Delphi/400

直感的に使える操作性を工夫した Webシステム開発（IntraWeb）の機能紹介

株式会社ミガロ.
システム事業部システム１課
田村　洋一郎

略　歴

生年月日：1983年9月27日
最終学歴：2006年　近畿大学　理工学部卒業
入社年月：2006年4月　株式会社ミガロ. 入社
社内経歴：2006年4月　システム事業部配属

現在の仕事内容：

RPGやDelphi/400などの開発経験を経て、現在は要件定義から安定稼働フォローまで、システム開発全般に携わっている。

株式会社ミガロ.
システム事業部システム１課
宮坂　優大

略　歴

生年月日：1982年11月19日
最終学歴：2006年　近畿大学　理工学部卒業
入社年月：2006年4月　株式会社ミガロ. 入社
社内経歴：2006年4月　システム事業部配属

現在の仕事内容：

主にDelphi/400を利用したシステムの受託開発・保守をメインに担当。設計から運用・フォローまでプロジェクト全般に関わることが多い。

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

Webシステムの需要は増加し、当社でも多くの受託開発を行っている。オンラインショッピングサイトなどの外部ユーザー向けWebシステム（ECサイト）では、さまざまなユーザーが利用するため、直感的に使える操作性が求められる。そこで本稿では、システム開発チームが実際の開発時に工夫し、実装したユーザーインターフェース機能を紹介する。

また今回紹介するWebシステムの開発手法は、Delphi/400を活用したIntraWebによるものである。IntraWebを使用すると、従来の GUI アプリケーションと同様にWeb システムを構築することができるため、ぜひ活用して頂きたい。

本稿では、以下のUI機能を紹介する。
【第2章】直感的に操作しやすいカレンダー機能
【第3章】視覚的に分かりやすいローディング画面
【第4章】フレームを使用した明細グリッド
今回の機能を実装した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。また、サンプルは、スタンドアロン　モードを使用して作成している。
なお本稿では、Delphi/400 11 AlexandriaとIntraWeb 15を使用し、動作環境は、MicrosoftのEdgeを採用する。また、Delphi/400におけるIntraWebの基本的な開発手順について本稿では割愛するが、ミガロ. のサイト内に詳しく紹介している資料があるのでそちらを参照して頂きたい。
<参照先>
https://www.migaro.co.jp/ts/27th/Session2.pdf

2. 直感的に操作しやすいカレンダー機能

本章では、カレンダー機能を紹介する。IntraWebには標準でTIWCalendarというカレンダーコンポーネントが存在する**【図1】**。複雑なソースコードを記述することなく、カレンダー機能を実装できる便利なコンポーネントである。

図 1　TIWCalendarのイメージ

Previous　　8月 2024　　Next

月	火	水	木	金	土	日
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

本章では、曜日毎に色やフォントを工夫し、直感的に操作しやすいカレンダー機能を実装する手法を紹介する。今回作成するカレンダーの完成イメージは**【図2】**の通りである。

図 2　今回紹介するカレンダーのイメージ

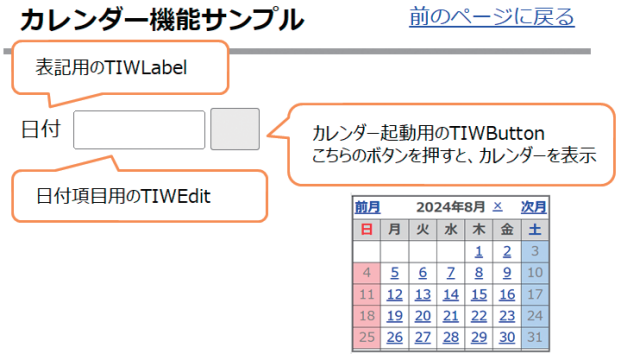
前月　　2024年8月 ×　　次月

日	月	火	水	木	金	土
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

2-1.サンプル画面

カレンダー機能を実装するためのサンプル画面を【図3】の通りに作成する。

図 3　サンプル画面



本稿で紹介するサンプルプログラムでは、トップページ【図4】を作成し、各章のサンプル画面に遷移可能としている。各章のサンプル画面には、トップページに戻るための「前のページに戻る」(TIWLink)を画面右上に配置している。

図 4　トップページ

2-2.カレンダー機能の構成

カレンダー機能は、ソースコードが煩雑にならないよう、クラス化による処理の共通化を行う。カレンダーのメイン処理を集約した「TIWSMPLCalendar」クラス、画面を閉じて呼出元の画面に戻るための処理を集約した「TIWSMPLLink」クラスの2つを作成する。これらのクラスを使用することで、カレンダー全体の機能を実装している【図5】。

図 5　カレンダー機能の構成

2-3.カレンダー(TIWSMPLCalendar)クラス

カレンダー(TIWSMPLCalendar)クラスの宣言部を【ソース1】、レイアウト情報(罫線、色、フォント等)や選択不可とする入力制御を行う処理を【ソース2】の通りに記述する。

ソース 1

カレンダー(TIWSMPLCalendar)クラスの宣言部

```
TIWSMPLCalendar = class(TIWCalendar)
  private
    { Private 宣言 }
    FYear, FMonth: Integer;                                        // 年月

    procedure CalendarRenderCell(ACell: TIWGridCell;
                                const ARow, AColumn: Integer);     // カレンダーのセル内容セット処
理
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;              // 生成時処理(コンストラクタ)
  end;
```

