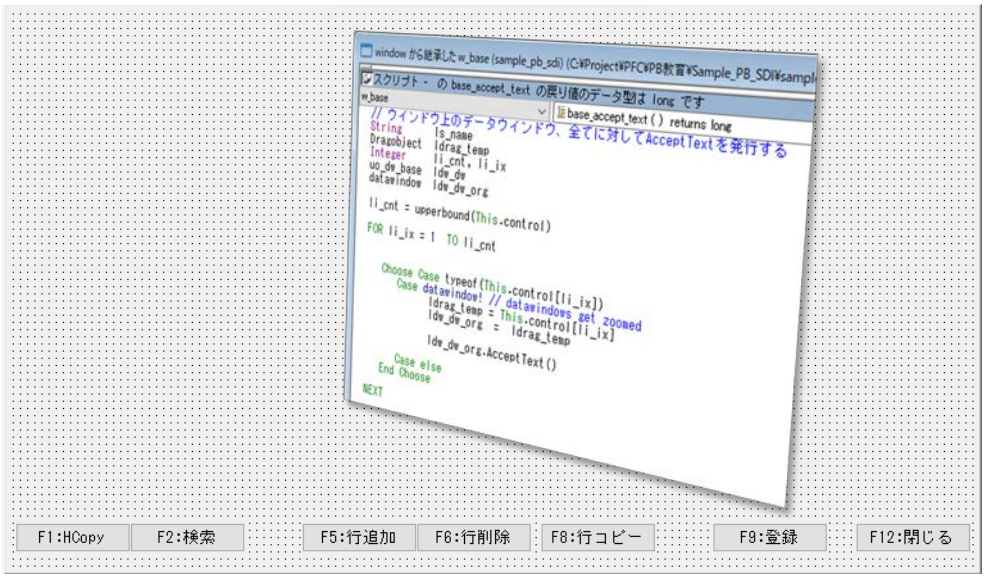
Appeon PowerBuilder 2017 クラス DW 機能



パワーフューチャー株式会社

目次

はじめに

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

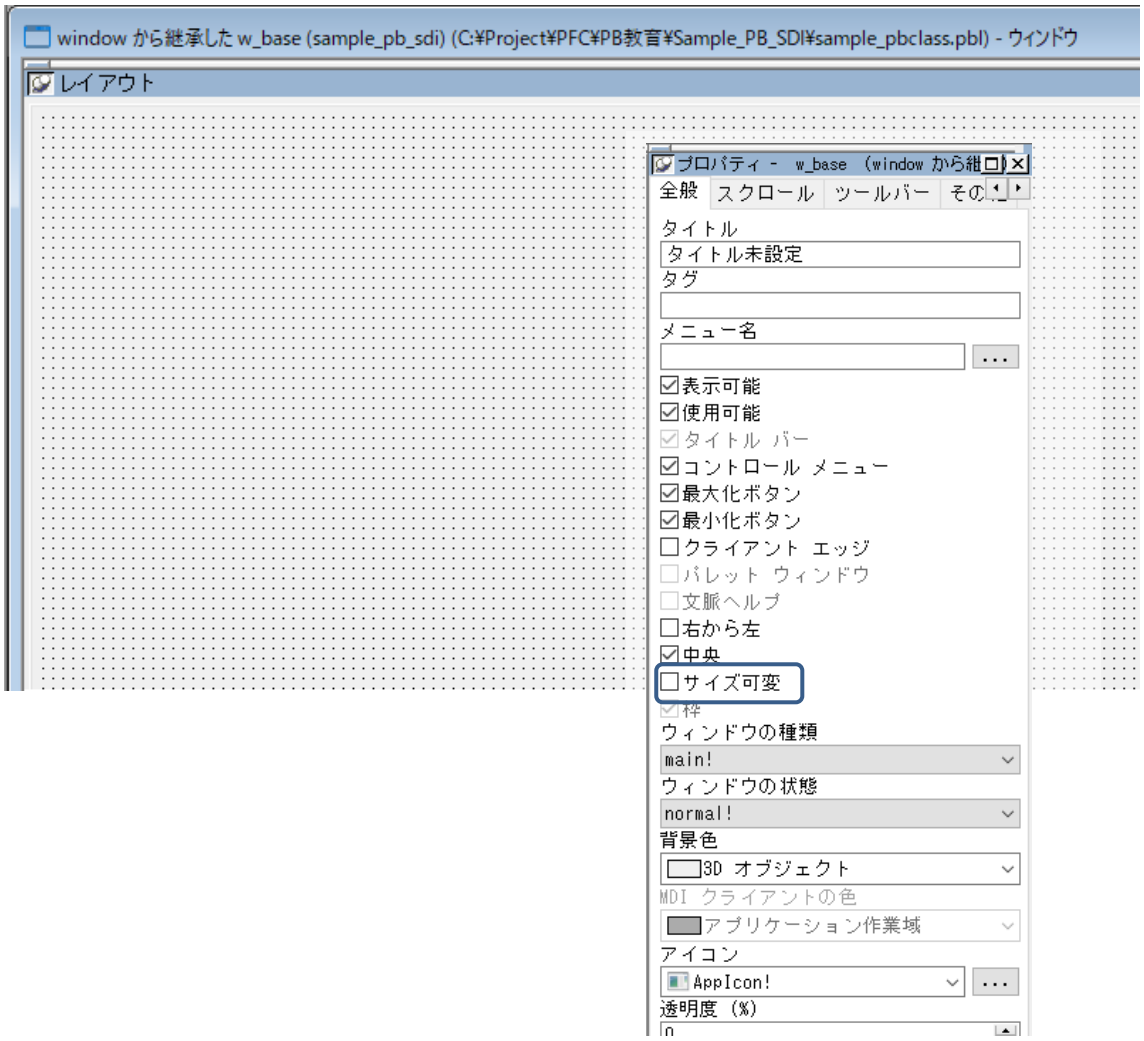
あらかじめ定義されているクラスからデータや処理手続きを受け継ぐことができ、下記の利点がある。

・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

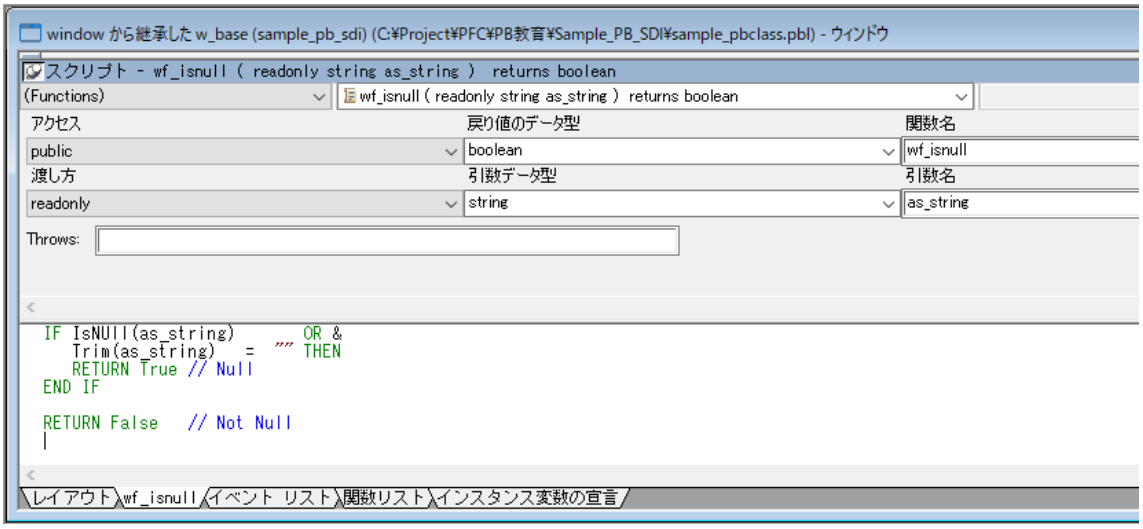
入門コースで作成したアプリケーションにクラスを用いて再デザインする。
Sample_pbclass.pbl を用意する。

①　ウインドウクラスの作成（その１）
w_base で保存する。

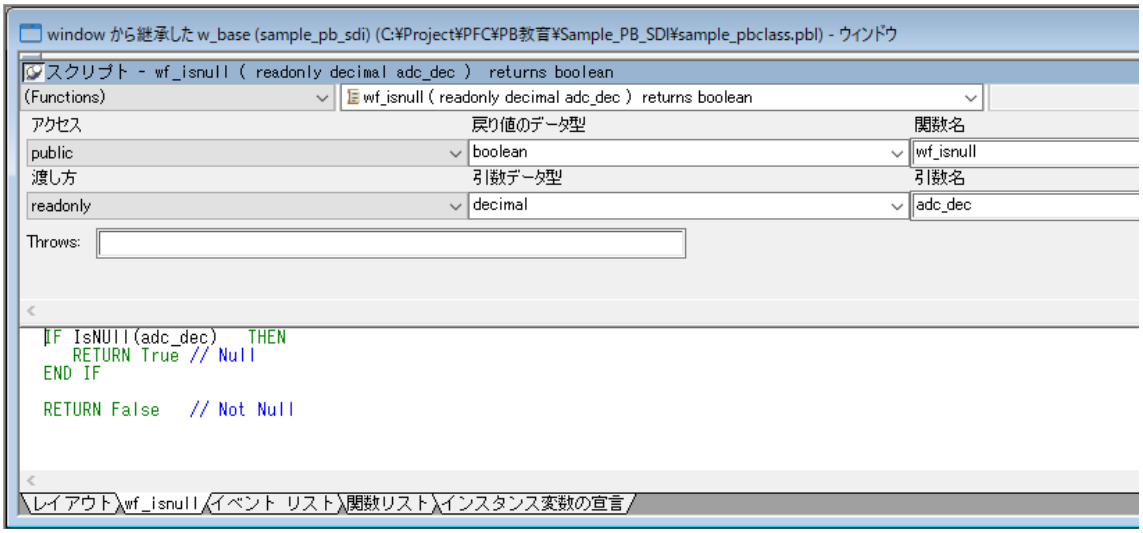


ウインドウ関数を作成する。

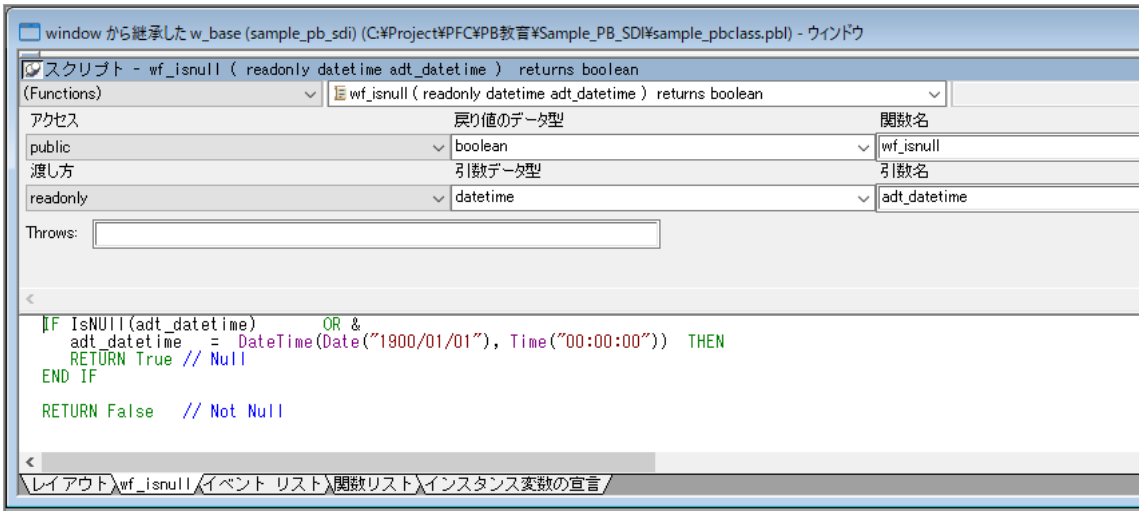
A）wf_isnulll(文字)



B）wf_isnulll(数値)



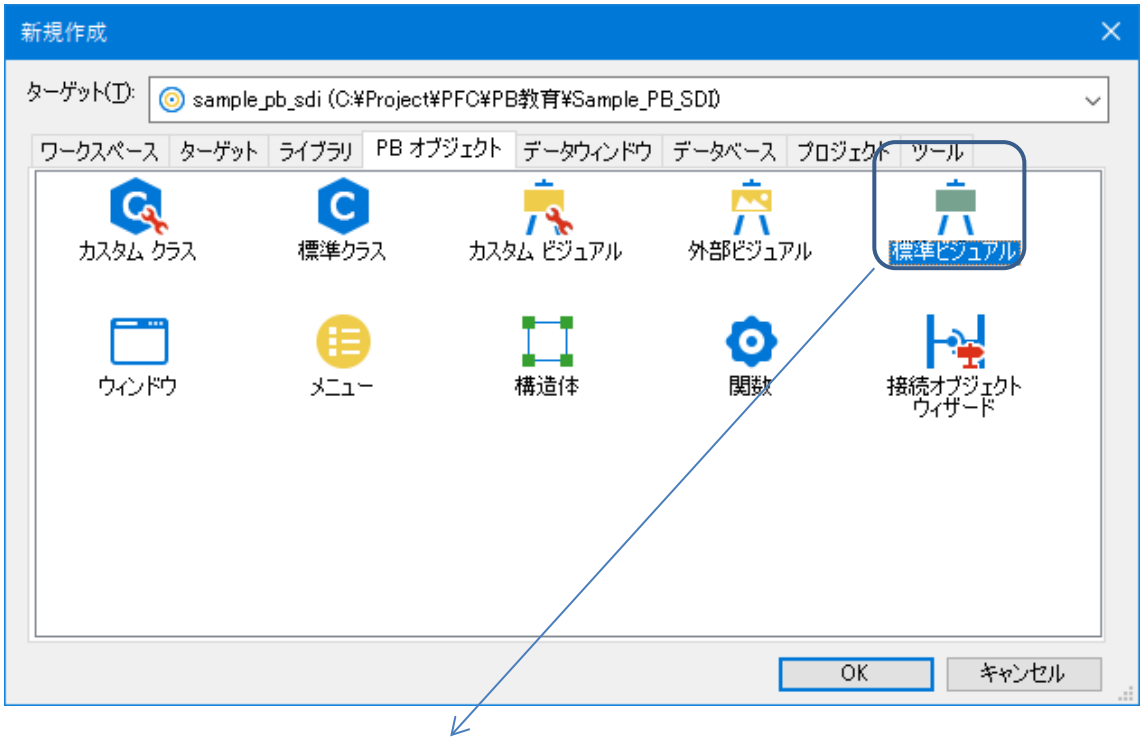
C）wf_isnulll(日付)

