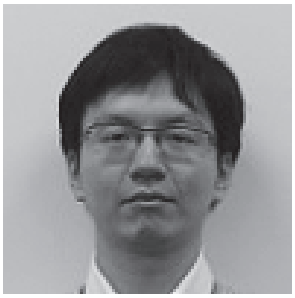
佐田 雄一

株式会社ミガロ.

システム事業部 システム1課

Webコンポーネントのカスタマイズ入門

**カスタムコンポーネントを開発できれば、さらなるWeb開発の効率化が見込める。
VCL for the Webならではのカスタムコンポーネント作成手法を述べる。**



略歴
1985年12月6日生
2009年甲南大学経営学部卒
2009年04月株式会社ミガロ. 入社
2009年04月システム事業部配属

現在の仕事内容
Delphi/400を利用したシステム開発や保守作業を担当。Delphiおよび Delphi/400のスペシャリストを目指して精進している。

1.はじめに

「VCL for the Web（IntraWeb）」はDelphi/400の開発機能の1つである。この機能を用いると、C/S（クライアント/サーバー型）アプリケーションの開発と同様の手法で、Webアプリケーションを作成することができる。

このVCL for the Webを活用することで、Web開発に馴染みの薄い開発者であっても、使い慣れたDelphi/400を用いて、容易にWebアプリケーションを作成することが可能になるだろう。

本稿では、Delphi/400を使用したWebアプリケーション開発のバリエーションをさらに広げていただけるよう、特にコンポーネントのカスタマイズ手法について取り上げ、くわしく紹介する。

2.カスタムコンポーネントを利用するメリット

Delphi/400では、開発機能の1つである“継承”を活用して、既存のコンポーネントから新しいコンポーネントを作成することができる。【図1】

このようにして作成されたコンポーネントは「カスタムコンポーネント」と呼ばれ、要件にあわせてさまざまな追加機能を組み込むことができる。

本稿では、VCL for the Webならではのカスタムコンポーネント作成手法について述べる。

なお、C/Sアプリケーションの開発におけるカスタムコンポーネントの作成手法については、『ミガロ. テクニカルレポート 2012』に、「カスタマイズコンポーネント入門～Delphi/400開発効率向上」を掲載している。わかりやすく解説されているので、ぜひ参考にしていただきたい。

3.Ajax・JavaScriptとの連携

VCL for the Web専用コンポーネントの特徴の1つとして、Ajaxと連携したAsyncイベントが利用できる点が挙げられる。

・Ajax

Ajaxとは「Asynchronous JavaScript + XML」の略で、JavaScriptによって実現する機能の総称である。Ajaxの仕組みを作成することは難しいが、Delphiではイベントで提供されているため、簡単に利用することが可能である。

例えば、C/Sアプリケーションの開発においては、TEditでフォーカスが抜けた際にOnExitイベントが実行される。一方、VCL for the WebアプリケーションのTIWEditにおいては、OnAsyncExitイベントが実行される。このイベントでは、HTML上でフォーカスが抜けたことをJavaScriptで検知し、非同期で処理を行うことが可能である。【図2】

・Asyncイベント

C/Sアプリケーションにおけるイベントの多くは、VCL for the WebアプリケーションではAsyncイベントで代替

図1

図1　コンポーネント継承の流れ

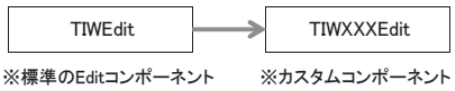


図2

図2　OnAsyncExit時に名称を取得する例

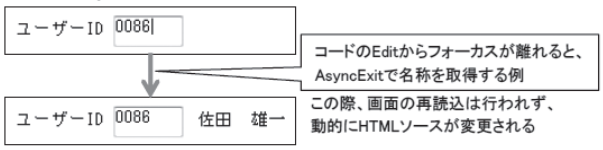


図3

図3　コンポーネントを新規作成する

図4

図4　フレームワークを選択する（Version XE3～）

されている。加えて、これらはAjaxの処理に依存しているため、中には実行タイミングや処理の種類が異なるものも存在する。

例えば、C/Sアプリケーションにおける TEdit の OnChange イベントは、1文字でも値が変更されたタイミングで実行される。それに対し、TIWEditのOnAsyncChangeイベントでは、フォーカスが抜けたタイミングで、値が変更されていれば実行される。