ソース 2

カレンダーのセル内容セット処理

```
procedure TIWSMPLCalendar.CalendarRenderCell(ACell: TIWGridCell; const ARow,
  AColumn: Integer);
var
  dDate: TDateTime;     // 日付
  iDate: Integer;       // 日付(数値)
  sDate: String;        // 日付(文字)
  iDayOfWeek: Integer;  // 曜日
begin
  inherited;

  // CSSのセット
  Css := 'calendar';

  // ----- タイトル(年月)セット処理 -----
  // タイトル部の場合
  if (ARow = 0) and (AColumn = 1) then
  begin
    // タイトル部をセット:yyyy年mm月
    if (Pos('年', ACell.Text) = 0) then
    begin
      // 年月を保管
      FYear  :=
        StrToIntDef(Copy(ACell.Text, Pos(' ', ACell.Text) + 1, Length(ACell.Text)), 0);
      FMonth :=
        StrToIntDef(Copy(ACell.Text, 1, Pos(' ', ACell.Text) - 2), 0);

      // セット
      ACell.Text := IntToStr(FYear) + '年' + IntToStr(FMonth) + '月';
    end;
  end;
```

レイアウト用のCSSのセレクタを指定
CSS側でレイアウト情報(罫線、色、フォント等)を制御

タイトルに
年月を表示

```
  // ----- 選択不可の設定曜日 -----
  if (ARow > 1) and (ACell.Text <> '') then
begin
    // 日付
    iDate := (FYear * 10000) + (FMonth * 100) + StrToIntDef(ACell.Text, 0);
    sDate := FormatFloat('0000/00/00', iDate);
    dDate := StrToDateTime(sDate + FormatDateTime('hh:mm:ss', Now));

    // 曜日を取得
    iDayOfWeek := System.DateUtils.DayOfTheWeek(dDate);

    // CSS側で処理するため、特定のCSSをセット
    // 土曜日の場合、選択不可
    if (iDayOfWeek = 6) then
    begin
      Css := Trim(Css + ' cal_disable cal_blue');
    end
    // 日曜日の場合、選択不可
    else if (iDayOfWeek = 7) then
    begin
      Css := Trim(Css + ' cal_disable cal_red');
    end;
  end;
end;
```

土曜日、日曜日を判定し、対象のCSSのセレクタを指定CSS側でセルの色や選択不可を制御

次に、カレンダー（TIWSMPLCalendar）クラスの生成時処理にカレンダー全体の基本情報や描画イベントに【ソース2】の関数を指定する【ソース3】。

ソース 3

カレンダー（TIWSMPLCalendar）の生成時処理

```
constructor TIWSMPLCalendar.Create(AOwner: TComponent);
begin
  inherited;

  // 初期化
  Caption      := '';   // キャプション
  Text         := '';   // テキスト
  Height       := 205;  // 高さ
  Width        := 3     // 幅
  ZIndex       := 0;    // 位置

  // フォント名
  Font.FontName                := cFontName;
  CalendarFont.FontName        := cFontName;
  CalendarHeaderFont.FontName  := cFontName;
  // フォントサイズ
  Font.Size                 := cFontSize + 2;
  CalendarFont.Size         := cFontSize + 2;
  CalendarHeaderFont.Size   := cFontSize + 2;
  // フォント色
  Font.Color                 := cFontColor;
  CalendarFont.Color         := cFontColor;
  CalendarHeaderFont.Color   := cFontColor;

  // タイトルは太字
  CalendarHeaderFont.Style := [fsBold];
```

カレンダー全体の基本情報を指定

```
    // 色
    BorderColors.Color      := clNone;
    BorderColors.Light      := clNone;
    BorderColors.Dark       := clNone;
    BGColor                 := clNone;
    CalendarColor           := clNone;
    AlternateCalendarColor  := clNone;
    CalendarHeaderColor     := clNone;

    // キャプション
    CaptionPrevious := '前月';
    CaptionNext     := '次月';

    // フレーム枠を描画
    UseFrame := True;

    // 週始まりは日曜日
    WeekStarts := wsSunday;

    // Asyncモード
    AsyncMode := True;
    // AsyncイベントでのSubmit
    SubmitOnAsyncEvent := False;

    // カレンダーのセル内容セット処理
    OnRenderCell := CalendarRenderCell;
  end;
```

描画イベントに先程記述した「カレンダーのセル内容セット処理」を指定

2-4.画面を閉じるリンク(TIWSMPLLink)クラス

画面を閉じるリンク(TIWSMPLLink)クラスの宣言部を**【ソース4】**の通りに記述する。

ソース 4

画面を閉じるリンク (TIWSMPLLink) クラスの宣言部

```
TIWSMPLLink = class(TIWLink)
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;          // 生成時処理(コンストラクタ)
  end;
```