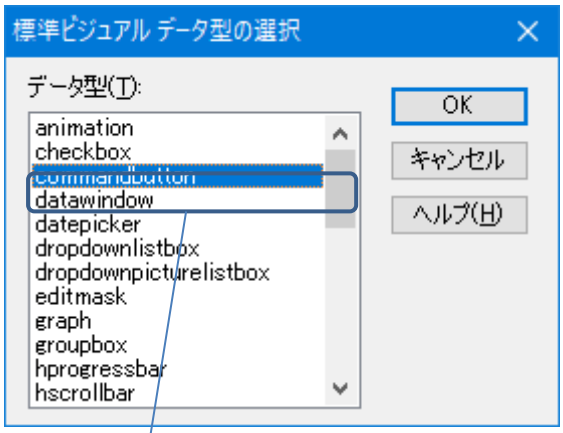


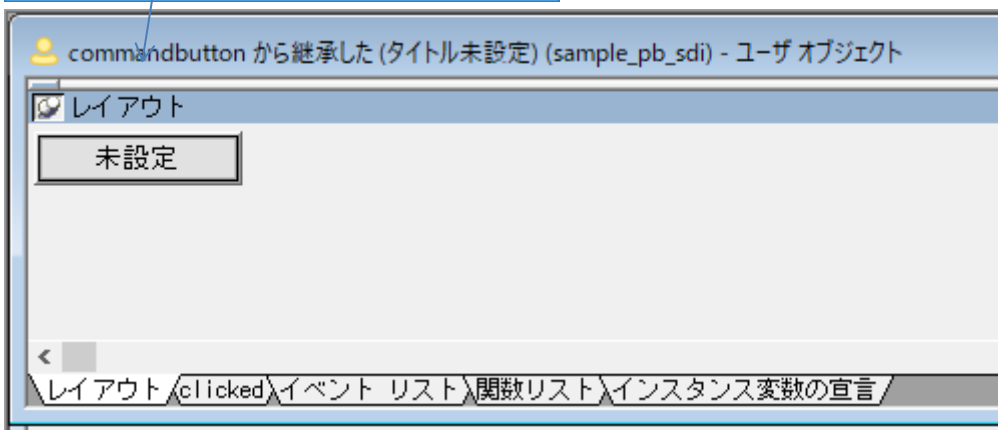
② コマンドボタンクラスの作成（その１）

ユーザオブジェクトで作成する。

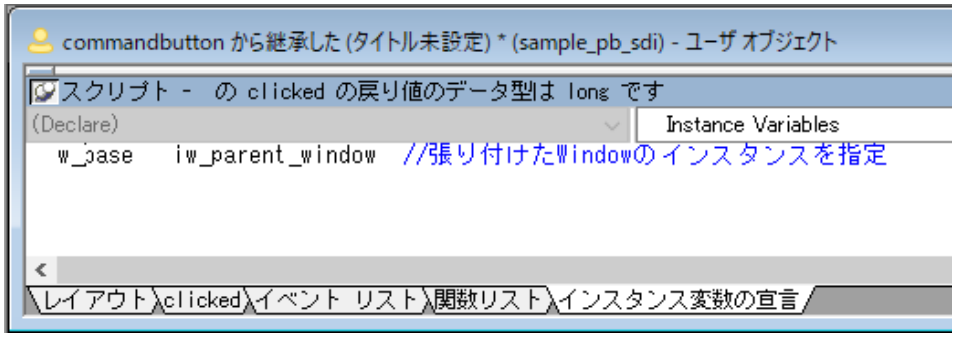
標準ビジュアルでデザインする。スクリプト記載後、保存する。





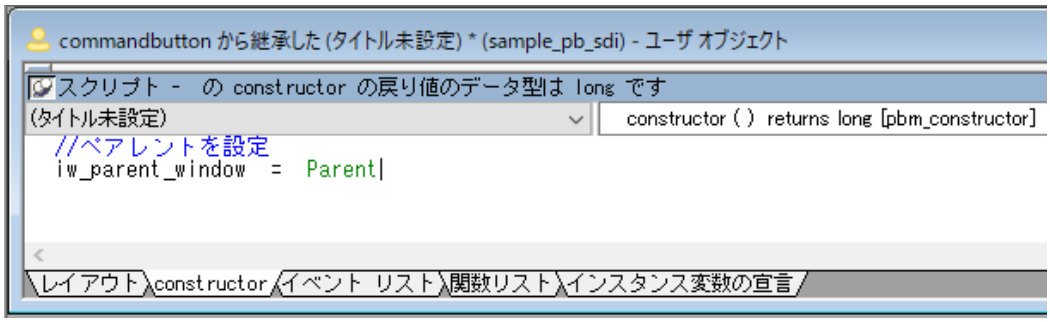


インスタンス変数にウインドウインスタンスを指定する。



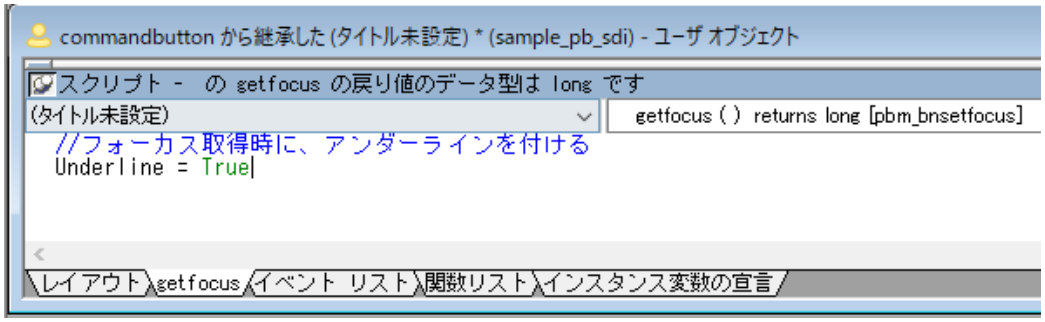
A）constructor イベント

ペアレントを設定する。



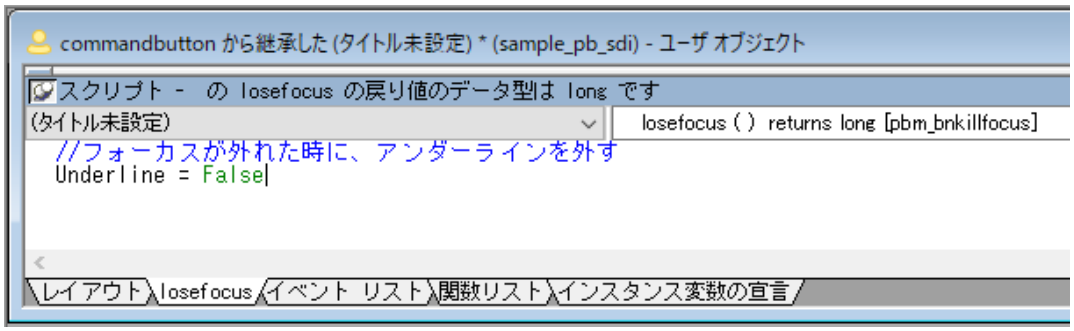
B）getfocus イベント

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。

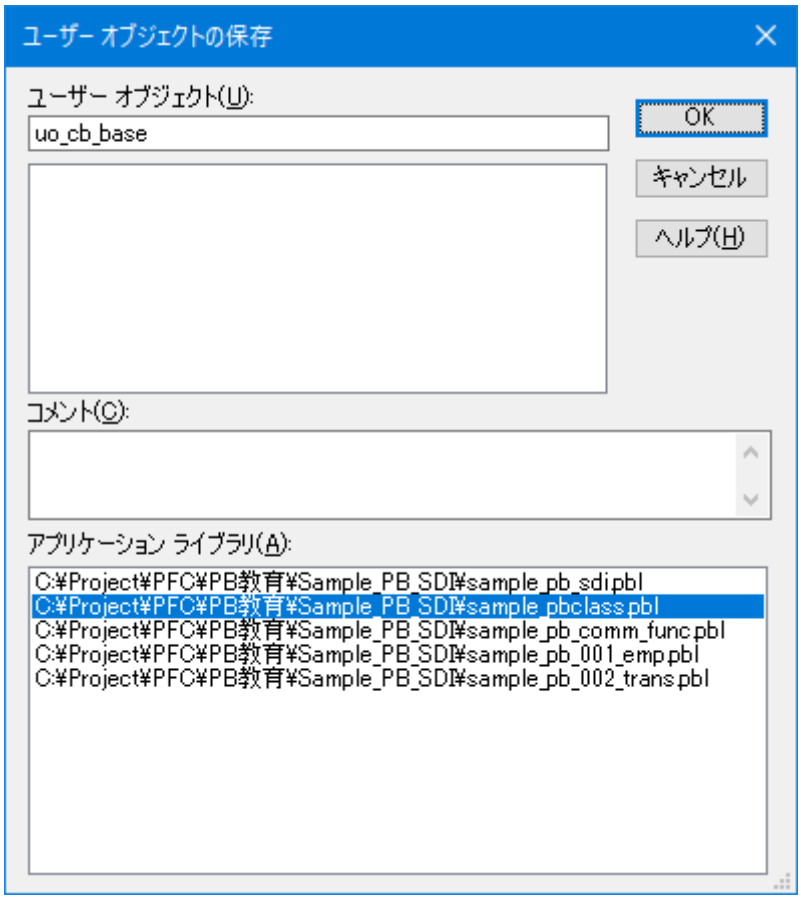


C）losefocus イベント

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。

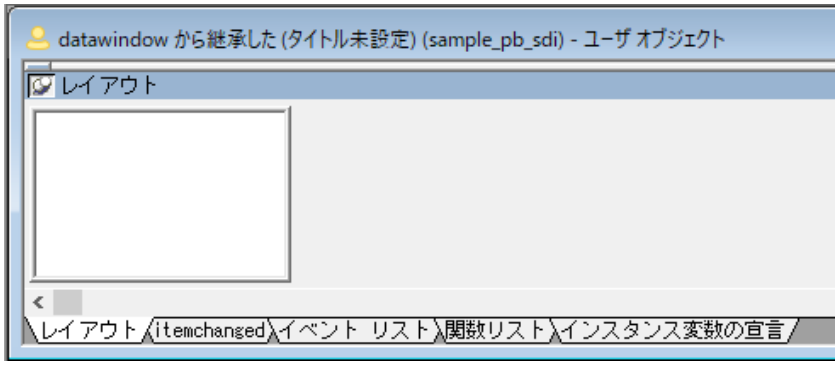


名前を付けて保存する。(uo_cb_base)