・JavaScript連携

VCL for the Webアプリケーションのコンポーネントは、JavaScriptと密接な関係がある。コンポーネントにJavaScriptを埋め込み、キー押下時やフォーカスセット時などの制御をJavaScriptで行うことが可能である。

コンポーネントのカスタマイズにおいて、実際に埋め込み、制御を行う手法については、次章でコンポーネントを作成しながら解説する。

※ Asynchronous =非同期

4.数値専用WebEditの作成例

カスタムコンポーネントを作成することで、処理の共通化を実現し、開発の効率化を図ることが可能になる。VCL for the Webのカスタムコンポーネントも同様であり、その特徴は前述の通りである。

ここからは、実際にVCL for the Webアプリケーション用のカスタムコンポーネントを作成していこう。

今回は例として、TIWEditを継承して「TIWNumEdit」という数値入力専用のWebEditを作成する。前述のAsyncイベントおよびJavaScriptを活用することで、C/Sアプリケーションのコンポーネントと同等の制御をWebアプリケーションでも実現できるだろう。

(1)コンポーネントの新規作成

C/Sアプリケーションで作成する際と同様に、コンポーネントソースを新規作成する。手順に沿って説明していこう。

なお、今回の開発環境はDelphi/400 Version XE3を使用している。

①コンポーネントの新規作成

メニューの［ファイル｜新規作成｜その他］を選択し、新規作成ウィンドウより「コンポーネント」を選択する。【図3】

②フレームワークの指定(Version XE3～)

フレームワーク指定のダイアログでは、VCL for Delphi Win32を選択する。【図4】

③継承元コンポーネントの指定

今回はTIWEditを継承するため、TIWEditを指定する。【図5】

④コンポーネント名とパレットページ、保存先の指定

クラス名に、今回作成するコンポーネント名であるTIWNumEditを指定する。

パレットページ名に、新規追加するコンポーネントパレットのパレットページ名を指定する。なお、今回はデフォルト値のSamplesを使用する。

ユニット名に、今回追加するカスタムコンポーネントのpasファイル名、および保存先を指定する。なお、今回は保存先をC:¥Projects¥TechRep¥SampleNumEdit.pasと設定する。【図6】

⑤コンポーネントの新規作成の完了

完了すると、図7の画面が表示される。【図7】

(2)宣言部の作成

今回の数値入力用WebEditでは、以下の機能を実装したいと思う。

・Value：Edit値を数値として設定/保持するための新規プロパティ。
・Text：既存のTextから変更し、読み取り専用プロパティにする。
・JavaScriptで不必要なキー押下を無効にする。
・フォーカスが抜けた時に、数値でない値がセットされていれば色を変更する。

これらを実現するため、ソース1のように宣言部を記述する。それぞれのプロパティやイベントの詳細については後述する。【ソース1】

① uses

HookEventsイベントの実装に必要なIWRenderContext、IWScriptEvents、色の指定を行うために必要なGraphicsを宣言部のusesに追加する。

② private宣言

入力された数値を内部で保管するための変数FValueをprivate部に宣言する。またSetValue、GetValue、GetTextについては後で自動生成されるため、この時点では記載不要である。

③ protected宣言

ScriptEventsの設定を行うためのHookEventsイベント、およびフォーカスが抜けた際に実行するEditExitイベントをprotected部に宣言する。また、HookEventsは上位クラスから継承されているため、宣言の後ろにoverride;を記述する。

④ public宣言

クラス生成時イベント（constructor）を使用するため、public部に宣言する。

⑤ published宣言

実際に画面に貼りつけた際にオブジェクトインスペクタに表示される変数Valueを宣言する。また、Textは上位クラスから存在するが、writeを指定しない（readのみとする）ことで、読み取り専用のプロパティになる。

各宣言部が完成したら、Ctrl + Shift + ［C］キーを同時に押下することで実装部、および先述したSetValue、GetValue、GetTextが自動生成される。

