Delphi/400

新規VCLコントロールを利用した ユーザーインターフェース改善テクニック

株式会社ミガロ.
システム事業部 システム2課
畑中　侑

略　歴

生年月日:1983年7月6日
最終学歴:2006年　京都産業大学 法学部卒業
ミガロ入社年月:2006年4月　株式会社ミガロ. 入社
社内経歴:2006年4月　システム事業部配属

現在の仕事内容:

システムの受託開発を担当しており、
要件確認から納品・フォロー、
保守作業に至るまで、
システム開発全般に携わっている。

1.はじめに

Windows7のサポート終了に伴い、Windows10対応を検討し、既存アプリケーションのバージョンアップを実施する事が非常に多い。アプリケーションをバージョンアップする際にソース変更作業を伴う場合は、バージョンアップ前後でアプリケーションの動作に差異がないかチェックする。Delphi/400アプリケーションのバージョンアップ作業を経験された方は既にご存知のことと思うがWindows10への正式対応版Delphi/400 10Seattle以前のアプリケーションをバージョンアップする際には、ソース変更作業が伴うため新環境での動作が以前の環境と同じ挙動かどうか確認されたことと思う。バージョンアップ作業はシステム担当者から見ると、アプリケーションをシステム基盤の最新OSに対応するという重要なミッションであるが、ユーザー目線からするとどうだろうか。使い慣れたアプリケーションではあるが、以前と同じ見た目、動作ではバージョンアップの恩恵を感じにくいのではないだろうか。
そこで本稿では、既存アプリケーションのコンポーネントを、Delphi/400 10Seattleから追加されている新規VCLコントロールに置き換えることで、ユーザーにバージョンアップの効果を実感してもらえるよう、簡単なUI(ユーザーインターフェース)改善テクニックをご紹介する。

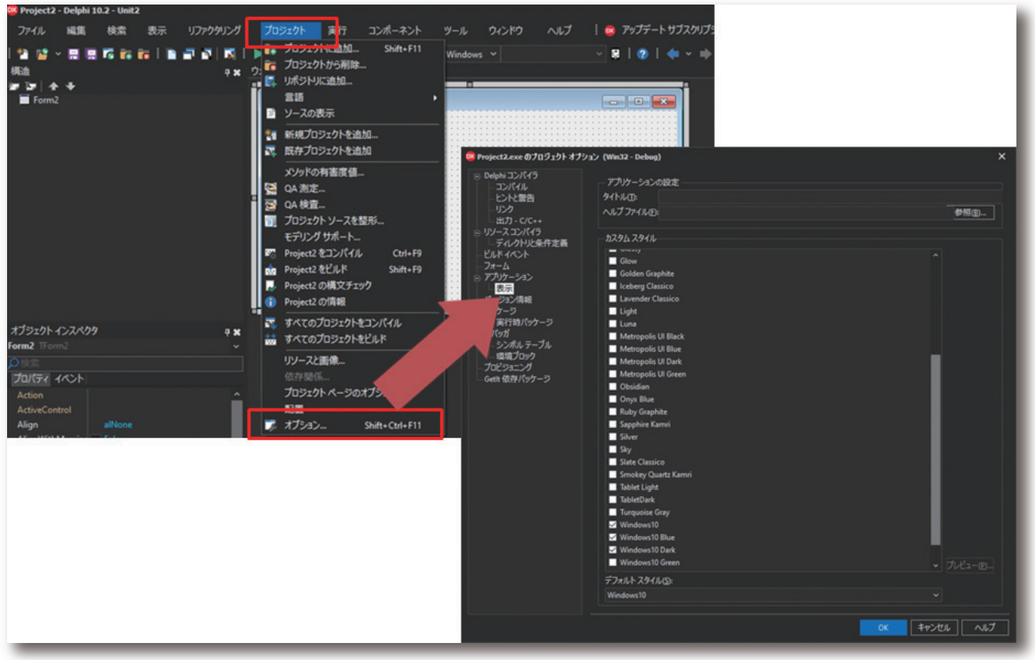
2. UI(ユーザーインターフェース)の改善

UIの改善といっても、大きく分けてビジュアル(視覚的)な改善と使い勝手に起因する操作性の改善が考えられる。本稿ではビジュアル的な改善テクニック2つと操作性に関連したレイアウト関する改善テクニックを説明する。
なお、ここではDelphi/400 10.2Tokyoを用いて操作方法をご紹介することとする。