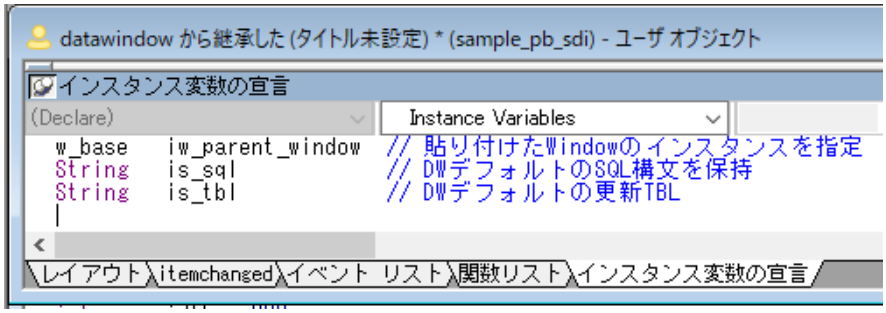


③　データウインドウクラスの作成（その１）

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。

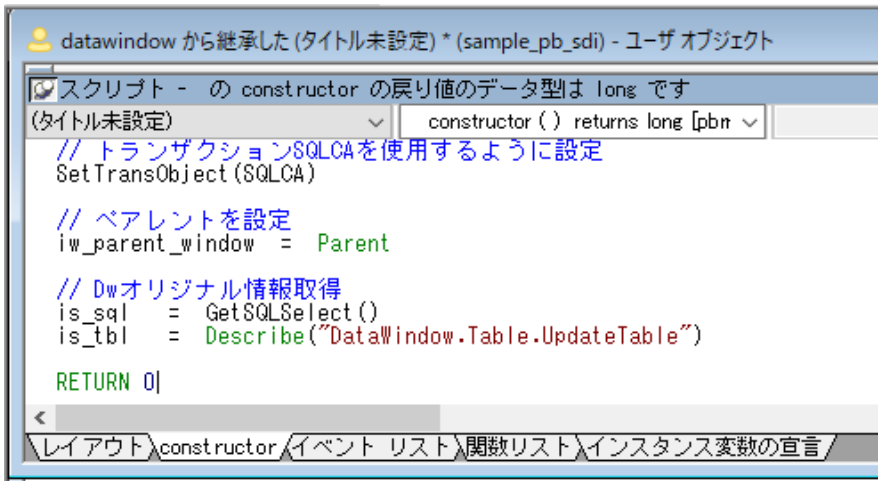


イベントにスクリプトを埋め込む。

A）constructor イベント

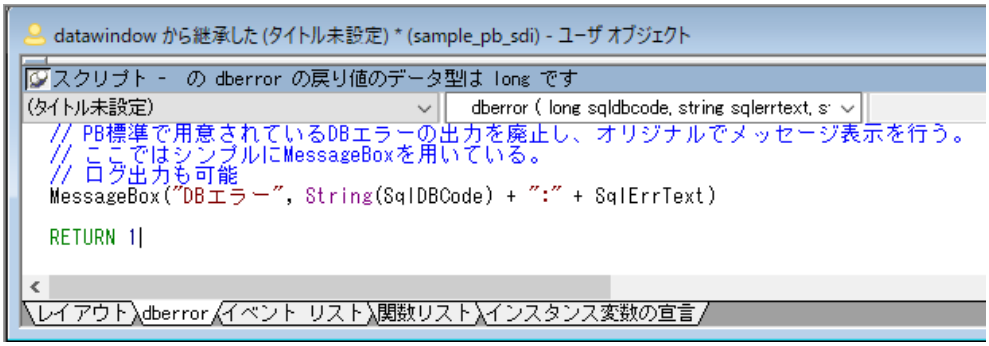
ペアレントを設定する。

Dw オリジナル情報を取得して保持する。



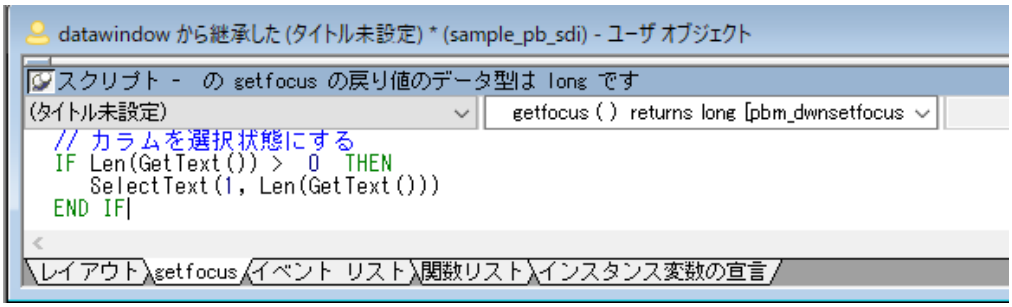
B）dberror イベント

PB 標準機能を廃止し、DB エラーの機能を組み込む。



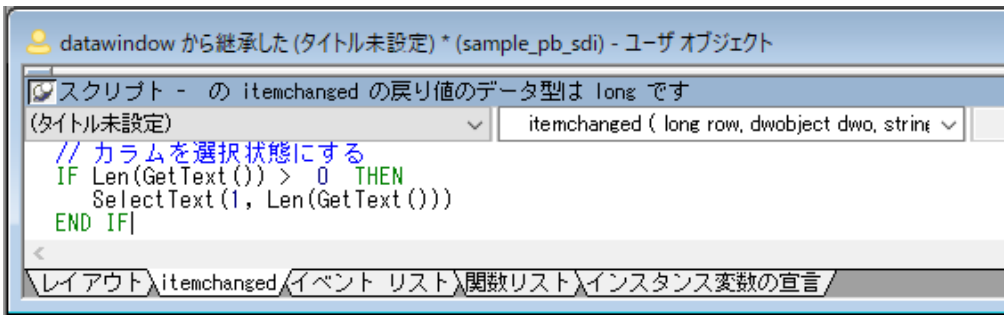
C）getfocus イベント

カラムを選択状態にする。



D）itemchanged イベント

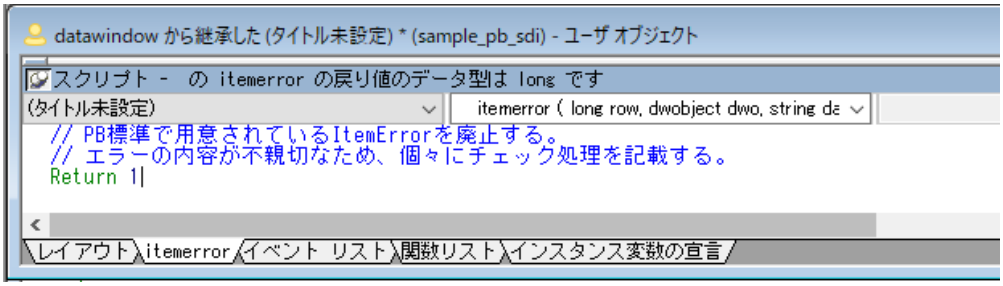
カラムを選択状態にする。



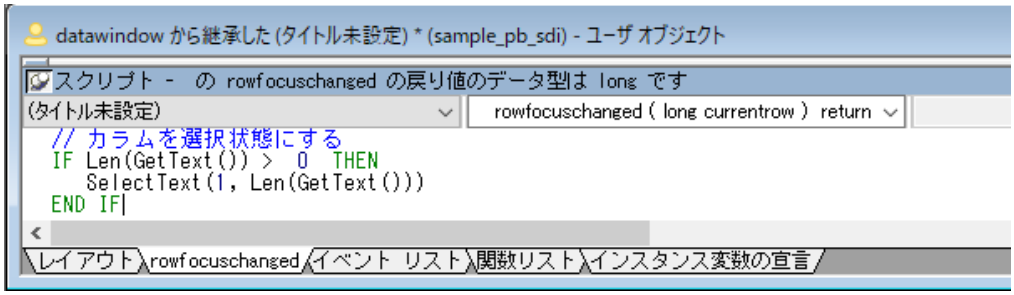
E）itemerror イベント

PB 標準機能を廃止する。（エラー出力なし）

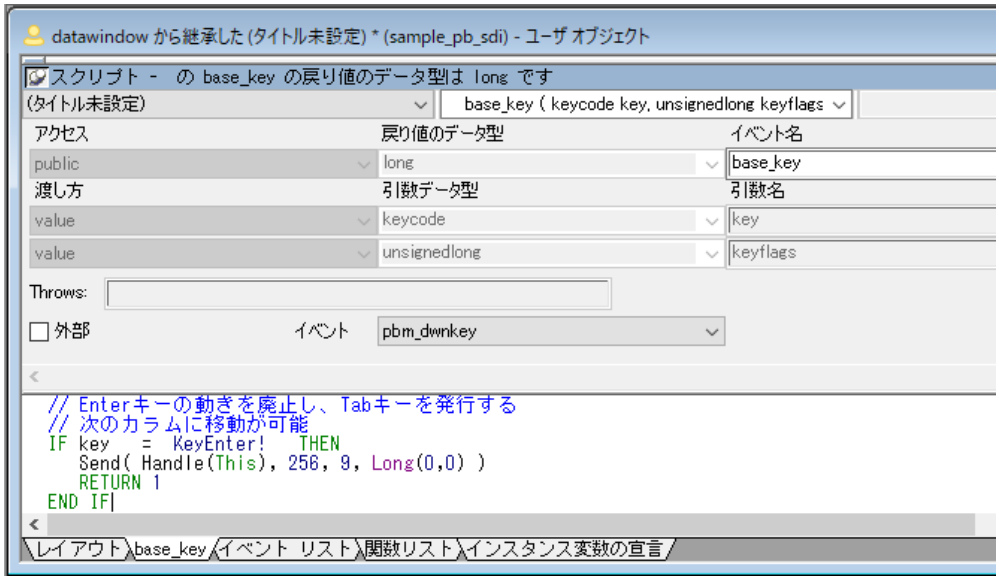
エラー処理は、各画面で実施する。



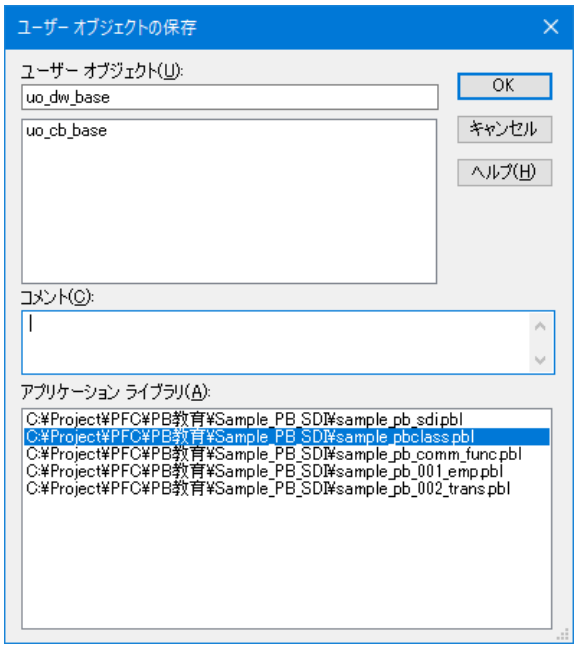
F）rowfocuschanged イベント
カラムを選択状態にする。



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

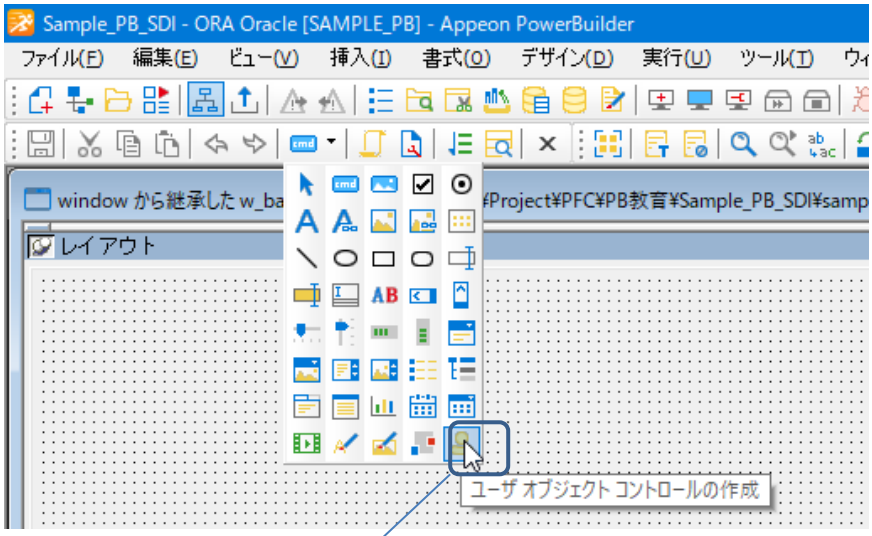


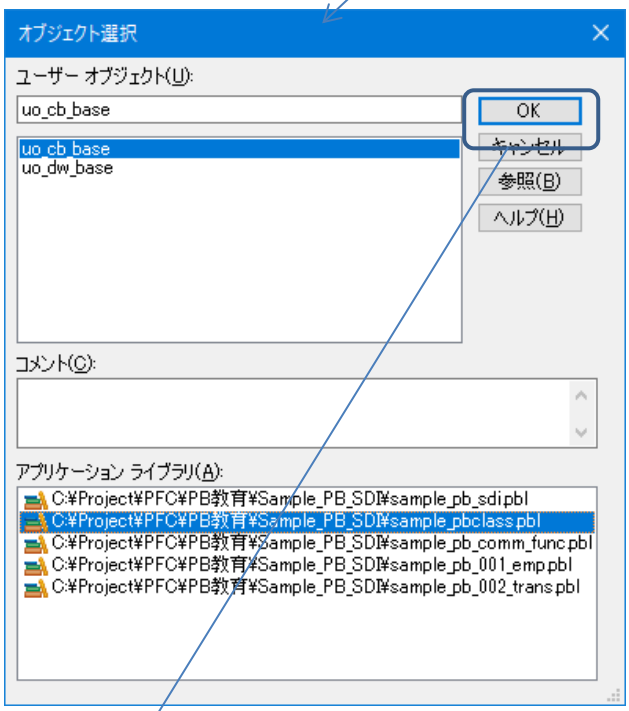
名前を付けて保存する。(uo_dw_base)



