福井 和彦

株式会社ミガロ.

システム事業部 プロジェクト推進室

[Delphi/400]

Windowsタブレット用にカスタムソフトウェアキーボードを実装

略歴
1972 年 3 月 20 日生まれ
1994 年 大阪電気通信大学工学部卒業
2001 年 4 月 株式会社ミガロ. 入社
2001 年 4 月 システム事業部配属

現在の仕事内容
主に Delphi/400 を使用したシステムの受託開発を担当しており、要件確認から納品・フォローに至るまで、システム開発全般に携わっている。また、Delphi/400 の導入支援やセミナーの講師も行っている。

1.はじめに

ここ数年、業務用としてタブレットを導入、または導入を検討する企業が増えている。そのタブレットを導入する際に重要な検討項目となるのが、OS の選択である。

Delphi/400 では iOS や Android のアプリケーションも開発できるが、PC と同様に Windows を搭載したタブレットを選択する企業も少なくない。それは、これまで購入してきた Windows ソフトウェアや、現在使用中の業務システムをタブレットでもそのまま使いたいという理由からである。

Windows タブレット用に新たにアプリケーションを開発する場合、その方法自体は、PC 用のアプリケーション開発と同じである。ただし Windows タブレットのソフトウェアキーボードは、iOS や Android のそれと比べると操作性に難があり、不便と感じられることがある。

そこで本稿では、ユーザーにとって操作性のよい Windows タブレット用ソフトウェアキーボードの実装方法を紹介する。

2.ソフトウェアキーボードについて

2-1. ソフトウェアキーボードとは

本題に入る前に、ソフトウェアキーボードについて少し説明する。ソフトウェアキーボードとは、物理的なキーボード機器を使用せずに画面上でキーボードを表示し、マウスクリックや画面タッチによって文字や数値等を入力できるソフトウェアである。

2-2. Windows 標準のソフトウェアキーボード

ここでは、Windows 8 搭載のタブレットに装備されている標準ソフトウェアキーボードについて基本的な動作を確認する。

ソフトウェアキーボードを表示するには、タスクバーに表示されているキーボードイメージのアイコンをタッチする。【図 1】

先ほど、Windows タブレットの標準ソフトウェアキーボードは、「操作性に難があり、不便と感じられることがある」と記したが、具体的には次のような点である。

①アプリケーションの入力項目にフォーカスが移っても、ソフトウェアキーボードは自動的に表示されず、タスクバーのキーボードアイコンをタッチして表示させなければならない。【図 2】
②ソフトウェアキーボードを開くと、アプリケーションの画面に被ってしまい入力項目が隠れてしまう場合がある。【図 3】

タブレット用のアプリケーションを開発する際、上記が問題となることが多い。そこで、それを改善するため標準のソフトウェアキーボードは使用せずに、アプリケーションの中でソフトウェアキーボード機能を実装する方法を紹介す

図1


ここをタッチする

図2

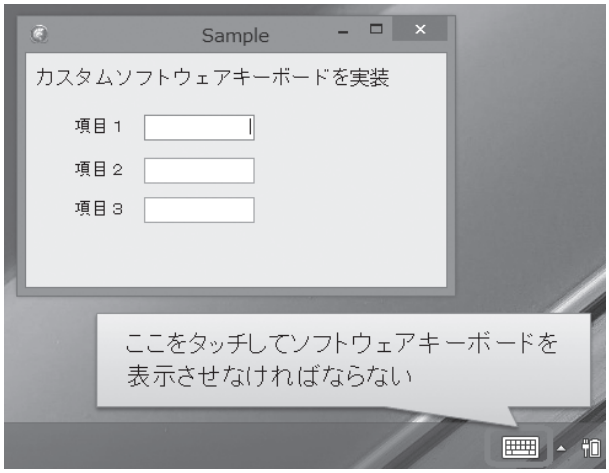

Sample
カスタムソフトウェアキーボードを実装
項目１
項目２
項目３
ここをタッチしてソフトウェアキーボードを
表示させなければならない

図3


Sample
カスタムソフトウェアキーボードを実装
項目１

る。

2-3. カスタムソフトウェアキーボードを独自実装

タブレットでは、ユーザーインターフェースとしてコンボボックスやボタン類を多用し、キーボード入力を極力少なくするのがセオリーである。しかし、数量や金額などの数値に関しては、リストから選択させるわけにもいかず、キーボードからの入力が必要になることが多い。そこで、これから紹介するソフトウェアキーボードは、タブレットで使用率が高い、テンキーのキーボードを題材として説明していく。

3.TTouchKeyboardコンポーネントでの実装

3-1. TTouchKeyboardコンポーネントの実装手順

Delphi/400 では、Ver.2010 から TTouchKeyboard コンポーネントが実装されている。このコンポーネントを使用することで、アプリケーション中にソフトウェアキーボード機能を簡単に実装できる。

