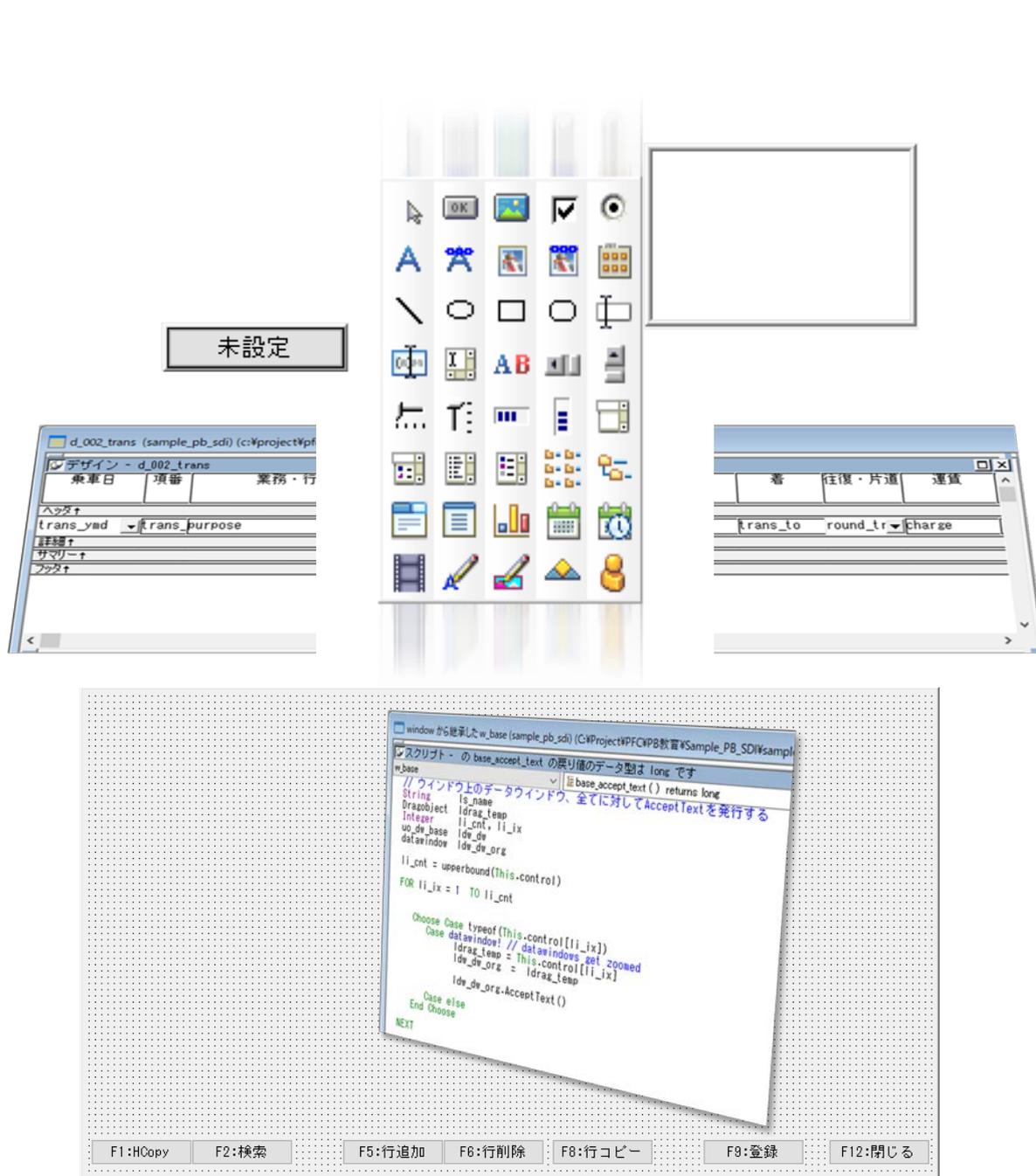


Appeon PowerBuilder 2017 クラス DW 機能



パワーフューチャー株式会社

目次

はじめに	3
1. クラスライブラリ	4
(1) Appeon PowerBuilder 2017 のクラスライブラリ	4
(2) クラスの作成	5
(3) クラスを用いたプログラミング	26
2. データウインドウ有効活用	36
(1) 有効な機能	36
(2) 開発で推奨する方式	42
3. まとめ	44

はじめに

本コースは、当社コース「**Appeon PowerBuilder 2017 入門コース**」を受講済で、**Appeon PowerBuilder 2017** による **DB 操作アプリケーション**の作成方法を理解できていることを前提に、クラスライブラリを用いた開発効率のアップとメンテナンス性の向上について学ぶ。

また、データウインドウの実践的な利用方法を学び、今後の開発に役立ててもらいたい。

本書は下記 2 ステップから構成される。

ステップ 1 では、**Appeon PowerBuilder 2017** 特有のクラスライブラリについて説明し、入門コースで作成したプログラムがどこまでシンプル化できるのか学んでいく。

ステップ 2 では、データウインドウの実践的な利用方法について例を用いて説明する。

1. クラスライブラリ

多くのシステム開発プロジェクトでは、一定のルール（規約）に沿って開発を行うために、C 言語クラス~.NET フレームワークに至るまで、オブジェクト指向プログラミングが用いられており、Appeon PowerBuilder 2017 のクラスでもシンプルかつプロジェクトに沿った基本コンセプトで設計が可能である。

(1) Appeon PowerBuilder 2017 のクラスライブラリ

Appeon PowerBuilder 2017 のクラスライブラリ機能として、カプセル化と継承があり、この組み合わせにより、シンプルで開発効率とメンテナンス性の向上、バグの発生の軽減が可能なシステムを構築できる。

① カプセル化

カプセル化とは、オブジェクトのデータ属性と処理手続き（関数やイベントのスクリプト）をオブジェクト定義に加えることである。

ここではイメージし易いように、共通関数と画面用の共通インタフェースについて述べる。

A) 共通関数

共通関数を関数群としてユーザオブジェクトにまとめることができる。

ユーザオブジェクトによる関数群のカプセル化は、便利なようだが Appeon PowerBuilder 2017 IDE のデバッガ機能でステップ実行ができないなど、開発作業において不便な点があることから、あまり推奨しない。

カプセル化とは意味合いが異なるが、PBL に関数をまとめることで代用できる手法である。

B) 画面用の共通インタフェース

各画面共通で利用できるインタフェース部品を作成することができる。

実装の際には、画面用のコントロールでデザインするのもよし、データウインドウを用いたリッチなインタフェースでデザインするのもよしとし、柔軟に対応する。

② 継承

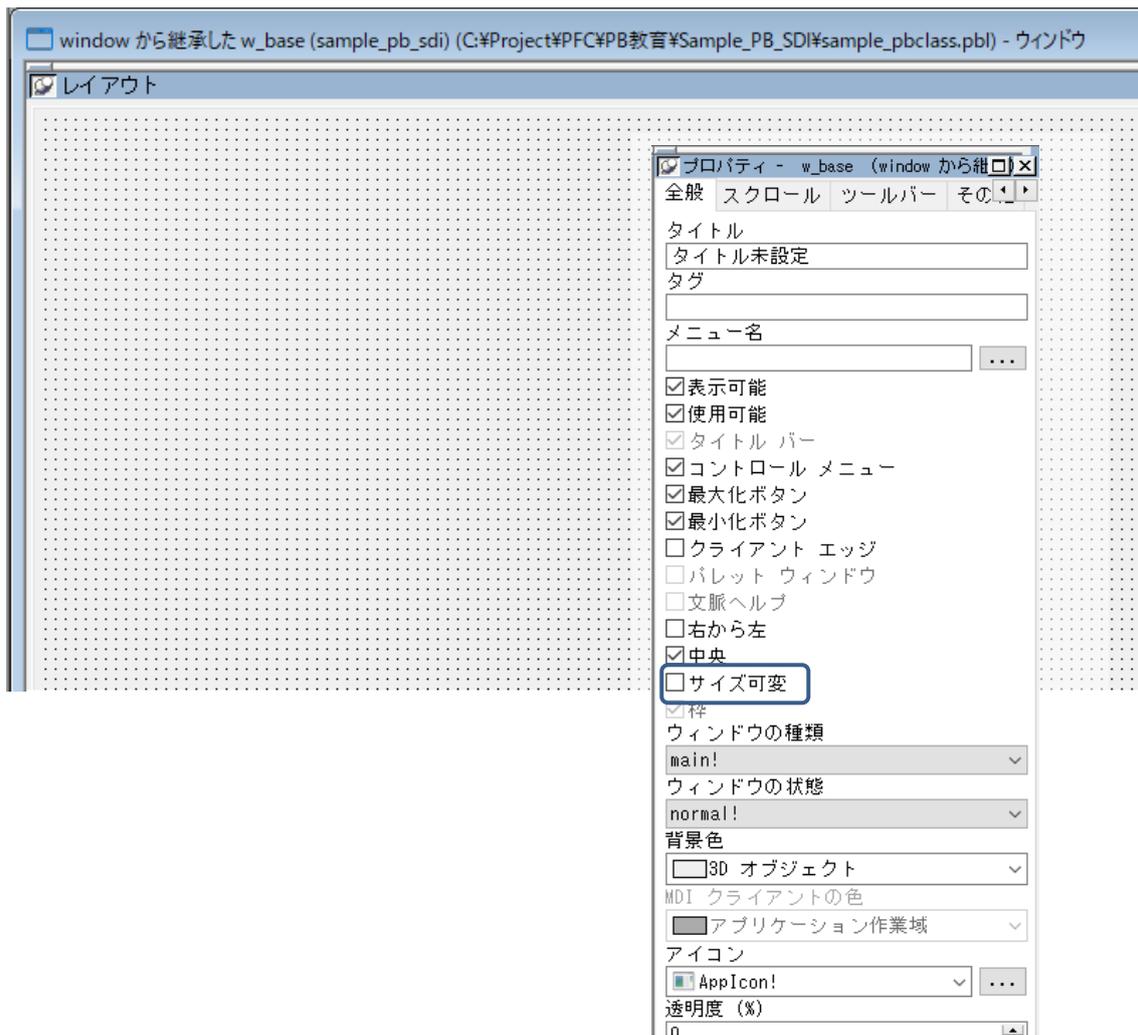
あらかじめ定義されているクラスからデータや処理手続きを受け継ぐことができ、下記の利点がある。

- 外観や動作させるためのスクリプトの一貫性が保障される。
- 先祖オブジェクトで行った修正は、全ての子孫に反映される。
- スクリプトの数が減り、バグ発生率が低減する。

(2) クラスの作成

入門コースで作成したアプリケーションにクラスを用いて再デザインする。
Sample_pbclass.pbl を用意する。

- ① ウィンドウクラスの作成 (その1)
w_base で保存する。



1. クラスライブラリ

(2) クラスの作成 ① ウィンドウクラスの作成 (その1)

ウィンドウ関数を作成する。

A) wf_isnull(文字)

The screenshot shows the 'Script' window for the function 'wf_isnull (readonly string as_string) returns boolean'. The function signature is 'wf_isnull (readonly string as_string) returns boolean'. The access is 'public', the return type is 'boolean', and the parameter 'as_string' is 'readonly' with type 'string'. The code is as follows:

```
IF IsNull(as_string) OR &
  Trim(as_string) = "" THEN
  RETURN True // Null
END IF

RETURN False // Not Null
```

B) wf_isnull(数値)

The screenshot shows the 'Script' window for the function 'wf_isnull (readonly decimal adc_dec) returns boolean'. The function signature is 'wf_isnull (readonly decimal adc_dec) returns boolean'. The access is 'public', the return type is 'boolean', and the parameter 'adc_dec' is 'readonly' with type 'decimal'. The code is as follows:

```
IF IsNull(adc_dec) THEN
  RETURN True // Null
END IF

RETURN False // Not Null
```

C) wf_isnull(日付)

The screenshot shows the 'Script' window for the function 'wf_isnull (readonly datetime adt_datetime) returns boolean'. The function signature is 'wf_isnull (readonly datetime adt_datetime) returns boolean'. The access is 'public', the return type is 'boolean', and the parameter 'adt_datetime' is 'readonly' with type 'datetime'. The code is as follows:

```
IF IsNull(adt_datetime) OR &
  adt_datetime = DateTime(Date("1900/01/01"), Time("00:00:00")) THEN
  RETURN True // Null
END IF

RETURN False // Not Null
```