2-1:アプリケーションのスタイル変更

まずは簡単なUI改善の一つとして手軽に行える視覚的な改善方法をご紹介する。
Windows環境で実行されるアプリケーションにはスタイルという概念がある。バージョンアップにて単にアプリケーションをコンバージョンした場合はWindows10環境で実行しても以前と同じ旧スタイルが使用され、ユーザー側からすると、きれいなビジュアル内に古いアプリケーションが起動され違和感を抱く一因になる。Delphi/400 XE3以降のバージョンからアプリケーションに適用されるスタイルを変更することができるため、これを用いることでWindows10ベースのスタイルやその他のスタイルに既存アプリケーションを変更しユーザーのもつイメージを変えることができる。
変更方法は次の手順となる。
開発環境より|プロジェクト|オプション|と順に操作し表示されるプロジェクトオプション画面にて【アプリケーション】配下にある【表示】にある「カスタム　スタイル」の該当項目にチェックを入れることで「デフォルトスタイル」の選択肢が増える。これを変更することでアプリケーション全体のスタイルを変更できる。**【図1】**

図 1　カスタムスタイルの設定



スタイル変更時の適用イメージは「カスタム スタイル」の該当項目を選択し「プレビュー」ボタンを押下すると確認できる。既存アプリケーションについても、このプロジェクトオプションより「カスタム　スタイル」を変更しコンパイルすると適用でき視覚的なイメージを一新した効果をユーザーに提供することが可能となる。

また、このスタイル変更はTStyleManager.SetStyleメソッドを用いることでソースでの記述も可能である。例えば既存アプリケーションのログイン画面や初期起動画面に、このスタイルをユーザーが選択できるような仕組みを組み込むことで、手軽にユーザーの好みにあったスタイルを提供することができる。実装方法は次の通りである。ここの実装例ではスタイルの選択方法はTComboBoxを用いたリストから行うことを想定している。あらかじめ、スタイルを選択し変更処理を搭載する画面のuses節に「Vcl.Themes」を追加する。これはスタイルを管理するクラスを用いるためである。**【ソース1】**

ソース 1

uses節への追加

```
unit Before2Frm;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes,
  Vcl.Graphics,  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,
  Vcl.WinXPanels, Data.DB, Datasnap.DBClient, Vcl.Grids, Vcl.DBGrids, Vcl.Menus,
  Vcl.CategoryButtons, Vcl.Buttons, Vcl.Themes;
```

次にTComboBoxのリストに選択肢を追加しておく。(実装例ではフォーム生成時のイベントOnFormCreateイベントで実装している。)スタイルの候補値はTStyleManager.StyleNamesにアクセスすることで得られるため、TComboBoxのItemsプロパティに加える。また現在適用されているスタイルはTStyleManager.ActiveStyle.Nameで取り出すことができるため、リストの初期選択値として利用する。**【ソース2】**

ソース 2

カスタムスタイルのリスト化

```
procedure TForm1.FormCreate(Sender: TObject);
var
  StyleName: string;
begin
  // 画面スタイルのリスト作成
  for StyleName in TStyleManager.StyleNames do
    cbxVclStyles.Items.Add(StyleName);

  // 現在スタイルを初期選択とする
  cbxVclStyles.ItemIndex := cbxVclStyles.Items.IndexOf(TStyleManager.ActiveStyle.Name);

end;
```


最後にこのTComboBoxでリストを選択した際に実行されるOnChangeイベントに、選択値を指定したTStyleManager.SetStyleメソッドを実行すれば完了である。**【ソース3】**