TTouchKeyboard コンポーネントをフォームに配置すると、ソフトウェアキーボードがフォーム上に表示される。またプロパティの設定だけで、通常の文字キーボードとテンキーのイメージを切り替えることができる。【図 4】

ここから、TTouchKeyboard を使用したソフトウェアキーボードの実装手順について説明する。

①コンポーネント配置

新規フォーム上に TEdit コンポーネントを 2 個と、TTouchKeyboard コンポーネントを配置する。【図 5】

② TTouchKeyboard のプロパティ設定

TTouchKeyboard コンポーネントの Layout プロパティを「NumPad」に設定し、テンキーイメージにする。【図 6】

③実装完了

TTouchKeyboard コンポーネントとフォームのサイズを整えて実装は完了となる。【図 7】

TTouchKeyboard コンポーネントの Layout プロパティを変更した以外、特にソースを記述する必要はないので、非常に簡易に実装できる。

3-2. TTouchKeyboard コンポーネントの機能

この節では、実際の動作について確認する。

アプリケーションを実行すると、フォーカスが Edit1 にある状態で画面が表示される。ソフトウェアキーボードの「1」「2」をタッチすると、Edit1 に「1」「2」がセットされる。【図 8】

次に、Edit2 にフォーカスを移してソフトウェアキーボードの「4」「5」をタッチすると、Edit2 に「4」「5」がセットされる。【図 9】

このように TTouchKeyboard コンポーネントは、同一フォーム上のアクティブなコンポーネントに対して値をセットするため、入力項目があるフォーム上に配置する必要はあるものの、ソフトウェアキーボードを簡単に実装したい場合に便利なコンポーネントである。

ただし、アプリケーションによっては、ソフトウェアキーボードを表示する領域が画面デザイン上、難しい場合もある。そうした場合には、アプリケーションとは別のウインドウにソフトウェアキーボードを分離させる必要がある。そこで、別のウインドウとしてソフトウェアキーボードを作成し、アプリケーションに実装する方法を次章で紹介していく。

4.カスタムソフトウェアキーボードを実装

4-1. ソフトウェアキーボードの開発手順

この章では、【図 10】のように入力画面とソフトウェアキーボードを別フォームとして作成し、ソフトウェアキーボードが入力画面に対してキーボードとして動作するように実装する方法を説明する。この実装方法では、簡易な TTouchKeyboard コンポーネントを使わずに独自にソフトウェアキーボードを作成するため、細かい制御を自由に実装できる。

まず、ソフトウェアキーボードを新しいフォームとして作成する。今回作成する画面イメージは【図 11】の通りである。

機能としては、数値入力、マイナス入力、小数点入力、BackSpace、フォーカスの移動（次項目、前項目）、入力項目値の全選択機能を実装する。では、作成手順を順番に説明する。

①コンポーネント配置

【図 11】に従って、新規フォーム上に TBitBtn コンポーネントを 16 個配置し、Caption プロパティを設定する。続いて【表 1】に従って、各 TBitBtn コンポーネントの Name プロパティを設定する。

②フォームのプロパティ設定

フォームの BorderStyle プロパティを「bsSingle」に設定して、ソフトウェアキーボードの画面サイズを変更不可にする。次に FormStyle プロパティを「fsStayOnTop」に設定し、ソフトウェアキーボードが常に手前で表示されるようにする。【図 12】

③グローバル変数の宣言

グローバル変数として DeActivateForm を記述しておく【ソース 1】。これは DeActivateForm 変数で受け取ったフォームに対して、別ウインドウのソフトウェアキーボードでボタンタッチされた結果を反映するためである。

④フォームの onCreate イベントハンドラの実装

フォームの onCreate イベントハンドラを次のように実装する。【ソース 2】

ここでは、btnKey0 ～ btnKey9、btnKeyDot、btnKeyMinus の Tag プロパティに、ボタンの Caption に該当する文字コードを設定しておく。今回は数値と記号のみだが、文字キーボードを実装する場合には「Ord('A')」といった形で、引数にアルファベットを指定することもできる。

次に、btnKeyBS の Tag プロパティには BackSpace キーに該当する制御コードを設定し、btnNext と btnPrior の Tag プロパティには、前項目への移動なのか、次項目への移動なのかを判断するための区分を設定しておく。ここで設定した内容は、後で説明する TBitBtn コンポーネントの onClick イベントハンドラで使用することになる。そして、ソフトウェアキーボードの Top と Left を指定し、初期表示位置を画面右下になるよう設定しておく。