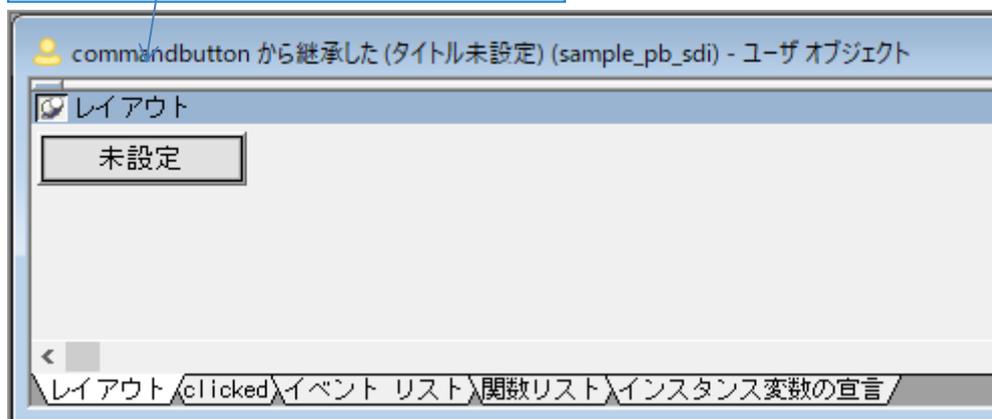
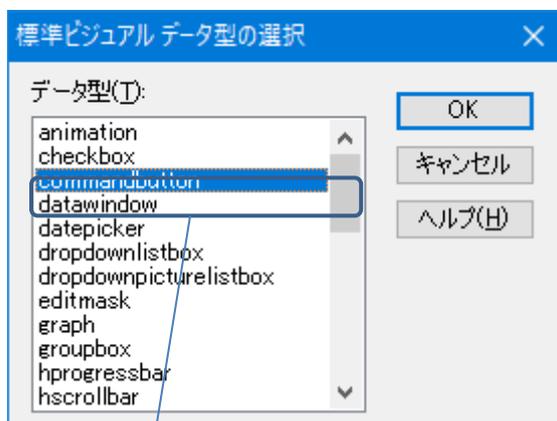
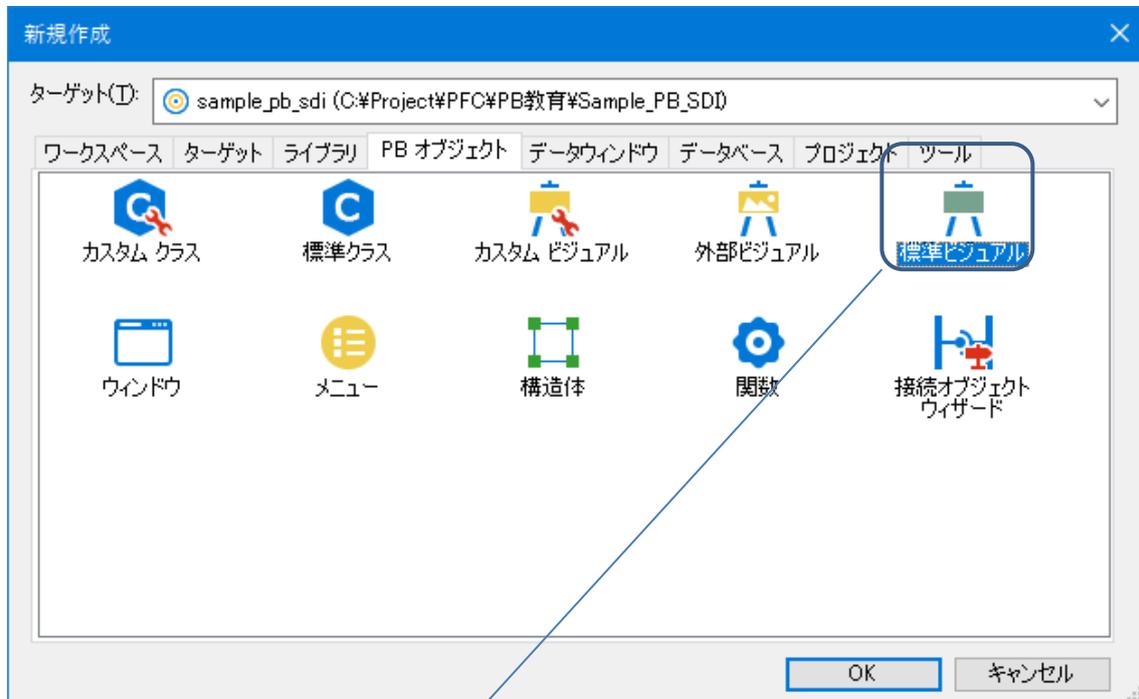
1. クラスライブラリ

(2)クラスの作成 ②コマンドボタンクラスの作成 (その1)

② コマンドボタンクラスの作成 (その1)

ユーザオブジェクトで作成する。

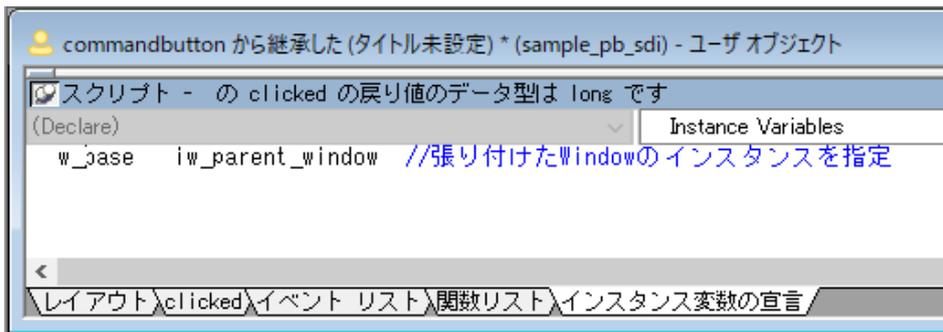
標準ビジュアルでデザインする。スクリプト記載後、保存する。



1. クラスライブラリ

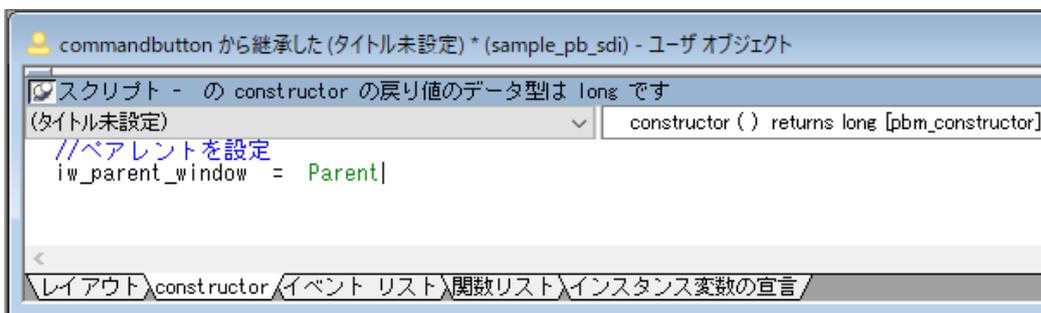
(2)クラスの作成 ②コマンドボタンクラスの作成 (その1)

インスタンス変数にウィンドウインスタンスを指定する。



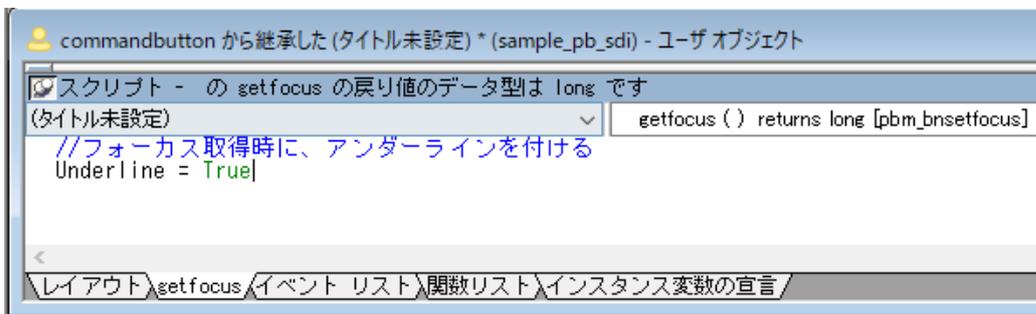
```
commandbutton から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[Script] clicked の戻り値のデータ型は long です
(Declare) Instance Variables
w_base iw_parent_window //張り付けたWindowのインスタンスを指定
<
レイアウト \ clicked \ イベント リスト \ 関数リスト \ インスタンス変数の宣言
```

A) constructor イベント
ペアレントを設定する。



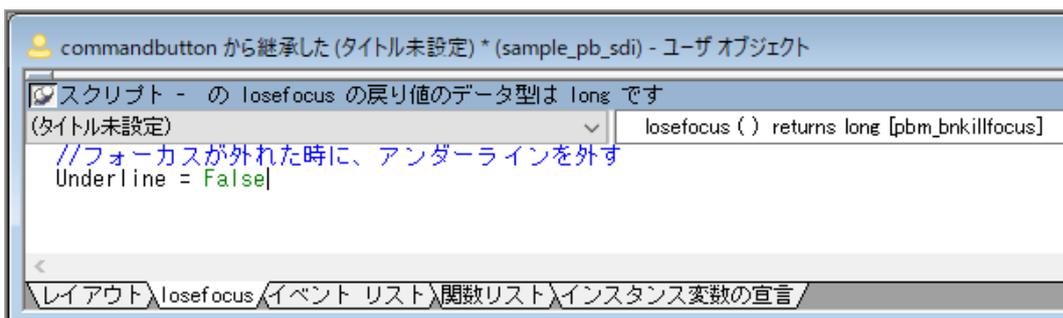
```
commandbutton から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[Script] constructor の戻り値のデータ型は long です
(タイトル未設定) constructor ( ) returns long [pbm_constructor]
//ペアレントを設定
iw_parent_window = Parent|
<
レイアウト \ constructor \ イベント リスト \ 関数リスト \ インスタンス変数の宣言
```

B) getfocus イベント
コマンドボタンにフォーカスがあるとき、確認し易いようにアンダーラインを表示する。



```
commandbutton から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[Script] getfocus の戻り値のデータ型は long です
(タイトル未設定) getfocus ( ) returns long [pbm_bnsetfocus]
//フォーカス取得時に、アンダーラインを付ける
Underline = True|
<
レイアウト \ getfocus \ イベント リスト \ 関数リスト \ インスタンス変数の宣言
```

C) losefocus イベント
コマンドボタンからフォーカスが外れたとき、アンダーラインを外す。



```
commandbutton から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[Script] losefocus の戻り値のデータ型は long です
(タイトル未設定) losefocus ( ) returns long [pbm_bnkillfocus]
//フォーカスが外れた時に、アンダーラインを外す
Underline = False|
<
レイアウト \ losefocus \ イベント リスト \ 関数リスト \ インスタンス変数の宣言
```

名前を付けて保存する。(uo_cb_base)

ユーザー オブジェクトの保存

ユーザー オブジェクト(U):
uo_cb_base

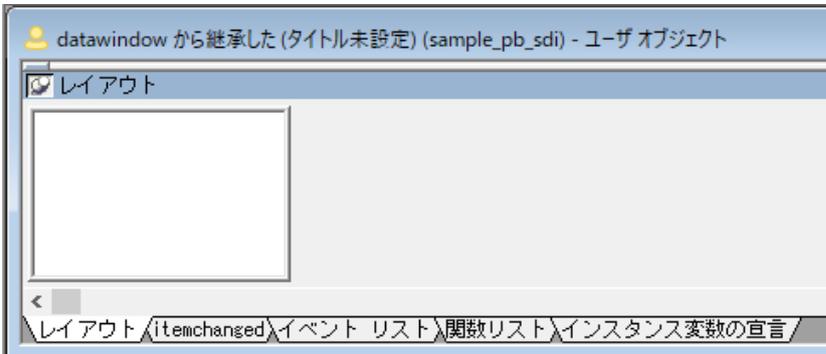
コメント(C):

アプリケーション ライブラリ(A):
C:\Project\PFC\PB教育\Sample_PB_SDI\sample_pb_sdi.pbl
C:\Project\PFC\PB教育\Sample_PB_SDI\sample_pbclass.pbl
C:\Project\PFC\PB教育\Sample_PB_SDI\sample_pb_comm_func.pbl
C:\Project\PFC\PB教育\Sample_PB_SDI\sample_pb_001_emp.pbl
C:\Project\PFC\PB教育\Sample_PB_SDI\sample_pb_002_trans.pbl

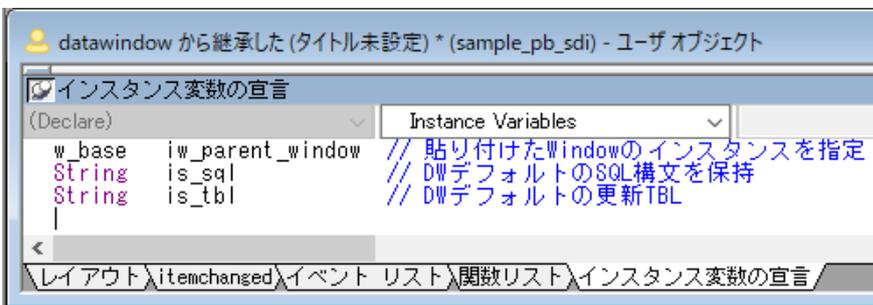
OK
キャンセル
ヘルプ(H)

③ データウインドウクラスの作成 (その1)

uo_cb_base 同様、ユーザオブジェクトの標準ビジュアルから選択する。



インスタンス変数にウインドウインスタンスを指定する。

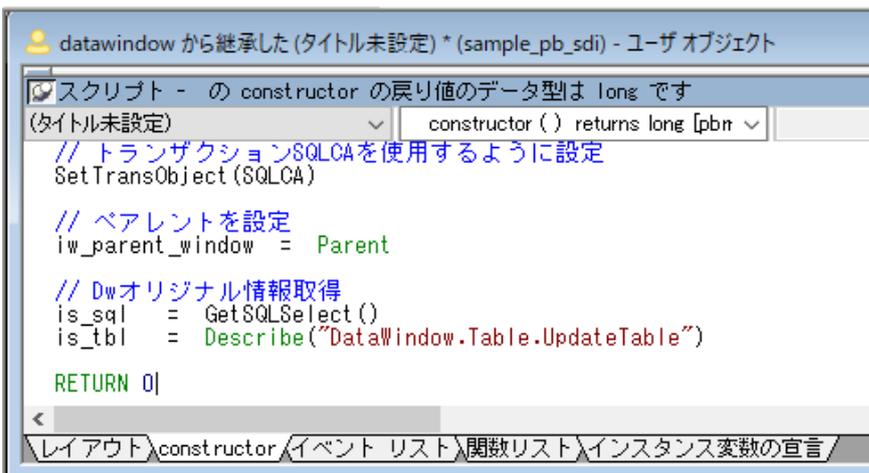


イベントにスクリプトを埋め込む。

A) constructor イベント

ペアレントを設定する。

Dw オリジナル情報を取得して保持する。



B) dberror イベント

PB 標準機能を廃止し、DB エラーの機能を組み込む。

```

datawindow から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[ ] スクリプト - の dberror の戻り値のデータ型は long です
(タイトル未設定) | dberror ( long sqlcode, string sqlerrtext, s
// PB標準で用意されているDBエラーの出力を廃止し、オリジナルでメッセージ表示を行う。
// ここではシンプルにMessageBoxを用いている。
// ログ出力も可能
MessageBox("DBエラー", String(SqlDBCode) + ":" + SqlErrText)

RETURN 1|
レイアウト dberror イベント リスト 関数リスト インスタンス変数の宣言

```

C) getfocus イベント

カラムを選択状態にする。

```

datawindow から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[ ] スクリプト - の getfocus の戻り値のデータ型は long です
(タイトル未設定) | getfocus ( ) returns long [pbm_dwnsetfocus
// カラムを選択状態にする
IF Len(GetText()) > 0 THEN
  SelectText(1, Len(GetText()))
END IF|
レイアウト getfocus イベント リスト 関数リスト インスタンス変数の宣言

