カスタムダイアログの作成(InstallScript プロジェクト)

検証したバージョン： InstallShield 2010 Premier Edition
対象プロジェクト： InstallScript プロジェクト

概要

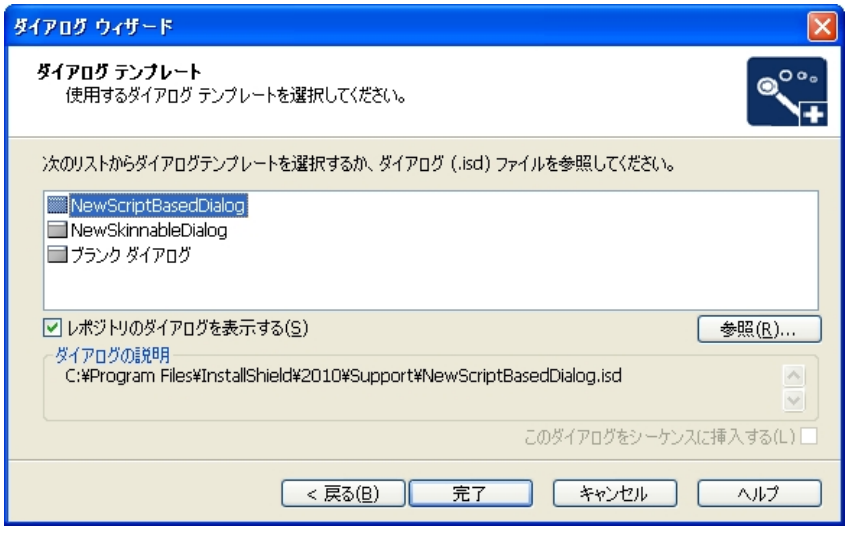
InstallScript プロジェクトにおいて、ユーザ定義のカスタムダイアログを使用する場合、ダイアログビューでのカスタムダイアログレイアウトの作成に加えて、カスタムダイアログのコントロール処理を行う InstallScript コードの記述が必要となります。この記事では、新規のカスタムダイアログを追加して、そのダイアログをインストーラに組み込む場合の簡単な手順について説明します。

A. 新規カスタムダイアログの追加

[ユーザー インターフェイス]－[ダイアログ]ビューにて、新規のダイアログを追加します。

1. [インストール デザイナ]－[ユーザー インターフェイス]－[ダイアログ]ビューにて、すべてのダイアログを右クリックして、コンテキストメニューより「ダイアログの新規作成」を選択します。

2. ダイアログウィザードが表示されますので、ダイアログのテンプレートより「NewScriptBaseDialog」を選び、「完了」ボタンを選択します。

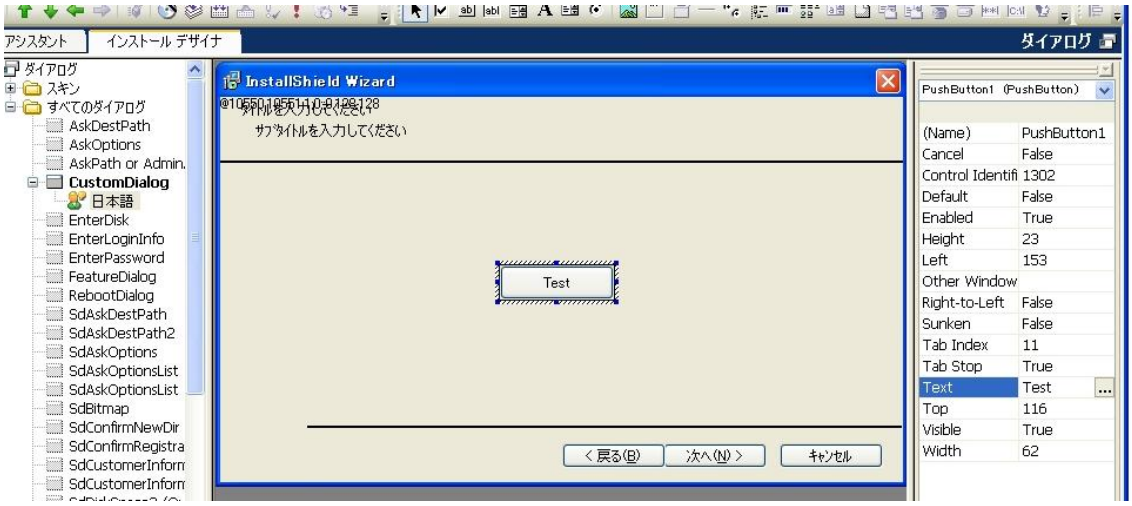


3. ダイアログの一覧にユーザ定義の新規ダイアログ「NewDialog」が追加されますので、任意の名称に変更します。(この手順では「CustomDialog」に設定)