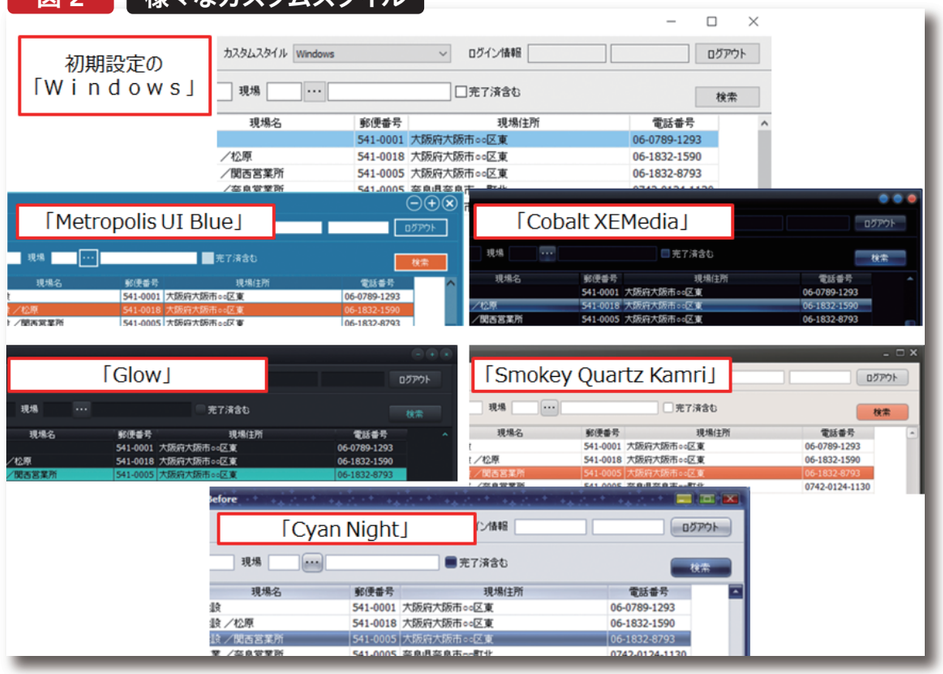
ソース 3

カスタムスタイルの適用

```
procedure TForm1.cbxVclStylesChange(Sender: TObject);
Begin
  // 選択スタイルの適用
  TStyleManager.SetStyle(cbxVclStyles.Text);
end;
```

上記の実装したものを実行すると、リストにてスタイルを変更すると画面全体がその選択されたスタイルに変更されることが確認できる。**【図2】**

図 2　様々なカスタムスタイル



スタイルを変更することで、画面の色味が変わり、それに合わせてコントロールのフォーカス時の反転カラーやタイトルバー上のボタン形状も変わる。例えば、初期設定のカスタムスタイル「Windows」では、表形式の選択行のカラーは青に対して、「MetropolisUIBlue」や「SmokeyQuartzKamri」では橙色となり、ボタンフォーカス時の反転も橙色となる。タイトルバーの「最小化」「最大化」「閉じる」の各機能ボタンの形状も「Cobalt XEMedia」や「Glow」では丸みを帯びた形となっている。「CyanNight」スタイルでは、タイトルバー部にきらめく大小の星が見て取れる。適用できるカスタムスタイルは40種あり、各々特長が違うため好みのスタイルをぜひ探して頂きたい。

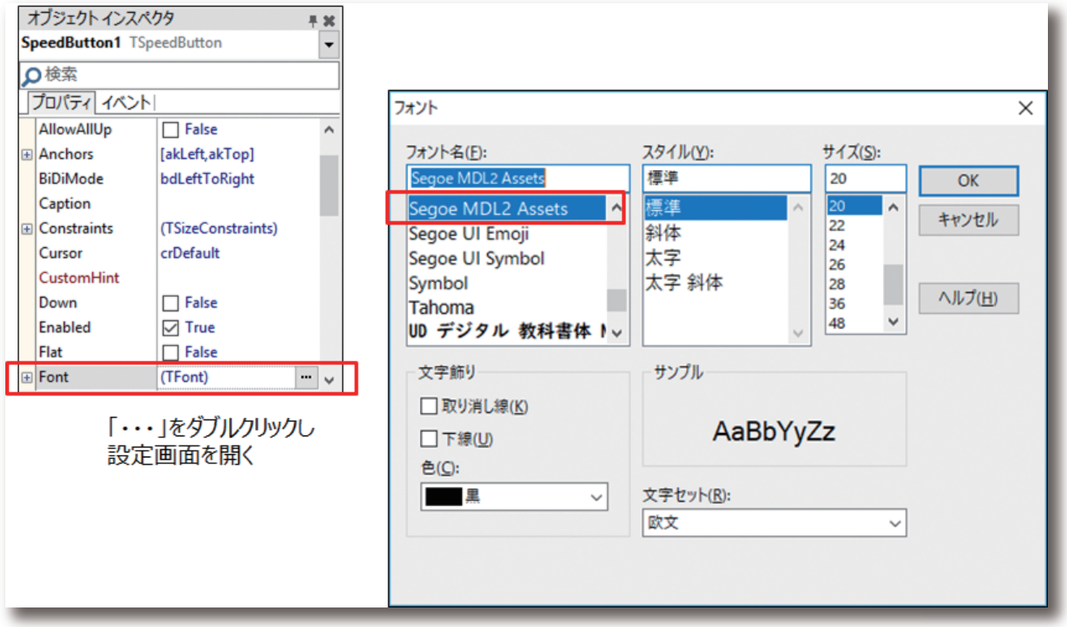
2-2:フォントを利用したアイコン変更

「カスタムスタイル」についてはアプリケーション全体の設定となるが、次は各画面に適用できる、手軽で視覚的な改善方法をご紹介する。

Windows10からは「Segoe MDL2 Assets」という記号群のフォントが追加されている。これを用いることで、今までアイコンイメージで補っていたボタンの機能内容を直感的に提供することができたり、単にアイコンイメージ+機能名の幅に合わせていた項目幅を節約したりできる。具体的な利用方法は、TSpeedButtonのCaptionを例にとって次の手順となる。