```

D) itemchanged イベント

カラムを選択状態にする。

```

datawindow から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[ ] スクリプト - の itemchanged の戻り値のデータ型は long です
(タイトル未設定) | itemchanged ( long row, dwoobject dwo, string
// カラムを選択状態にする
IF Len(GetText()) > 0 THEN
  SelectText(1, Len(GetText()))
END IF|
レイアウト itemchanged イベント リスト 関数リスト インスタンス変数の宣言

```

E) itemerror イベント

PB 標準機能を廃止する。(エラー出力なし)

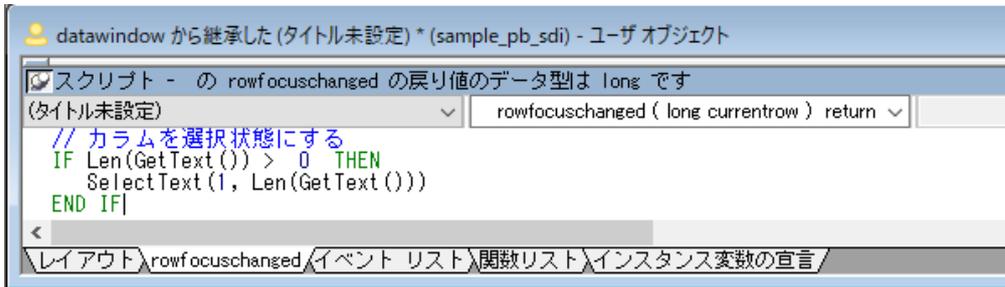
エラー処理は、各画面で実施する。

```

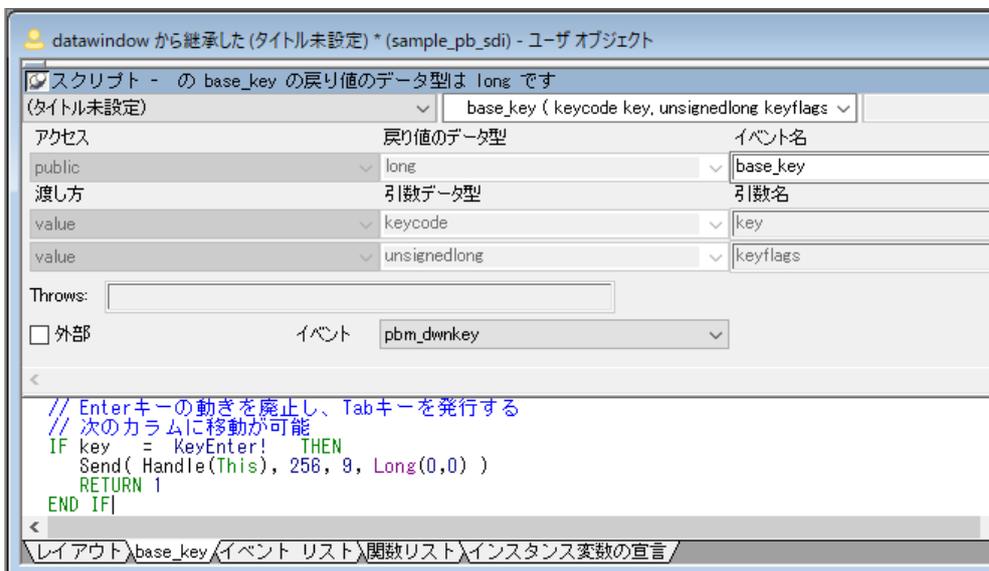
datawindow から継承した (タイトル未設定) * (sample_pb_sdi) - ユーザオブジェクト
[ ] スクリプト - の itemerror の戻り値のデータ型は long です
(タイトル未設定) | itemerror ( long row, dwoobject dwo, string de
// PB標準で用意されているItemErrorを廃止する。
// エラーの内容が不親切なため、個々にチェック処理を記載する。
Return 1|
レイアウト itemerror イベント リスト 関数リスト インスタンス変数の宣言

```

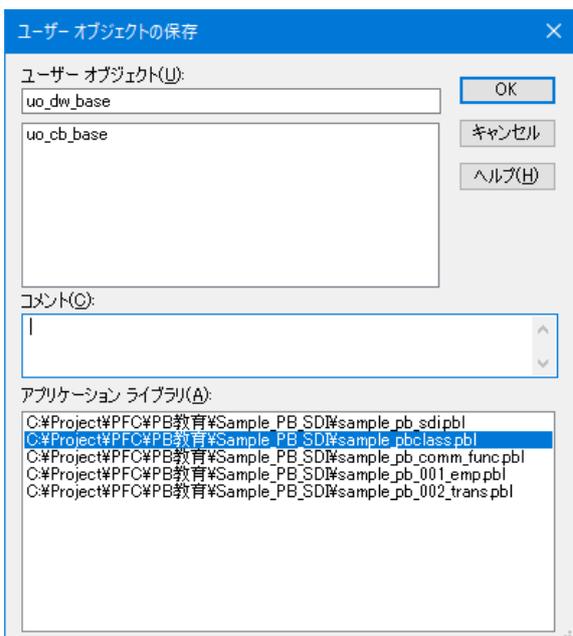
F) rowfocuschanged イベント
 カラムを選択状態にする。



G) base_key イベント (ユーザ定義イベント)
 Enter 押下で Tab を動作させる。



名前を付けて保存する。(uo_dw_base)

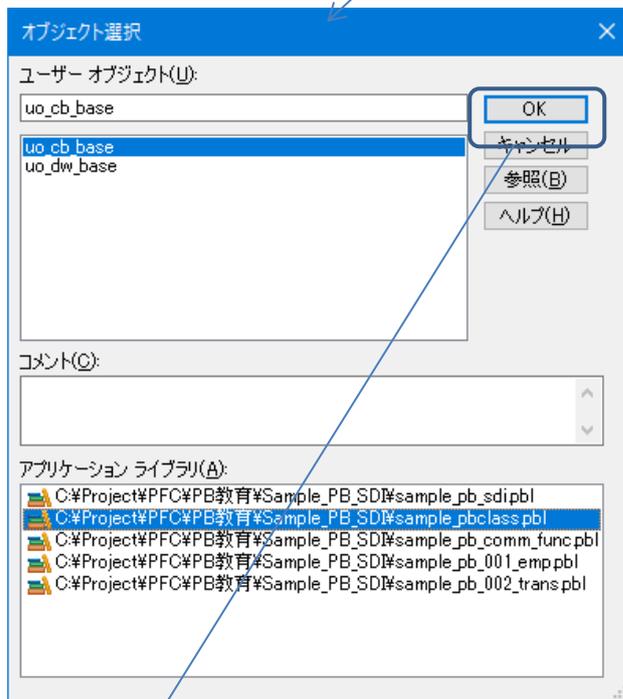
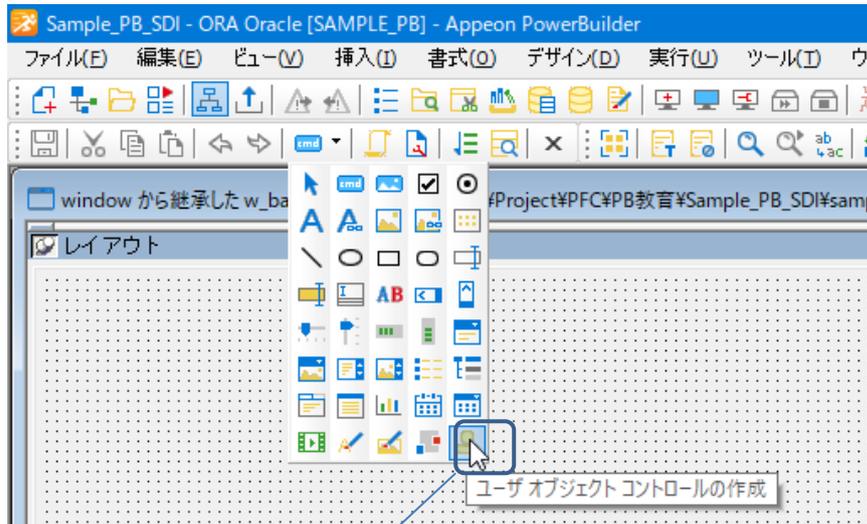


1. クラスライブラリ

(2)クラスの作成 ④ウインドウクラスの作成 (その2)

④ ウインドウクラスの作成 (その2)

ウインドウ上にシステム共通で使用するボタンを配置する。

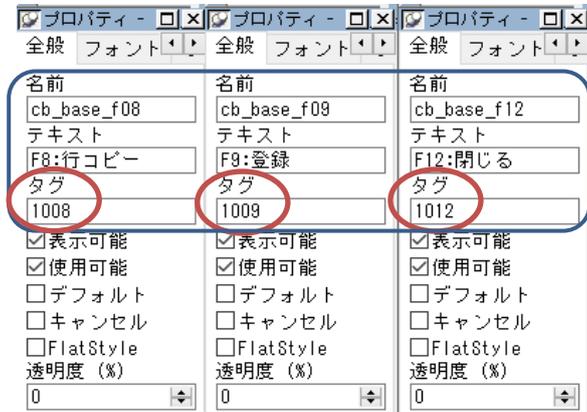


それぞれ配置する。
タブオーダーは0に設定する。

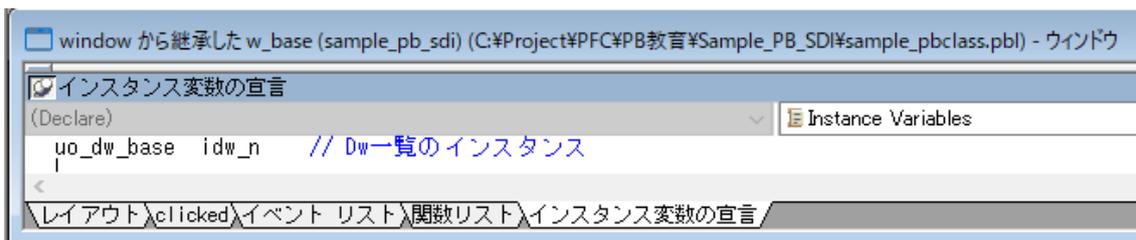


1. クラスライブラリ
 (2)クラスの作成 ④ウインドウクラスの作成 (その2)

プロパティに値を設定する。



インスタンス変数の設定



クラスイベントを記載する。

A) base_accept_text イベント

画面上全てのデータウインドウに AcceptText を発行する。

The screenshot shows the Visual Basic IDE with the Properties window for the `base_accept_text` event. The event name is set to `base_accept_text` and the return type is `long`. The script code is as follows:

```

// ウィンドウ上のデータウインドウ、全てに対してAcceptTextを発行する
String    ls_name
Dragobject ldrag_temp
Integer   li_cnt, li_ix
uo_dw_base ldw_dw
datawindow ldw_dw_org

li_cnt = upperbound(This.control)
FOR li_ix = 1 TO li_cnt

    Choose Case typeof(This.control[li_ix])
        Case datawindow! // datawindows get zoomed
            ldrag_temp = This.control[li_ix]
            ldw_dw_org = ldrag_temp

            ldw_dw_org.AcceptText()

        Case else
            End Choose
    NEXT
RETURN 0

```

B) base_f01 イベント

画面コピーのスク립トを組み込む。

The screenshot shows the Visual Basic IDE with the Properties window for the `base_f01` event. The event name is set to `base_f01` and the return type is `long`. The script code is as follows:

```

Long    ilg_print
String  ls_printer

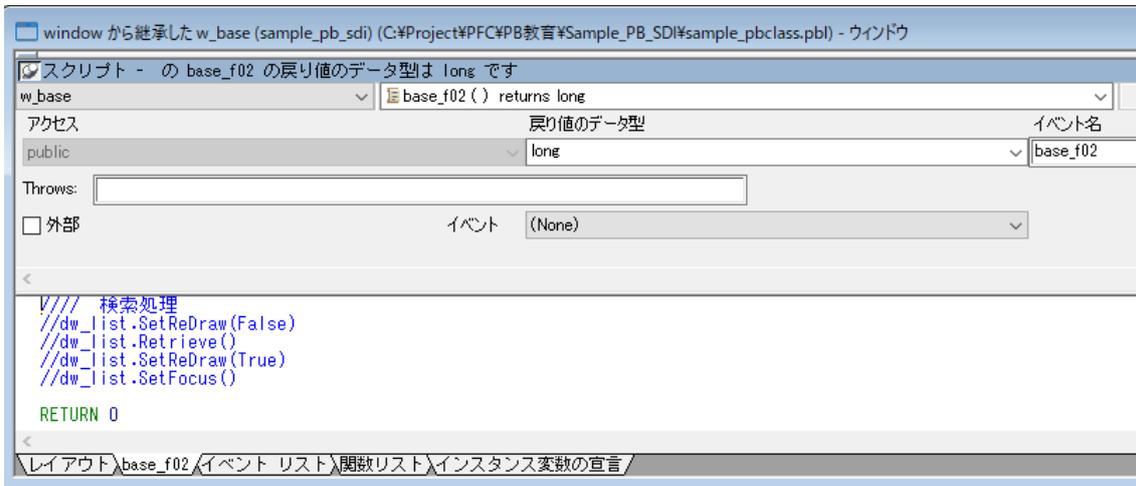
IF MessageBox("Question", "画面イメージを印刷しますか?", Question!, YesNo!, 1) = 1 THEN
    ilg_print = PrintOpen()
    Print(ilg_print, 0, 0, 7800, 7000)
    PrintClose(ilg_print)
END IF

RETURN 0