④　ウインドウクラスの作成（その２）

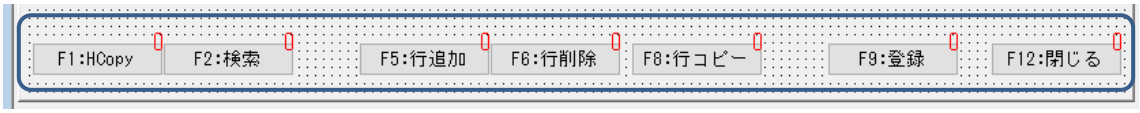
ウインドウ上にシステム共通で使用するボタンを配置する。





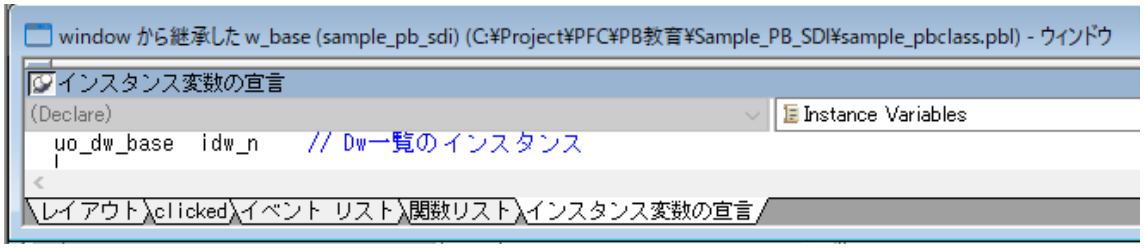
それぞれ配置する。

タブオーダーは０に設定する。



プロパティに値を設定する。

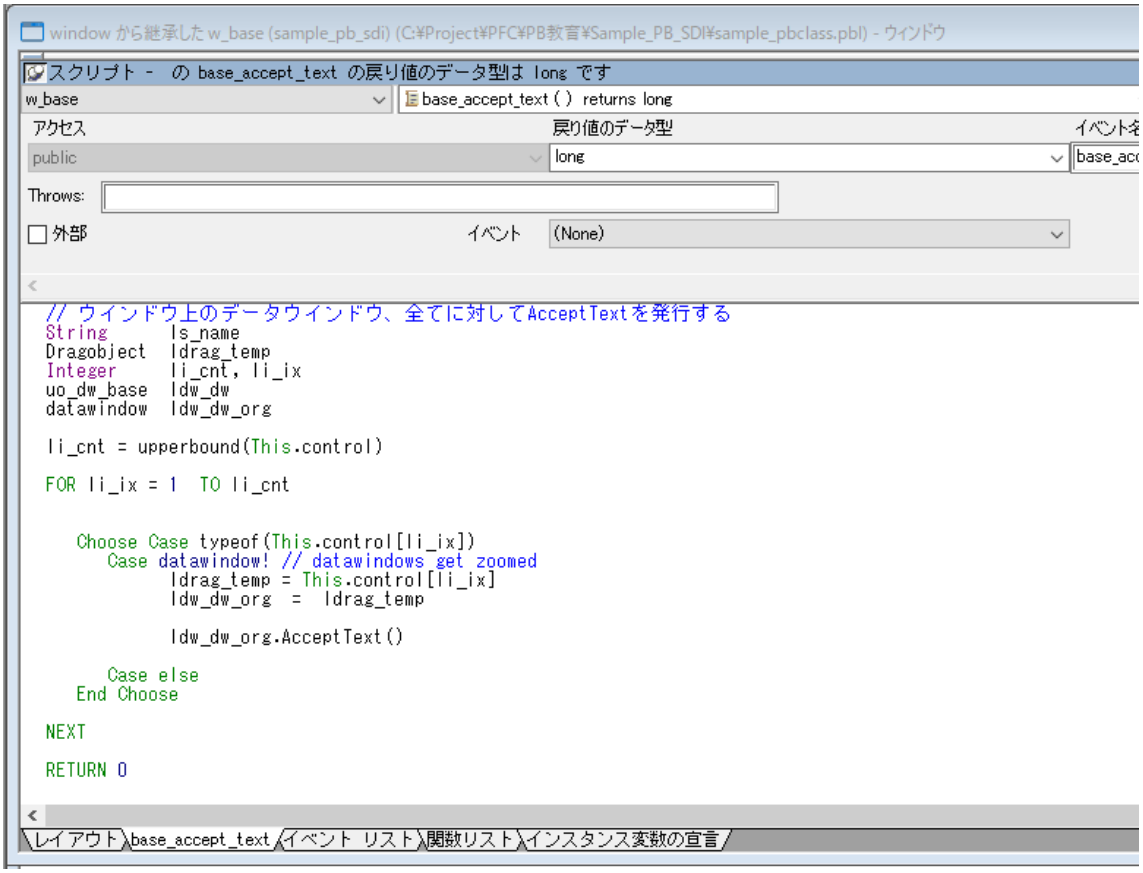
インスタンス変数の設定



クラスイベントを記載する。

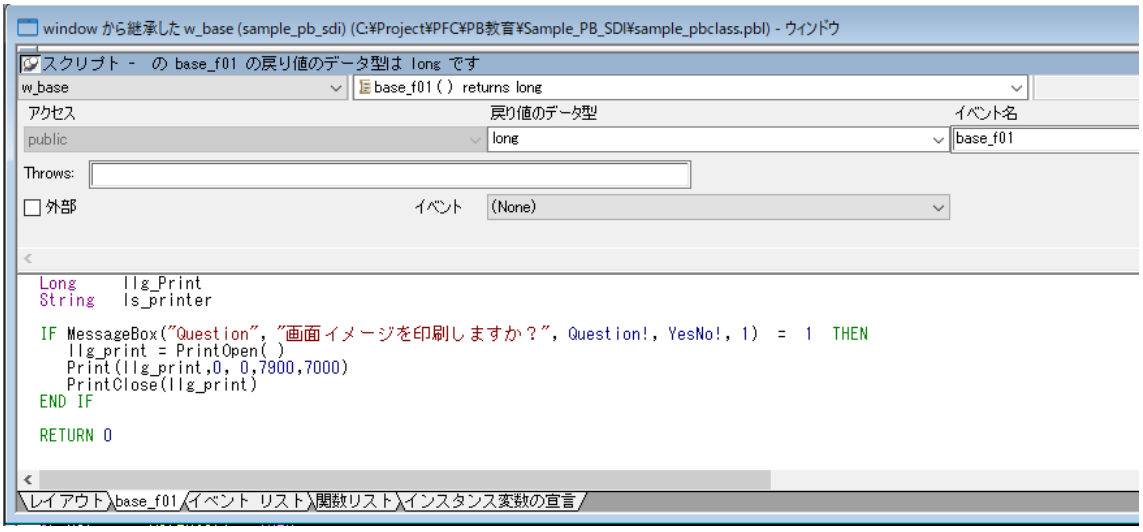
A）base_accept_text イベント

画面上全てのデータウインドウに AcceptText を発行する。



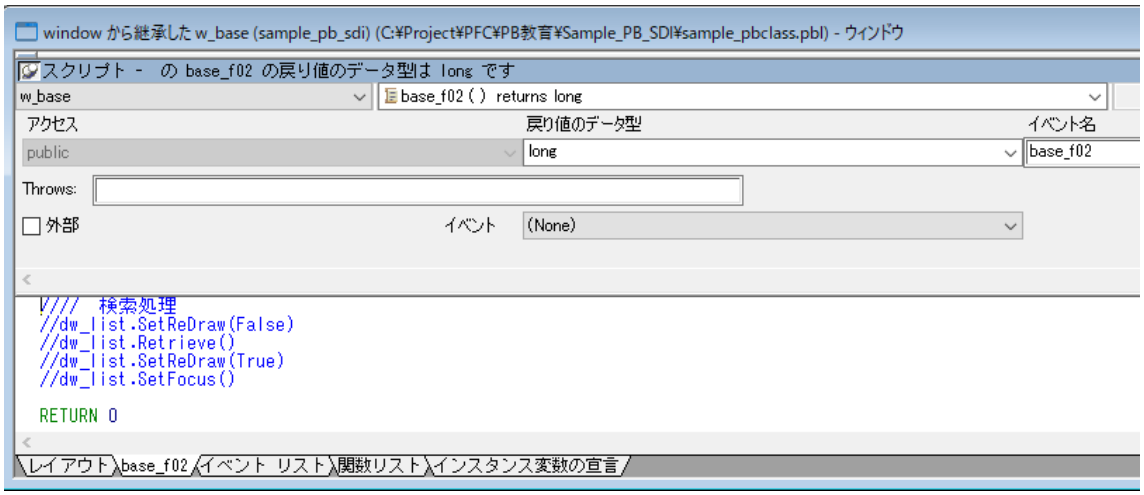
B）base_f01 イベント

画面コピーのスクリプトを組み込む。



C）base_f02 イベント

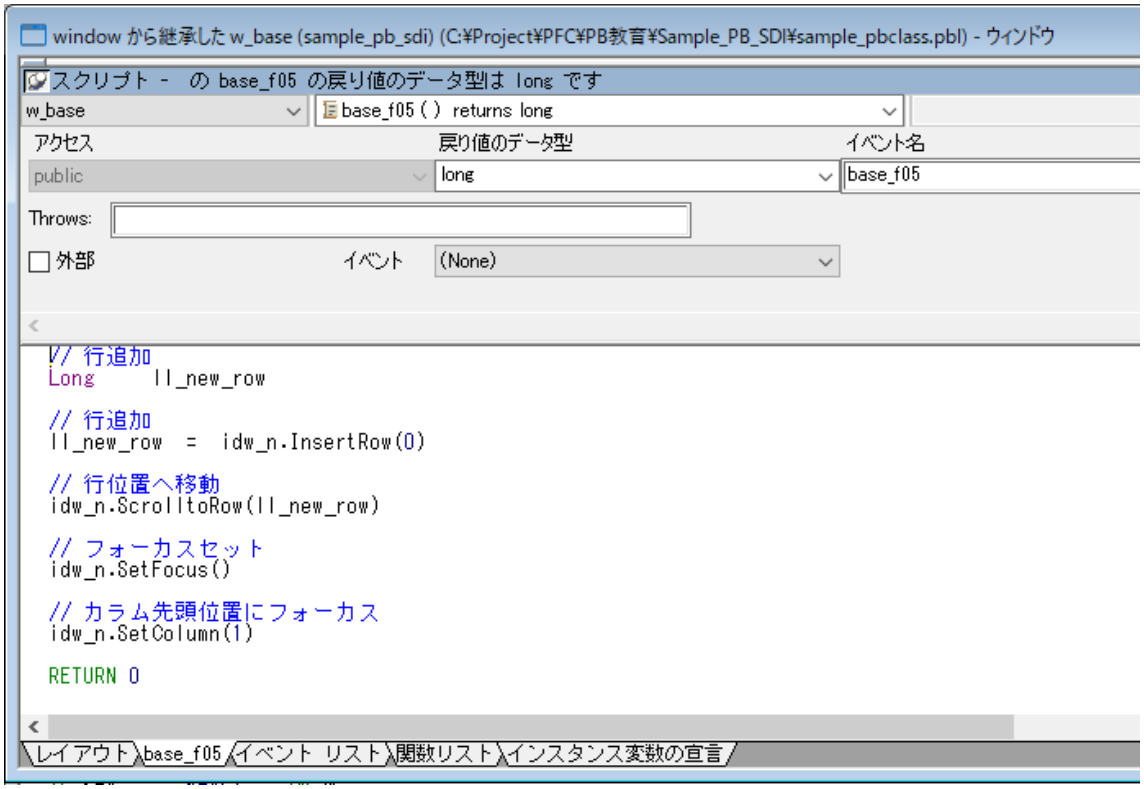
検索処理は、画面ごとに異なるため、サンプルスクリプトを記載する。



D）base_f05 イベント

行追加処理を組み込む。

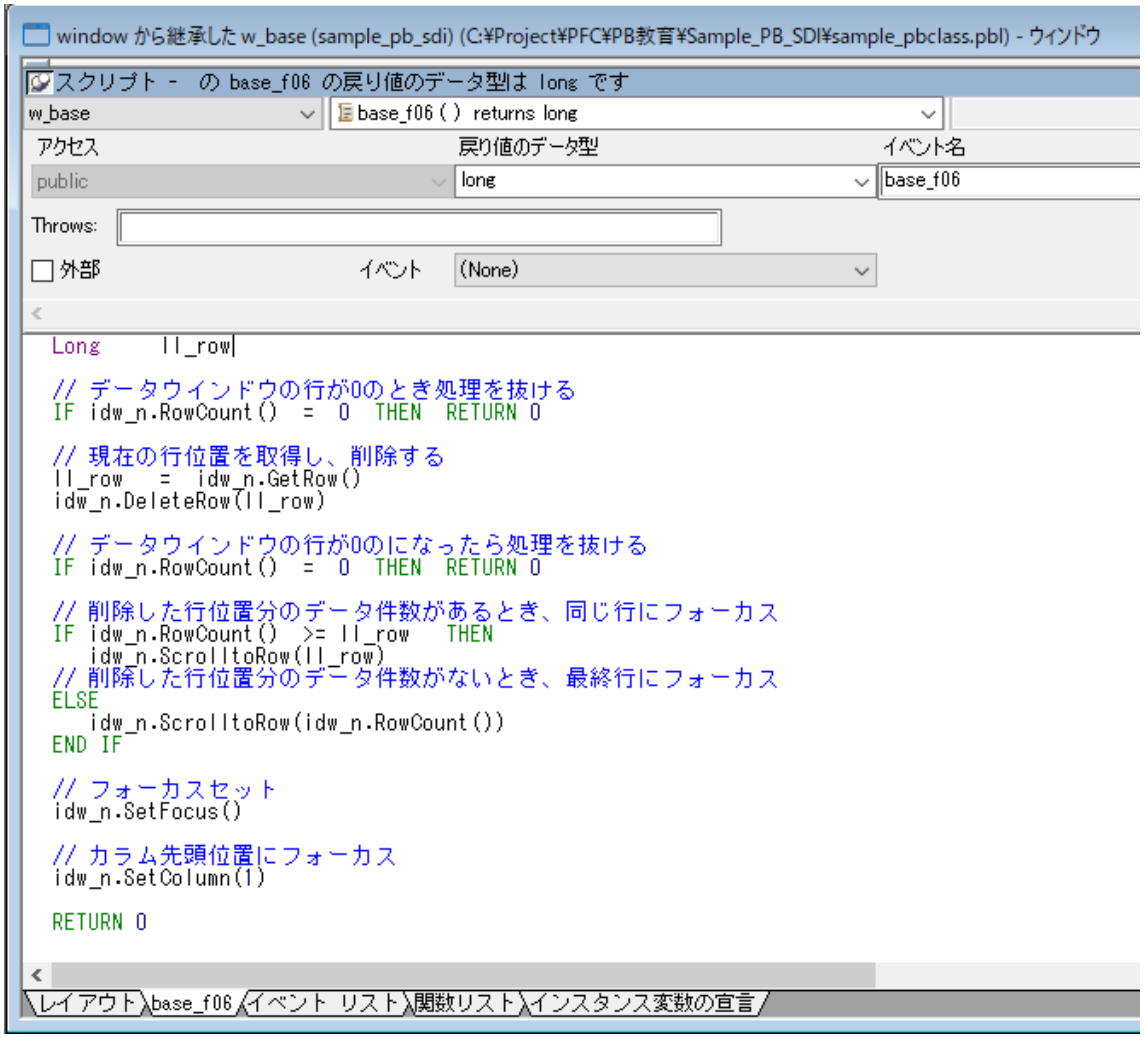
ｄｗ一覧インスタンスの idw_n に対して処理を実施する。



E）base_f06 イベント

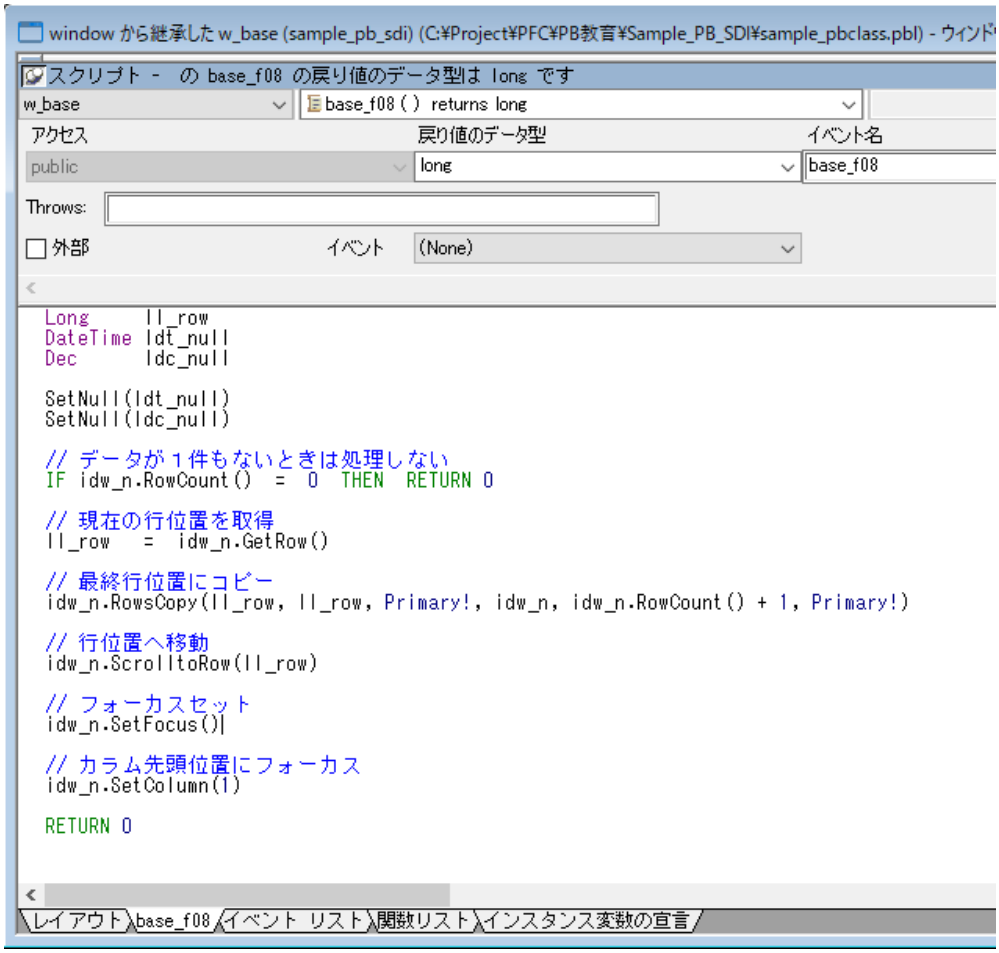
