照）

■ カスタムコンポーネント

- カスタムコンポーネントの使用方法(2)

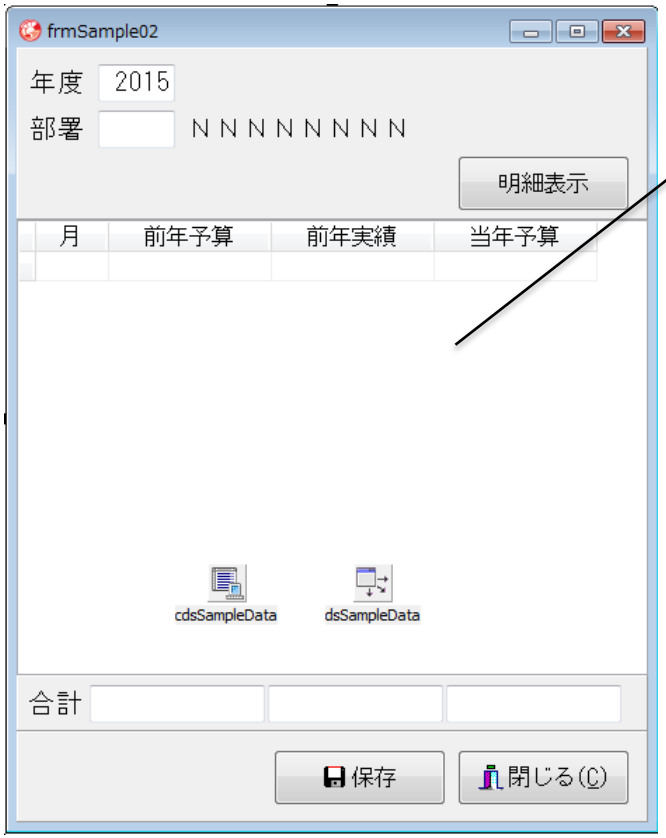
TClientDatasetのonCalcFieldsイベント

```
procedure TfrmSample02.cdsSampleDataCalcFields(DataSet: TDataSet);
begin
  with cdsSampleData do
  begin
    // 前年実績＜当年予算 の場合、警告！
    if (FieldByName('WKZNJS').AsCurrency < FieldByName('WKTNYS').AsCurrency) then
      FieldByName('CHGKBN').AsInteger := 4
    else
      FieldByName('CHGKBN').AsInteger := 0;
  end;
end;
```

前年実績＜当年予算 の場合に当年予算列(4列目)の背景色を変更する
TMGRGridの実装機能を使用するのに記述するロジックはこれだけ！

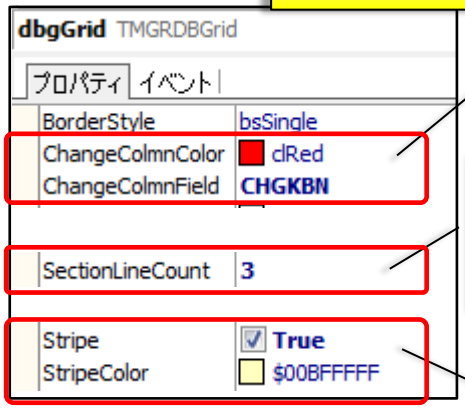
■ カスタムコンポーネント

- カスタムコンポーネントの使用方法(3)



TMGRDBGridをフォームに貼り付けプロパティを設定

■ 特定のセルのみ背景色を変更する
・ChangeColmnColorプロパティに色を設定する
・ChangeColmnFieldプロパティに、色を変更する列数を保管する
DataSsetに連結するDataSsetの項目を指定する



■ 区切り線を表示する
・SectionLineCountプロパティを設定するだけ！

■ 1行毎に背景色を変更して分かりやすく明細を表示する
・StripeプロパティをTrueにしてStripeColorプロパティに色を設定するだけ！

■ グリッドを分かりやすく表現する

- カスタムコンポーネントを利用することのメリット
  - 機能ごとに個別に実装する必要がなくなる
  - 機能ごとに個別にテストをする必要がなくなる

開発効率の向上、品質の向上に繋がる

■ まとめ

- 分かりやすいユーザーインターフェースの実現
  - ◆スケジュールボード
    - 「直観的」に全体のスケジュールを捉えることができる
      → グリッド＋パネルによるスケジュールバーイメージを実現
    - 誰でも簡単に操作できる操作性
      →ドラッグ &ドロップによるスケジュールの変更・移動を実現
  - ◆カスタムコンポーネント
    - 「直観的」にチェック箇所を捉えることができる
      → 特定セルの背景色を変更する機能を実現
    - 分かりやすい明細の表示
      → 区切り線の表示、1行毎に明細の背景色を変更する機能を実現

ご静聴ありがとうございました