次に、画面を閉じるリンク(TIWSMPLLink)クラスの生成時処理に基本情報を指定する【ソース5】。

ソース 5

画面を閉じるリンク（TIWSMPLLink）の生成時処理

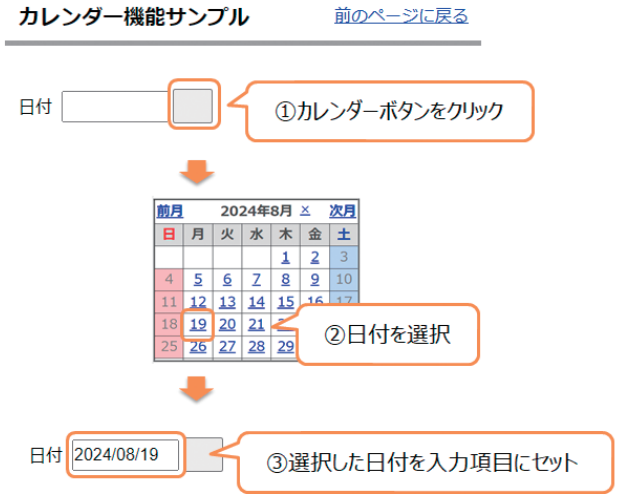
```
constructor TIWSMPLLink.Create(AOwner: TComponent);
begin
  inherited;

  // 初期化
  Font.FontName := cFontName;    // フォント名
  Font.Size     := cFontSize;    // フォントサイズ
  Font.Color    := cFontColor;   // フォント色
  Height        := 19;           // 高さ
  ZIndex        := 0;            // 位置

  // ボタンのAsyncClick時、画面をロック(ウエイトカーソルを表示)するよう変更
  LockOnAsyncEvents := [aeClick];

  // Asyncイベントでの Submit
  SubmitOnAsyncEvent := False;
end;
```

画面を閉じるリンクの基本情報を指定

2-5. カレンダーボタンクリック時処理

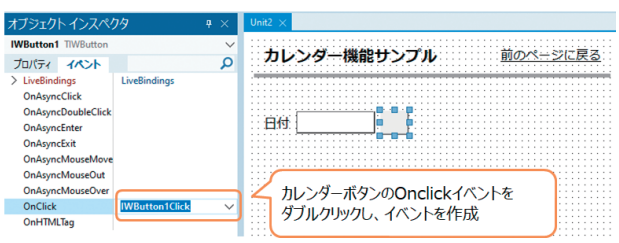
前述の2-3～2-4で実装したクラスを使用し、カレンダーボタンクリック時にカレンダーを表示、カレンダーの日付選択時に入力項目に日付をセットする処理を記述する【図6】。

図 6 サンプル画面の挙動イメージ



まず、カレンダーボタンのOnClickイベントを作成する【図7】。次に、前述で実装したクラスを使用し、カレンダーを作成する処理を記述する【ソース6】。

図 7 OnClickイベントの作成



ソース 6

カレンダーボタンのクリック時処理

```
procedure TIWForm2.IWButton1Click(Sender: TObject);
var
  dSetDate: TDateTime;     // 日付値(セット用)
  iDate: Integer;          // 日付値
  objParent: TWinControl;  // 親オブジェクト
  regParent: TIWRegion;    // メインRegion
  iTopSelf: Integer;       // Top位置
begin
  // 初期化
  objParent := IWForm2;  // 親オブジェクト
  iTopSelf  := 0;        // Top位置

  // 入力値を内部保持
  iDate := Self.DateValue;

  // セットする日付を内部保持
  if (TryStrToDate(FormatFloat('0000/00/00', iDate), dSetDate)) then
  begin
    dSetDate := dSetDate;
  end
  else
  begin
    dSetDate := Now;
  end;

  // 親オブジェクトを取得
  iTopSelf  := iTopSelf + objParent.Top;  // Top位置

  // メインRegion
  regParent := TIWRegion(objParent);

  // カレンダー関連の作成
  if (FDateCalendar = nil) then
  begin
    // ----- カレンダーの生成 -----
    FDateCalendar := TIWSMPLCalendar.Create(Self);
    with FDateCalendar do
    begin
      Parent                := regParent;
      Name                  := Self.Name + '_Calendar';
      BGColor               := clWebWhite;
      Left                  := Self.Left;
      Top                   := iTopSelf + IWButton1.Top + IWButton1.Height + 10;
      StartDate             := DateOf(dSetDate);
      OnDateChange          := CalendarDateChange;  // カレンダーの日付選択処理
      FreeNotification(Self);
    end;

    // ----- 画面を閉じるリンクの生成 -----
    FlnkClose := TIWSMPLLink.Create(Self);
    with FlnkClose do
    begin
      Parent       := FDateCalendar.Parent;
      Name         := Self.Name + '_Calendar_FlnkClose';
      Left         := FDateCalendar.Left + 185;
      Top          := FDateCalendar.Top  + 2;
      Width        := Height;
      Caption      := '×';
      Font.Color   := clWebBlue;
      Font.Size    := Self.Font.Size + 3;
      OnAsyncClick := CalendarlnkCloseAsyncClick;  // リンククリック処理
      FreeNotification(Self);
    end;
  end

  // カレンダー関連を破棄
  else
  begin
    CalendarDestroy;
  end;
end;
```

カレンダー
(TIWSMPLCalendar)
クラスを生成

画面を閉じるリンク
(TIWSMPLLink)
クラスを生成

カレンダー機能の実装は以上である。

3. 視覚的に分かりやすいローディング画面

WEBアプリケーションで、入力内容をサーバーへ送信する、ページ数の多いレポートを出力するなど、時間のかかる処理を行う場合があるだろう。その際、ブラウザ側はサーバーからのレスポンス待ちの状態となり、画面が固まっているように見えてしまう。そこで、ローディング画面の表示により処理中を示すことで、ユーザーの不安感を減らすことができる。本章では、Delphi/400とCSSを使用し、処理中にローディング画面を表示する方法を紹介する。