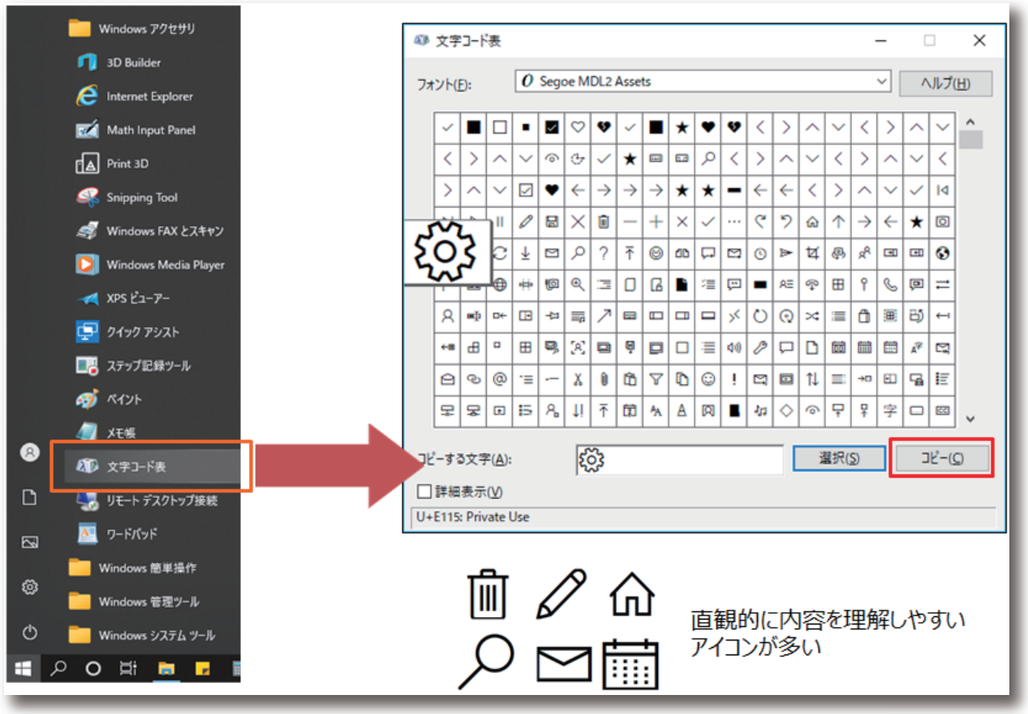
まずはTSpeedButtonのFontプロパティの設定画面にて、「フォント名」に「Segoe MDL2 Assets」を選択する。【図3】

図 3　TSpeed Buttonプロパティの設定



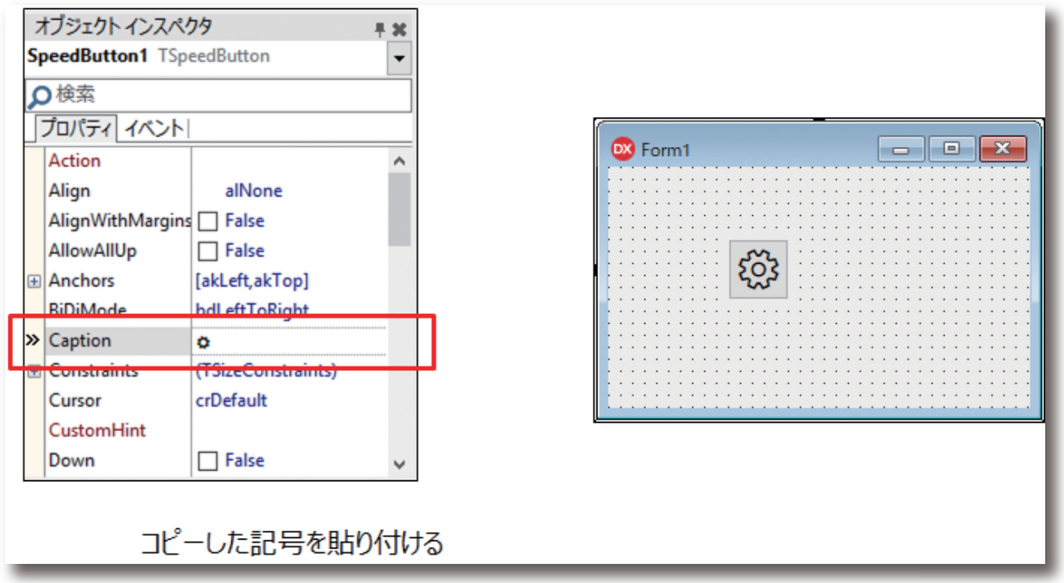
次にWindowsの文字コード表を起動する。方法はWindowsボタンより|Windowsアクセサリ|文字コード表|を選択するとダイアログが表示される。利用したい記号を選択し、ダイアログ上のコピーボタンを押下しクリップボードに保存する。【図4】