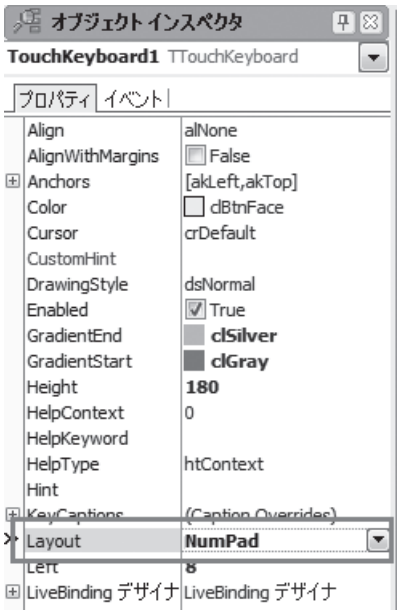
図4


通常のキーボード
テンキー
プロパティの設定で切り替え

図5


Sample
Edit1
Edit2

図6


オブジェクト インスペクタ
TouchKeyboard1 TTouchKeyboard
プロパティ
イベント
Align alNone
AlignWithMargins False
Anchors [akLeft,akTop]
Color clBtnFace
Cursor crDefault
CustomHint
DrawingStyle dsNormal
Enabled True
GradientEnd clSilver
GradientStart clGray
Height 180
HelpContext 0
HelpKeyword
HelpType htContext
Hint
KeyCaptions (Caption Overrides)
Layout NumPad
Left 8
LiveBinding デザイナ

⑤ TBitBtn コンポーネントの onClick イベントハンドラ

各 TBitBtn コンポーネントの onClick イベントハンドラを実装する。イベントハンドラはボタンごとに用意するのではなく、機能ごとに共通化して 3 つのイベントハンドラを用意する。【ソース 3】

各イベントハンドラのソースは、【ソース 4】のようになる。そして、各 TBitBtn コンポーネントの onClick イベントハンドラを次のように設定する。【表 2】

btnKeyBoardClick では、DeActivate Form 変数で受け取ったフォームを BringToFront を使用してアクティブにし、フォームのアクティブコンポーネントに対して TBitBtn の Tag プロパティに設定されている文字コードや制御コードを、PostMessage 関数を使用してメッセージ送信している。この実装で、ソフトウェアキーボードの数値、記号、「BS」を押した結果が入力項目に反映されるようになる。

btnSelectALL で は、btnKeyBoard Click 同様にフォームをアクティブにし、フォームのアクティブコンポーネントに対して値の全選択命令をメッセージ送信している。

btnFocusControl では、btnKeyBoard Click 同様にフォームをアクティブにし、TBitBtn の Tag プロパティに設定されている区分によってフォームのアクティブコンポーネントの次項目または前項目にフォーカスを移動させる命令をメッセージ送信している。

以上でソフトウェアキーボードの作成は完了である。

4-2. 入力画面への実装手順

続いて、入力画面へソフトウェアキーボードを実装する方法について説明する。実装の手順は、次の通りである。

①入力画面の作成

前節のソフトウェアキーボードと同じプロジェクト内に、新規フォームを追加して TEdit コンポーネントを 3 個配置する【図 13】。そして、ソフトウェアキーボードのユニットを参照しておく。また、プロジェクトオプションのフォームの設定では、ソフトウェアキーボードも自動生成の対象としておく。【図 14】

②フォームの onShow イベントハンドラの実装

ソフトウェアキーボードは【図 14】で自動生成されているため、フォームの onShow イベントでは、ソフトウェアキーボードを Show することで表示させることができる。【ソース 5】

③フォームの onDeactivate イベントハンドラの実装

ソフトウェアキーボードのボタンを押した際、入力中のフォームからソフトウェアキーボードへ制御が移る。この時にソフトウェアキーボードの Deactivate Form 変数に入力中のフォームをセットしておくため、フォームの onDeactivate イベントハンドラを次のように実装する。【ソース 6】

以上で入力画面への実装は完了である。

4-3. カスタムソフトウェアキーボードの機能

実行して動作を確認する。アプリケーションを実行すると入力画面とソフトウェアキーボードが表示され、入力画面の Edit1 にフォーカスが設定される。【図 15】

入力画面とソフトウェアキーボードは別々のフォームになっているので、ソフトウェアキーボードは画面上の好きな位置に移動できる。

実際にソフトウェアキーボードを使用して入力してみよう。ソフトウェアキーボードの「1」「2」をタッチすると Edit1 に「1」「2」がセットされ、「全選択」をタッチすると、入力内容が全選択された状態となる。【図 16】

この状態で Edit1 をタッチ長押しでポップアップメニューを開き、コピーを選択すると、入力中の値をコピーすることもできる。

次に、「BS」をタッチすると、Edit1 に入力されていた「12」がクリアされる。さらに「次項目」を押せば Edit2 へ移り、「前項目」を押せば Edit1 へフォーカスを戻すことができる。