3-1.前提条件

本章では、商品マスタメンテナンス画面を想定し、検索時、更新時にローディング画面を表示させる。今回は**【図8】**のDDSより作成された商品マスタ（SHOHINM）を使用する。

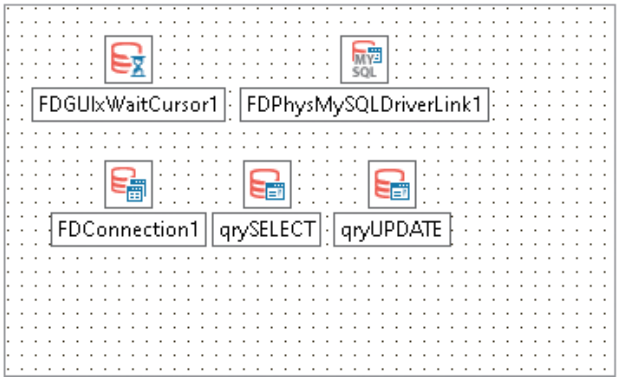
図 8　商品マスタのファイルレイアウト

```
A*****************************************************************************
A*        FILE-ID     :     SHOHINM
A*        FUNCTION    :     商品マスタ
A*****************************************************************************
A                                      UNIQUE
A           R SHOHINMR                 TEXT('  商品マスタ  ')
A             SHSHCD         10A       COLHDG('  商品コード  ')
A             SHSHNM         50O       COLHDG('  商品名  ')
A             SHSRNM         30O       COLHDG('  商品略称  ')
A             SHSRKN          9P 2     COLHDG('  仕入価格  ')
A             SHJURY          7P 2     COLHDG('  重量  ')
A             SHZKKB          1A       COLHDG('  在庫管理区分  ')
A             SHCRDT          8S 0     COLHDG('  作成日付  ')
A             SHCRTM          6S 0     COLHDG('  作成時間  ')
A             SHCRUS         10A       COLHDG('  作成ユーザー  ')
A             SHUPDT          8S 0     COLHDG('  作成日付  ')
A             SHUPTM          6S 0     COLHDG('  作成時間  ')
A             SHUPUS         10A       COLHDG('  作成ユーザー  ')
A           K SHSHCD
```

3-2.IBMiデータベースへの接続

接続先情報を設定するためのiniファイルを準備し、【図9】の通りにデータモジュールにコンポーネントを配置する。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 9　データモジュール:コンポーネント配置



IBMiデータベースへ接続する処理【**ソース7**】を記述する。第4章のサンプル画面からも呼び出せるように、publicセクションで共通メソッドとして作成する。

ソース 7

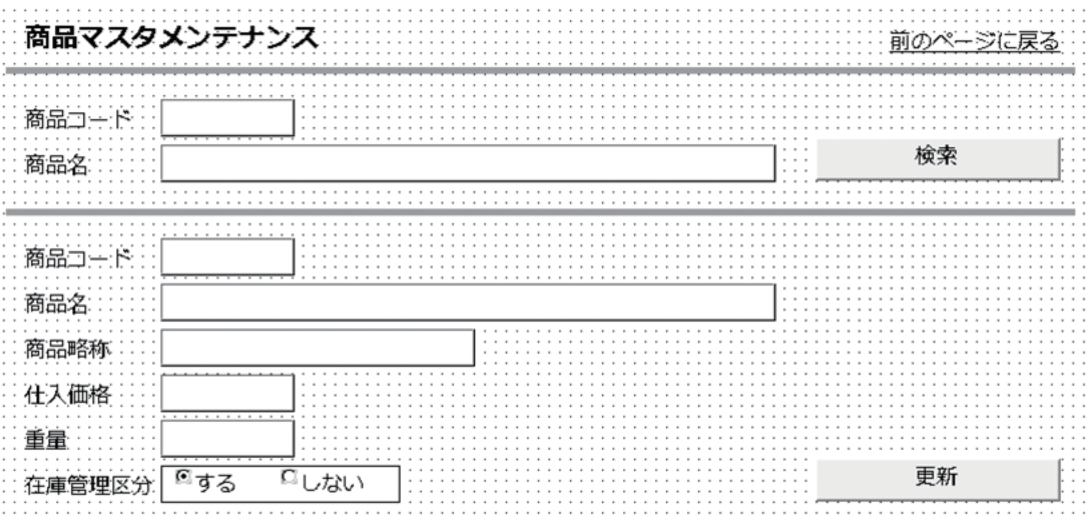
データモジュール接続開始処理

```
procedure TDataModule1.OpenDataBase;
begin
  // FireDAC接続
  if (not FDConnection1.Connected) then
  begin
    try
      FDConnection1.Params.Values['Database']     := IWServerController.Database;
      FDConnection1.Params.Values['ODBCAdvanced'] := 'LibraryOption=';
      FDConnection1.Params.Values['User_Name']    := IWServerController.User_Name;
      FDConnection1.Params.Values['Password']     := IWServerController.Password;
      FDConnection1.LoginPrompt := False;
      FDConnection1.Connected := True;
    except
      raise;
    end;
  end;
end;
```