図 4　文字コード表の起動



最後にTSpeedButtonのCaptionプロパティにコピーした記号をペーストする。(記号によっては"・"と表示されるものもある)
以上で操作は完了である。【図5】

図 5　コピーした記号の貼り付け



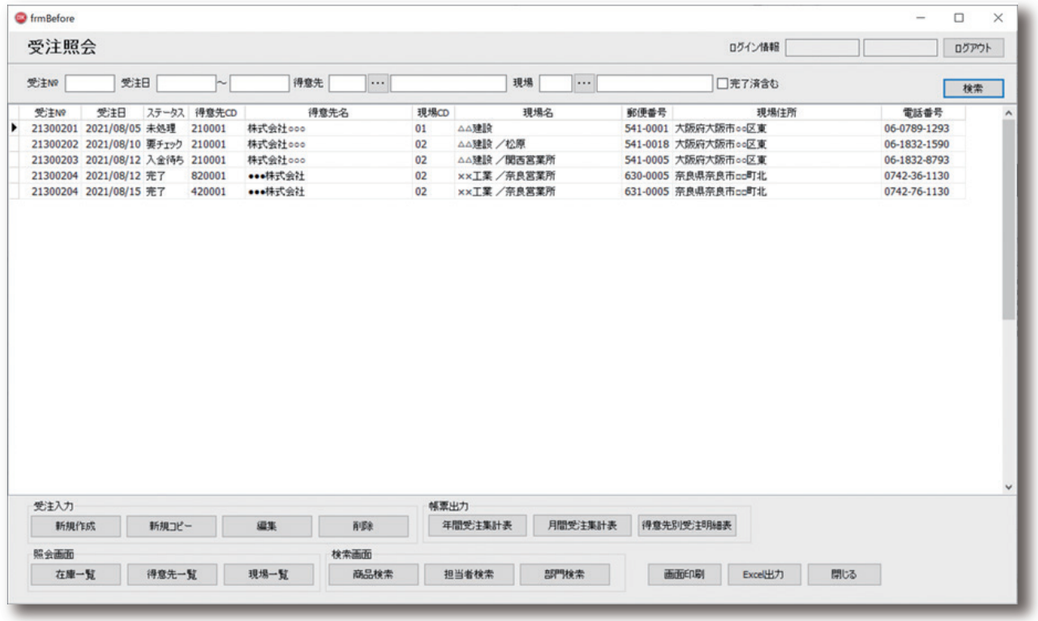
上記の操作で表示される文字コード表を確認すると、多数収録されていることが分かる。従来、削除イメージに用いられてきた「ゴミ箱」アイコンや編集イメージの「鉛筆」アイコン、メイン機能を表す「ホーム」アイコンがいわゆる今風のデザインで用意されている。これらを元に既存アプリケーションで共通的に用いていたアイコンイメージを差し替えたり、画面項目幅によりアイコンイメージの割り当てを避けていた項目に新たに割り当てたりすることでユーザーにはスタイリッシュなイメージを持ってもらえるのではないだろうか。

2-3：TSplitViewの活用で省スペース化

最後に操作性に関する改善テクニックをご紹介する。
既存アプリケーションを長年ユーザー要望に応えながら拡張していくにあたって、特定機能の画面が増え、それに伴い画面展開するためのボタン配置も増え画面レイアウトがボタン群で覆われている、またボタンに機能名を表すための表記も、場合によっては多くの文字数を要し項目幅をボタン群で統一するとスペースに難があるといったレイアウトに関する悩みはないだろうか。そのように雑多なボタン群をTSplitViewを使うことでスマートなレイアウトにすることができる。