このようにソフトウェアキーボードを独自に作成する場合は、TTouchKeyboard コンポーネントよりも実装に手間がかかるが、別ウインドウとして制御したり、独自の機能を実装できる利点がある。

ここで、Windows タブレットのソフトウェアキーボードが、どのように改善されたのかをまとめておく。

1 つ目の「アプリケーションの入力項目にフォーカスが移っても、ソフトウェアキーボードは自動的に表示されず、タスクバーのキーボードアイコンをタッチしなければならない」に関しては、ソフトウェアキーボードを常に表示させておくことで、入力項目にフォーカスが移っても、すぐに入力できるようになり解決できる。

2 つ目の「ソフトウェアキーボードを開くと、アプリケーションの画面に被ってしまい入力項目が隠れてしまう場合がある」に関しては、画面起動時にソフトウェアキーボードを入力項目に被らない位置に表示させることで解決できる。ただし、今回の説明で使用した入力画面は項目も少なく画面サイズも小さいため、入力項目に重ならない位置に表示できた。しかし、入力項目が多い画面では、ソフトウェアキーボードがどうしても入力項目に被ってしまうケースが出てくる。

そこで、次節では入力項目にソフトウェアキーボードが重なってしまった場合に、ソフトウェアキーボードの位置を変える拡張方法について紹介する。

4-4. カスタムソフトウェアキーボードの移動

ここでは、【図 17】のように Edit2 にソフトウェアキーボードが被っている状態で、Edit2 にフォーカスを移した際にソフトウェアキーボードを下方向にずらす調整方法を説明する。

この制御を実装するには、入力項目である各 TEdit コンポーネントの onEnter イベントハンドラを利用して実装する。イベントハンドラは【ソース 7】のように共通化した 1 つのイベントハンドラとして用意する。

イベントハンドラのソースを【ソース 8】に示す。

作成した共通イベントハンドラは、各 TEdit コンポーネントの onEnter イベントに設定する。

【ソース 8】について、ソースコードのポイントを説明する。

図7

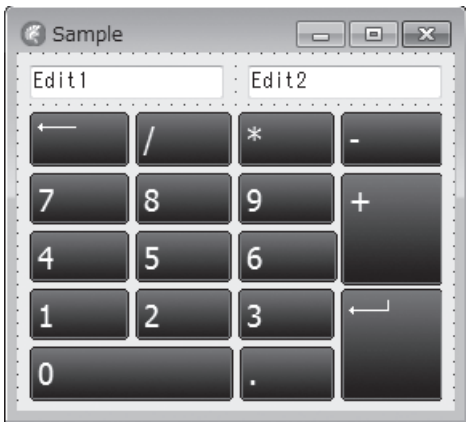

Sample
Edit1
Edit2
←
/
*
-
7
8
9
+
4
5
6
1
2
3
↵
0
.

図8

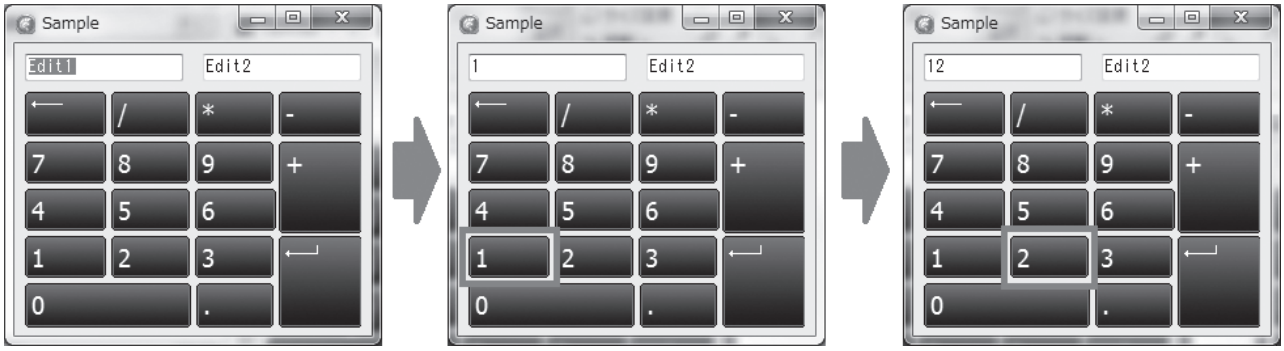
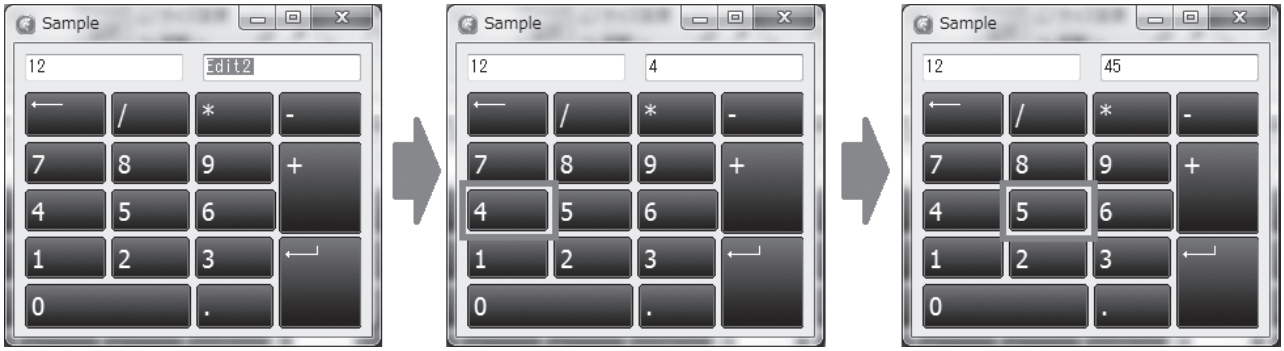