```

C) base_f02 イベント

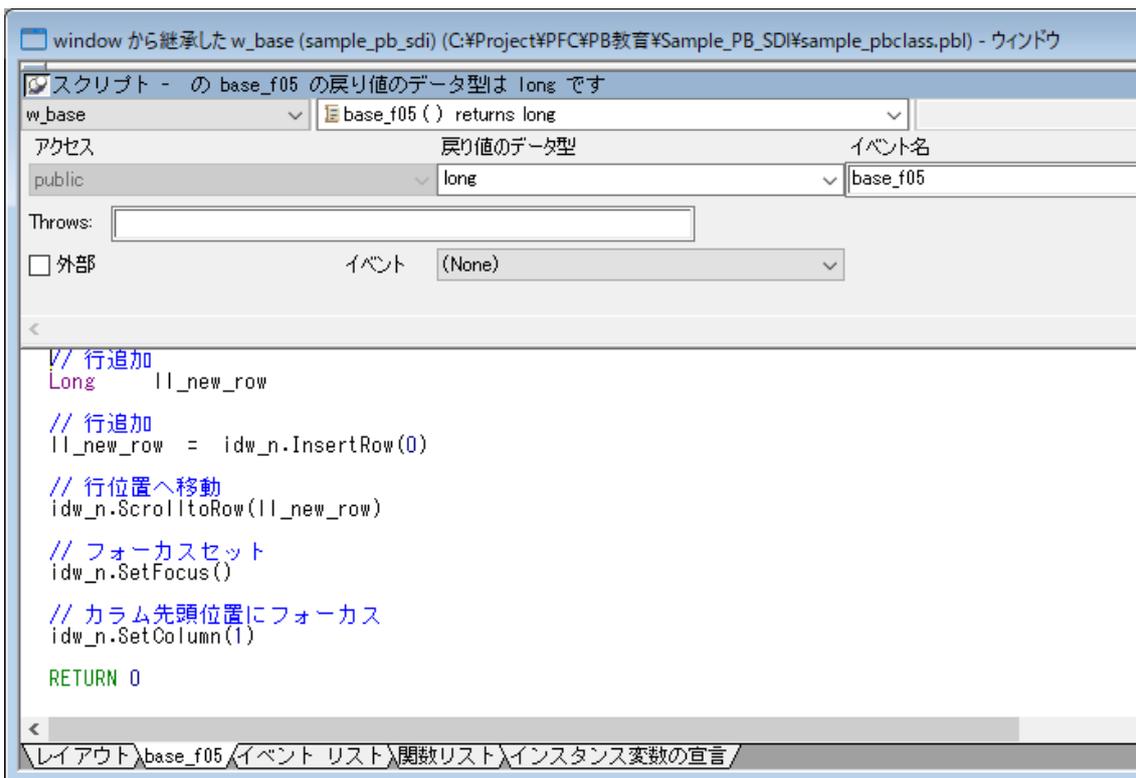
検索処理は、画面ごとに異なるため、サンプルスクリプトを記載する。



D) base_f05 イベント

行追加処理を組み込む。

d w 一覧インスタンスの `idw_n` に対して処理を実施する。



E) base_f06 イベント

行削除処理を組み込む。

d w 一覧インスタンスの idw_n に対して処理を実施する。

Script - の base_f06 の戻り値のデータ型は long です

アクセス	戻り値のデータ型	イベント名
public	long	base_f06

Throws:

外部 イベント (None)

```

Long    ll_row
// データウインドウの行が0のとき処理を抜ける
IF idw_n.RowCount() = 0 THEN RETURN 0
// 現在の行位置を取得し、削除する
ll_row = idw_n.GetRow()
idw_n.DeleteRow(ll_row)
// データウインドウの行が0になったら処理を抜ける
IF idw_n.RowCount() = 0 THEN RETURN 0
// 削除した行位置分のデータ件数があるとき、同じ行にフォーカス
IF idw_n.RowCount() >= ll_row THEN
    idw_n.ScrollToRow(ll_row)
// 削除した行位置分のデータ件数がないとき、最終行にフォーカス
ELSE
    idw_n.ScrollToRow(idw_n.RowCount())
END IF
// フォーカスセット
idw_n.SetFocus()
// カラム先頭位置にフォーカス
idw_n.SetColumn(1)
RETURN 0

```

レイアウト base_f06 イベント リスト 関数リスト インスタンス変数の宣言

F) base_f08 イベント

行コピー処理を組み込む。

Script - base_f08 の戻り値のデータ型は long です

w_base base_f08() returns long

アクセス	戻り値のデータ型	イベント名
public	long	base_f08

Throws:

外部 イベント (None)

```

Long      ll_row
DateTime  ldt_null
Dec       ldc_null

SetNull(ldt_null)
SetNull(ldc_null)

// データが 1 件もないときは処理しない
IF idw_n.RowCount() = 0 THEN RETURN 0

// 現在の行位置を取得
ll_row = idw_n.GetRow()

// 最終行位置にコピー
idw_n.RowsCopy(ll_row, ll_row, Primary!, idw_n, idw_n.RowCount() + 1, Primary!)

// 行位置へ移動
idw_n.ScrollToRow(ll_row)

// フォーカスセット
idw_n.SetFocus()

// カラム先頭位置にフォーカス
idw_n.SetColumn(1)

RETURN 0

```

レイアウト | base_f08 | イベント リスト | 関数リスト | インスタンス変数の宣言

G) base_f09 イベント

登録処理は、画面ごとに異なるため、サンプルスクリプトを記載する。

The screenshot shows the Visual Basic IDE with the following details:

- Window title: window から継承した w_base (sample_pb_sdi) (C:\Project\PF\教育\Sample_PB_SDI\sample_pbclass.pbl) - ウィンドウ
- Script name: base_f09 の戻り値のデータ型は long です
- Access: public
- Return type: long
- Event name: base_f09
- Throws: (empty)
- External: 外部
- Event: (None)

```

//Long    ll_row
//Long    ll_find
//String  ls_XXXXXXXXXX
//// 入力を確定させる
//EVENT  base_accept_text()
////
//// 全行チェックする
//FOR ll_row = 1 TO dw_list.RowCount()
//// 主キー重複チェック
////
//// 入力変更がない行はスキップする
// IF dw_list.GetItemStatus(ll_row, 0, Primary!) = NotModified! OR &
// dw_list.GetItemStatus(ll_row, 0, Primary!) = New! THEN
//// CONTINUE
// END IF
////
//// 項番のチェック
// ls_XXXXXXXXXX = dw_list.Object.XXXXXXXXXX[ll_row]
// IF wf_isnull(ls_XXXXXXXXXX) THEN
//     MessageBox("入力エラー", "必須入力です。")
//     dw_list.ScrollToRow(ll_row)
//     dw_list.SetFocus()
//     dw_list.SetColumn("XXXXXXXXXXXX")
//     RETURN 0
// END IF
// NEXT
////
//// 更新する
// IF dw_list.Update() = 1 THEN
//     COMMIT;
// ELSE
//     ROLLBACK;
//     RETURN 0
// END IF
// dw_list.Retrieve(gs_emp_cd)
// dw_list.SetFocus()

RETURN 0

```

レイアウト base_f09 イベント リスト 開数リスト インスタンス変数の宣言

H) base_f12 イベント

終了処理を組み込む。

The screenshot shows the Visual Basic IDE with the following details:

- Window title: window から継承した w_base (sample_pb_sdi) (C:\Project\PF\教育\Sample_PB_SDI\sample_pbclass.pbl) - ウィンドウ
- Script name: base_f12 の戻り値のデータ型は long です
- Access: public
- Return type: long
- Event name: base_f12
- Throws: (empty)
- External: 外部
- Event: (None)

```

Close(This)

RETURN 0

```

レイアウト base_f12 イベント リスト 開数リスト インスタンス変数の宣言

I) base_action_post イベント

F キー押下、F キー連動ボタン押下時に Call されるイベントを組み込む。

The screenshot shows the IDE interface for configuring the `base_action_post` event. The top part displays the event configuration table:

アクセス	戻り値のデータ型	イベント名
public	long	base_action_post

Below the table, the event is configured with the following settings:

- 渡り方: 引数データ型
- value: integer
- 引数名: ai_no
- Throws: (empty)
- 外部: イベント: (None)

The implementation code in the script editor is as follows:

```

SetPointer (HourGlass!)
CHOOSE CASE ai_no
CASE 1001
EVENT base_f01()
CASE 1002
EVENT base_f02()
CASE 1005
EVENT base_f05()
CASE 1006
EVENT base_f06()
CASE 1008
EVENT base_f08()
CASE 1009
EVENT base_f09()
CASE 1012
EVENT base_f12()
CASE ELSE
END CHOOSE
RETURN 0
  
```

J) base_action イベント

入力確定させ、他のウインドウ動作も完結した状態でF キーボタンを動作させたいため、`base_action_post` イベントを POST CALL する。

The screenshot shows the IDE interface for configuring the `base_action` event. The top part displays the event configuration table:

アクセス	戻り値のデータ型	イベント名
public	long	base_action

Below the table, the event is configured with the following settings:

- 渡り方: 引数データ型
- value: integer
- 引数名: ai_no
- Throws: (empty)
- 外部: イベント: (None)

The implementation code in the script editor is as follows:

```

EVENT base_accept_text()
POST EVENT base_action_post(ai_no)
RETURN 0
  
```

K) key イベント

F キーに連動する動作を組込む。

Script - の key の戻り値のデータ型は long です

w_base key (keycode key, unsignedlong keyflags) returns long [pbm_keydown]

アクセス	戻り値のデータ型	イベント名
public	long	key

渡り方

value	引数データ型	引数名
value	keycode	key
value	unsignedlong	keyflags

Throws:

外部 イベント pbm_keydown

```

V// キーイベントにて、Fキーに対応するイベントをCallする
CHOOSE CASE key
CASE KeyF1!
  IF cb_base_f01.Enabled = True THEN
    cb_base_f01.EVENT Clicked()
  END IF
CASE KeyF2!
  IF cb_base_f02.Enabled = True THEN
    cb_base_f02.EVENT Clicked()
  END IF
CASE KeyF5!
  IF cb_base_f05.Enabled = True THEN
    cb_base_f05.EVENT Clicked()
  END IF
CASE KeyF6!
  IF cb_base_f06.Enabled = True THEN
    cb_base_f06.EVENT Clicked()
  END IF
CASE KeyF8!
  IF cb_base_f08.Enabled = True THEN
    cb_base_f08.EVENT Clicked()
  END IF
CASE KeyF9!
  IF cb_base_f09.Enabled = True THEN
    cb_base_f09.EVENT Clicked()
  END IF
CASE KeyF12!
  IF cb_base_f12.Enabled = True THEN
    cb_base_f12.EVENT Clicked()
  END IF
CASE ELSE
  END CHOOSE
RETURN
  
```

レイアウト key イベント リスト 関数リスト インスタンス変数の宣言

L) open イベント

共通仕様をコメントで記載する。

Script - の open の戻り値のデータ型は long です

w_base open () returns long [pbm_open]

```

V/// Fキー設定
/// 123456789012
/// wf_fkey("11xx11x01xx1")
///
/// Dw一覧の設定
/// idw_n      = dw_list
///
/// 検索処理
/// EVENT base_f02()
RETURN 0
  
```

レイアウト open イベント リスト 関数リスト インスタンス変数の宣言

N) wf_fkey イベント

F キーの使用可否を設定する関数を作成する。

Script Editor - wf_fkey (string as_string) returns (none)

(Functions) wf_fkey (string as_string) returns (none)

アクセス	戻り値のデータ型	関数名
public	(None)	wf_fkey
渡し方	引数データ型	引数名
value	string	as_string

Throws:

```

// F1
IF Mid(as_string, 1, 1) = "1" THEN
    cb_base_f01.Enabled = True
ELSE
    cb_base_f01.Enabled = False
END IF

// F2
IF Mid(as_string, 2, 1) = "1" THEN
    cb_base_f02.Enabled = True
ELSE
    cb_base_f02.Enabled = False
END IF

// F5
IF Mid(as_string, 5, 1) = "1" THEN
    cb_base_f05.Enabled = True
ELSE
    cb_base_f05.Enabled = False
END IF

// F6
IF Mid(as_string, 6, 1) = "1" THEN
    cb_base_f06.Enabled = True
ELSE
    cb_base_f06.Enabled = False
END IF

// F8
IF Mid(as_string, 8, 1) = "1" THEN
    cb_base_f08.Enabled = True
ELSE
    cb_base_f08.Enabled = False
END IF

// F9
IF Mid(as_string, 9, 1) = "1" THEN
    cb_base_f09.Enabled = True
ELSE
    cb_base_f09.Enabled = False
END IF

// F12
IF Mid(as_string, 12, 1) = "1" THEN
    cb_base_f12.Enabled = True
ELSE
    cb_base_f12.Enabled = False
END IF

```

レイアウト \ wf_fkey \ イベント リスト \ 関数リスト \ インスタンス変数の宣言

⑤ データウインドウクラスの作成 (その2)

A) base_key イベント

データウインドウ上でFキーを押下した際の機能を組み込む。

The screenshot shows the Visual Basic IDE with the Properties window and the Script Editor. The Properties window is set to the 'base_key' event of the 'uo_dw_base' control. The event signature is 'base_key (keycode key, unsignedlong keyflags) return'. The Properties window shows the following configuration:

アクセス	戻り値のデータ型	イベント名
public	long	base_key
渡し方	引数データ型	引数名
value	keycode	key
value	unsignedlong	keyflags

The Script Editor shows the following code:

```
// Enterキーの動きを廃止し、Tabキーを発行する
// 次のカラムに移動が可能
IF key = KeyEnter! THEN
    Send( Handle(This), 256, 9, Long(0,0) )
    RETURN 1
END IF

// Fキーを動作させる
IF key = KeyF1! OR &
key = KeyF2! OR &
key = KeyF5! OR &
key = KeyF6! OR &
key = KeyF8! OR &
key = KeyF9! OR &
key = KeyF12! THEN
    iw_parent_window.POST EVENT Key( Key, Keyflags)
END IF

RETURN 0
```

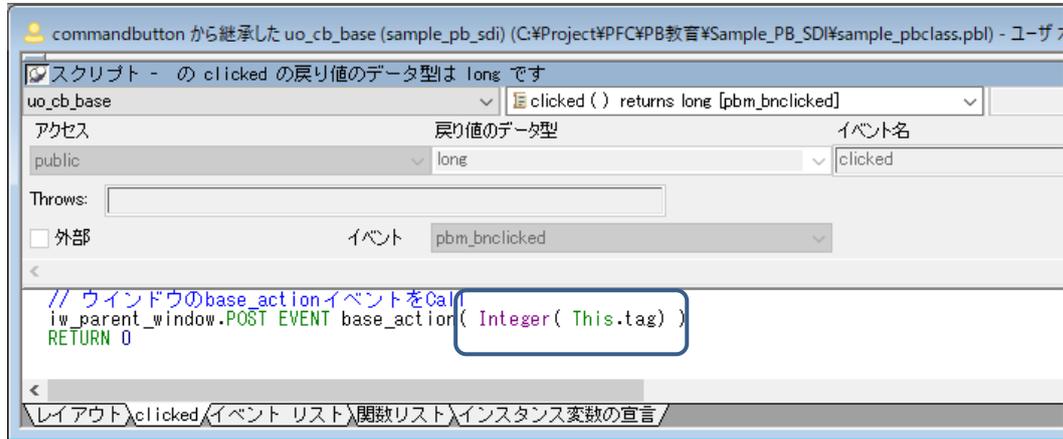
The code is annotated with blue comments and a blue box highlights the F-key handling logic. The bottom status bar shows the current view: 'レイアウト base_key イベント リスト 関数リスト インスタンス変数の宣言'.