(3)Createイベントの作成

Createは、コンポーネントの生成時に実行されるイベントである。

画面にコンポーネントを配置した際や、ソース内で動的にコンポーネントを生成した際のプロパティの初期値は、ここで設定した値となる。

記述例については、ソース2を参考にしていただきたい。これらの中で注意が

図5

図5 継承元コンポーネントを指定する

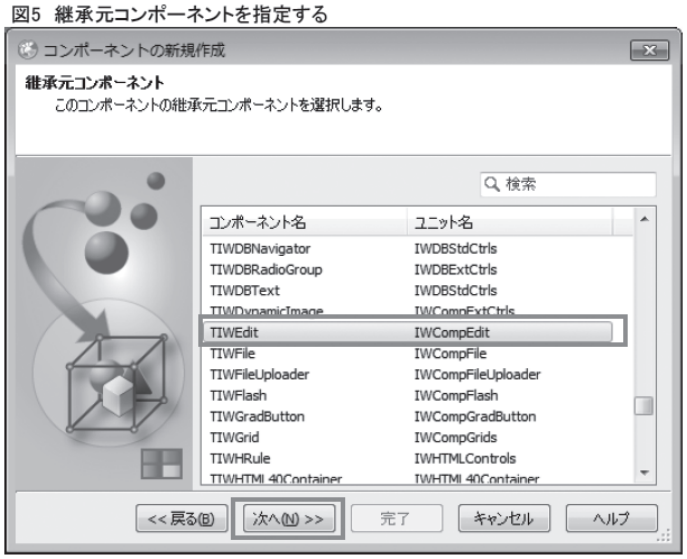


図6

図6 コンポーネント名とパレットページ、保存先を指定する

図7

図7 コンポーネントソース新規作成完了時のソース

必要なプロパティやイベントは、次の2点である。【ソース2】

・Font.FontName
Createイベント内でフォント名を指定する場合、「MSゴシック」のように日本語が入っていると正しく認識しないため、英語で指定する。

・EditExit
OnAsyncExitイベントにEditExitを代入することで、フォーカスが抜けた際にEditExitイベントが実行されるようになる。このため、EditExitイベントの引数はOnAsyncExitと同じものを設定しておく。

(4) HookEventsイベントのコーディング

コンポーネント内でJavaScriptを利用したイベントを実行させることができるScriptEventsプロパティを利用して、キー押下時に、数値入力と無関係のキー押下を無効にする処理を実装する。

これを実現するため、HookEventsイベントにおいて、ScriptEventsプロパティに対してJavaScriptを設定する。なお、本稿ではJavaScript言語の詳細説明については割愛するが、参考文献や解説しているWebサイトが多く存在するので、比較的修得がしやすい言語である。

ソースの記述については、ソース3を参考にしていただきたい。【ソース3】

(5) ValueとTextの読み取り/書き出し時イベントの設定

今回はEditの値を数値として保持させるため、既存のTextを読み取り専用プロパティに変更し、新たに数値型(Double)のValueプロパティを作成して値の入出力を行う。

記述例については、ソース4を参考にしていただきたい。【ソース4】

・GetText
GetTextはTextの値を読み取るためのイベントで、他のクラスからTextを呼び出すと、このイベントを通して値が渡される。Textではなくinherited Textと記述することで、画面にセットされている文字列をそのまま返すことができる。

・GetValue
GetValueはValueの値を読み取るためのイベントで、GetTextと同様に他のクラスからValueを呼び出すと、このイベントを通して値が渡される。

・SetValue
SetValueは逆にValueに値を書き込むためのイベントで、内部保管用のFValue変数および画面のTextにセットされた数値を代入する。

(6) EditExitイベントのコーディング

EditExit手続きを使用することで、Editからフォーカスが抜けた時に、Textに入力した文字列を数値に変換して内部保管値に代入する処理を行う。

その際、コピー&ペースト等によって数値でない文字列がセットされた状態でフォーカスが抜けた場合には、背景色を赤に変更し、内部保管値に初期値(0)を代入する。

また、数値の判定に成功した際は、背景色をもとに戻す処理を記述している。ただし、無色を示すclNoneをBGColorプロパティにセットしても色の変更が行われないため、他の色(今回の場合ウィンドウ色)を指定する必要がある。

ソース記述については、ソース5を参考にしていただきたい。【ソース5】

以上で、TIWNumEditコンポーネントのコーディングは完了である。

(7) インストールと画面への実装

作成したコンポーネントを実際に使用するには、パッケージを新規作成してインストールを行う。手順に沿って説明しよう。

①パッケージを新規作成する
メニューの［ファイル｜新規作成｜その他］を選択し、新規作成ウィンドウより「パッケージ」を選択する。【図8】

パッケージを作成したら、プロジェクトを任意の場所に名前を付けて保存する。(今回作成したユニットと同じディレクトリ内が望ましい。)今回はSamplePackage.bplとして保存する。