Sample
Edit1
Edit2
1
12

図9


Sample
12
Edit2
4
45

位置の調整については、最初にTEditコンポーネントの位置（Top、Left）とソフトウェアキーボードの位置（Top、Left）を、スクリーンを基準にして算出する。

次に、重なり合う部分を計算するIntersectRect関数を使用して、TEditコンポーネントとソフトウェアキーボードの重なりを求める。今回は上下方向の重なりを判断している（if rRes.Height > 0 then）。横方向の重なりを判断したい場合は、rRes.Widthを使用することで判断できる。

そして、重なりがあった場合にソフトウェアキーボードを、TEditの底辺がTop位置になるよう下方向に移動させるのだが、その前に下方向に移動した場合のソフトウェアキーボードの底辺の位置を計算し（iBot := rCon.Top + rCon.Height + frmKeyBoard.Height + 5）、スクリーン下にはみ出ないかを判断する（if iBot > Screen.Height then）。

スクリーン下にはみ出る場合は、TEditのTop位置がソフトウェアキーボードの底辺になるよう上方向に移動し（frmKeyBoard.Top := rCon.Top - frmKeyBoard.Height - 5）、はみ出なければ下方向に移動させる（frmKeyBoard.Top := rCon.Top + rCon.Height + 5）。

実装内容は以上である。

次に、実際にアプリケーションを実行して動作を確認する。Edit2とEdit3にソフトウェアキーボードが重なっている状態で、フォーカスをEdit1→Edit2→Edit3と移動させると、ソフトウェアキーボードがフォーカスの移動に合わせて下方向に移動していくことが確認できる。【図18】

また、Edit3にフォーカスが移動した際、ソフトウェアキーボードがスクリーン下にはみ出る場合は、【図19】のように上方向に移動する。

こうした実装を行うことで、前節で述べていた入力項目が多い画面であっても、フォーカスの移動に合わせてソフトウェアキーボードの位置を調整し、入力項目を見えるようにできるので、ユーザーが使用する際に非常に便利である。また、この方法を用いれば、離れた位置にあるソフトウェアキーボードを入力項目の近くに移動させたり、入力項目からフォーカスが抜けた時に画面外にソフトウェアキーボードを移動させる、といった制御も可能である。

5.まとめ

本稿では、Windowsタブレットの標準ソフトウェアキーボードが持つ不便さの解消を目的に、アプリケーションによる実装方法を紹介した。またデスクトップPCやノートPCで使用するアプリケーションの場合でも、今回のソフトウェアキーボードを実装することで、キーボードを使用することなくマウスのみで入力操作が可能な画面設計を行える。

実装方法については、TTouchKeyboardコンポーネントを使った簡単な方法と、独自にソフトウェアキーボードを作成する方法を取り上げたが、どちらも有効な方法なので、アプリケーションの画面設計や用途によって使い分けるとよい。

また今回は、単純なテンキーのキーボードを題材にしたが、同様の方法で文字キーボードの実装や、新しい機能の追加も可能である。

Windowsアプリケーションについても、タブレットなどのスマートデバイス端末での使用が増えてきている。そうした開発を行う中で、デスクトップPCやノートPCにはなかった新しい課題に取り組むことも多い。

開発方法はこれまでと同様であっても、使用するデバイスが変わると、使い勝手や求められるユーザーインターフェースも違ってくる。こうしたことを常に頭において設計を工夫することが、これからのアプリケーション開発では重要である。