(2)クラスの作成 ⑥コマンドボタンクラスの作成 (その2)

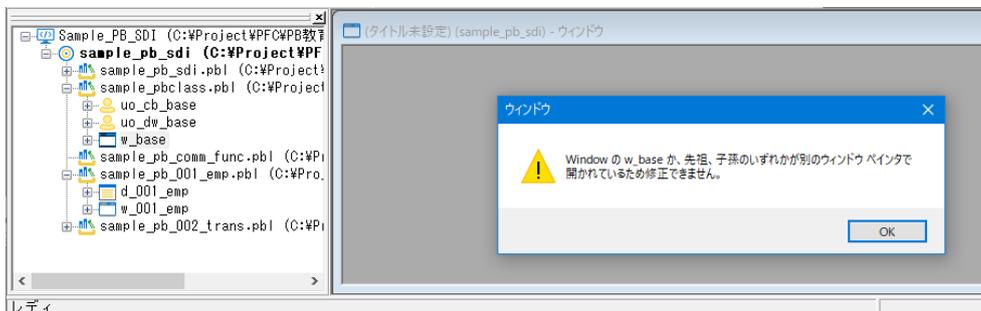
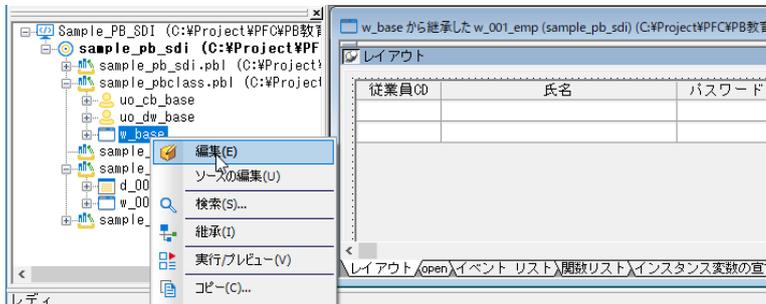
⑥ コマンドボタンクラスの作成 (その2)

A) clicked イベント

ウインドウの **base_action** イベントを Call する。
 引数には、プロパティのタグ値を渡す。



※注意 クラスで作成したスクリプトの参照
 次項より、クラスを用いたプログラミングを行うが
 クラスオブジェクトと継承後のオブジェクトは同時に編集しようとする
 と下記メッセージが表示されるため、どちらかを閉じて編集する。
 継承後のオブジェクト同士は編集可能。



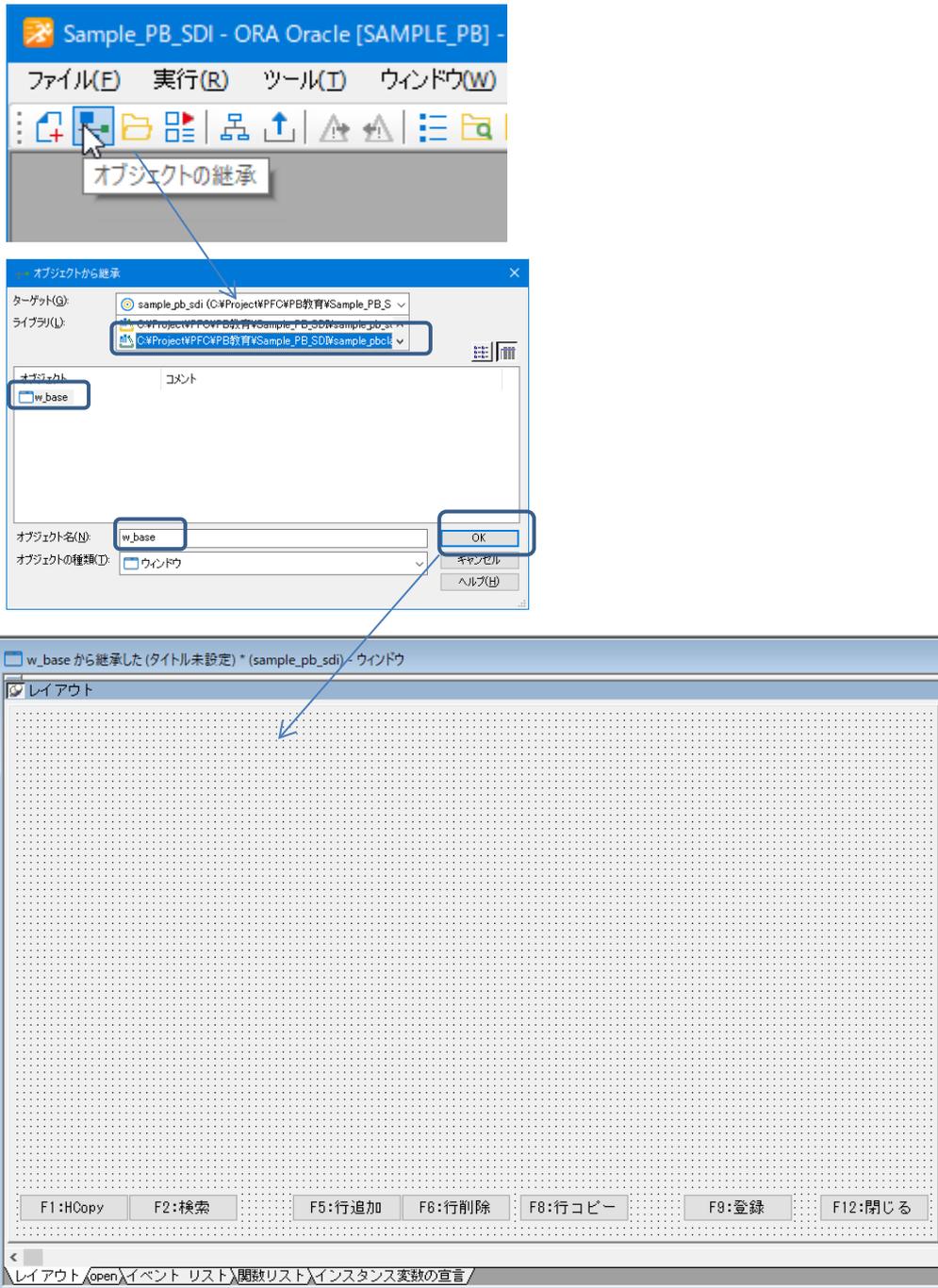
(3) クラスを用いたプログラミング

今回、説明を簡易的にするために、下記の2画面についてクラスを用いて作成する。

- ・従業員マスタメンテナンス
- ・交通費精算入力

① 従業員マスタメンテナンス

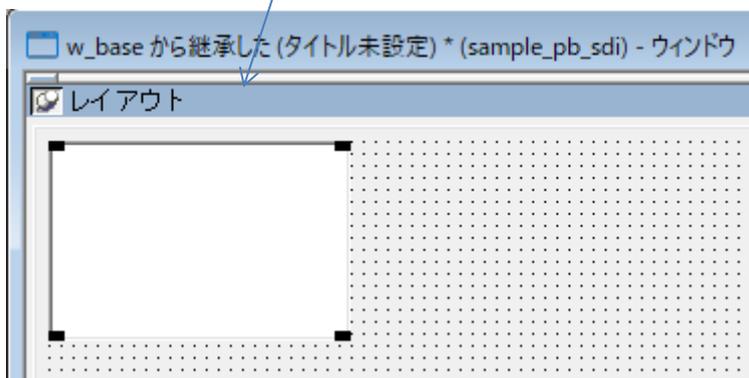
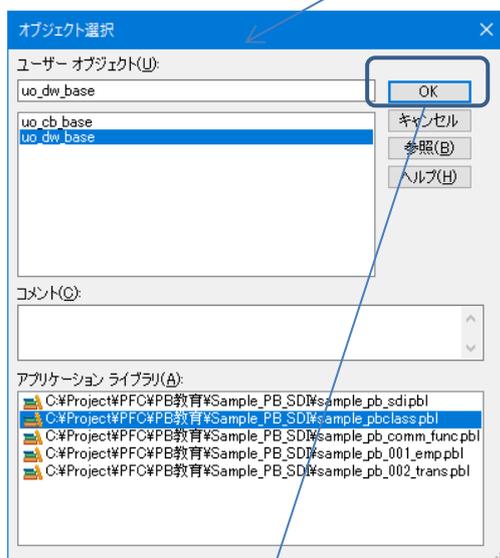
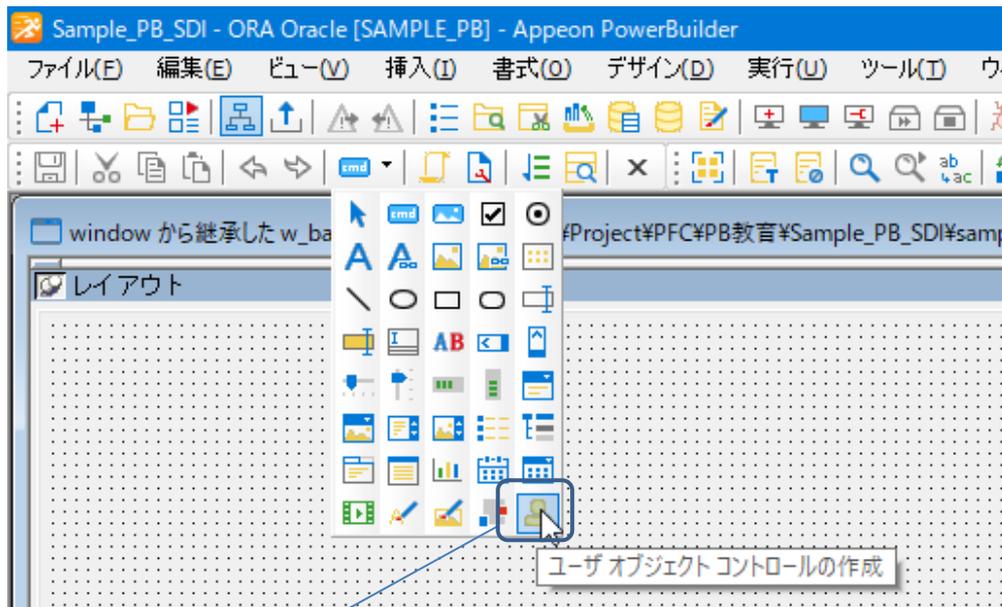
ウインドウ (w_base) を継承する。