4. ダイアログを編集するためにユーザ定義のダイアログ「CustomDialog」を展開して[日本語]を選びます。

5.右に表示されるダイアログエディタにて、ダイアログの編集を行います。 この手順では、ラベル名を「Test」に設定された PushButton コントロールを新規に追加します。

※PushButton コントロールは、コントロールツールバーの以下のボタンより追加可能です。

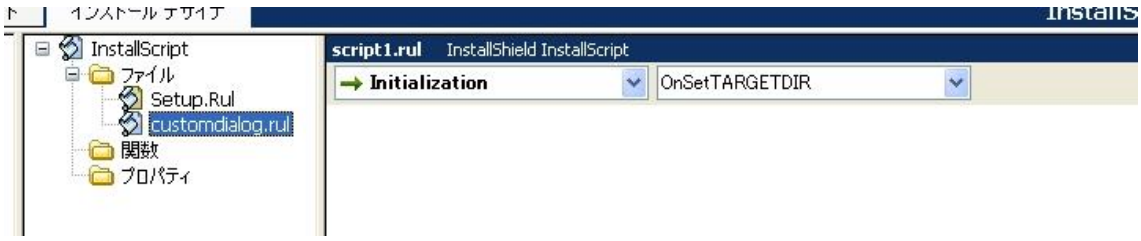


B. カスタムダイアログ用スクリプトファイルの追加

次に、カスタムダイアログの動作を記述するための Rul ファイルを新規に追加します。
※以下の手順で行う関数のサンプルコードは付録として最後に記述してあります。

1.[動作とロジック]－[InstallScript]ビューにて、[ファイル]を右クリックしてコンテキストメニューより「スクリプトファイルの新規作成」を選びます。

2.スクリプトの一覧に新規 Rul ファイル「script1.rul」が追加されるので、任意の名称に変更します。
（この手順では「customdialog.rul」に設定）



3. customdialog.rul を選択します。 右に表示される スクリプトエディタにて、カスタムダイアログの動作の実体となる関数のプロトタイプと実装を追加します。 以下は関数のスケルトンの記述例

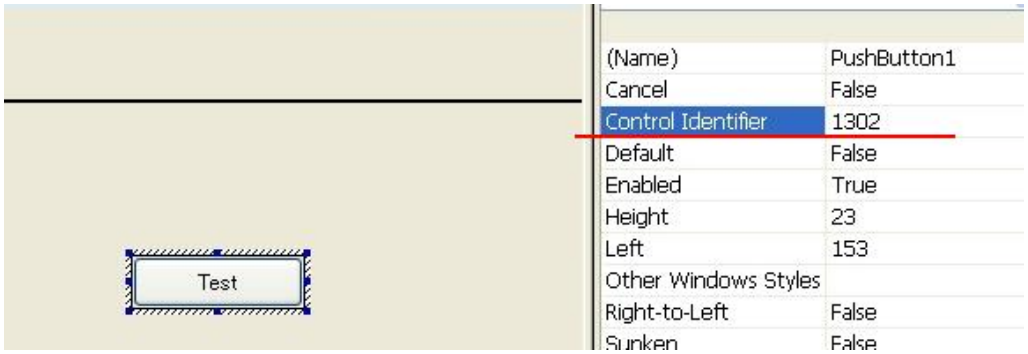
customdialog.rul

```
//プロトタイプ宣言
prototype NUMBER CustomDialog();

//カスタムダイアログを表示する関数 “CustomDialog” の定義
function NUMBER CustomDialog()

begin

end;
```