M

図10

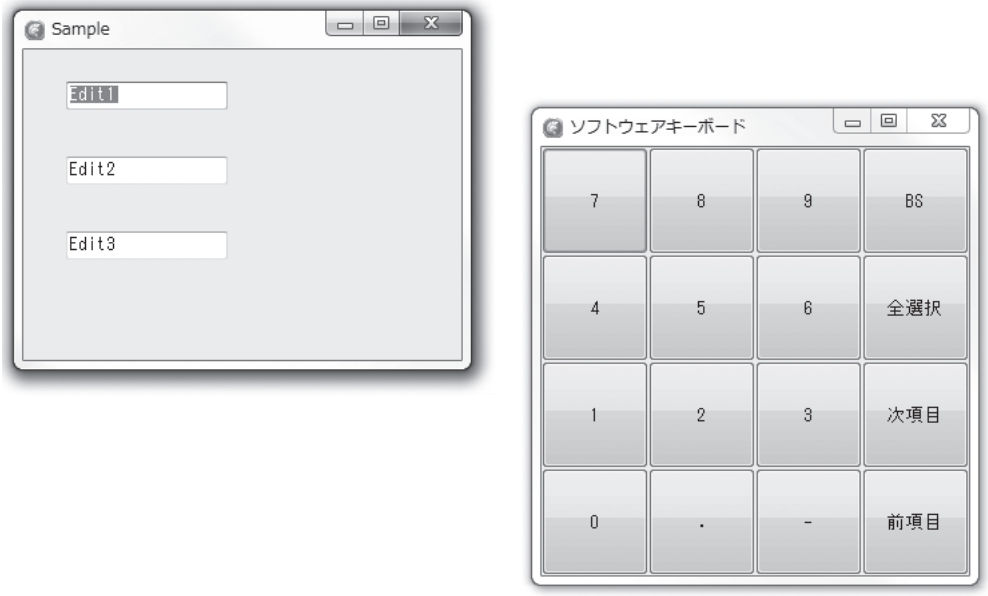


図11

表1

TBitBtnコンポーネントのNameプロパティの設定

Caption	Name
0	btnKey0
1	btnKey1
2	btnKey2
3	btnKey3
4	btnKey4
5	btnKey5
6	btnKey6
7	btnKey7
8	btnKey8
9	btnKey9

Caption	Name
.	btnKeyDot
-	btnKeyMinus
BS	btnKeyBS
全選択	btnSelAll
次項目	btnNext
前項目	btnPrior

図12

ソース1

```
var
  frmKeyBoard: TfrmKeyBoard;
  DeActivateForm: TForm;

implementation
```

ソース2

```
procedure TfrmKeyBoard.FormCreate(Sender: TObject);
begin
  // TBitBtnのTagプロパティを設定
  btnKey7.Tag      := Ord('7');
  btnKey4.Tag      := Ord('4');
  btnKey1.Tag      := Ord('1');
  btnKey0.Tag      := Ord('0');

  btnKey8.Tag      := Ord('8');
  btnKey5.Tag      := Ord('5');
  btnKey2.Tag      := Ord('2');
  btnKeyDot.Tag    := Ord('.');

  btnKey9.Tag      := Ord('9');
  btnKey6.Tag      := Ord('6');
  btnKey3.Tag      := Ord('3');
  btnKeyMinus.Tag  := Ord('-');

  btnKeyBS.Tag     := $08;

  btnNext.Tag      := 0;
  btnPrior.Tag     := 1;

  // ソフトウェアキーボードの初期表示位置を設定
  Top  := Screen.WorkAreaTop  + Screen.WorkAreaHeight - Height;
  Left := Screen.WorkAreaLeft + Screen.WorkAreaWidth  - Width;
end;
```

ソース3

```
  procedure btnKeyBoardClick(Sender: TObject);
  procedure btnSelectALL(Sender: TObject);
  procedure btnFocusControl(Sender: TObject);
private
  { Private 宣言 }
public
  { Public 宣言 }
end;
```

ソース4

btnKeyBoardClick

```
procedure TfrmKeyBoard.btnKeyBoardClick(Sender: TObject);
begin
  if Assigned(DeActivateForm) then
  begin
    DeActivateForm.BringToFront;
    PostMessage(DeActivateForm.ActiveControl.Handle, WM_CHAR, (Sender as TBitBtn).Tag, 0);
  end;
end;
```

DeActivateForm変数で受け取ったフォームをアクティブにする

TBitBtnのTagプロパティに設定されている
キャラクターコードや制御コードをメッセージ送信

btnSelectALL

```
procedure TfrmKeyBoard.btnSelectALL(Sender: TObject);
begin
  if Assigned(DeActivateForm) then
  begin
    DeActivateForm.BringToFront;
    PostMessage(DeActivateForm.ActiveControl.Handle, EM_SETSEL, 0, -1);
  end;
end;
```