行削除処理を組み込む。

ｄｗ一覧インスタンスの idw_n に対して処理を実施する。



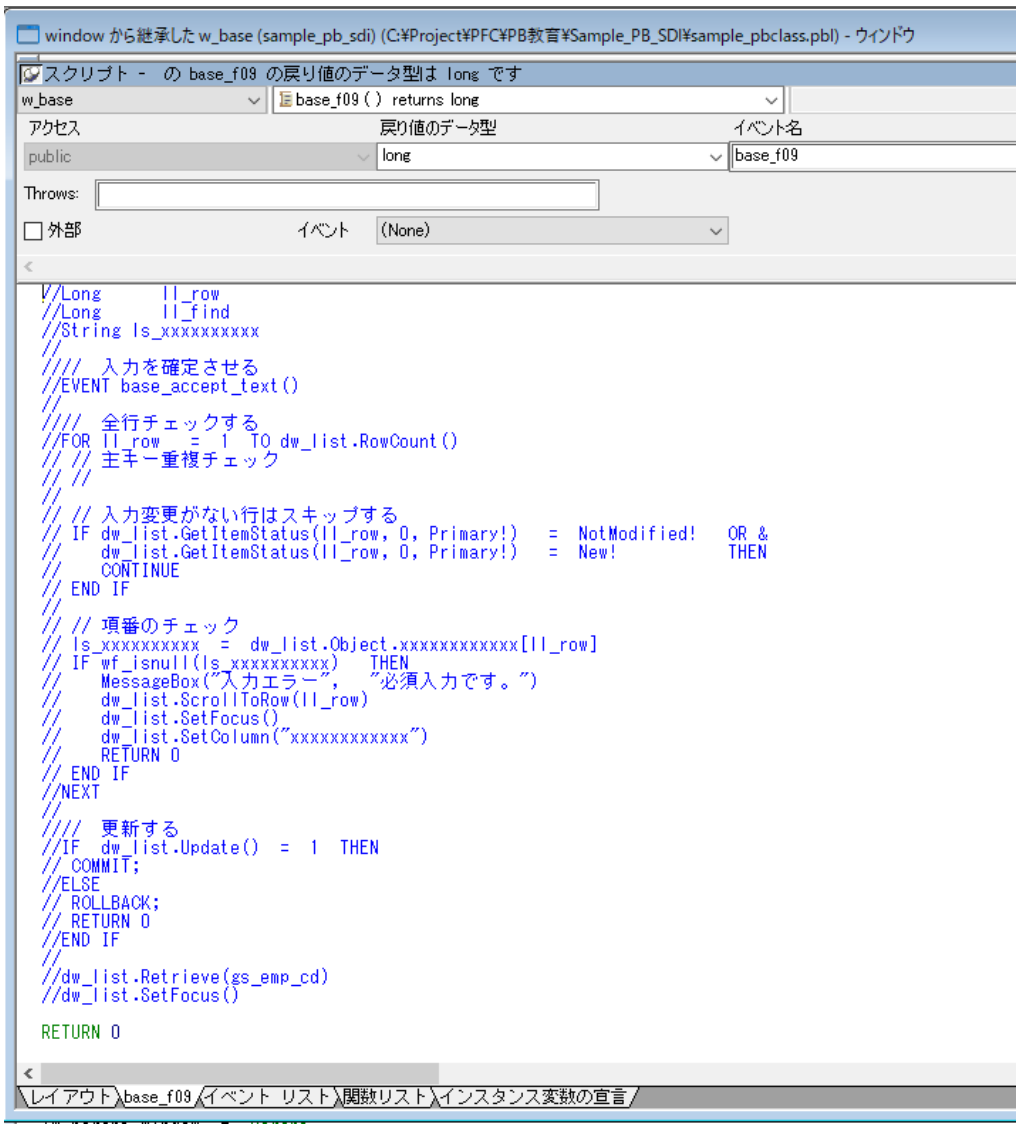
F）base_f08 イベント

行コピー処理を組み込む。



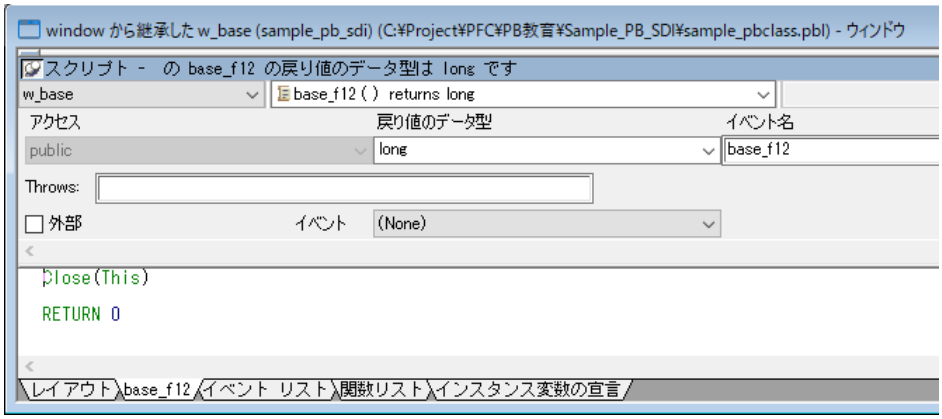
G）base_f09 イベント

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。



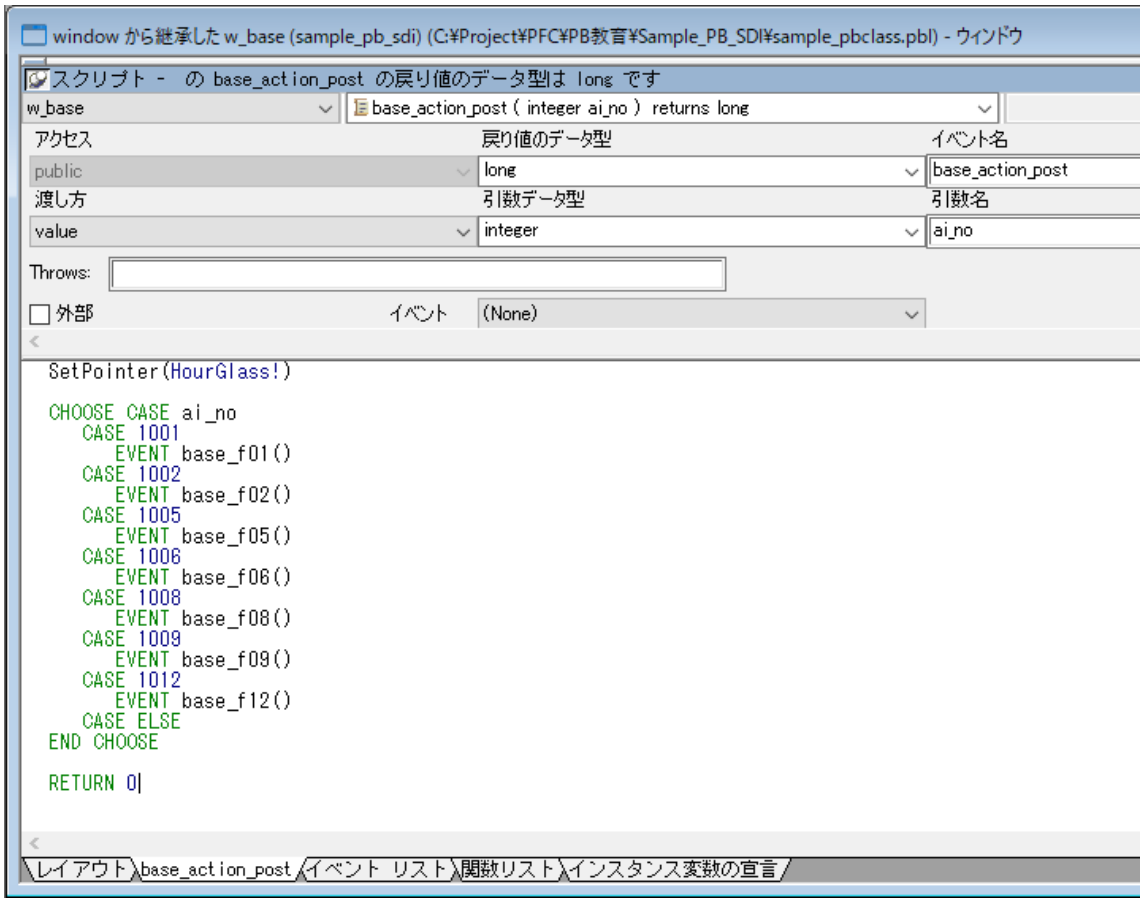
H）base_f12 イベント

終了処理を組み込む。



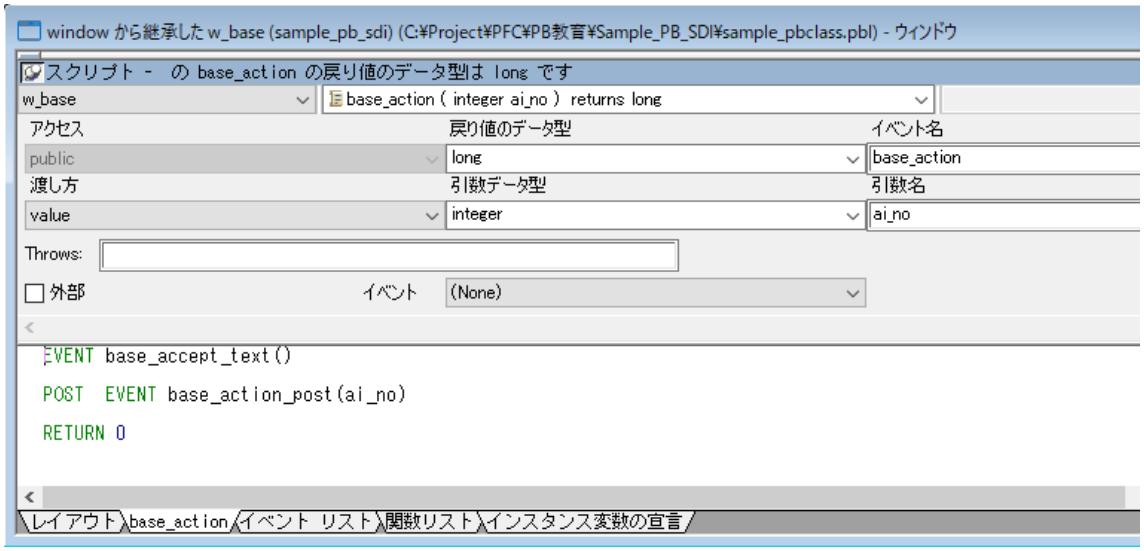
I）base_action_post イベント

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。



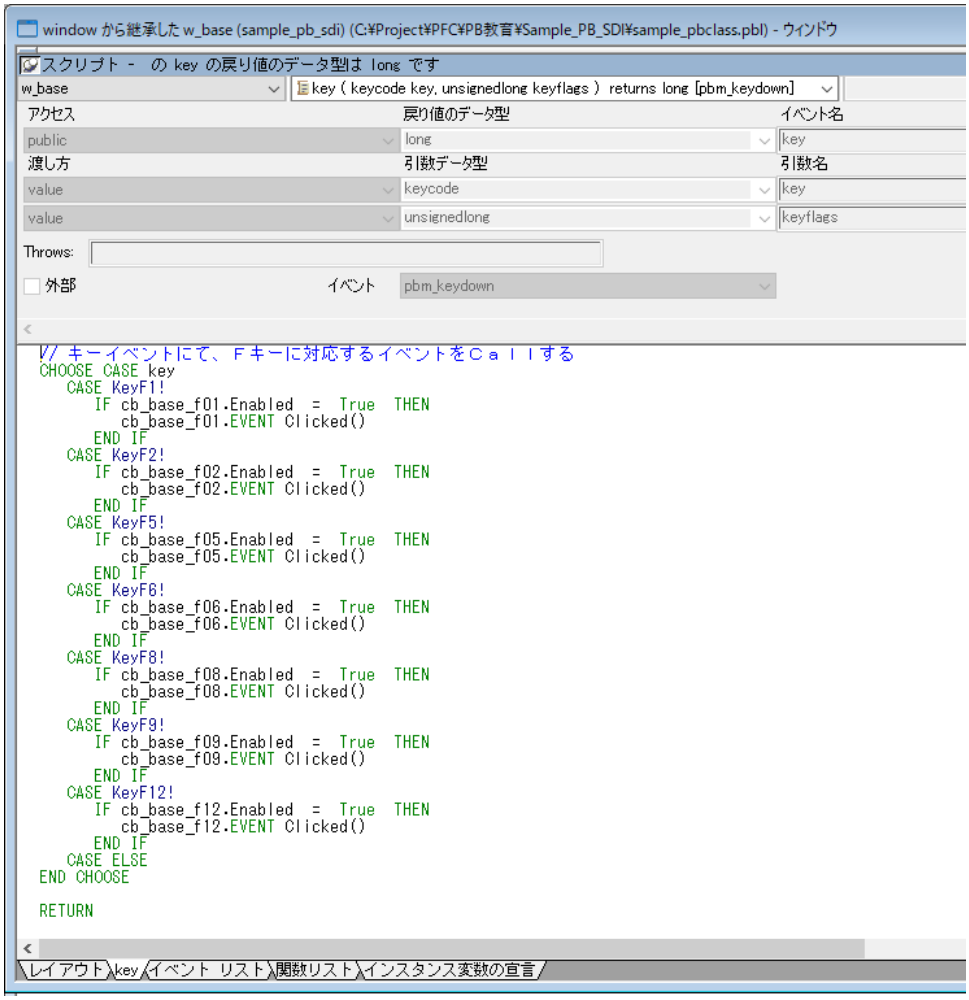
J）base_action イベント

入力を確定させ、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。



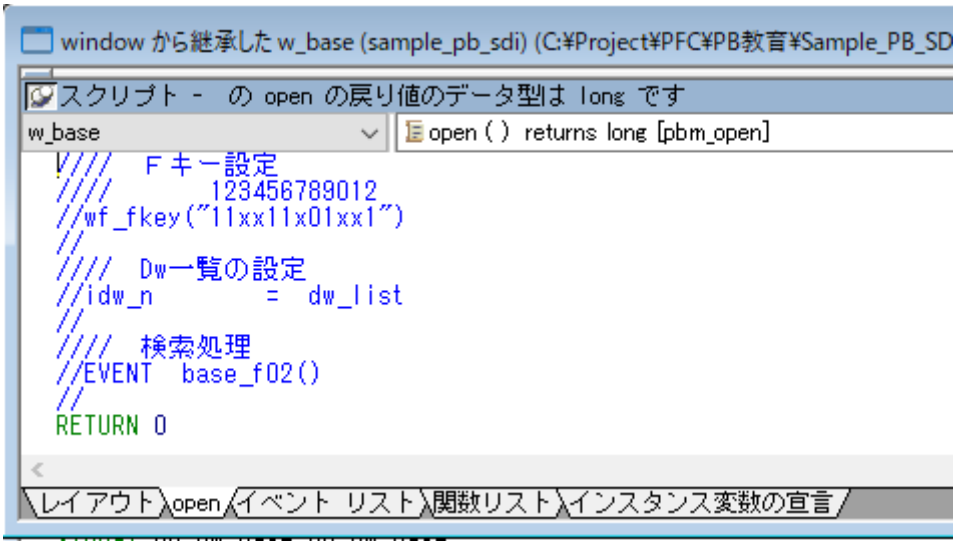
K）key イベント

Ｆキーに連動する動作を組込む。



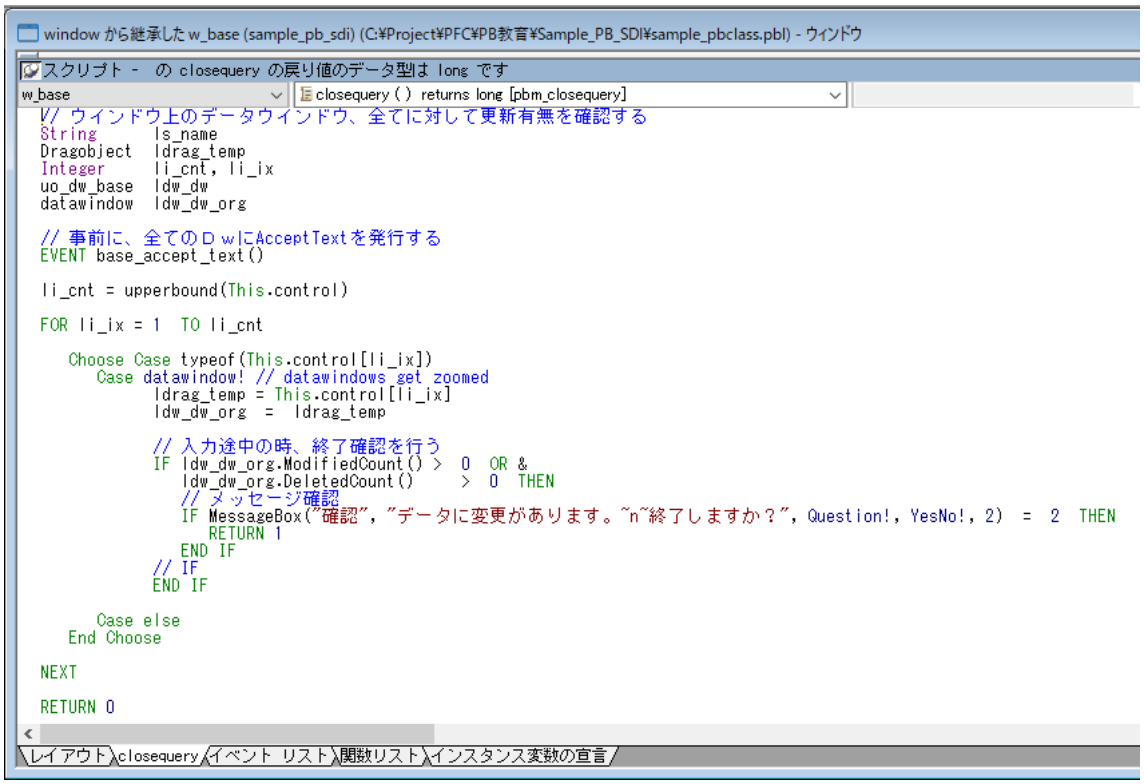
L）open イベント

共通仕様をコメントで記載する。