なお、既存のパッケージに今回作成したユニットを追加する場合は、ここの手順は不要である。

②パッケージにユニットを追加
次に、メニューの［プロジェクト｜プロジェクトに追加...］を選択し、今回作成したユニットをパッケージに追加する。【図9】

③インストール
図9のプロジェクトマネージャ内にあるSamplePackage.bplを右クリックし、再構築(ビルド)を行う。

再構築が正常に行えたことを確認したら、再度SamplePackage.bplを右クリックし、「インストール」を選択すると、コンポーネントのインストールが実行される。

インストールが正常に完了すると、ダイアログが表示される。【図10】

④画面への組み込み
インストールが完了したら、VCL for the Webの新規プロジェクトを作成すると、デザイン画面のツールパレットにTIWNumEditが追加される。【図11】

TIWNumEditを選択して画面内をクリックすると、図12のような項目が配置される。見た目はカスタム前のTIWEditと異なっているが、TIWEditと同じようにプロパティやイベントの設定を行うことが可能である。【図12】

⑤ライブラリパス設定と実行
作成したカスタムコンポーネントを使用したプロジェクトのコンパイルが通るようにするには、Delphi/400開発環境のライブラリパスの設定を行う必要がある。

メニューの［ツール｜オプション］を選択し、今回作成したSampleNumEdit.pasの保管先ディレクトリをライブラリパスの一覧に追加する。【図13】

ここまでの手順で、今回作成したTIWNumEditを使用する準備はすべて完了である。

アプリケーションにコンポーネントを組み込んでコンパイルすれば、TIWNumEditの機能動作を確認するこ

ソース1

ソース1 宣言部の記述

```
unit SampleNumEdit;

interface

uses
  System.SysUtils, System.Classes, Vcl.Controls, IWVCLBaseControl,
  IWBaseControl, IWBaseHTMLControl, IWControl, IWCompEdit,
  IWRenderContext, IWScriptEvents, Graphics;

type
  TIWNumEdit = class(TIWEdit)
  private
    { Private 宣言 }
    FValue: Double;                                          // 実際に値を保管している変数
    procedure SetValue(const Value: Double);                 // Valueに書き込む時の処理
    function GetValue: Double;                               // Valueが読み取られる時の処理
    function GetText: String;                                // Textが読み取られる時の処理
  protected
    { Protected 宣言 }
    procedure HookEvents(AContext: TIWPageContext40;
                         AScriptEvents: TIWScriptEvents); override; // JavaScriptの設定
    procedure EditExit(Sender: TObject; EventParams: TStringList); // フォーカスが抜けた時の処理
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;        // コンポーネント生成時処理
  published
    { Published 宣言 }
    property Value: Double read GetValue write SetValue;     // 数値として値保持
    property Text: String read GetText;                      // 既存のTextを読み取り専用にする
  end;
```

HookEventsで使用するため、usesに追加

色(TColor)を使用するため、usesに追加

writeを指定しないことで、Textが読み取り専用になる

ソース2

ソース2 Createイベントの記述

```
constructor TIWNumEdit.Create(AOwner: TComponent);
begin
  inherited;
  Alignment      := taRightJustify; // 右揃え(Version2009以降で対応)
  Font.FontName  := 'MS GOTHIC';    // フォント名(英語で指定)
  Font.Size      := 10;             // フォントサイズ
  OnAsyncExit    := EditExit;       // OnAsyncExitのタイミングでEditExitを実行
  Self.BGColor   := clWindow;       // 背景色の初期値(ウインドウ色)
  FValue         := 0;              // 実数値の初期値
  inherited Text := '0';            // 画面に表示する文字列の初期値
end;
```