C. EzDefineDialog 関数 によりカスタムダイアログのロードを行う

手順[A]で新規追加を行ったカスタムダイアログを表示するため、EzDefineDialog 関数によりダイアログのロードを行います。

1. カスタムダイアログ上からの入力を判定するため、コントロール ID の定義を行います。 各コントロールに割り当てられている ID 値は、[ダイアログエディタ]の以下のプロパティにて確認可能です。



（カスタムダイアログに一番初めに配置されるコントロールの既定の ID は 1302 です）

2. [スクリプト]ビューにて、customdialog.rul を選択して、コードの最上部に以下の定義を追加します。

```
//ダイアログ上のコントロールの ID を定義
#define TEST_BUTTON 1302
```

なお、デフォルトで配置されている「戻る」「次へ」ボタンに関しては既存のシステム定数 ***SD_PBUT_OK*** ・***SD_PBUT_BACK*** が使用可能のため定義を行う必要はありません。

3. begin の後ろに EzDefineDialog 関数の呼び出しを行うエントリを追加します。（網掛けされている箇所が追加分です。）

```
begin

//EzDefineDialog 関数によりカスタムダイアログのロードを実行
EzDefineDialog(
      "CustomDialog",//ダイアログの管理名を指定。この名称がダイアログ管理を行う別の関数でも使用される
      ISUSER,        //カスタムダイアログを含む DLL ファイル名。システム定数 ISUSER を指定する
      "CustomDialog",//ダイアログビューで新規作成したカスタムダイアログ名
      NULL);         //NULL で固定
```

D. WaitOnDialog 関数を使用したメッセージループの追加

次に、カスタムダイアログからの入力情報を元に特定の処理を実行する、カスタムダイアログの動作の実体となる、メッセージループを追加します。

1. メッセージループの制御構造となる While 文・Switch 文を追加します。 各文の制御に使用する変数、および while,switch を使用した構文を CustomDialg.rul へ追加します。

```
function NUMBER CustomDialog()

   NUMBER nCtrl,nReturn,nvCustomDialog;
    BOOL bDone;

begin

//EzDefineDialog 関数によりカスタムダイアログのロードを実行
EzDefineDialog(
      "CustomDialog",//ダイアログの管理名を指定。この名称がダイアログ管理を行う別の関数でも使用される
      ISUSER,        //カスタムダイアログを含む DLL ファイル名。システム定数 ISUSER を指定する
      "CustomDialog",//ダイアログビューで新規作成したカスタムダイアログ名
      NULL);         //NULL で固定

//メッセージループ
while(!bDone)

    //ステータスにより処理を分岐
    switch(nCtrl)

    endswitch;//switch 終了

endwhile;//ループ終了
```

2. WaitOnDialog 関数 の呼び出しを追加

ループの開始時に WaitOnDialog 関数を追加して、ダイアログからの入力を受け取ります。While 文の開始直後に以下のコードを追加します。

```
//メッセージループ
while(!bDone)

//EzDefineDialog 関数でロードしたダイアログからの情報を取得
nCtrl = WaitOnDialog("CustomDialog");

    //ステータスにより処理を分岐
    switch(nCtrl)
```

3. switch 文に各ステータスに対応した処理を追加

WaitOnDialog 関数により取得したダイアログのステータスは、変数 **nCtrl** へ格納されますので、この値を使用して、各ステータスに対応した処理を switch 文内に追加します。例えば、カスタムダイアログ上に配置したボタン「Test」が押された場合の動作は以下のように定義します。

```
    //ステータスにより処理を分岐
    switch(nCtrl)

        case TEST_BUTTON: //Test ボタンが押された場合
            MessageBox("Test ボタンが押されました",INFORMATION);

    endswitch;//switch 終了
```

以下はその他デフォルトで配置されているボタンに対する処理を追加した場合のサンプルコードです。

```
//ステータスにより処理を分岐
switch(nCtrl)
    case TEST_BUTTON: //Test ボタンが押された場合
        MessageBox("Test ボタンが押されました",INFORMATION);

    case SD_PBUT_OK: // 次へ ボタンが押された場合
        nReturn = SD_PBUT_OK;
        bDone = TRUE; //ループ終了フラグ ON

    case SD_PBUT_BACK: // 戻る ボタンが押された場合
        nReturn = SD_PBUT_BACK;
        bDone = TRUE; //ループ終了フラグ ON

    case SD_PBUT_CANCEL: // 閉じる ボタン(右上の×ボタン)が押された場合
        MessageBox("閉じる ボタンが押されました",INFORMATION);
        Do(EXIT);

     default:
            // デフォルト動作
            if(SdIsStdButton( nCtrl ) && SdDoStdButton( nCtrl )) then
                bDone = TRUE;
            endif;
endswitch; //switch 終了
```