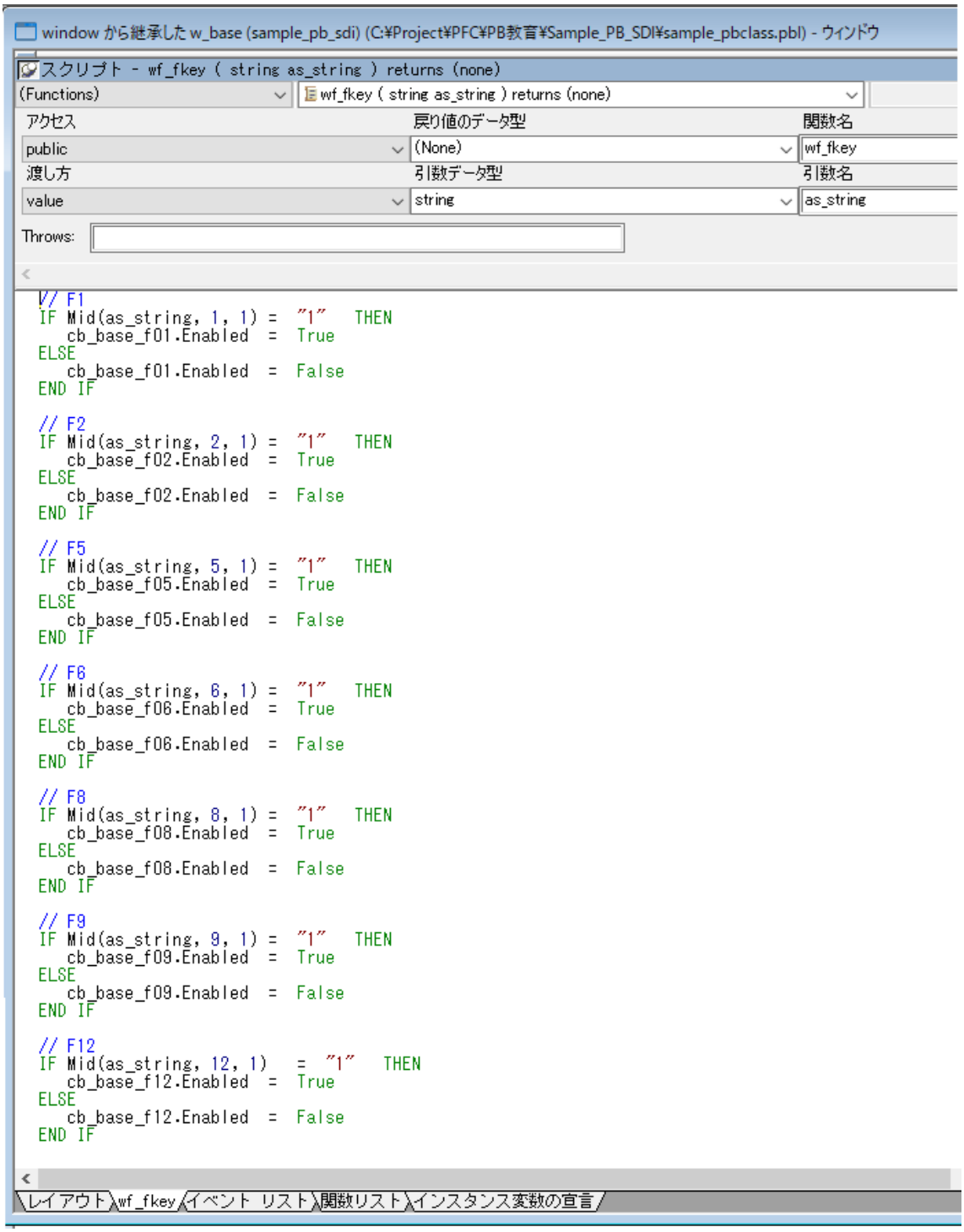
M）closequery イベント

終了時にデータウインドウの更新確認を組み込む。



N）wf_fkey イベント

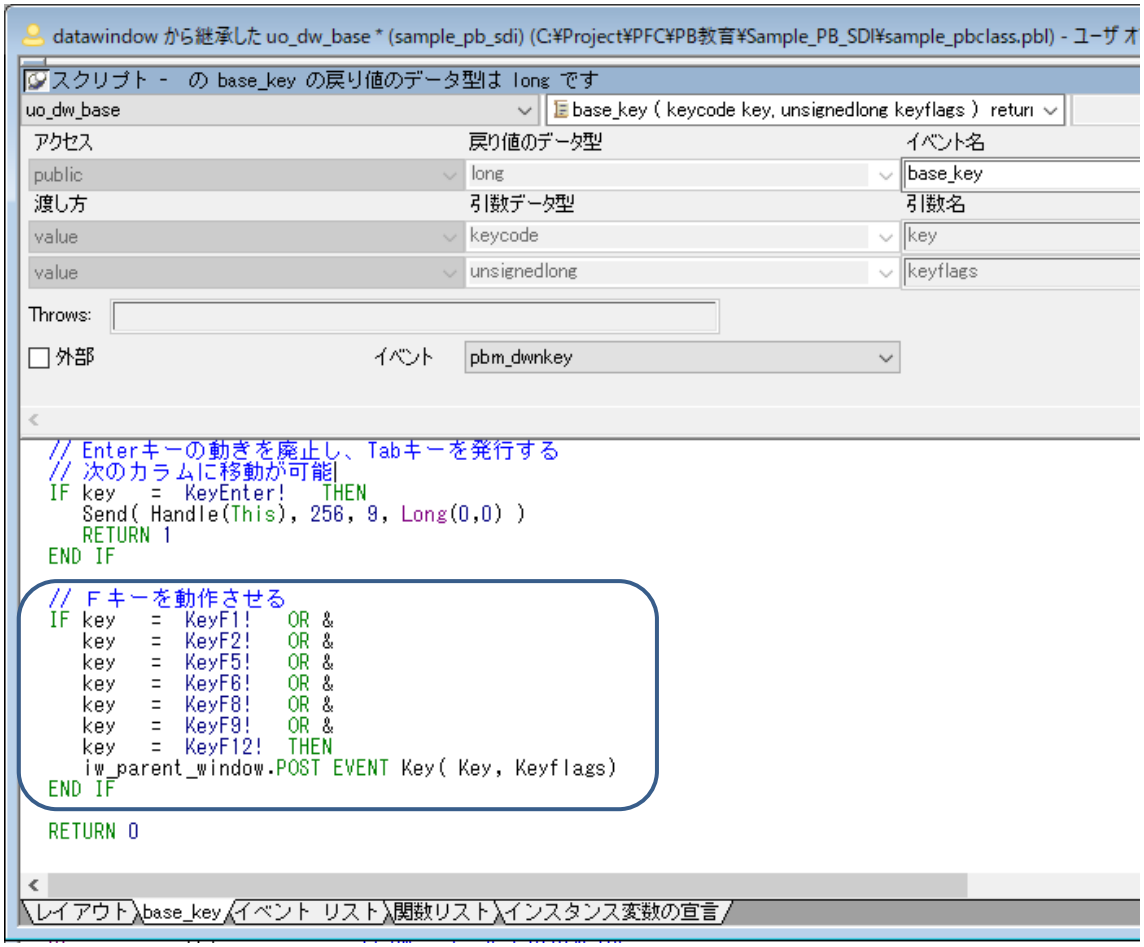
Ｆキーの使用可否を設定する関数を作成する。



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

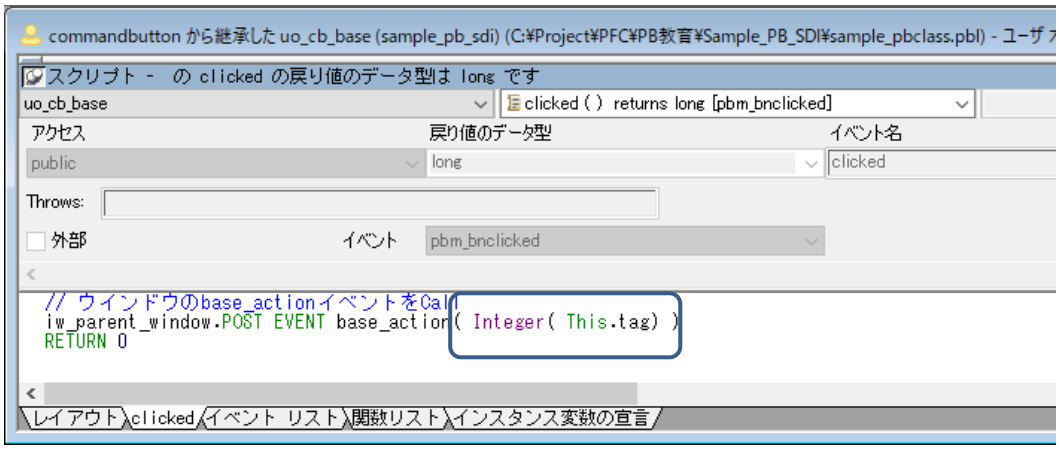
データウインドウ上でＦキーを押下した際の機能を組み込む。



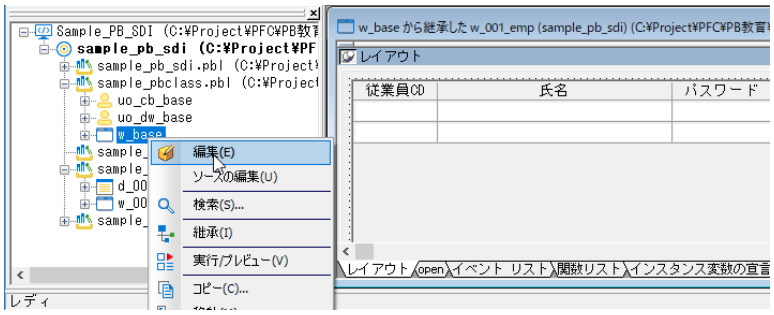
⑥　コマンドボタンクラスの作成（その２）

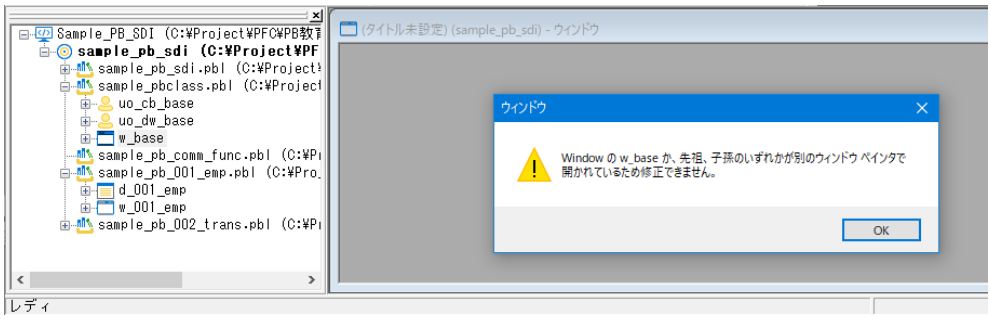
A）clicked イベント

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。

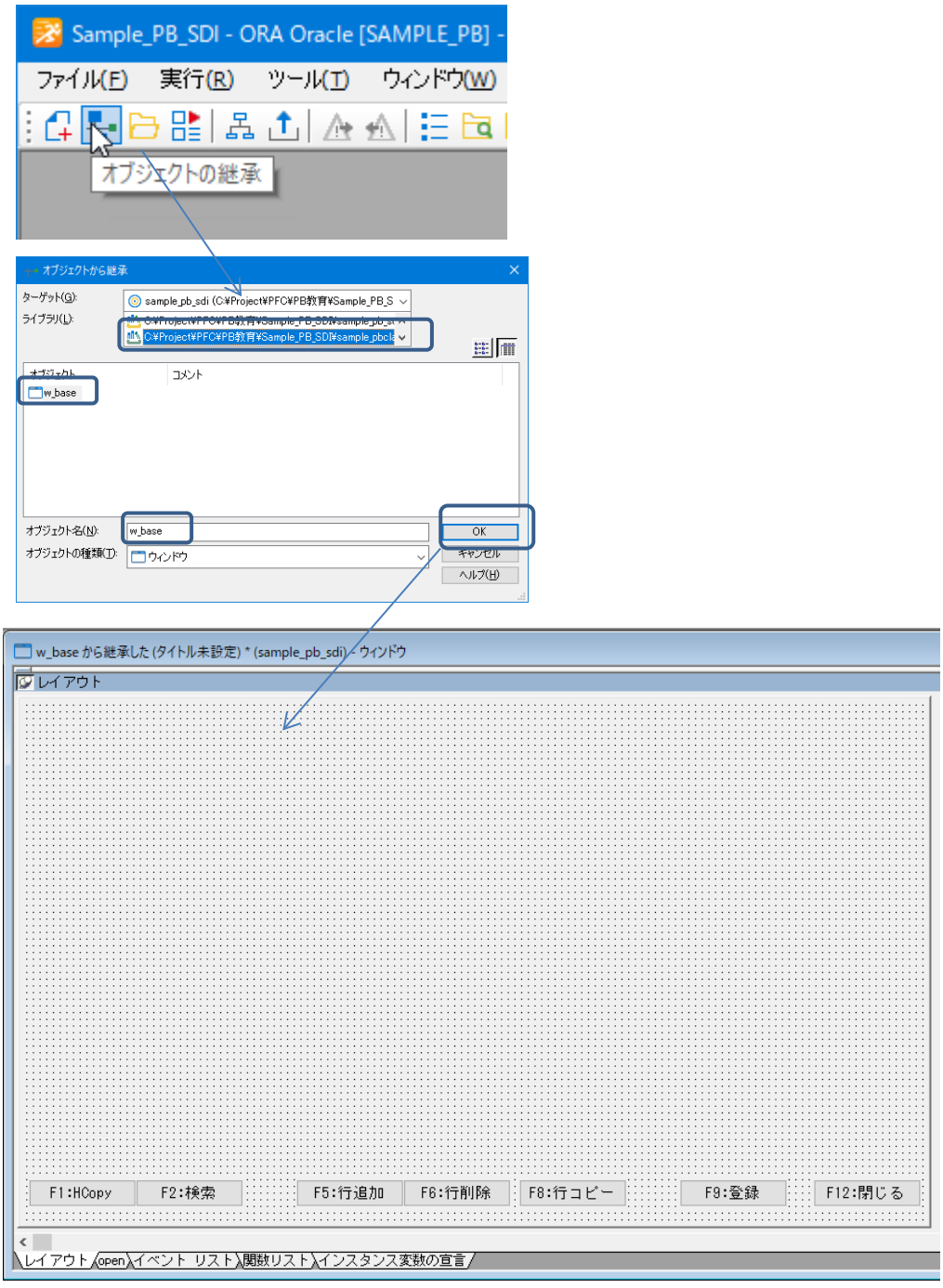




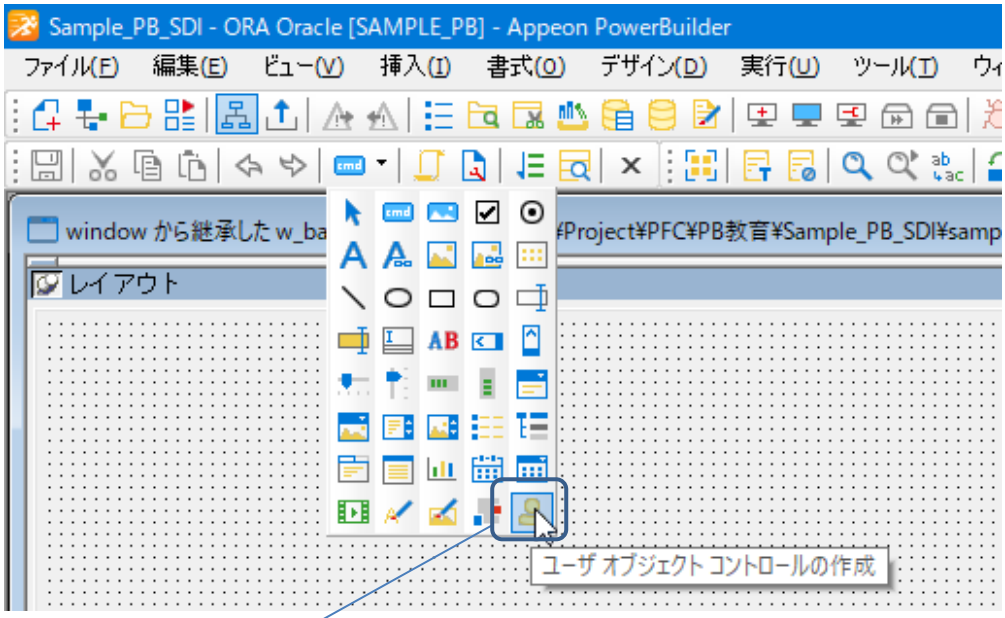
（３）　クラスを用いたプログラミング

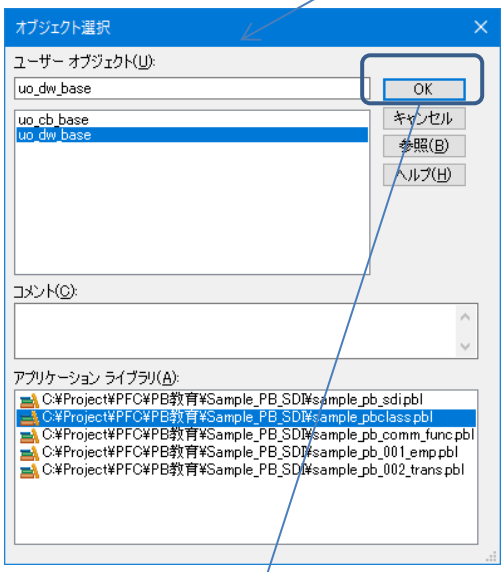
今回、説明を簡易的にするために、下記の２画面についてクラスを用いて作成する。
・従業員マスタメンテナンス
・交通費精算入力