1. クラスライブラリ

(3)クラスを用いたプログラミング ①従業員マスタメンテナンス

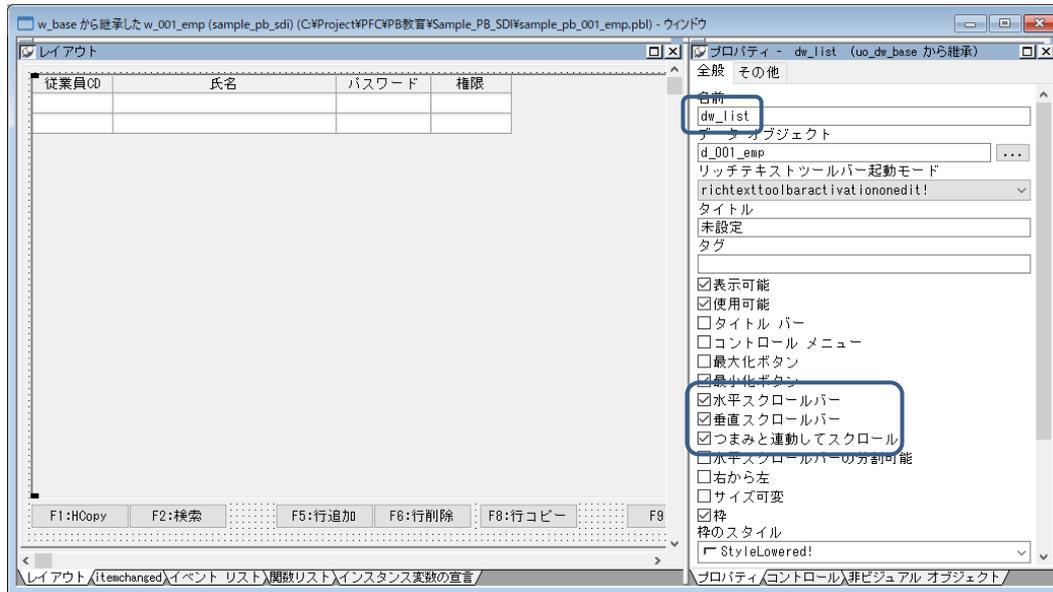
データウインドウを配置する。



1. クラスライブラリ

(3)クラスを用いたプログラミング ①従業員マスタメンテナンス

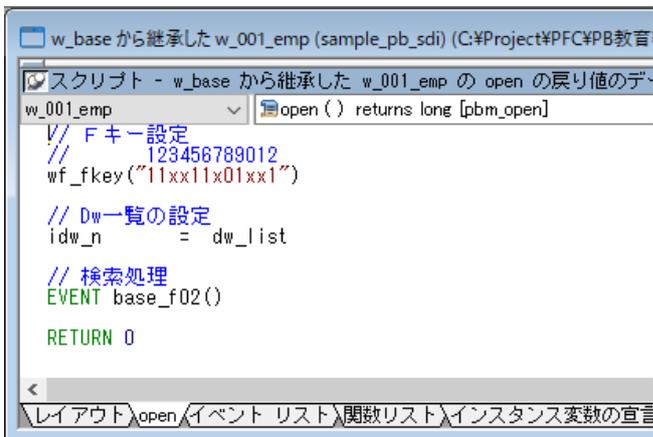
データウインドウを設定する。



スクリプトを記載する。

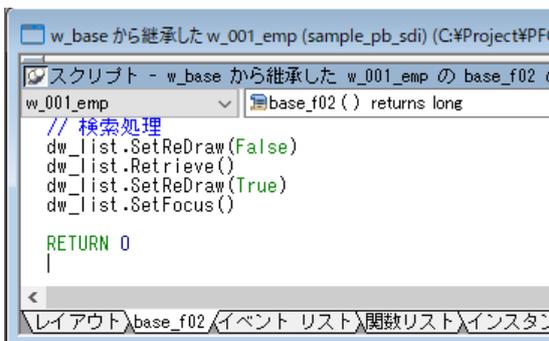
A) open イベント

継承元に記載してコメントを例に記載する。



B) base_f02 イベント

継承元に記載してコメントを例に記載する。



※この時点で、入力チェック、更新処理を除いて動作することが確認できる。

C) base_f09 イベント

コラムチェック、更新処理を記載する。

```

w_base から継承した w_001_emp (sample_pb_sdi) (C:\Project\PC教育\Sample_PB_SDI\sample_pb_001_emp.pbl) - ウィンドウ
スクリプト - w_base から継承した w_001_emp の base_f09 の戻り値のデータ型は long です
w_001_emp base_f09() returns long
Long ll_row
Long ll_find
String ls_emp_cd
String ls_emp_name
String ls_emp_pass

// 入力を確定させる
EVENT base_accept_Text()

// マスタメンテナンスのため、全行チェックする
FOR ll_row = 1 TO dw_list.RowCount()
// 主キーの値を取得
ls_emp_cd = dw_list.Object.emp_cd[ll_row]
IF wf_isnull(ls_emp_cd) THEN
ELSE
// 主キー重複チェック
IF ll_row <> dw_list.RowCount() THEN
// 既存行の下を検索
ll_find = dw_list.Find("emp_cd='" + ls_emp_cd + "'", ll_row + 1, dw_list.RowCount())
// 同一キーが存在する
IF ll_find > 0 THEN
MessageBox("キー重複", "[" + ls_emp_cd + "]" + &
String(ll_row) + "行目と" + &
String(ll_find) + "行目が重複しています。")

dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn(1)
RETURN 0
END IF
END IF
END IF
// 入力変更がない行はスキップする
IF dw_list.GetItemStatus(ll_row, 0, Primary!) = NotModified! OR &
dw_list.GetItemStatus(ll_row, 0, Primary!) = New! THEN
CONTINUE
END IF
// 主キーが未入力
IF wf_isnull(ls_emp_cd) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("emp_cd")
RETURN 0
END IF
// 氏名が未入力
ls_emp_name = dw_list.Object.emp_name[ll_row]
IF wf_isnull(ls_emp_name) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("emp_name")
RETURN 0
END IF
// パスワードが未入力
ls_emp_pass = dw_list.Object.emp_pass[ll_row]
IF wf_isnull(ls_emp_pass) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("emp_pass")
RETURN 0
END IF
NEXT

// 更新する
IF dw_list.Update() = 1 THEN
COMMIT;
ELSE
ROLLBACK;
RETURN 0
END IF

dw_list.Retrieve()
dw_list.SetFocus()

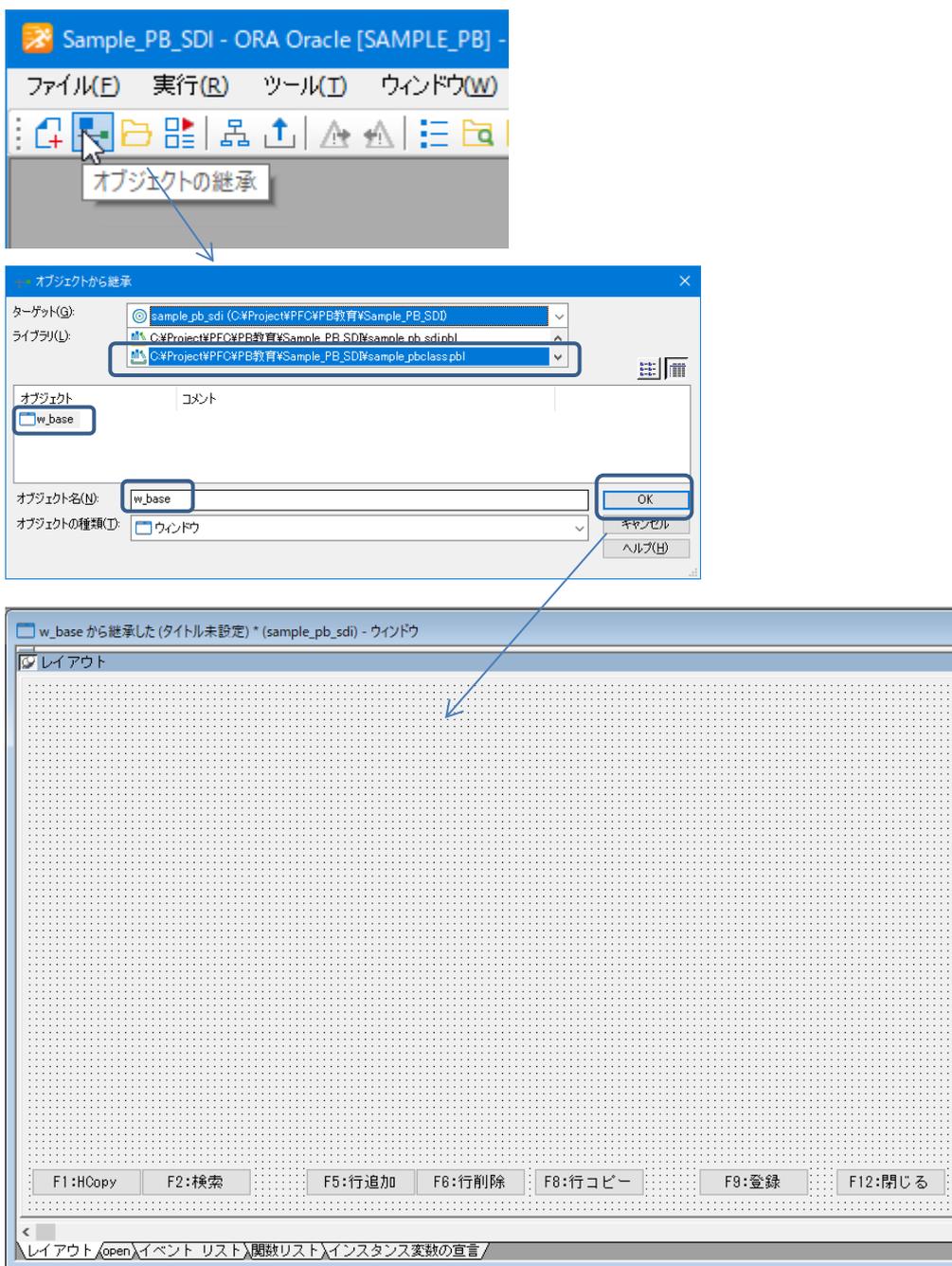
RETURN 0
レイアウト\base_f09\イベント リスト\関数リスト\インスタンス変数の宣言

```

1. クラスライブラリ
(3)クラスを用いたプログラミング ②交通費精算入力

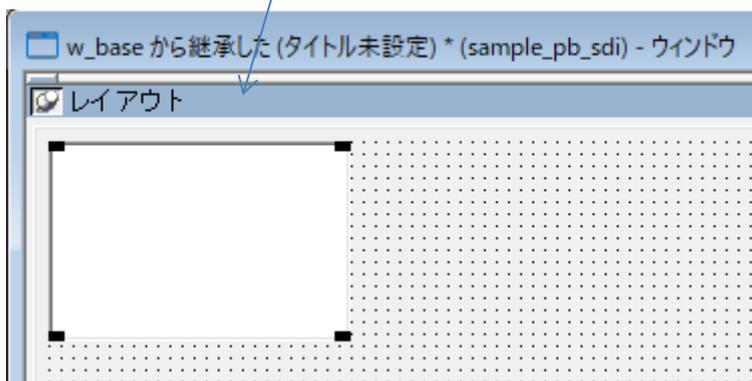
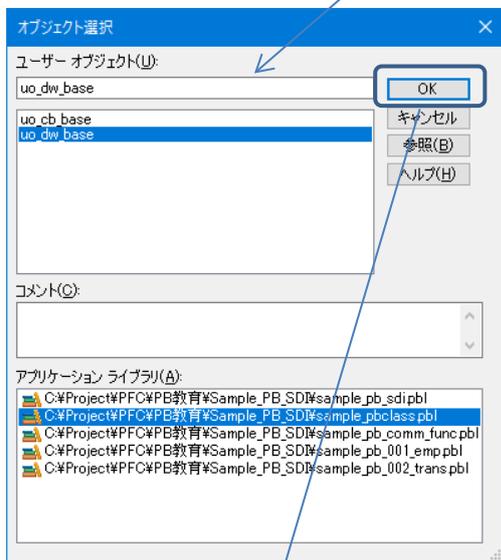
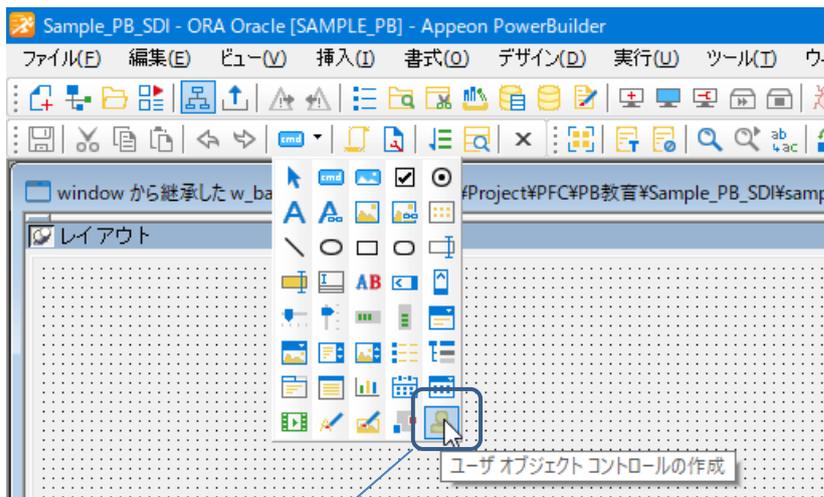
② 交通費精算入力

ウインドウ (w_base) を継承する。



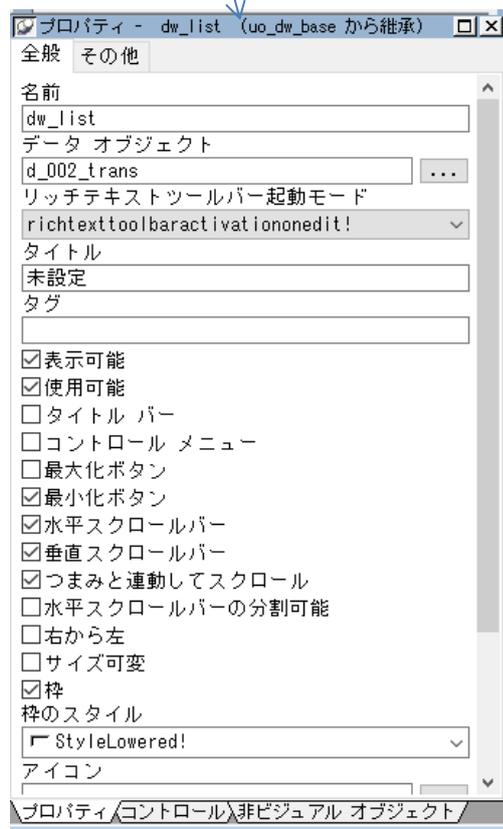
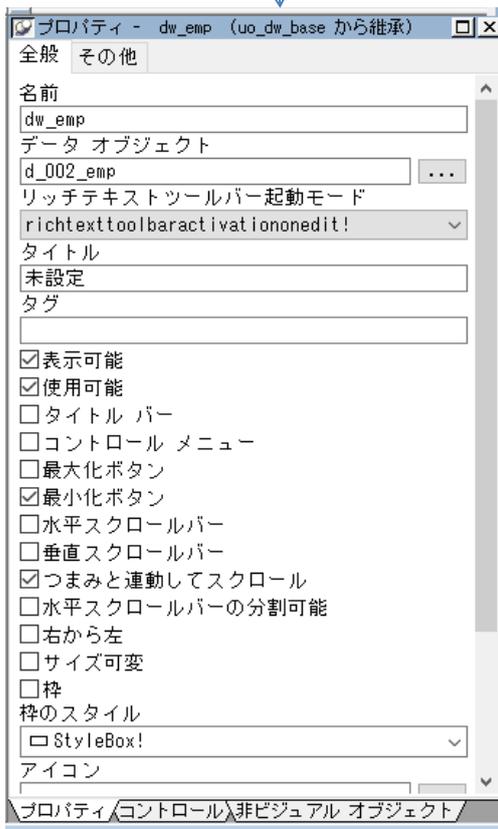
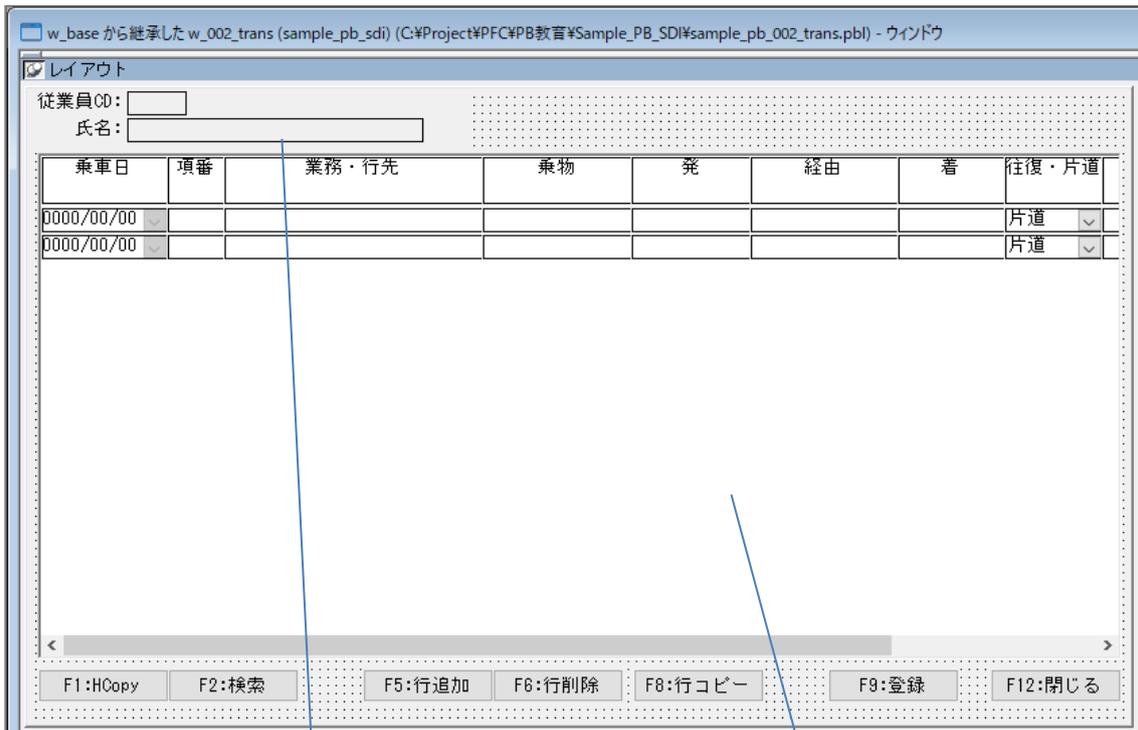
1. クラスライブラリ
(3)クラスを用いたプログラミング ②交通費精算入力