DeActivateForm変数で受け取ったフォームをアクティブにする

値の全選択命令をメッセージ送信

btnFocusControl

```
procedure TfrmKeyBoard.btnFocusControl(Sender: TObject);
begin
  if Assigned(DeActivateForm) then
  begin
    DeActivateForm.BringToFront;
    PostMessage(DeActivateForm.Handle, WM_NEXTDLGCTL, (Sender as TBitBtn).Tag, 0);
  end;
end;
```

DeActivateForm変数で受け取ったフォームをアクティブにする

TBitBtnのTagプロパティに設定されている区分によって、
次項目or前項目にフォーカスを移動させる命令をメッセージ送信

表2

TBitBtnコンポーネントのonClickイベントハンドラの設定

TBitBtn	onClick
btnKey0	btnKeyBoardClick
btnKey1	btnKeyBoardClick
btnKey2	btnKeyBoardClick
btnKey3	btnKeyBoardClick
btnKey4	btnKeyBoardClick
btnKey5	btnKeyBoardClick
btnKey6	btnKeyBoardClick
btnKey7	btnKeyBoardClick
btnKey8	btnKeyBoardClick
btnKey9	btnKeyBoardClick

TBitBtn	onClick
btnKeyDot	btnKeyBoardClick
btnKeyMinus	btnKeyBoardClick
btnKeyBS	btnKeyBoardClick
btnSelAll	btnSelectALL
btnNext	btnFocusControl
btnPrior	btnFocusControl

図13

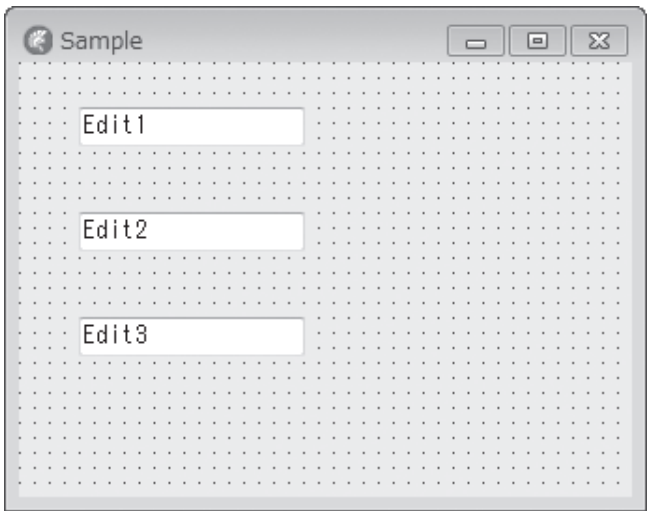


図14

ソース5

```
procedure TForm1.FormShow(Sender: TObject);
begin
  // ソフトウェアキーボードを表示
  frmKeyBoard.Show;
end;
```

ソース6

```
procedure TForm1.FormDeactivate(Sender: TObject);
begin
  // ソフトウェアキーボードの"DeactivateForm"に自身をセット
  DeactivateForm := Self;
end;
```