3-3.商品マスタメンテナンス画面の作成

新規画面を作成し、【**図10**】の通りにコンポーネントを配置する。

図 10　商品マスタメンテナンス画面:コンポーネント配置



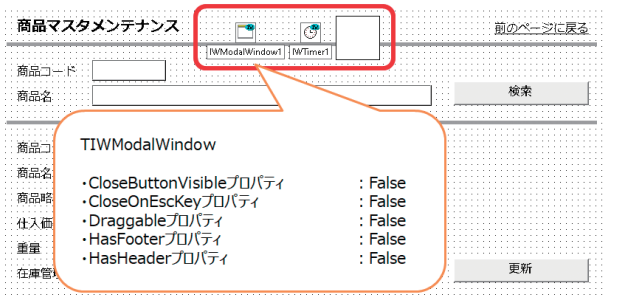
3-4.ローディング画面の作成

今回実装するローディング画面は、TIWFrame、TIWModalWindow、CSSを使用して作成する。まず、TIWFrameを新規作成し、**【図11】**の通りにコンポーネントを配置し、プロパティ値の設定を行う。また、ローディング中の文言表示用のプロパティを宣言し、画面から呼び出された際に、任意の文言を表示できるようにしておく。

図 11　ローディング画面(フレーム):コンポーネント配置

次に、3-3で作成した画面に、フレーム表示用のコンポーネントを追加で配置する**【図12】**。
TIWModalWindowを使用することで、フレームを画面へモーダル表示することができる。

図 12　ローディング画面(フレーム):コンポーネントの追加



ローディング画面表示用の共通関数を画面側に記述する**【ソース8】**。共通関数の引数には、ローディング画面表示中の文言、表示中に実行するイベントを準備し、画面側からの指定文言の表示、指定イベントの実行を行う。またTIWTimerのOnAsyncTimerメソッドに、対象イベントの実行、処理完了後にローディング画面を破棄する処理を記述する**【ソース9】**。

ソース 8

ローディング画面の表示

```
procedure TIWForm3.ShowLoadingFra(AEvent: TNotifyEvent;
  ALoadingStr: String = '');
var
  Frame1: TIWFrame1;
begin
  // ローディング画面の破棄
  IWModalWindow1.Close;
  FreeAndNilList(IWRegion4);

  // イベントのセット
  FAfterEvent := AEvent;
```

```
    // ローディング画面の生成
    Frame1              := TIWFrame1.Create(IWRegion4);
    Frame1.Parent
    Frame1.LoadingStr  := ALoadingStr;
    Frame1.SetSpinnerPosition;

    // フレーム(ローディング画面)を画面にレンダリングする
    IWModalWindow1.ContentElement := IWRegion4;
    IWModalWindow1.Show;

    // タイマー開始
    IWTimer1.Enabled := True;
  end;
```

ソース 9

TIWTimer の OnAsyncTimer メソッド

```
procedure TIWForm3.IWTimer1AsyncTimer(Sender: TObject;
  EventParams: TStringList);
begin
  // タイマー停止
  IWTimer1.Enabled := False;

  // ローディング画面表示中のイベントを実行
  if Assigned(FAfterEvent) then
  begin
    try
      FAfterEvent(nil);
    finally
      // ローディング画面の破棄
      FAfterEvent := nil;
      IWModalWindow1.Close;
      FreeAndNilList(IWRegion4);
    end;
  end;
end;
```

ここで、TIWModalWindowのプロパティ・メソッドについて、ポイントとなる内容を一部抜粋して説明する。

(1)CloseButtonVisibleプロパティ
・True:フレーム右上の「×」ボタンを表示

(2)CloseOnEscKeyプロパティ
・True:Escapeキー押下でフレームを終了

(3)Draggableプロパティ
・True:マウスでフレームを移動できる

(4)HasFooter/HasHeaderプロパティ
・True:フレームのフッター/ヘッダー部を表示

(5)ContentElementプロパティ
・フレームをレンダリングするTControlを指定
※ここでは、【図12】で追加したTIWRegionとする

(6)Showメソッド
・ContentElementプロパティで指定したTControlに対し、フレームをレンダリングする

3-5.ローディング画面の組み込み

3-3で作成した画面の検索ボタンのOnAsyncClick処理、更新ボタンのOnAsyncClick処理のそれぞれで、3-4で作成したローディング画面表示用の共通関数を呼び出す**【ソース10～11】**。また、検索ボタンの押下時に商品マスタデータを画面にセットする処理、更新ボタン押下時に商品マスタを更新する処理を作成する。詳細なソースについてはここでは割愛するが、3-2で配置したTFDQueryを使用し、SQL文を発行して各処理を実行する。

ソース 10

検索ボタンの OnAsyncClick 処理

```
procedure TIWForm3.IWButton1AsyncClick(Sender: TObject;
  EventParams: TStringList);
var
  sLoadingStr: String;
begin
  sLoadingStr := '検索中です。<br>しばらくお待ちください。';
  ShowLoadingFra(IWButton1Click, sLoadingStr);
end;
```

ソース 11

更新ボタンの OnAsyncClick 処理

```
procedure TIWForm3.IWButton2AsyncClick(Sender: TObject;
  EventParams: TStringList);
var
  sLoadingStr: String;
begin
  sLoadingStr := '更新中です。<br>しばらくお待ちください。';
  ShowLoadingFra(IWButton2Click, sLoadingStr);
end;
```

3-6.CSSの組み込み

次に、CSSの定義について説明する。

(1)ローディング中の文字の中央揃え

・CSSファイルに**【図13】**のように記述する。また、3-4のローディング画面のスピナー用TIWLabelのCssプロパティに先ほど記述したセレクタ「center_hor　center_ver」を指定する。今回のように複数指定する場合は、セレクタ間を半角スペースで繋ぐ。