データウインドウを配置する。



1. クラスライブラリ
 (3)クラスを用いたプログラミング ②交通費精算入力

データウインドウを設定する。

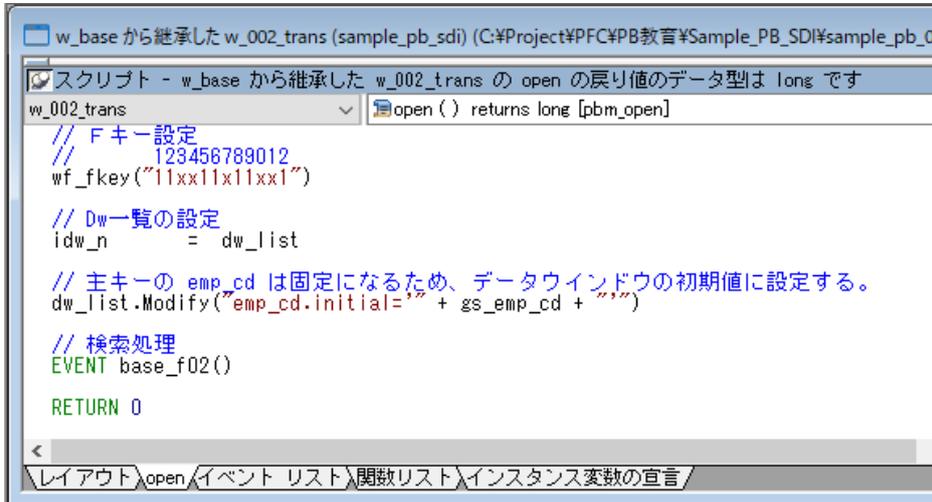


1. クラスライブラリ
(3)クラスを用いたプログラミング ②交通費精算入力

スクリプトを記載する。

A) open イベント

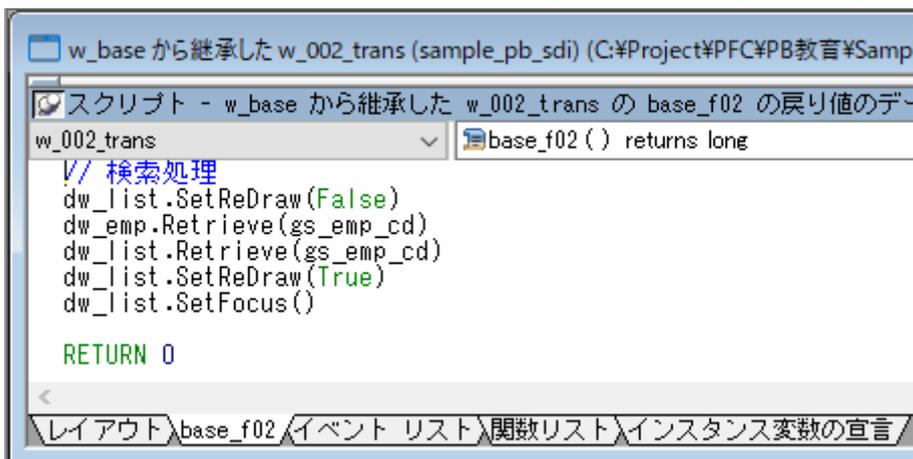
継承元に記載してコメントを例に記載する。
dw_list の初期値設定を行う。



```
w_base から継承した w_002_trans (sample_pb_sdi) (C:\Project\PF\教育\Sample_PB_SDI\sample_pb_0
[ ] スクリプト - w_base から継承した w_002_trans の open の戻り値のデータ型は long です
w_002_trans open () returns long [pbm_open]
// Fキー設定
// 123456789012
wf_fkey("11xx11x11xx1")
// Dw一覧の設定
idw_n = dw_list
// 主キーの emp_cd は固定になるため、データウィンドウの初期値に設定する。
dw_list.Modify("emp_cd.initial=" + gs_emp_cd + ",")
// 検索処理
EVENT base_f02()
RETURN 0
<
レイアウト\open\イベント リスト\関数リスト\インスタンス変数の宣言/
```

B) base_f02 イベント

継承元に記載してコメントを例に記載する。



```
w_base から継承した w_002_trans (sample_pb_sdi) (C:\Project\PF\教育\Sample_PB_SDI\sample_pb_0
[ ] スクリプト - w_base から継承した w_002_trans の base_f02 の戻り値のデータ型は long です
w_002_trans base_f02 () returns long
// 検索処理
dw_list.SetReDraw(False)
dw_emp.Retrieve(gs_emp_cd)
dw_list.Retrieve(gs_emp_cd)
dw_list.SetReDraw(True)
dw_list.SetFocus()
RETURN 0
<
レイアウト\base_f02\イベント リスト\関数リスト\インスタンス変数の宣言/
```

C) base_f08 イベント

行コピーのスク립トを記載する。

```

w_base から継承した w_002_trans (sample_pb_sdi) (C:\Project\PFC\PB教育\Sample_P...
スク립ト - w_base から継承した w_002_trans の base_f08 の戻り値のデータ型
w_002_trans  base_f08 ( ) returns long
Long         ll_row
DateTime     ldt_null
Dec          ldc_null

SetNull(ldt_null)
SetNull(ldt_null)

// 最終行 (追加行) を指定
ll_row = dw_list.RowCount()
// コピー先のデータを「新規」イメージにする
dw_list.Object.trans_ymd[ll_row] = ldt_null
dw_list.Object.trans_seq[ll_row] = ldc_null
dw_list.Object.request[ll_row]   = "0"
dw_list.Object.settle[ll_row]    = "0"

// 新規行と同じステータスにする
dw_list.SetItemStatus(ll_row, 0, Primary!, NewModified!)
dw_list.SetItemStatus(ll_row, 0, Primary!, NotModified!)

RETURN 0

```

コピーした行のステータスを変更する必要がある。

クラスで実施している RowsCopy の結果、コピーされた行のステータスは NewModify! になり、SQL Insert の対象になる。

行コピー後に何も変更なく画面を閉じたいときもあり、その都度、終了確認メッセージが表示されるのは煩わしいため、ステータスを New! の状態にする。

※この時点で、入力チェック、更新処理を除いて動作することが確認できる。

1. クラスライブラリ
(3)クラスを用いたプログラミング ②交通費精算入力

D) base_f09 イベント
カラムチェック、更新処理を記載する。

```
w_base から継承した w_002_trans (sample_pb_scd) (C:\Project\PF\PC教育\Sample_PB_SDI\Sample_pb_002_trans.pbl) - ウィンドウ
スク립ト - w_base から継承した w_002_trans の base_f09 の戻り値のデータ型は long です
w_002_trans base_f09 () returns long
Long ll_row
Long ll_find
DateTime ldt_trans_ymd
Long ll_trans_seq
String ls_string
Dec ldc_decimal

// 入力を確定させる
EVENT base_accept_text()

// 全行チェックする
FOR ll_row = 1 TO dw_list.RowCount()
// 主キー重複チェック
ldt_trans_ymd = dw_list.Object.trans_ymd[ll_row]
ll_trans_seq = dw_list.Object.trans_seq[ll_row]
// 項番0はNull扱する
IF ll_trans_seq = 0 THEN
SetNull(ll_trans_seq)
END IF
IF wf_isnull(ldt_trans_ymd) AND wf_isnull(ll_trans_seq) THEN
ELSE
IF ll_row <> dw_list.RowCount() THEN
// 既存行の下を検索
ll_find = dw_list.Find( "trans_ymd=DateTime(Date(~~~~ + String(ldt_trans_ymd, "yyyy/mm/dd") + "~~~~),Time("00:00:00"))" &
" AND trans_seq=" + String(ll_trans_seq), ll_row + 1, dw_list.RowCount())
// 同一キーが存在する
IF ll_find > 0 THEN
MessageBox("キー重複", String(ldt_trans_ymd, "yyyy/mm/dd") + "[" + String(ll_trans_seq) + "]" + &
String(ll_row) + "行目と" + &
String(ll_find) + "行目が重複しています。")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn(1)
RETURN 0
END IF
END IF
END IF
// 入力変更がない行はスキップする
IF dw_list.GetItemStatus(ll_row, 0, Primary!) = NotModified! OR &
dw_list.GetItemStatus(ll_row, 0, Primary!) = New! THEN
CONTINUE
END IF
// 乗車日のチェック
ldt_trans_ymd = dw_list.Object.trans_ymd[ll_row]
IF wf_isnull(ldt_trans_ymd) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("trans_ymd")
RETURN 0
END IF
// 項番のチェック
ll_trans_seq = dw_list.Object.trans_seq[ll_row]
IF wf_isnull(ll_trans_seq) THEN
MessageBox("入力エラー", "必須入力です (1,2,3...の順で入力)")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("trans_seq")
RETURN 0
END IF
// 業務・行先が未入力
ls_string = dw_list.Object.purpose[ll_row]
IF wf_isnull(ls_string) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("purpose")
RETURN 0
END IF
// 発が未入力
ls_string = dw_list.Object.trans_from[ll_row]
IF wf_isnull(ls_string) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("trans_from")
RETURN 0
END IF
// 着が未入力
ls_string = dw_list.Object.trans_to[ll_row]
IF wf_isnull(ls_string) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("trans_to")
RETURN 0
END IF
// 運賃が未入力
ldc_decimal = dw_list.Object.charge[ll_row]
IF ldc_decimal = 0 OR &
wf_isnull(ldc_decimal) THEN
MessageBox("入力エラー", "必須入力です")
dw_list.ScrollToRow(ll_row)
dw_list.SetFocus()
dw_list.SetColumn("charge")
RETURN 0
END IF
NEXT
// 更新する
IF dw_list.Update() = 1 THEN
COMMIT;
ELSE
ROLLBACK;
RETURN 0
END IF
dw_list.Retrieve(gs_emp_cd)
dw_list.SetFocus()
RETURN 0
<
レイアウト base_f09 イベント リスト 閉鎖リスト インスタンス実行の宣言
```

2. データウインドウ有効活用

データウインドウは様々な機能を有しており、システム開発に有効な機能もあれば、統一ルールに従った開発手法では不向きな機能もある。

ここでは、よりよいシステムの提供のために、活用すべき事項と、別途方式をとった方がよい事項について例を挙げて記載する。

(1) 有効な機能

① 更新特性の活用

データウインドウペインタでデータウインドウを作成する際に、様々なケースを想定した動作を、更新特性を用いて実現することができる。

A) 排他制御（楽観的排他制御：ロックしない整合性検証）

行ロックなどの悲観的排他制御を用いない、更新可能カラムの整合性を検証する排他制御が可能である。

（タイムスタンプ方式に似ている）

データの更新（Update 文）を行う際に、対象のデータが他のユーザによって更新されていないかどうか判定するためには、下図の設定を行う。★

この設定により、更新可能なカラムで **WHERE** 句が生成され、整合性を確認することができる。

整合性がとれないときは排他エラーとし、ユーザには更新処理を再度実施するように促す。

B) 更新処理の高速化

行ロックなどの悲観的排他制御により整合性が保障されているときは、下図のように設定を行う。★

この設定により、主キーのみで WHERE 句が生成されるため、パフォーマンスの改善となる。

C) キーカラムの更新

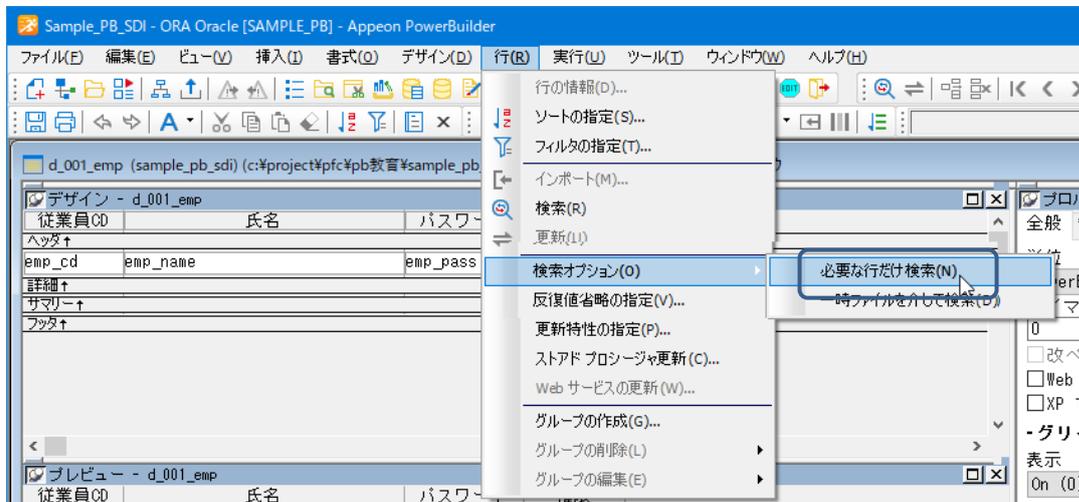
インタフェースの作りにもよるが、「Delete 文の後 Insert 文を使用」を推奨する。★