4. ダイアログの終了・メモリの開放追加

最後にメッセージループを抜けた後の処理を追加します。 メッセージループの終了後通常ダイアログは別のダイアログ処理に遷移しますので、ダイアログ自体の終了およびメモリの開放を行う必要があります。 ダイアログの終了およびメモリの開放を行うため、EndDialog・ReleaseDialog 関数の呼び出しをループの終了後(endWhile; の後)に追加します。

```
endswitch;
endwhile;

//ダイアログの終了処理、メモリの開放
EndDialog("CustomDialog");
ReleaseDialog("CustomDialog");
```

5. Return 文 を追加

カスタムダイアログの最終選択ステータスを戻り値として返却するため(「次へ」ボタンによりメッセージループを抜けたのか、「戻るボタンにより」 メッセージループを抜けたのか 等） カスタム関数 CustomDialog の最後に return 文を追加します。

```
//ダイアログの終了処理、メモリの開放
EndDialog("CustomDialog");
ReleaseDialog("CustomDialog");

//値の返却
return nReturn;
```

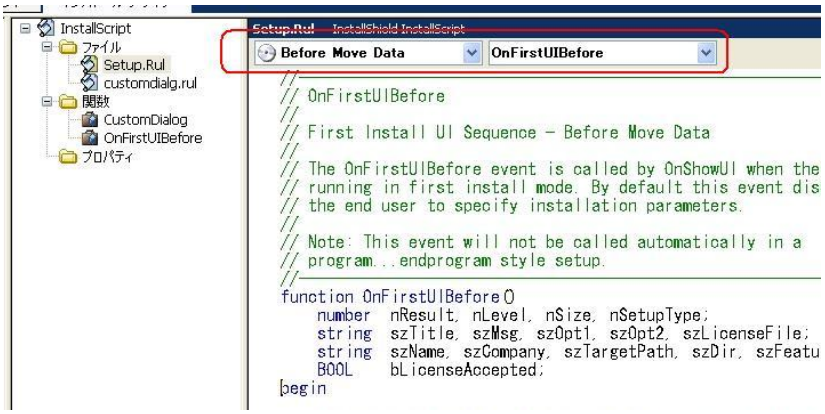
※変数 **nReturn** は switch 内でダイアログからの入力ステータスが代入された変数です

E. 作成したカスタムダイアログを表示するエントリを追加

手順B～Dによりユーザー定義のカスタムダイアログ関数（CustomDialog）が作成されましたので、次にInstallScript のダイアログの呼び出しを行っているイベント(OnFirstUIBefore)に、カスタムダイアログ関数 CustomDialog の呼び出しを行うコードを追加します。

1. [動作とロジック]－[InstallScript]ビューにて、デフォルトの Rul ファイル[Setup.Rul]を選択します。

2. 右に表示されるスクリプトエディタ上部のコンボボックスを[Before Move Data]－[OnFirstUIBefore]に切り替えます。



3. イベント OnFirstUIBefore がコードに追加されることを確認します。 このままの状態では別の rul ファイルに定義された関数の呼び出しを行うことができないため、Setup.rul ファイルの上部の include 文の下に以下の定義を追加します。

```
#include "ifx.h"
//カスタムダイアログ用スクリプトファイルのインクルード
#include "CustomDialog.rul"
```

4. ユーザー定義のカスタムダイアログ関数 CustomDialog の呼び出しを OnFirstUIBefore 追加します。この手順では、SdWelcome 関数により一番初めのダイアログが表示された後にカスタムダイアログが表示されるように、以下の箇所にコードを追加します。

```
                                                    :
                                                    :
Dlg_SdWelcome:
      szTitle = "";
      szMsg = "";
      //{{IS_SCRIPT_TAG(Dlg_SdWelcome)
      nResult = SdWelcome( szTitle, szMsg );
      //}}IS_SCRIPT_TAG(Dlg_SdWelcome)
      if (nResult = BACK) goto Dlg_Start;

//ユーザー定義のカスタムダイアログ関数 CustomDialog の呼び出しを追加
Dlg_CustomDialog:
            nResult = CustomDialog();
            if (nResult = SD_PBUT_BACK ) goto Dlg_SdWelcome;

Dlg_SdLicense2:
      szTitle = "";
                                                    :
                                                    :
```