図 13　CSS:文字の中央揃えの設定

CSS：水平中央揃え

```
.center_hor {
  justify-content: center;
  height: 100%;
}
```

CSS：垂直中央揃え

```
.center_ver {
  display: flex;
  align-items: center;
}
```

実行結果：CSSの設定なし

検索中です。
しばらくお待ちください。

実行結果：CSSの設定あり

検索中です。
しばらくお待ちください。

(2)ローディング・スピナーのアニメーション

・CSSファイルに【**図14**】のように記述する。また、3-4のローディング画面のスピナーのアニメーション表示用TIWLabelのCssプロパティに先ほど記述したセレクタ「spinner」を指定する。

以上で、処理中にローディング画面を表示するプログラムは完成である。上記プログラムを実行し、検索ボタン押下時処理の結果を確認する。ボタン押下時に指定した文言のローディング画面が表示され、検索処理完了後はローディング画面が終了していることが確認できる【**図15～17**】。

図 14　CSS:スピナーのアニメーション表示

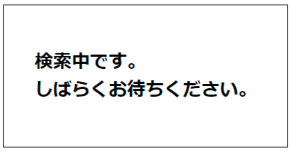
CSS：基本設定

```
/*スピナーの位置などの基本設定や、
 回転させるアニメーションの設定*/
.spinner {
  width: 32px;
  height: 32px;
  margin: 10px auto;
  border: 4px #ddd solid;
  border-top: 4px #2e93e6 solid;
  border-radius: 50%;
  animation: sp-anime 1.0s infinite linear;
}
```

CSS：アニメーション

```
/*スピナーの角度を指定し、回転させる設定*/
@keyframes sp-anime {
  100% {
    transform: rotate(360deg);
  }
}
```

実行結果：CSSの設定なし



実行結果：CSSの設定あり

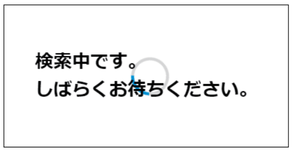


図 15　実行結果①:検索ボタン押下前

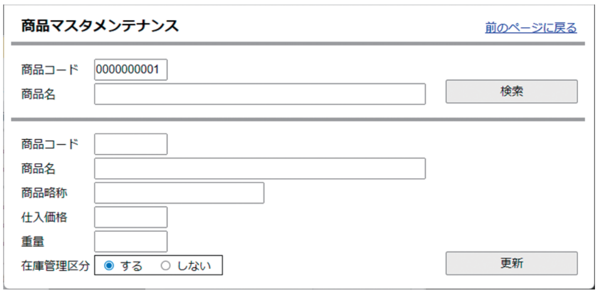


図 16　実行結果②:検索ボタン押下時

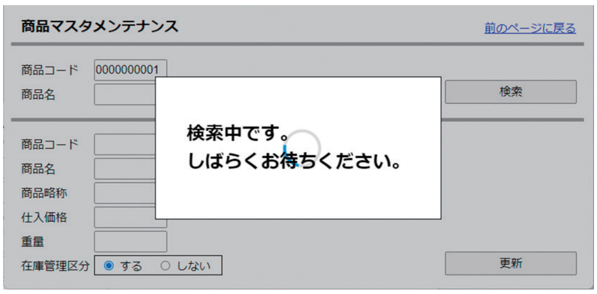
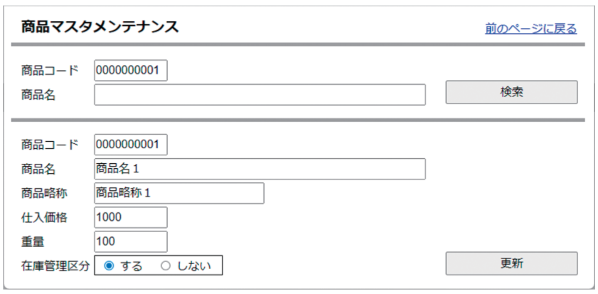


図 17　実行結果③:検索ボタン押下後



4. フレームを使用した明細グリッド

本章では、明細をフレーム形式で作成・使用する方法を紹介する。

4-1.フレームを使用する利点

IntrawebにはTIWGridという明細形式でデータを表示できるコンポーネントが存在する。TIWGridは、タイトルや項目間の幅など、実際にプログラムを実行するまで、その内容を確認することができず、少々扱い辛い印象を受ける【図18】。TIWGridでは、タイトルや幅など明細のビジュアルを統合開発画面上で確認することができないため、実行してビジュアルを確認する必要があり、レイアウト調整に時間がかかる【図19】。

今回紹介する明細のフレームでは、統合開発画面上でビジュアルを確認しながら、レイアウトを調整することができるため、効率よく開発することができる【図20】。

図 18 標準のIWGrid(開発画面)

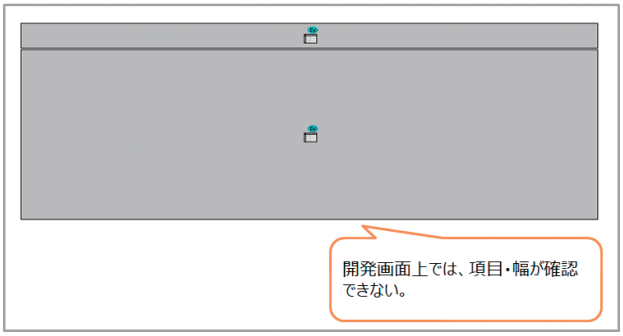


図 19 標準のIWGrid(実行画面)