DB ベンダー製品によるが、主キーの変更は DB 再編成がされるため、削除してから挿入する方がパフォーマンスの向上を期待できる。

2. データウインドウ有効活用 (1) 有効な機能

② 必要データの検索

データ件数が多いとき、必要とするデータを先頭から表示されることができる。

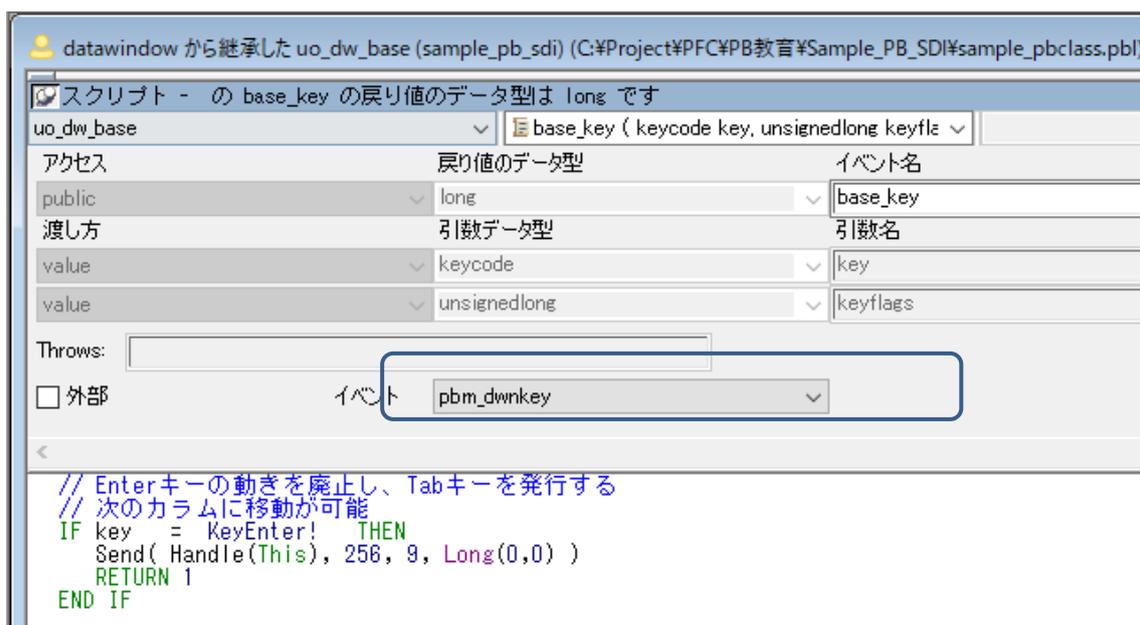


注意点として、SQL 文に ORDER BY 句の記載がある場合、DB 側で全ての検索処理が終わってから表示が始まるので、これを意識した開発が必要である。

③ Enter キー押下による Tab キー制御

既に入門コースでも実施してきたが、Windows では Enter キーの押下は次の行に移動するのが標準であった。

しかし、Enter キーは旧来から次のフィールドに移動するのが一般的であり、キー操作も簡易であることから、Enter キー押下の動作を Tab キーに置き換える。



④ 行・カラムのステータスの活用

データウインドウの行またはカラムは、以下のステータスで管理されている。

Dw.GetItemStatus 関数、Dw.SetItemStatus 関数を用いる。

行、ステータス	意味
New!	新規行、入力変更なし
NewModified!	新規行、入力変更あり
NotModified!	検索行、入力変更なし
DataModified!	検索行、入力変更あり

カラム、ステータス	意味
NotModified!	入力変更なし
DataModified!	入力変更あり

A) コピーした行を New! (新規、変更なし) にしたい

本コースで実施しているが、コピーされた行は SQL Insert の対象となり、入力変更ありの状態になっている。

コピーしただけで修正していないのであれば、変更なしの状態にしたい。

本コースで画面を閉じる際に、データウインドウの変更有無を判定しているため、その対象から外したい。

⑤ カラムデータのチェック方法

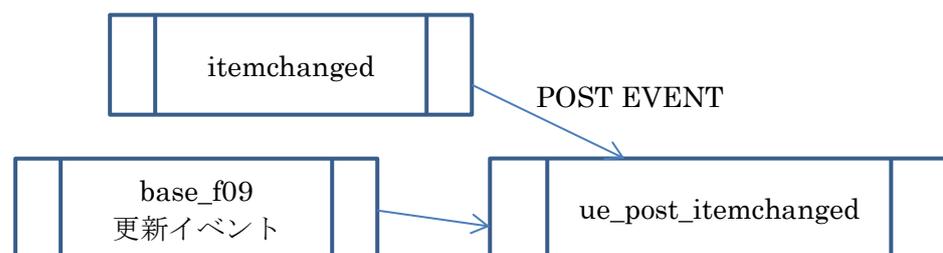
本コースでは登録前に入力項目のチェックを行っているが、入力操作中にエラーチェックを実施したいケースがある。

入力変更のタイミングでは ItemChanged イベントが動作する。

但し、ItemChanged イベント内で GetItemXXXX 関数を用いて値を取得しても、入力変更前の値が取得され、入力チェックのプログラミングがスムーズにいかない。(GetText 関数で対応可能だが)

GetItemXXXX 関数で新しい値が取得できるタイミングは、ItemChanged イベントが終わってからになるため、この特性を生かし、別途ユーザイベント (ue_post_Itemchanged) を用意して、ItemChanged イベントから POST Call し、このイベント内でカラム値のチェック処理を記載するシンプルな方式を用いる。

また、更新処理からも Call できる仕組みを構築し、コードの重複を避けることもできる。



2. データウインドウ有効活用 (1)有効な機能

⑥ データウインドウのテスト技法

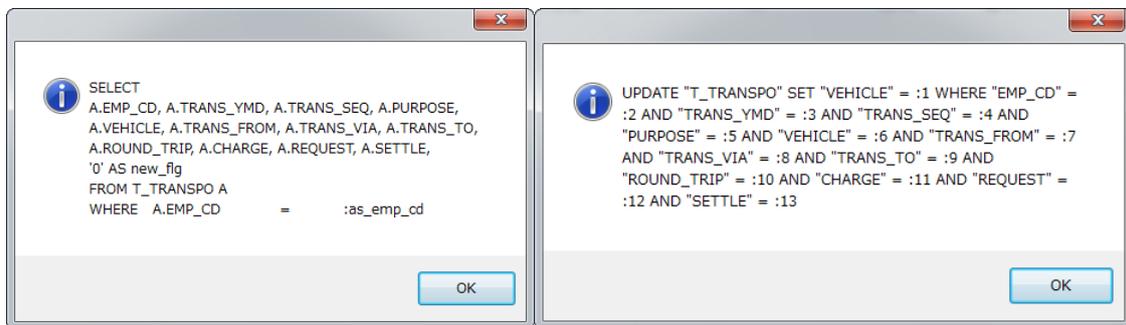
データウインドウがDBMSに発行するSQL文を確認することができる。(SQLPreview イベント)

引数の `sqlsyntax` にて SQL 構文を確認することができる。

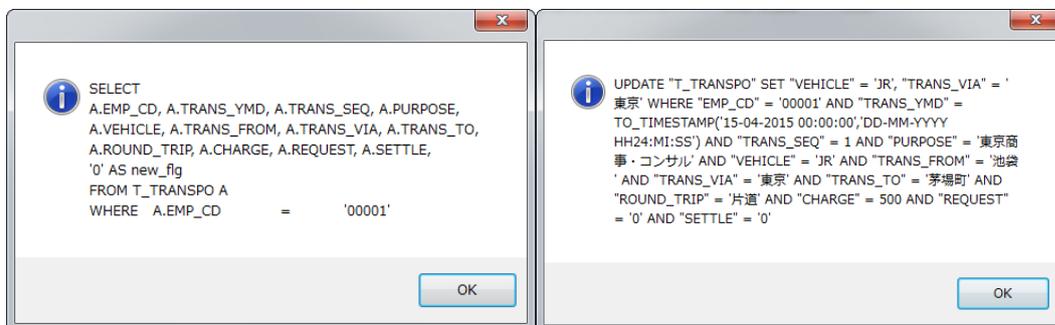
但し、DB 接続の際にバインド抑制が OFF (デフォルト) のとき、SQL 文中の値がバインド引数で表示されるため、値の確認をとることができない。

テストにより値の確認が必要な場合は、バインド抑制を ON にする。また、常に SQL 構文をログに記録する場合も ON の方が望ましい。

【バインド抑制 OFF】★



【バインド抑制 ON】★

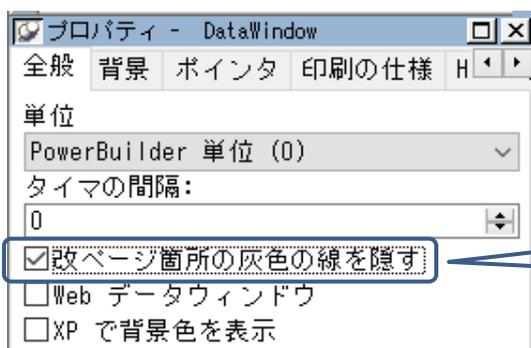


```
// DB接続設定
// 一般的にアプリケーション起動時にDB接続を行う。
SQLCA.DBMS      = "ORA Oracle"
SQLCA.ServerName = "orcl"
SQLCA.LogId     = "SAMPLE_PB125"
SQLCA.LogPass   = "SAMPLE_PB125"
SQLCA.AutoCommit = False
//SQLCA.DBParm  = "CommitOnDisconnect='No',NLS_Charset='Local'"
SQLCA.DBParm    = "CommitOnDisconnect='No',NLS_Charset='Local',DisableBind=1"
```

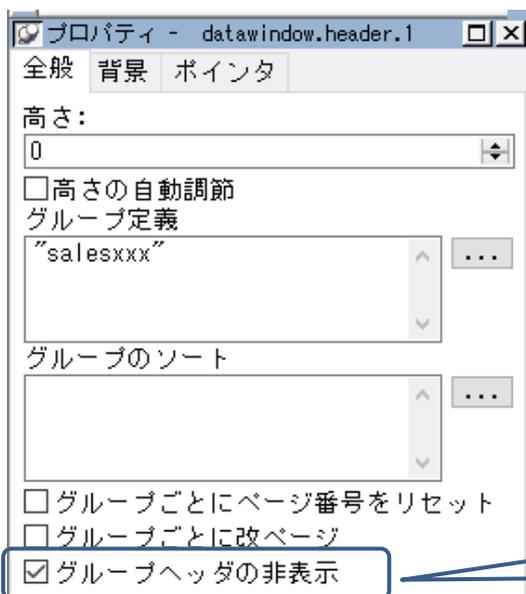
2. データウインドウ有効活用 (1)有効な機能

- ⑦ グループデザインでのデータウインドウで縦方向の空欄を消す
グループデザインされたデータウインドウを画面で表示し、縦方向にスクロールした際に空欄が生まれる現象がある。

これを回避するには下記設定を行う。



データウインドウプロパティ
デフォルト OFF を ON にする。



グループヘッダプロパティ
デフォルト OFF を ON にする。

(2) 開発で推奨する方式

⑧ DBError イベント

データウインドウの検索 (Retrieve 関数) や更新 (Update 関数) を行い、DB サーバからエラーが返された際に起動されるイベントであり、標準のままでは Appion PowerBuilder 2017 が自動的にエラー表示を行う便利な仕組みであるが、ユーザに親切的なインタフェースを提供するために、ひと工夫入れることを推奨する。

既に入門コースでも実施してきたが、イベントの戻り値を用いて制御を変更する。

下表、Return 2 と 3 はシステム構築上不向きなため検討対象から除外する。

戻り値	エラーメッセージ	トランザクションの DBError
Return 0	表示する	発生させる
Return 1	表示しない	発生させる
Return 2	表示する	無視する
Return 3	表示しない	無視する

ここでは、Return 1 で復帰し、ユーザに理解しやすい方式をとる。

このイベントでは、DB サーバからのメッセージがそのまま表示されるため、ユーザにとっても理解し辛い。

DB ベンダーごとに DB エラー番号を持っているため、ある程度は事前に調査を行い、わかり易い表示内容にした方がよい。

また、メッセージを表示するだけでなく、ログ出力機能を組み込むこともできる。

イベント引数からは、エラー起因のデータウインドウの行も取得できるため、早期の問題解決を考慮した設計が可能である。

2. データウインドウ有効活用 (2)改良を推奨する方式

⑨ ItemError イベント

データウインドウで入力を行った際に、データ型に見合わない入力がされたときに起動されるイベントであり、標準のままでは **Appeon PowerBuilder 2017** が自動的にエラー処理を行い、わかり辛いメッセージが表示されるため、ひと工夫入れることを推奨する。

既に入門コースでも実施してきたが、イベントの戻り値を用いて制御を変更する。

戻り値	エラーメッセージ	データの受入
Return 0	表示する	受入れない
Return 1	表示しない	受入れない
Return 2	表示しない	受入れる
Return 3	表示しない	受入れないがフォーカスは移動

スムーズな入力操作を考慮した場合、**Return 1** が最も適切と考えられる。

Appeon PowerBuilder 2017 側からのエラーメッセージを止め、データ型に見合わない入力があるときはフォーカス移動できない。

Appeon PowerBuilder 2017 標準機能に関連する、入力条件則とエラーメッセージの設定が可能であるが、**ItemError** イベントを標準機能で用いないことを前提とした場合は、利用してはならない。

設定の仕様が難解であるのと、メンテナンス性が悪いため推奨しない。

3. まとめ

本書にて、Appeon PowerBuilder 2017 クラスライブラリとデータウインドウの便利な利用法を解説してきた。

クラスライブラリの設計は、ただ単に継承利用が便利だけでなく、開発の規則性を持たせるなど基本コンセプトを持たせることが重要である。

スクリプトを記載する箇所を限定させ、サンプルスクリプトをコメントで記載しておくことで、開発者への規約説明が最小限で済み、スクリプトの規則性が成り立つ。

その結果、ビジネスロジックの構築に注力でき、不具合発生率の削減、問題点の早期解決に繋がっていく。

データウインドウは、DBMS とのデータのやりとりが便利であるのは当然であり、それよりもデータウインドウが持つイベントと関数をうまく活用することが大事である。

最後に、これまでの開発経験から参考として述べるが、データウインドウの SQL 構文は手書きでデザインし、構文が複雑になるケースもあるが、1回の Retrieve で完結させることで、ウインドウのスクリプトが少なく済む、問題点の切り分けを明確にすることができる。

以上