F. 動作のテスト

カスタムダイアログを追加する場合の基本的な手順は以上となります。 実際にビルドを行い、カスタムダイアログの動作を確認します。

1. ツールバーの[ビルド]ボタンをクリックして(手順 2 の画像の赤い丸のついたボタン)ビルドを実行します。ビルドは[F7]キーからも実行可能です。

2. ツールバーの[実行]ボタンをクリックしてビルドしたインストーラを実行します。(画像の青い丸の付いたボタン)

3. 起動されたインストーラを「次へ」ボタンにより進めます。 一番初めに表示されるようこそダイアログの次のタイミングで作成を行ったカスタムダイアログが表示されることを確認します。また、配置したボタンをクリックした際に、Switch 文内で定義した処理が行われることも確認します。

G. サイレントモード時の動作の設定

以下は応用的な内容となりますが、InstallScript インストーラがサイレントモードで動作した場合に、カスタムダイアログがどのように動作するか(どのように選択されたものとして実行を行うか)を記述する方法についてです。以下は、サイレントインストール時のカスタムダイアログの動作を定義するサンプルコードとなります。

```
Begin
//Begin 直後に挿入
//---------------サイレントモード時の動作--------------------------
if(MODE=SILENTMODE) then

        //SdMakeName 関数により iss ファイルより読み出しを行うエントリを指定
        SdMakeName( szAppKey, "CustomDialog", "" , nvCustomDialog );

        //SilentReadData 関数により iss ファイル読み出しを行う箇所を指定して、値を取得
        //この場合、3 番目の引数に DATA_NUMBER を指定しているため、nCtrl に値が返却される
        SilentReadData( szAppKey, "Result", DATA_NUMBER, szTemp, nCtrl );

        return nCtrl;//サイレントモード時の動作の終了後 カスタム関数を終了する
endif;
//-------------------------------------------------------------

//EzDefineDialog 関数によりカスタムダイアログのロードを実行
EzDefineDialog(
```

1. SdMakeName 関数により応答ファイル(iss)ファイルより読み出しを行うセクション名を追加します。
   SdMakeName 関数は、このダイアログが応答ファイル内のどのセクションから値を取得するかを指定するために使用されます。

2. SilentReadData 関数により指定したセクションから値の読み出しを行う。
   SilentReadData 関数は SdMakeName 関数にて指定したセクションより値の取得を行う関数です。
   サイレントモード時は通常この関数を使用して、応答ファイルに記載されたダイアログの動作の結果を取得します。

3. ダイアログが表示される前に return 文によりカスタム関数を終了させます。
   サイレントインストール時の動作は、手順 C～E より追加を行ったカスタムダイアログの表示箇所よりも前に配置する必要があります。また、サイレントインストール時の動作の記述が終わった後は、サイレントインストール時にカスタムダイアログの表示が行われてしまわないように return 文により関数を終了させる必要があります。

H. iss ファイル記録モード時の動作の設定

InstallScript 形式インストーラにて応答ファイル(iss ファイル)の生成を行う場合、テキストエディタ等を使用して直接作成する方法と、/r オプションを指定して各ダイアログの選択状態が記録された応答ファイルを自動生成する方法があります。作成したカスタムダイアログを /r オプションの記録モードに対応させる場合、専用のコードを追加する必要があります。以下は記録モード時にダイアログの選択状態を応答ファイルへ記録するサンプルコードになります。

```
//ダイアログの終了処理、メモリの開放
EndDialog("CustomDialog");
ReleaseDialog("CustomDialog");

//------------------iss ファイル記録モード時の動作-----------------
//return により関数が終了する前の段階でダイアログの状態を記録します。
if(MODE=RECORDMODE) then

    //SdMakeName 関数により書き込みを行う iss ファイル内のエントリを指定
    SdMakeName( szAppKey, "CustomDialog", "", nvCustomDialog );

    // SilentWriteData によりデータの書き込みを実行。
    // この場合 変数 nCtrl( ダイアログの選択状態 を保持)が記録される
    SilentWriteData( szAppKey, "Result", DATA_NUMBER, "", nCtrl );
endif;
//-----------------------------------------------------------------
```