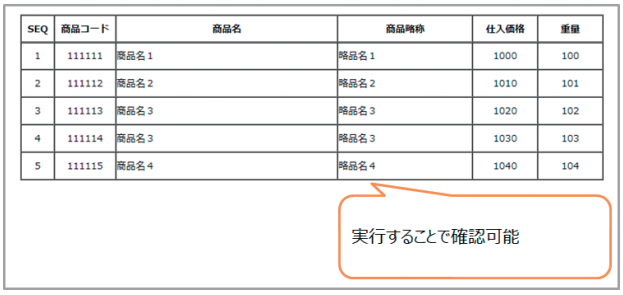
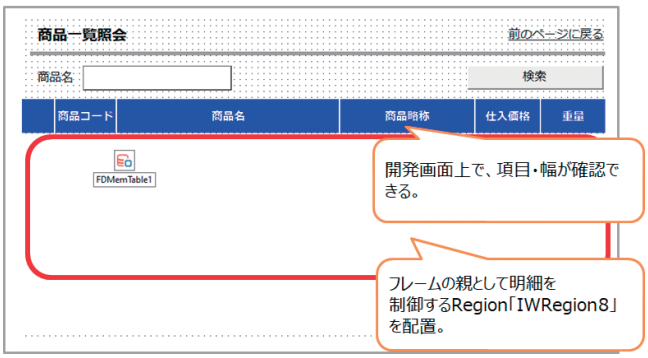


図 20 フレームを用いた場合の開発画面



そこで次節より、商品一覧照会を想定し、フレームを利用した明細形式のプログラム作成例を紹介する。

4-2.今回作成する画面について

今回作成するプログラムのコンポーネントの配置は【図20】の通りである。明細フレームの親として使用するIWRegion8を明細部に配置している。

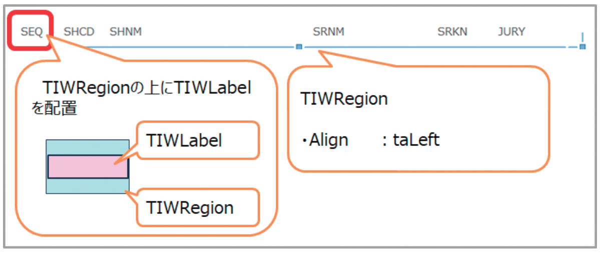
商品マスタについては、第3章と同様の商品マスタ(SHOHINM)を使用する。

4-3.明細フレームの作成

(1)コンポーネントの配置

・【図21】の通りにコンポーネントを配置する。

図 21 明細フレームの構成



(2)明細フレームのプロパティ宣言

・明細フレームには明細項目をそれぞれプロパティとして宣言する【ソース12】。

ソース 12

プロパティの宣言部

```
property SEQ : Integer  read GetSEQ  write SetSEQ;        // SEQ
property SHCD: String   read GetSHCD write SetSHCD;       // 商品コード
property SHNM: String   read GetSHNM write SetSHNM;       // 商品名
property SRNM: String   read GetSRNM write SetSRNM;       // 商品略称
property SRKN: Currency read GetSRKN write SetSRKN;       // 仕入価格
property JURY: Currency read GetJURY write SetJURY;       // 重量
```

(3)「GET項目名」メソッド

・各プロパティごとに、該当項目のLabelのCaptionを返却する処理を記述する【ソース13】。

ソース 13

「GET 項目名」メソッド

```
function TIWFrame2.GetSEQ: Integer;
begin
  Result := StrToIntDef(IWLabel1.Caption, 0);
end;
```

データ表示用Labelの値を受け取る

(4)「SET項目名」メソッド

・各プロパティごとに、該当項目のLabelのCaptionにプロパティ値をセットする処理を記述する【ソース14】。

ソース 14

「Set 項目名」メソッド

```
procedure TIWFrame2.SetSEQ(const Value: Integer);
begin
  IWLabel1.Caption := IntToStr(Value);
end;
```

受け取った値をデータ表示用Labelにセットする

以上で、明細フレームの準備は完了である。

4-4.メイン画面の作成

検索ボタン押下時、第3章と同様の方法でIBMiへ接続し、SQLを発行して画面で指定の条件で商品マスタのデータを取得する。取得したデータはTFDMemTableへ転送する【ソース15】。

ソース 15

データの取得処理

```
// 商品マスタを参照
qrySELECT.Close;
qrySELECT.SQL.Clear;
qrySELECT.SQL.Text := ' SELECT SHSHCD, SHSHNM, SHSRNM, SHSRKN, SHJURY '
                    + ' FROM SHOHINM ';

// 商品名
if (IWEdit1.Text <> '') then
begin
  qrySELECT.SQL.Text := qrySELECT.SQL.Text + ' WHERE SHSHNM LIKE :SHSHNM ';
  qrySELECT.ParamByName('SHSHNM').AsString := '%' + IWEdit1.Text + '%';
end;

// データ取得
qrySELECT.Open;

try
  // 取得したデータをTFDMemTableへ転送
  FDMemTable1.Close;
  FDMemTable1.AppendData(qrySELECT, False);
finally
  qrySELECT.Close;
end;
```