ソース3

ソース3 HookEventsイベントの記述

```
procedure TIWNumEdit.HookEvents(AContext: TIWPageContext40;
  AScriptEvents: TIWScriptEvents);
const
  // OnKeyDownスクリプトイベント
  // 入力可能文字列を指定する関数
  sScript='m = String.fromCharCode(event.keyCode); '
    +'if("0123456789abcdefghi¥b¥r¥t".indexOf(m, 0) >= 0){ '
    +'  return true; } '
    +'else if (event.keyCode == 46){ '  // Deleteキー
    +'  return true; }'
    +'else if (event.keyCode == 39){ '  // 右方向キー
    +'  return true; }'
    +'else if (event.keyCode == 37){ '  // 左方向キー
    +'  return true; } '
    +'else if (event.keyCode == 96){ '  // テンキーの0
    +'  return true; } '
    +'else if (event.keyCode == 190){ ' // 小数点
    +'  return true; }'
    +'else if (event.keyCode == 110){ ' // テンキーの小数点
    +'  return true; } '
    +'else if (event.keyCode == 189){ ' // マイナス
    +'  return true; } '
    +'else if (event.keyCode == 109){ ' // テンキーのマイナス
    +'  return true; } '
    +'else if (event.ctrlKey == true && event.keyCode == 67){ ' // Ctrl+C
    +'  return true; } '
    +'else if (event.ctrlKey == true && event.keyCode == 86){ ' // Ctrl+V
    +'  return true; } '
    +'else if (event.ctrlKey == true && event.keyCode == 88){ ' // Ctrl+X
    +'  return true; } '
    +'else if (event.ctrlKey == true && event.keyCode == 90){ ' // Ctrl+Z
    +'  return true; } '
    +'else{ return  false; } ';
begin
  inherited;
  // 編集不可の場合処理しない
  if (not Editable) or (not Enabled) or (ReadOnly) then
  begin
    Exit;
  end;

  // OnKeyDown時に、スクリプトイベントを実施する
  AScriptEvents.HookEvent('OnKeyDown', sScript);
end;
```

押下されたキーのキーコードやCtrlキーの押下有無をもとに判定を行う。 図6～図7と同様、trueが返らなかった場合は以降の処理が行われない

ここでScriptEventsをセット

とができるだろう。【図 14】

5.まとめ

近年、Web アプリケーションやスマートデバイス用アプリケーションへの需要が高まり、C/S アプリケーションだけではなく、Web アプリケーションでのシステム構築が多くなってきている。

今回は Web のカスタムコンポーネント開発手法を説明したが、カスタムコンポーネントを開発できるようになれば、さらなる Web 開発の効率化が見込めるだろう。

Web アプリケーション開発に難しさを感じておられる開発者の皆様にとって、本稿が新たなアプリケーション開発の一歩となる内容であれば幸いである。

M

ソース4

ソース4　GetText/GetValue/SetValueの記述

```
function TIWNumEdit.GetText: String;
begin
  Result := inherited Text; // 画面のTextの文字値を渡す
end;

function TIWNumEdit.GetValue: Double;
begin
  Result := FValue; // 実際に値を保管している変数FValueの値を返す
end;

procedure TIWNumEdit.SetValue(const Value: Double);
begin
  inherited Text := FloatToStr(Value); // 画面のTextに代入された値を表示
  FValue := Value; // 実際に値を保管している変数FValueに代入
end;
```

他のユニット等から「Text」プロパティが参照されると、この関数を通じて値を返す

他のユニット等から「Value」プロパティが参照されると、この関数を通じて値を返す

他のユニット等から「Value」プロパティに値が代入されると、この関数を通じてセットする

ソース5

ソース5　EditExit手続きの記述

```
procedure TIWNumEdit.EditExit(Sender: TObject; EventParams: TStringList);
var
  dCheck: Double; // 値変換用変数
begin

  if (not TryStrToFloat(inherited Text, dCheck)) then
  begin
    Self.BGColor := clRed;    // 入力値が数値でない場合、背景色を赤に変更
    FValue := 0;              // 実際に値を保管している変数FValueに0を代入
  end
  else
  begin
    Self.BGColor := clWindow; // 入力値が数値の場合、背景色をウインドウ色に変更
    FValue := dCheck;         // 実際に値を保管している変数FValueに入力値を代入
  end;
end;
```