1. SdMakeName 関数により応答ファイル(iss)ファイルに書き込みを行うセクション名を指定します
   SdMakeName 関数はこのダイアログの動作を応答ファイル内のどのセクションに記録するかを指定するために使用されます。

2. SilentWriteData 関数により指定したセクションにダイアログで選択された内容を記録します。
   SilentWriteData 関数は指定した変数を応答ファイルへ書き込みます。通常は、実際に選択されたダイアログの結果(「次へ」ボタンが押された等)を保持する変数(上記例では nCtrl)を書き込み対象として指定します。ダイアログ上にその他選択が必要なコントロール(ラジオボタン等)が配置されている場合も、SilentWriteData 関数を別途呼び出して、適宜記録を行います。

I. 付録(サンプルコード全文)

customdialog.rul

```
//ダイアログ上のコントロールの ID を定義
#define TEST_BUTTON 1302

//プロトタイプ宣言
prototype NUMBER CustomDialog();

//カスタムダイアログを表示する関数の定義
function NUMBER CustomDialog()

NUMBER nCtrl,nReturn,nvCustomDialog;
BOOL bDone;

STRING szTemp;

begin
//Begin 直後に挿入
//---------------サイレントモード時の動作---------------------------
if(MODE=SILENTMODE) then

        //SdMakeName 関数により iss ファイルより読み出しを行うエントリを指定
        SdMakeName( szAppKey, "CustomDialog", "" , nvCustomDialog );

        //SilentReadData 関数により iss ファイル読み出しを行う箇所を指定して、値を取得
        //この場合、3 番目の引数に DATA_NUMBER を指定しているため、nReturn に値が返却される
        SilentReadData( szAppKey, "Result", DATA_NUMBER, szTemp, nCtrl );

        return nCtrl;
endif;
//-----------------------------------------------------------------

//EzDefineDialog 関数によりカスタムダイアログのロードを実行
EzDefineDialog(
    "CustomDialog",//ダイアログの管理名を指定。この名称がダイアログ管理を行う別の関数でも使用される
    ISUSER,          //カスタムダイアログを含む DLL ファイル名。システム定数 ISUSER を指定する
    "CustomDialog",//ダイアログビューで新規作成したカスタムダイアログ名
    NULL);           //NULL で固定

//メッセージループ
while(!bDone)

//EzDefineDialog 関数でロードしたダイアログからの情報を取得
nCtrl = WaitOnDialog("CustomDialog");

//ステータスにより処理を分岐
switch(nCtrl)

    case TEST_BUTTON: //Test ボタンが押された場合
        MessageBox("Test ボタンが押されました",INFORMATION);

    case SD_PBUT_OK: // 次へ ボタンが押された場合
        nReturn = SD_PBUT_OK;
        bDone = TRUE; //ループ終了フラグ ON

    case SD_PBUT_BACK: // 戻る ボタンが押された場合
        nReturn = SD_PBUT_BACK;
        bDone = TRUE; //ループ終了フラグ ON

    case SD_PBUT_CANCEL: // キャンセル ボタンが押された場合
        MessageBox("キャンセル ボタンが押されました",INFORMATION);
        Do(EXIT);

     default:
```

```
            // デフォルト動作
            if(SdIsStdButton( nCtrl ) && SdDoStdButton( nCtrl )) then
                  bDone = TRUE;
            endif;

endswitch;//switch 終了
endwhile;//ループ終了

//ダイアログの終了処理、メモリの開放
EndDialog("CustomDialog");
ReleaseDialog("CustomDialog");

//-------------------iss ファイル記録モード時の動作------------------
if(MODE=RECORDMODE) then

      //SdMakeName 関数により書き込みを行う iss ファイルを指定
      SdMakeName( szAppKey, "CustomDialog", "", nvCustomDialog );

      // SilentWriteData によりデータの書き込みを実行
      SilentWriteData( szAppKey, "Result", DATA_NUMBER, "", nCtrl );
endif;
//-------------------------------------------------------------------

//値の返却
return nReturn;

end;
```