取得したデータをTFDMemTableへ転送

メイン画面では、次のような処理を記述する。

(1)明細の初期化

・明細フレームを解放するための共通関数【ソース16】を作成し、明細作成時の処理の先頭で実行し、明細フレームの初期化を行う。

ソース 16

FreeAndNilList 関数の内容

```
procedure TIWForm4.FreeAndNilList(ARegion: TIWRegion);
var
  i, iCnt: Integer;
  fraList: TFrame;
begin
  // コンポーネントの数を内部保持
  iCnt := ARegion.ComponentCount;

  // コンポーネントの数分、処理を行う
  for i := iCnt - 1 downto 0 do
  begin
    if (ARegion.Components[i] is TFrame) then
    begin
      fraList := TFrame(ARegion.Components[i]);
      FreeAndNil(fraList);
    end;
  end;
end;
```

指定したTIWRegion上にあるフレームを全て解放する

(2)明細作成処理

・TFDMemTableのレコードの数だけ明細フレームを生成する処理を記述する。

・レコード1行を1つのフレームに設定する。

(3)明細フレームのプロパティ設定

・取得したデータを4-3で宣言したフレームのプロパティにセットする【ソース17】。

ソース 17

FreeAndNilList 関数の内容

```
procedure TIWForm4.SetList;
var
  i: Integer;
  Frame2: TIWFrame2;
begin
  // 明細破棄・変数初期化
  FreeAndNilList(IWRegion8);
  FFraCnt := 0;
  Frame2 := nil;

  // 明細作成
  FDMemTable1.First;
  for i := 1 to FDMemTable1.RecordCount do
  begin
    Frame2 := TIWFrame2.Create(IWRegion8);
    with Frame2 do
    begin
      // Nameが重複するとエラーになるため、重複しないよう連番を与える
      Name := 'Frame2_' + IntToStr(i);
      Align := alTop;
      Parent := IWRegion8;
```

```
      // ＜明細データのセット＞
      // フレームに定義しているプロパティに値をセットする
      SEQ  := i;                                          // 明細連番
      SHCD := FDMemTable1.FieldByName('SHSHCD').AsString;  // 商品コード
      SHNM := FDMemTable1.FieldByName('SHSHNM').AsString;  // 商品名
      SRNM := FDMemTable1.FieldByName('SHSRNM').AsString;  // 商品略称
      SRKN := FDMemTable1.FieldByName('SHSRKN').AsInteger; // 仕入価格
      JURY := FDMemTable1.FieldByName('SHJURY').AsInteger; // 重量
      // 色をセット
      case (i mod 2) of
        1:   IWFrameRegion.Color := clWebWhite;  // 奇数行：白
        else IWFrameRegion.Color := $00A4FEF9;   // 偶数行：黄
      end;
    end;

    // 次レコードへ
    Inc(FFraCnt);
    FDMemTable1.Next;
  end;

  // 明細の高さを設定
  if (Frame2 <> nil) then
  begin
    IWRegion8.Height := Frame2.Height * FFraCnt;
  end;
end;
```

フレームを明細レコード数分生成。
同時に設定及びデータをセットする

(4)明細背景色の設定

・明細が見やすくするように奇数行は背景を白色、偶数行は背景を黄色に設定した。

(5)明細の高さを設定

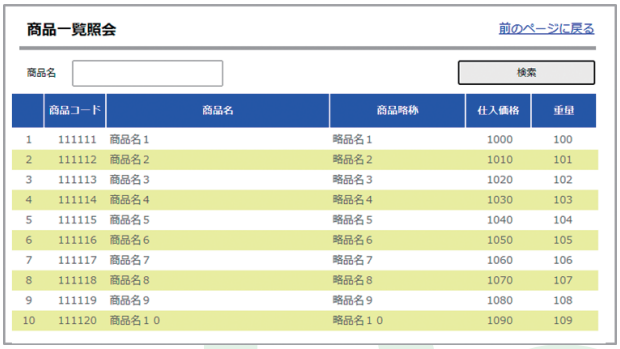
・明細フレームを生成したTIWRegionの高さを設定する。

以上で、フレームを利用した明細形式のプログラムは完成である。

4-5.プログラムの実行

上記プログラムを実行し、検索ボタン押下時処理の結果を確認する。ボタン押下時に商品マスタのデータが明細フレームに表示されることが確認できる【図22】。

図 22　検索ボタン実行後



	商品コード	商品名	商品略称	仕入価格	重量
1	111111	商品名１	略品名１	1000	100
2	111112	商品名２	略品名２	1010	101
3	111113	商品名３	略品名３	1020	102
4	111114	商品名４	略品名４	1030	103
5	111115	商品名５	略品名５	1040	104
6	111116	商品名６	略品名６	1050	105
7	111117	商品名７	略品名７	1060	106
8	111118	商品名８	略品名８	1070	107
9	111119	商品名９	略品名９	1080	108
10	111120	商品名１０	略品名１０	1090	109

本章では、商品一覧照会機能をフレームを利用して作成してきた。明細をフレーム形式で作成することで、設計画面で明細の高さや幅、色などを自由に設定することができる。似たような画面を展開する場合、フレームを利用することで、記述するロジックを削減する事ができ、開発工数を短縮する事が可能である。

5.さいごに

本稿では、ユーザーが直感的に操作することができるよう、システム開発チームが実際の開発時に工夫し、実装したユーザーインターフェース機能を紹介した。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に役立てて頂ければ幸いである。

なお、今回紹介したプログラム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/2024report_intraweb.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）