TryStrToFloat関数は、実数変換成功時はTrueが返り、dCheckに変換値が代入される
変換失敗時はFalseが返る

図8

図8　パッケージの新規作成

図9

図9　パッケージにユニットを追加

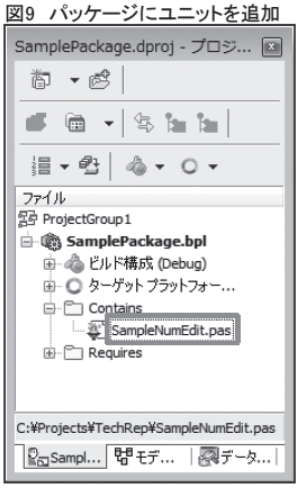


図10

図10　インストール完了

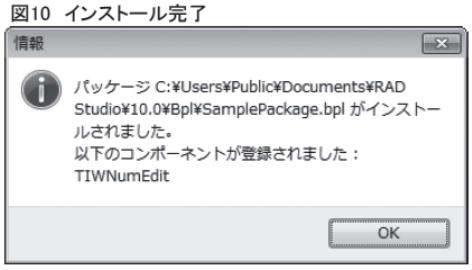


図11

図11　ツールパレット

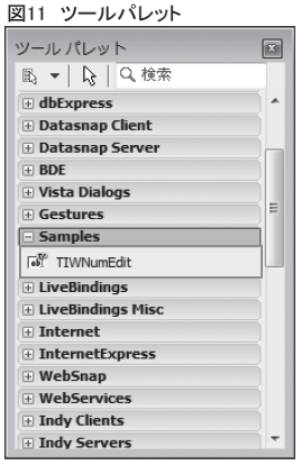
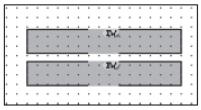


図12

図12　コンポーネント配置例



※カスタムコンポーネントを配置すると、
デザイン画面上の見た目が異なる

図13

図13 ライブラリパス設定

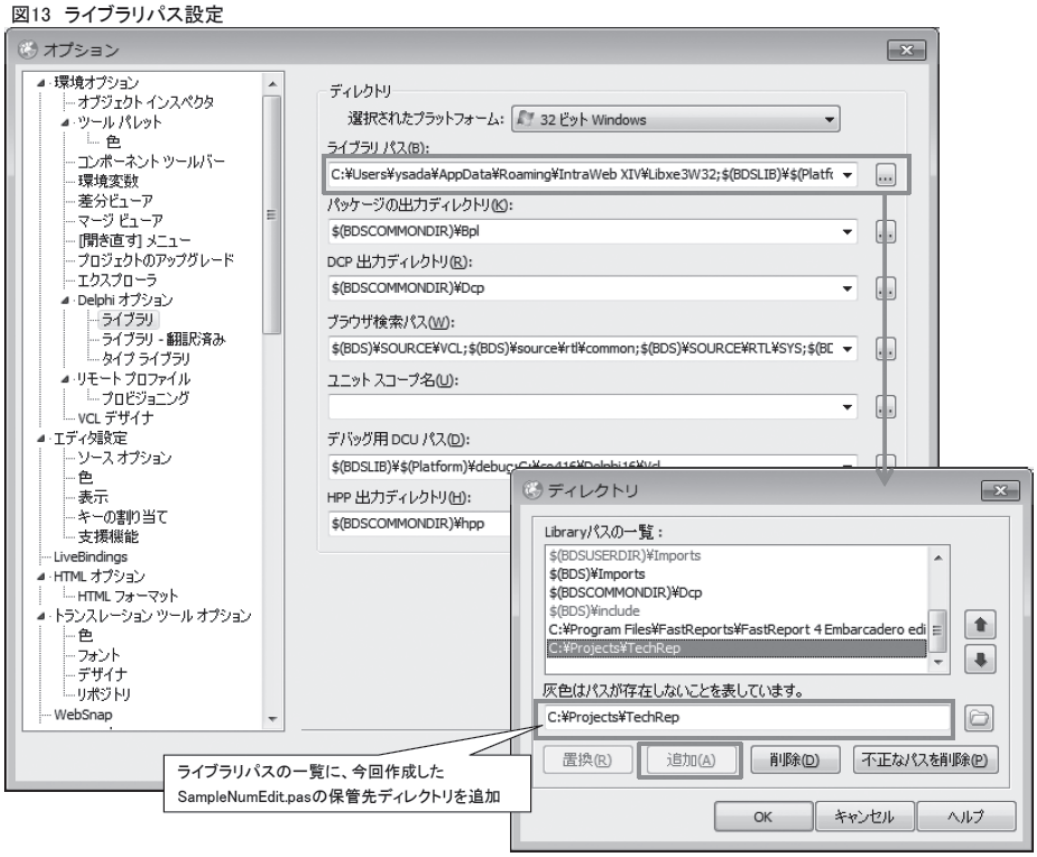
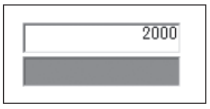


図14

図14 コンポーネント実行例



※ブランクは数値でないため、色が変化する