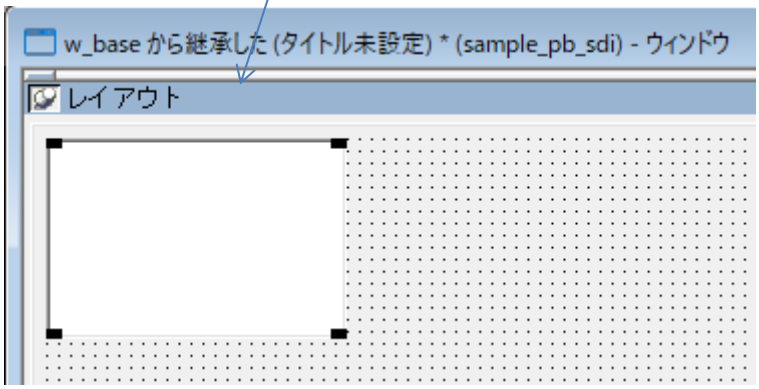
① 従業員マスタメンテナンス
ウインドウ（w_base）を継承する。



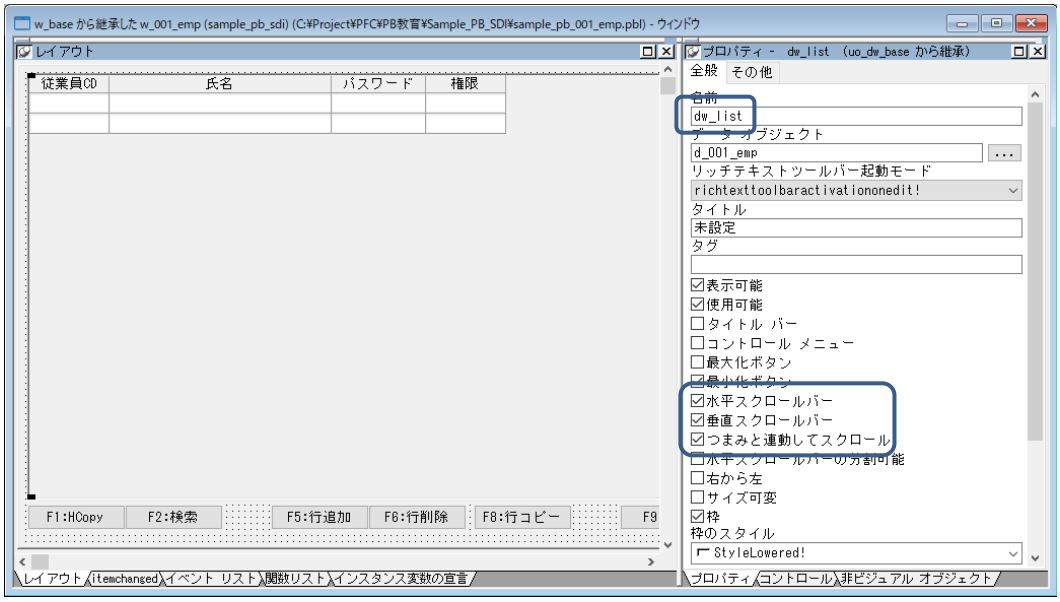
データウインドウを配置する。







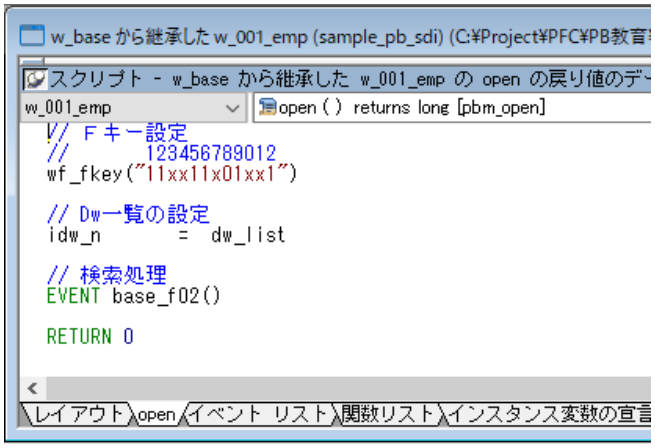
データウインドウを設定する。



スクリプトを記載する。

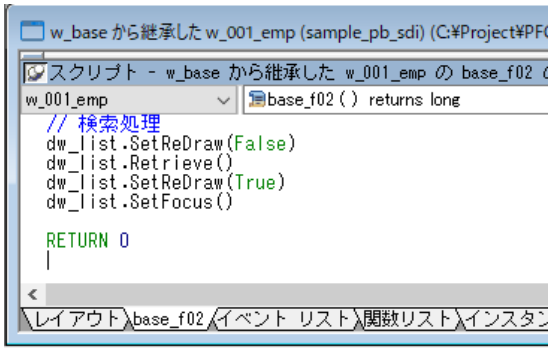
A）open イベント

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

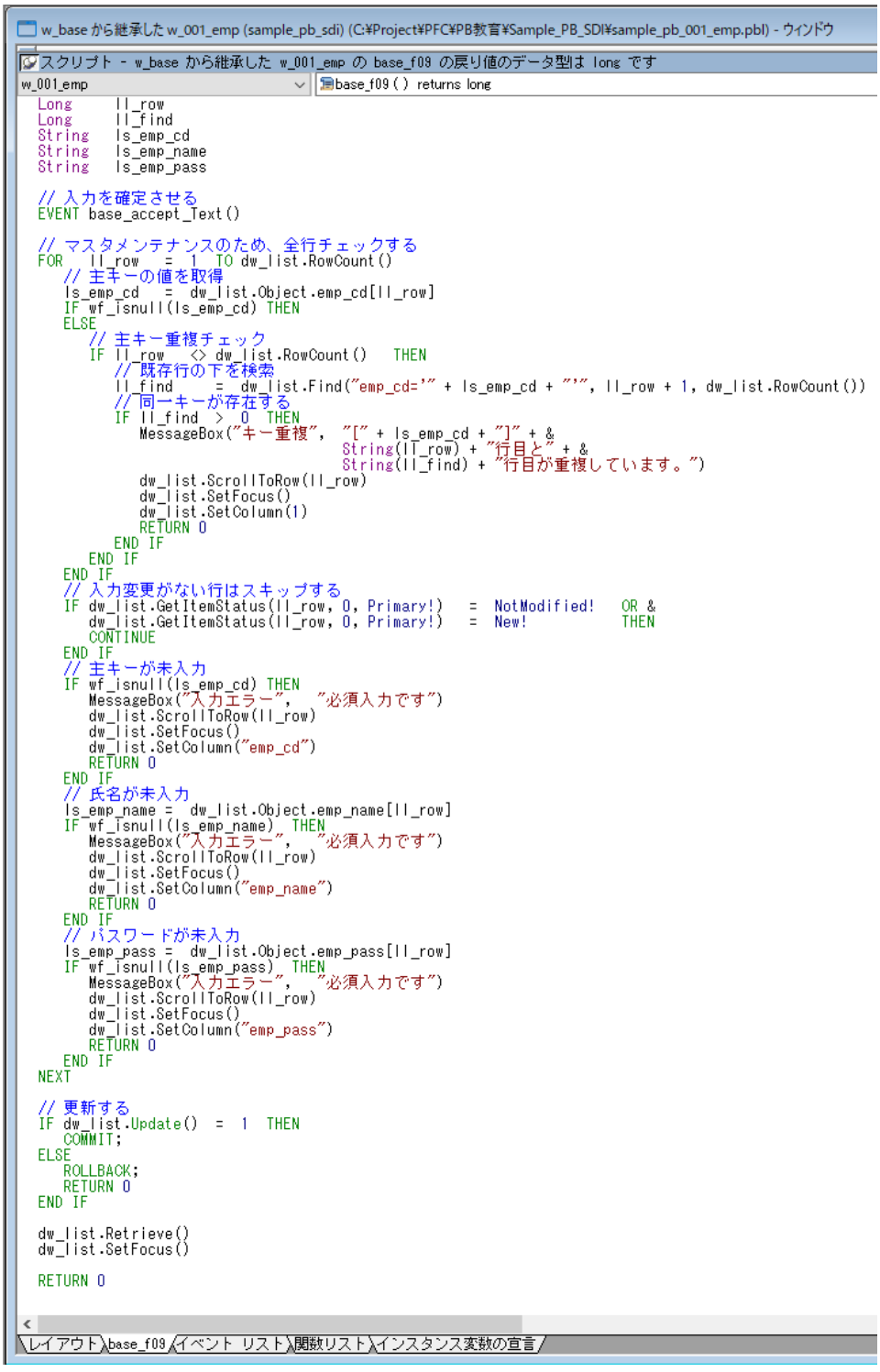
継承元に記載してコメントを例に記載する。



※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。

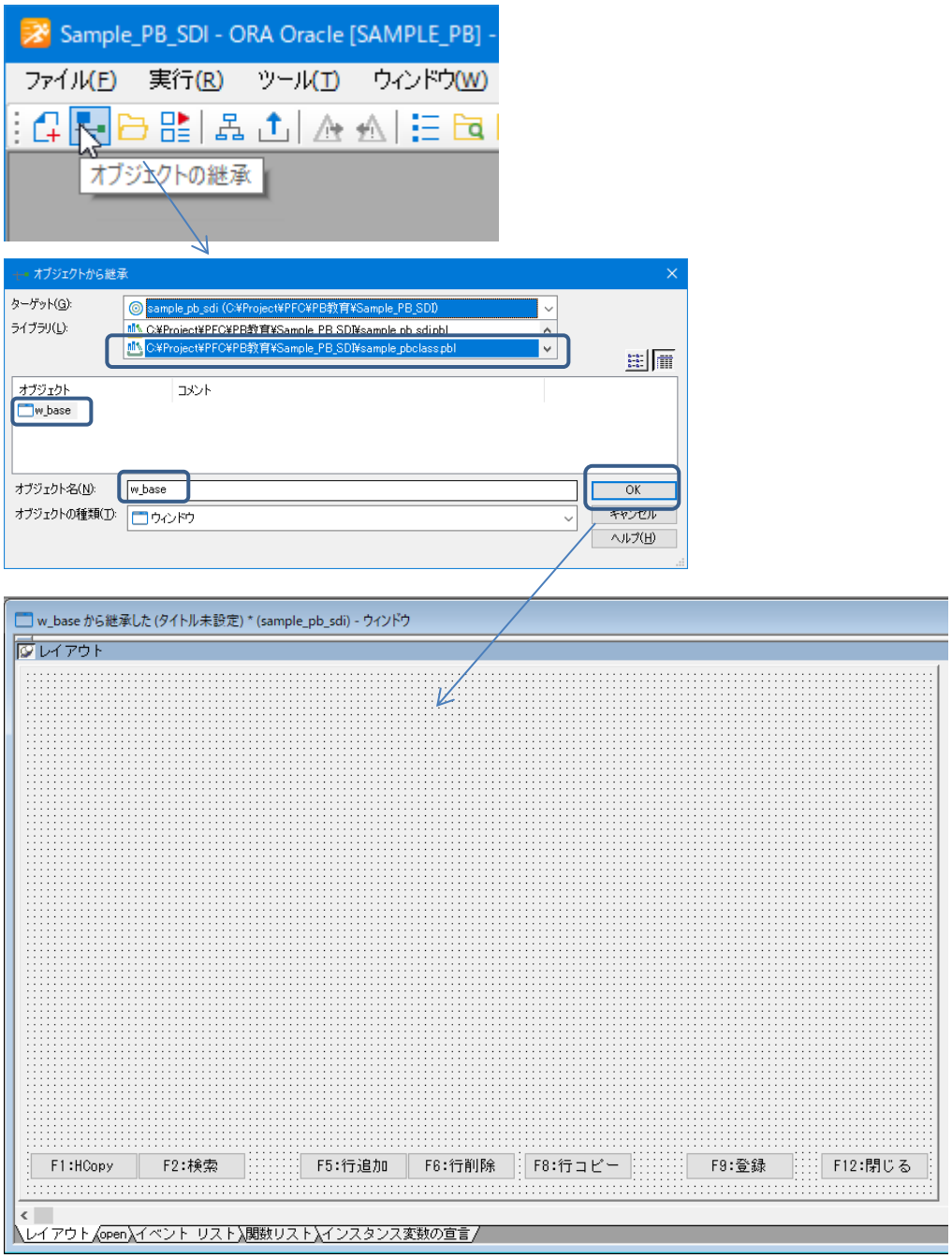
C）base_f09 イベント

カラムチェック、更新処理を記載する。

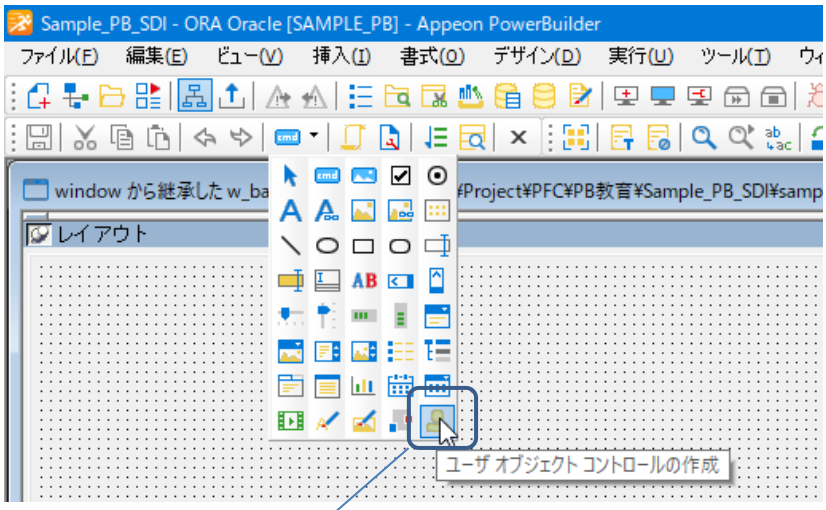


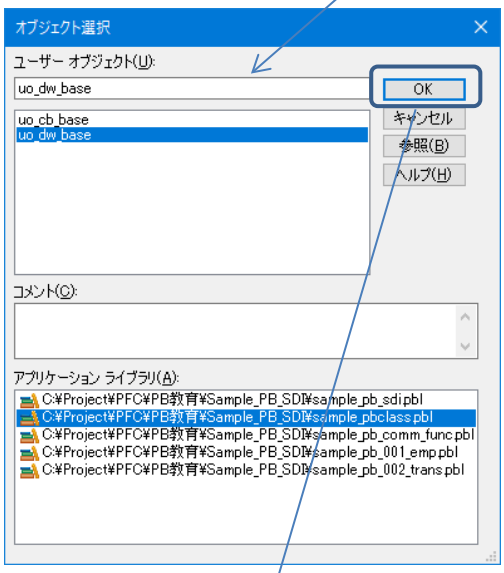
② 交通費精算入力

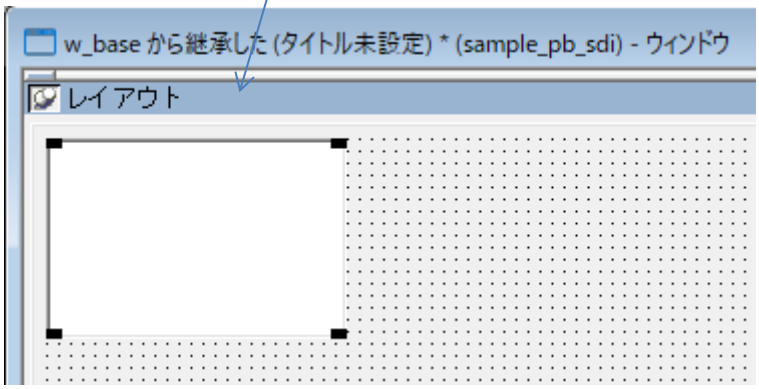
ウインドウ（w_base）を継承する。



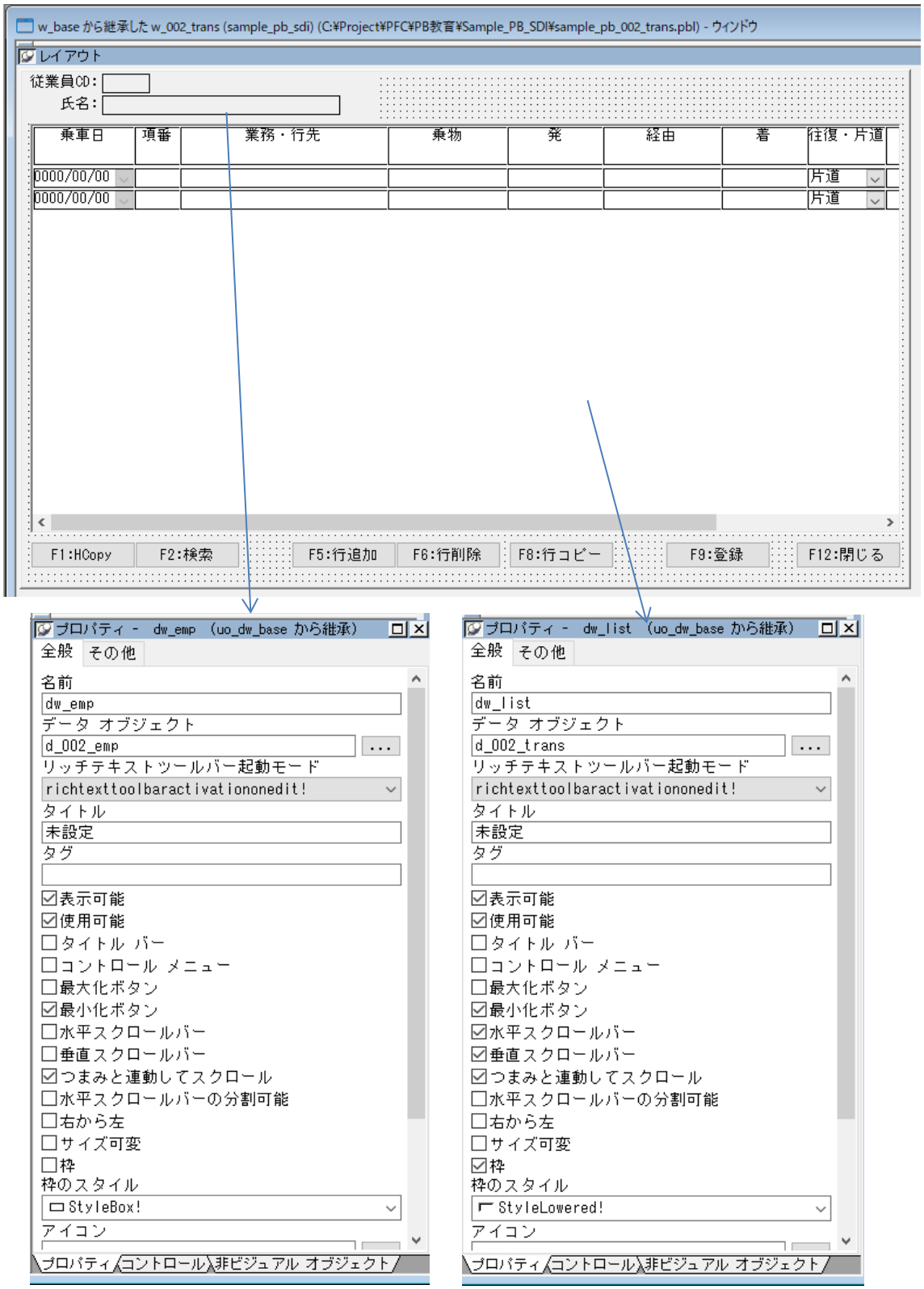
データウインドウを配置する。







データウインドウを設定する。

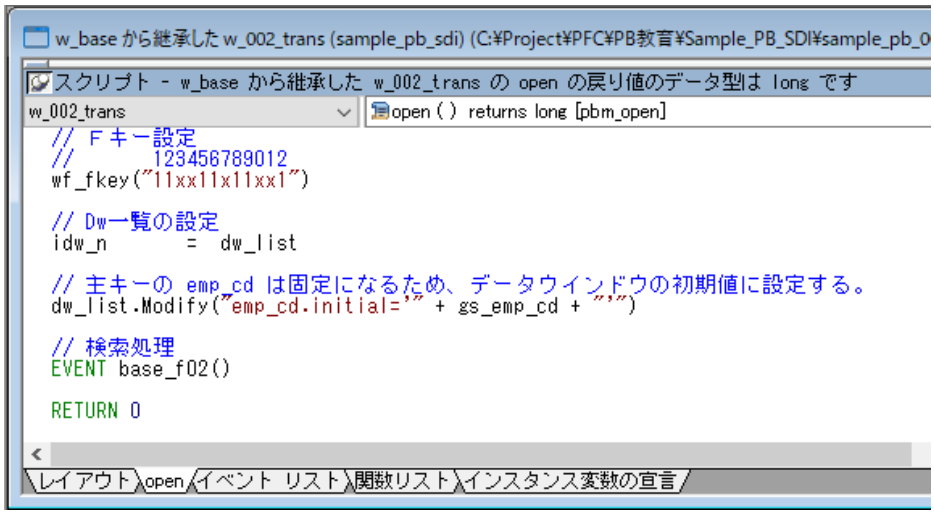


スクリプトを記載する。

A）open イベント

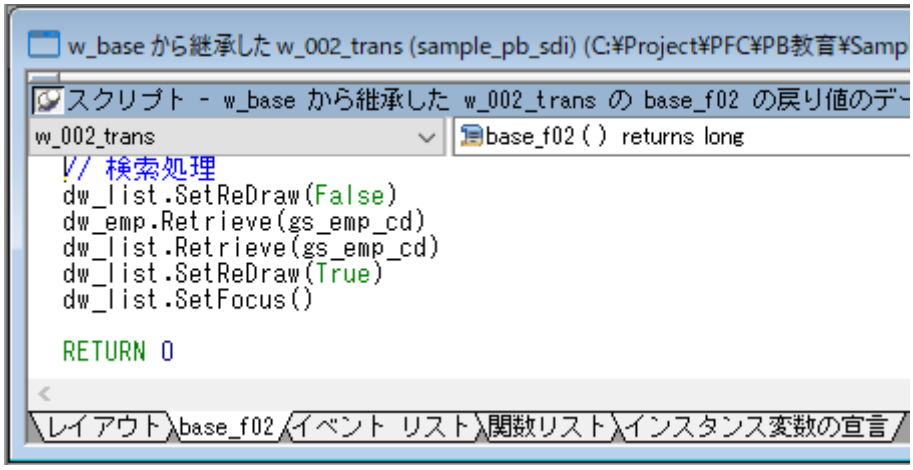
継承元に記載してコメントを例に記載する。

dw_list の初期値設定を行う。



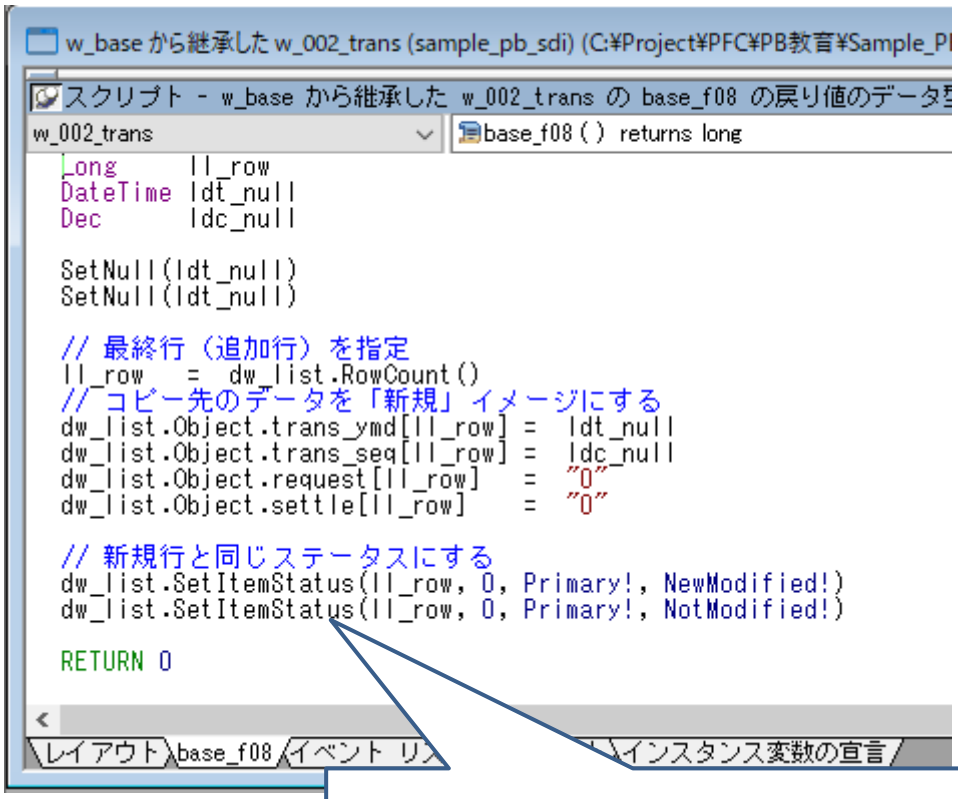
B）base_f02 イベント

継承元に記載してコメントを例に記載する。



C）base_f08 イベント

行コピーのスクリプトを記載する。



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。

D）base_f09 イベント

カラムチェック、更新処理を記載する。

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。

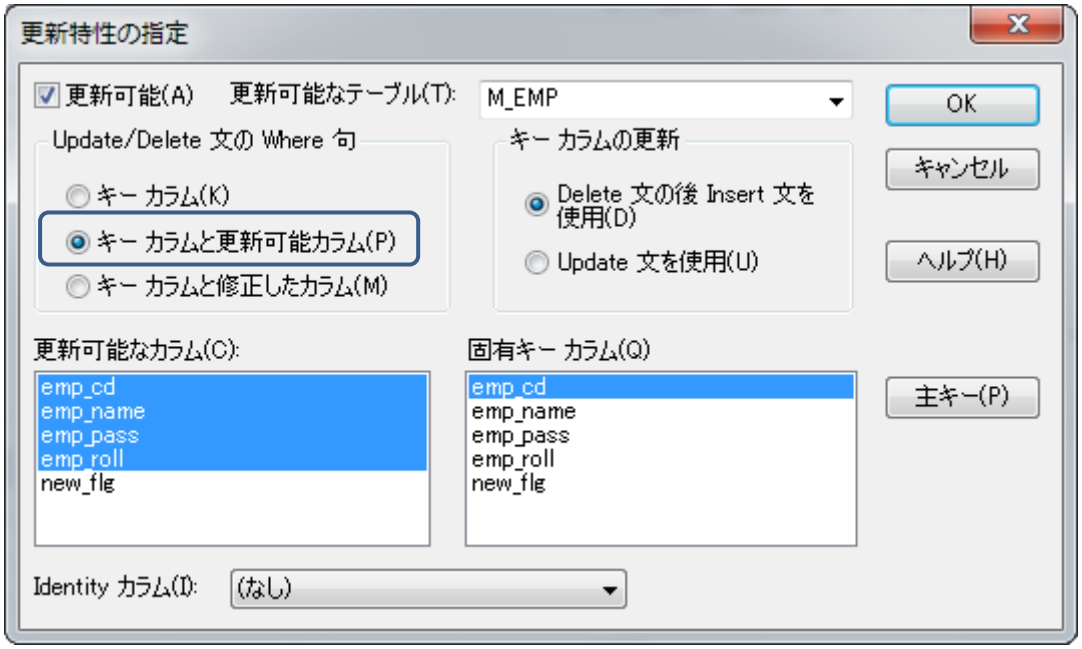
（１）　有効な機能

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）

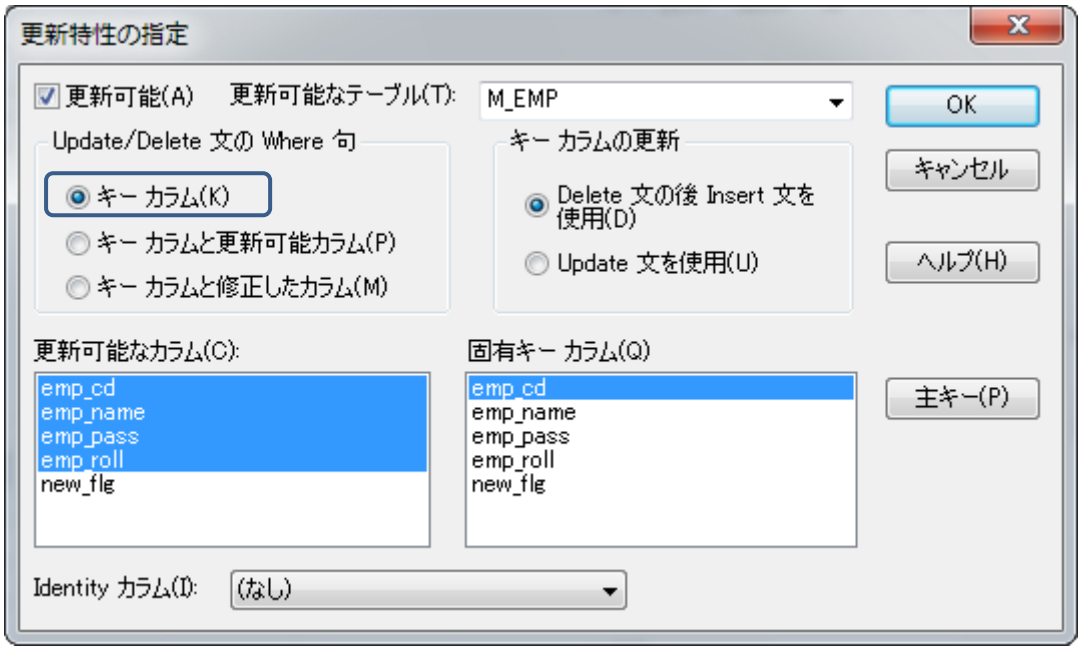
データの更新（Update 文）を行う際に、対象のデータが他のユーザによって更新されていないかどうか判定するためには、下図の設定を行う。★



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

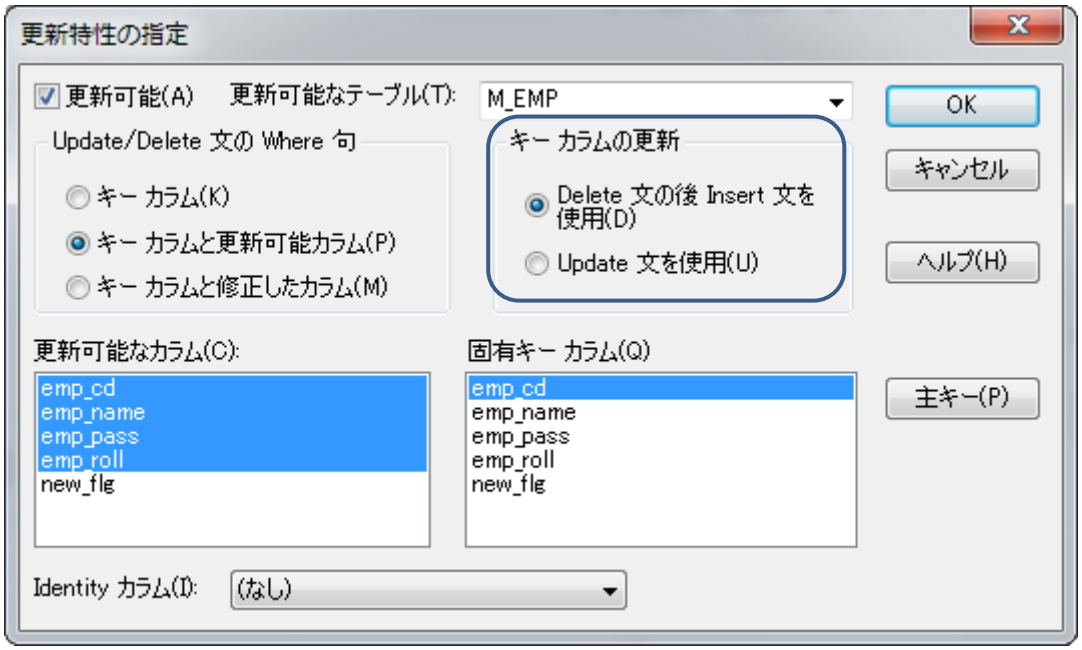
行ロックなどの悲観的排他制御により整合性が保障されているときは、下図のように設定を行う。★



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

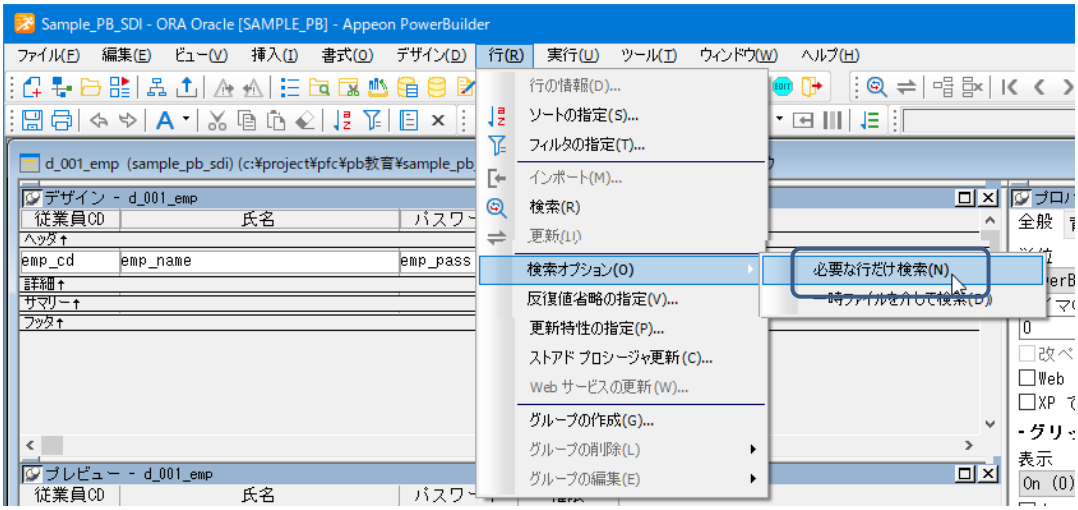