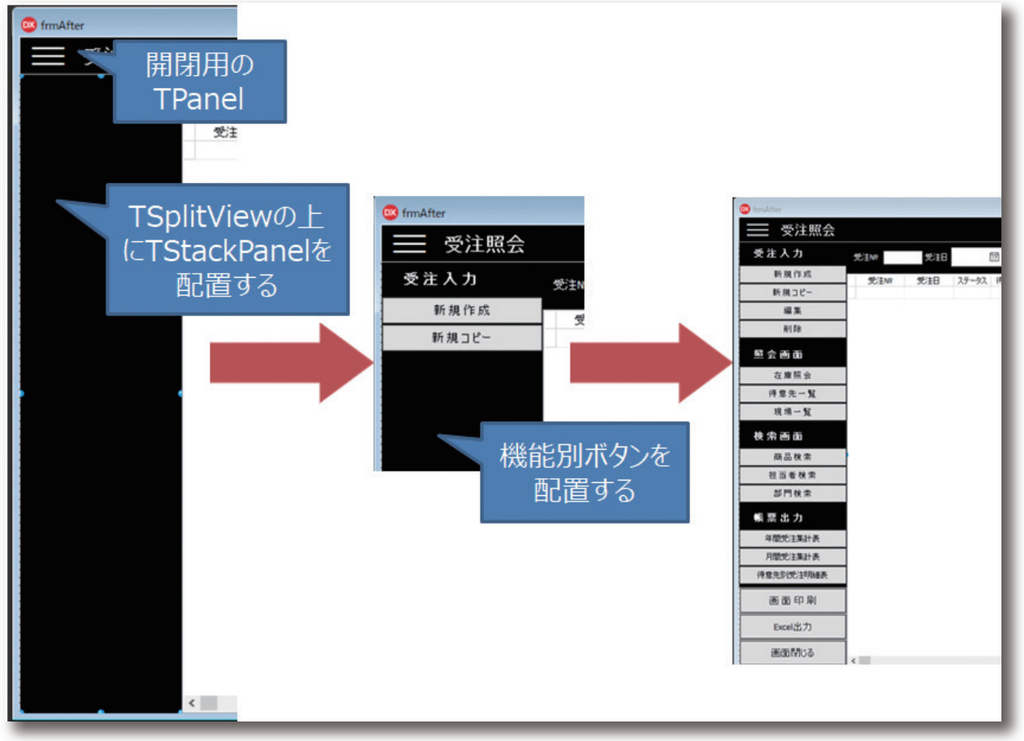
TSplitViewは開閉可能なコンテナ領域を提供でき、普段は閉じておきスペースを省略、必要に応じて開いてそのスペースを活用できるようにする。また今回、TStackPanelを用いてボタン群の管理を容易にできる工夫を行うことにする。TStackPanelは直接レイアウトに作用するものではないが、複数のコントロールを配下に管理でき設計時の配置変更に役立つ。具体的な利用方法を説明するにあたり、変更前の画面を紹介する。【図6】

図 6　変更前の画面イメージ



変更前の画面は明細形式の照会機能となり、上部に検索条件、中央部には照会データエリア、下部にデータ編集指示や帳票出力、その他機能画面への遷移を行うボタン群を配置した画面構成となっている。このボタン群をTSplitViewを用いて開閉可能なサブメニューとする。
まず、開閉を指示するために画面上部にTPanelを配置する。例では背景色を持たせるためにTPanelを用いたがボタンでもよい。Captionには「2-2」でご紹介した文字コード表より三本ラインを指定している。次に該当画面にTSplitViewを配置する。今回は画面左側に開閉スペースを設けるため、Placementプロパティを「svpLeft」に設定する。**【図7】**

図 7　コントロールの配置



TSplitViewの開閉はOpenedプロパティのTrue（開く）/False（閉じる）で操作するため、先ほど配置したTPanelのOnClickイベントにOpenedプロパティの切り替えを実装する。**【ソース4】**