インタフェースの作りにもよるが、「Delete 文の後 Insert 文を使用」を推奨する。★



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 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。

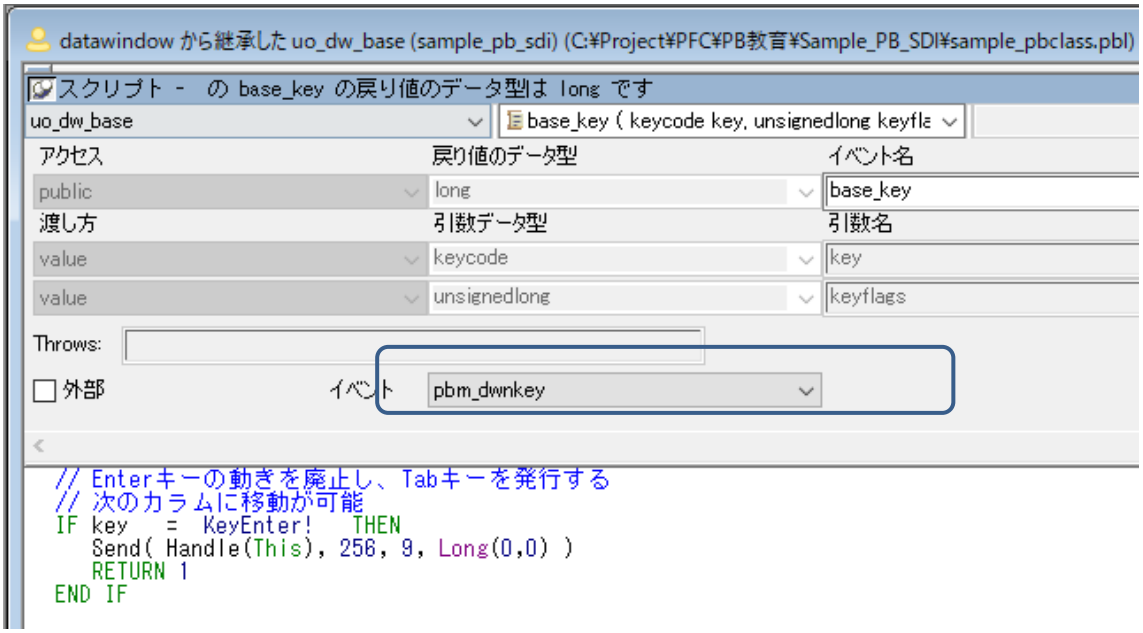


注意点として、SQL 文に ORDER BY 句の記載がある場合、DB 側で全ての検索処理が終わってから表示が始まるので、これを意識した開発が必要である。

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
NewModofied!	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

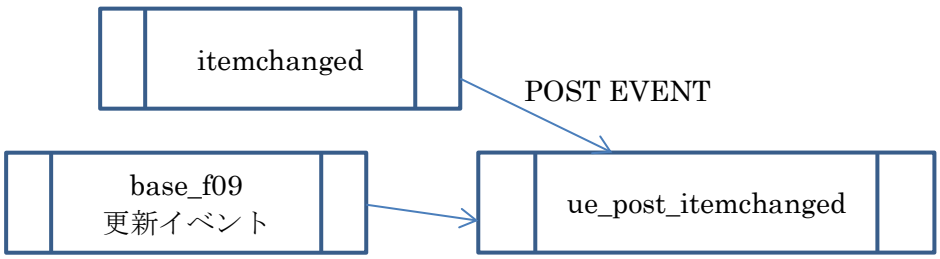
本コースでは登録前に入力項目のチェックを行っているが、入力操作中にエラーチェックを実施したいケースがある。

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。



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）

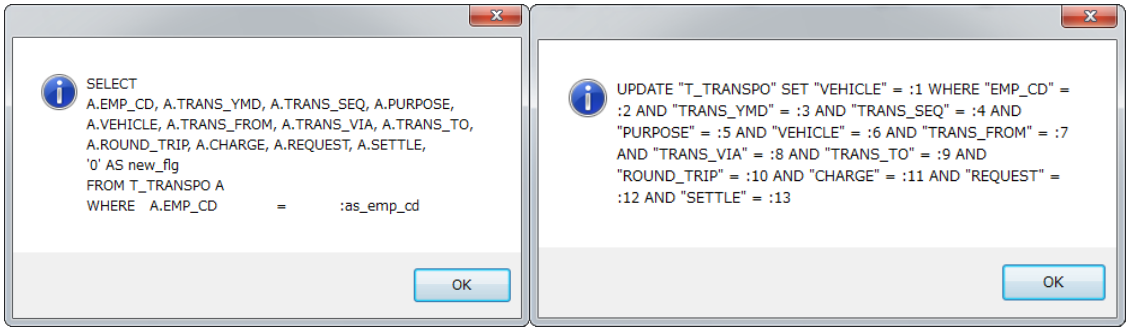
引数のsqlsyntaxにてSQL構文を確認することができる。

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。

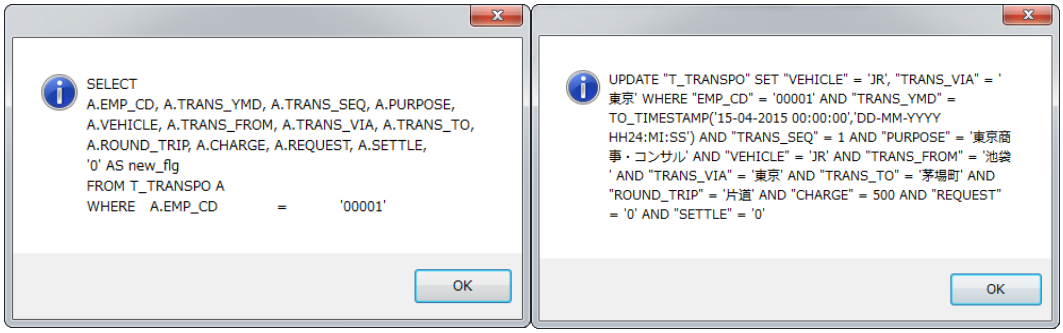
テストにより値の確認が必要な場合は、バインド抑制をONにする。

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】★

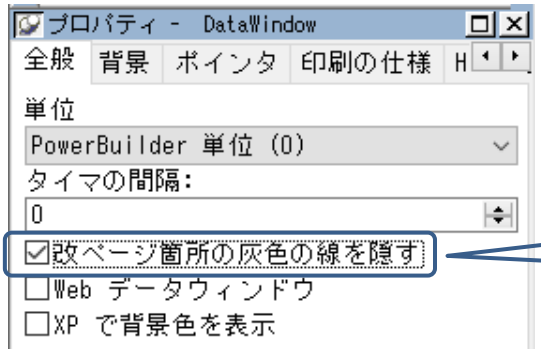



```
// DB接続設定
// 一般的にアプリケーション起動時にDB接続を行う。
SQLCA.DBMS        = "ORA Oracle"
SQLCA.ServerName  = "orcl"
SQLCA.LogId       = "SAMPLE_PB125"
SQLCA.LogPass     = "SAMPLE_PB125"
SQLCA.AutoCommit  = False
//SQLCA.DBParm    = "CommitOnDisconnect='No',NLS_Charset='Local'"
SQLCA.DBParm      = "CommitOnDisconnect='No',NLS_Charset='Local',DisableBind=1"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

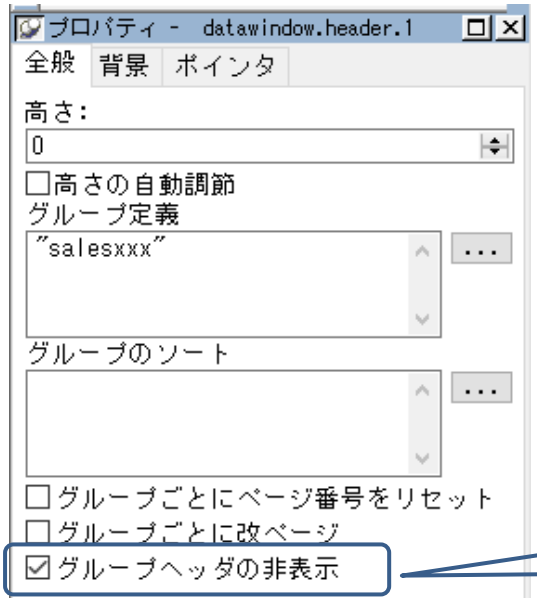
⑦　グループデザインのデータウインドウで縦方向の空欄を消す

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 Appeon 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はなく、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。

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なだけで終わらせず、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