ソース 4

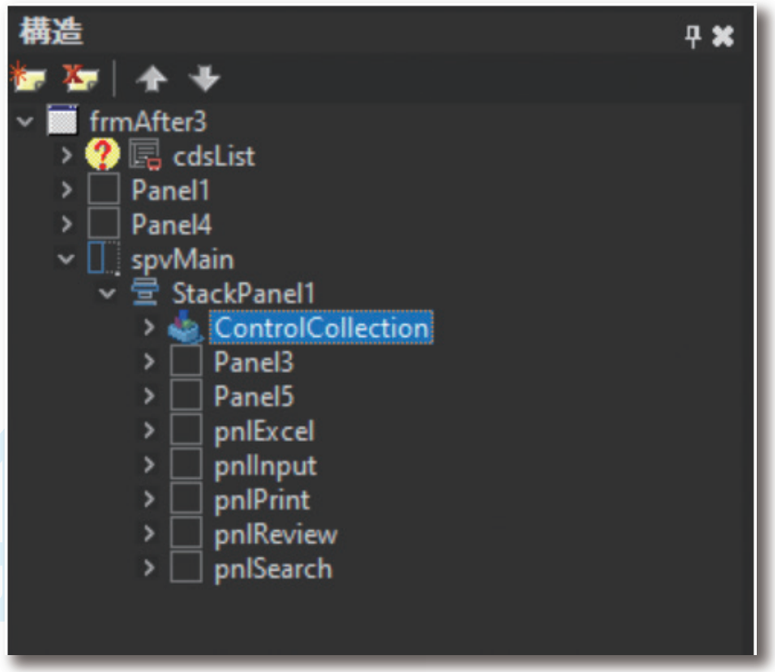
TSplitViewの開閉

```
procedure TForm1.pnlMenuClick(Sender: TObject);
begin
  // 横からスライド表示するアニメーションの適用(True:適用する False:適用しない)
  spvMain.UseAnimation := False;    // ちらつき防止のため本サンプルでは適用しない

  if spvMain.Opened then
    spvMain.Opened := False
  else
    spvMain.Opened := True;
end;
```

配置したTSplitView上にTStackPanelを配置し、Alignプロパティを「alClient」に設定する。さらに配置したTStackPanel上に機能カテゴリを表すパネルを配置、その下にボタン群を配置する。この際、各Alignプロパティを「alTop」として上詰めで配置していく。TStackPanel上にパネル、ボタンを配置すると、開発画面の「構造」エリアのTStackPanel配下にあるControlCollectionのさらに配下にTStackPanelControlItemが増えていくのが分かる。**【図8】**

図 8　TStack Panel Control Item

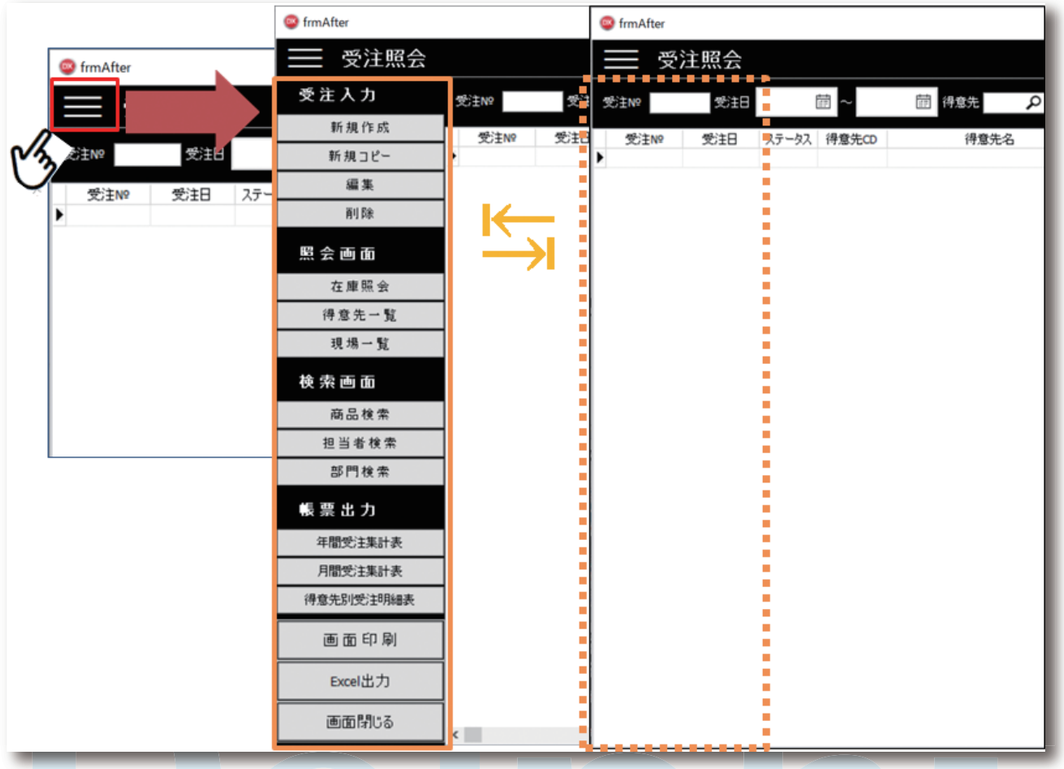


TStackPanelControlItemのControlプロパティを確認すると、配置したパネルやボタンが紐づいていることが分かる。この「構造」エリアのTStackPanelControlItemをドラッグアンドドロップし上位（または下位）に移動することで画面上の配置も合わせて入れ替わることができるため、今後、例えばボタンの配置を変更したい際にはメンテナンスが楽になる。最後にTSplitViewの下記プロパティを好みに応じて設定する。

・DisplayMode：TSplitViewのOpenedプロパティのTrue（開く）設定時のモーション指定
-svmDocked：隣接コントロールも同時に右に移動する
-svmOverlay：隣接コントロールに被さるようにTSplitViewが表示される

・CloseStyle：TSplitViewのOpenedプロパティのFalse（閉じる）設定時のスタイル指定。
-svcCollapse：完全に閉じた状態となる（非表示）
-svcCompact：CompactWidthの設定値に応じたスペースが表示される。以上の実装でアプリケーションを実行すると、開閉パネルをクリックするたびに画面左のボタンエリアが表示/非表示されることが確認できる。これで画面レイアウトの下部を占めていたボタン群を、任意で開閉できるエリアに移動しメイン部のスペースを広げることができた。
【図9】

図 9　TSplit Viewの開閉イメージ

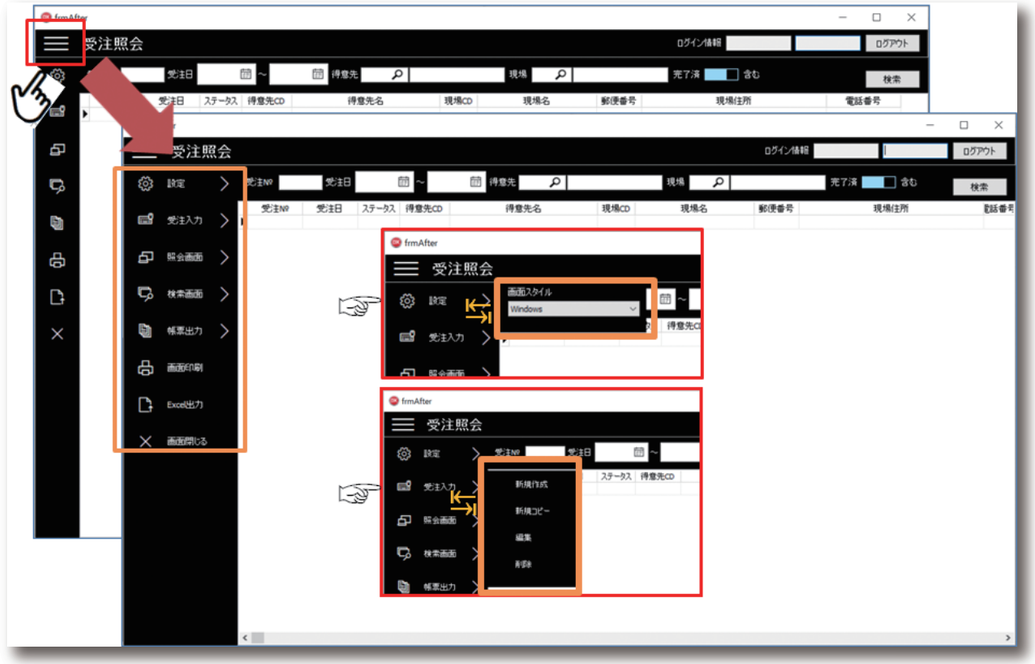


3.さいごに

本稿では、既存アプリケーションの一部のコントロールを、Delphi/400に新しく追加されたコントロールへ置き換えることでユーザーにバージョンアップの効果を実感してもらえる簡単なUI改善テクニックをご紹介した。

一つの参考として図「変更前の画面イメージ」を用いた、適用後のイメージを見て頂きたい。**【図10】**

図 10　UI改善テクニック適用例



画面左側にカスタムスタイルの変更を指示するための「設定」をはじめ、「受注入力」や「帳票出力」のような画面遷移や出力用の指示項目をカテゴリごとに開閉可能なTSplitViewに格納している。「設定」や「受注入力」については、「Segoe MDL2 Assets」フォントを用いた記号を配し、クリックするとさらに選択項目部を展開するようにしている。また指示項目についてはColorプロパティが適用可能なTPanelを用い画面上部と同じ色味にすることで、ビジュアルを画面内で統一している。

もしバージョンアップを単にコンバージョンで対応されたままの場合は、このテクニックを用い、UIの面でもバージョンアップの効果をユーザーに伝える役に立てて頂ければ幸いである。