図15

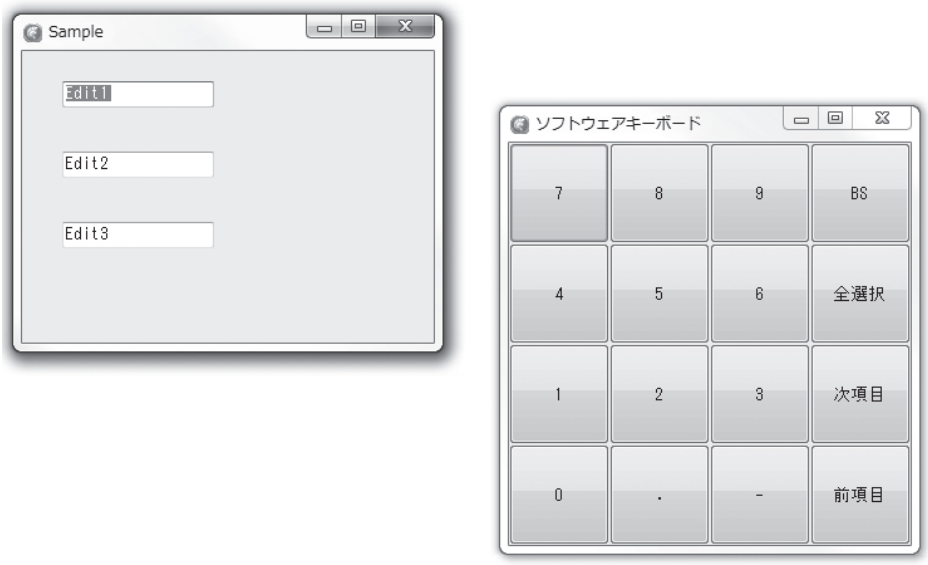
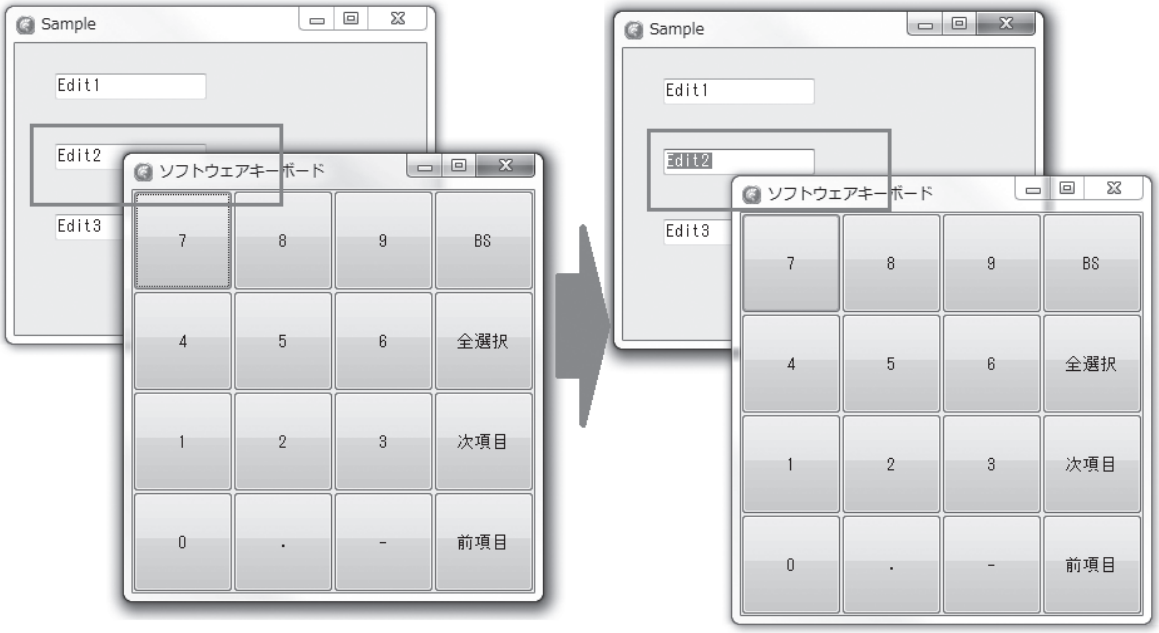

Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目

図16


Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目

図17


Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目
Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目

ソース7

```
    procedure FormDeactivate(Sender: TObject);
    procedure EditEnter(Sender: TObject);
  private
    { Private 宣言 }
```

ソース8

```
procedure TForm1.EditEnter(Sender: TObject);
var
  rCon, rFrm, rRes: TRect;
  pCon: TPoint;
  iBot: Integer;
begin
  if (Sender is TWinControl) and (Assigned(frmKeyBoard)) then
  begin
    // コンポーネントの位置をスクリーン座標で求める
    pCon.X := (Sender as TWinControl).Left;
    pCon.y := (Sender as TWinControl).Top;
    pCon   := Form1.ClientToScreen(pCon);

    // コンポーネントのRect
    rCon.Top    := pCon.Y;
    rCon.Left   := pCon.X;
    rCon.Height := (Sender as TWinControl).Height;
    rCon.Width  := (Sender as TWinControl).Width;

    // キーボードのRect
    rFrm.Top    := frmKeyBoard.Top;
    rFrm.Left   := frmKeyBoard.Left;
    rFrm.Height := frmKeyBoard.Height;
    rFrm.Width  := frmKeyBoard.Width;

    // コンポーネントとフォームの重なりを求める
    IntersectRect(rRes, rCon, rFrm);

    // 重なりが有った場合キーボードを移動
    if rRes.Height > 0 then
    begin
      // キーボードを移動した結果のフォームの下位置を求める
      // [TEditのTop]+[TEditのHeight]+[キーボードのHeight]+[調整値]
      iBot := rCon.Top + rCon.Height + frmKeyBoard.Height + 5;

      // キーボードがスクリーンの外にはみ出したかどうかを判断
      if iBot > Screen.Height then
        frmKeyBoard.Top := rCon.Top - frmKeyBoard.Height - 5  // TEditの上に移動
      else
        frmKeyBoard.Top := rCon.Top + rCon.Height + 5;        // TEditの下に移動
    end;
  end;
end;
```

図18

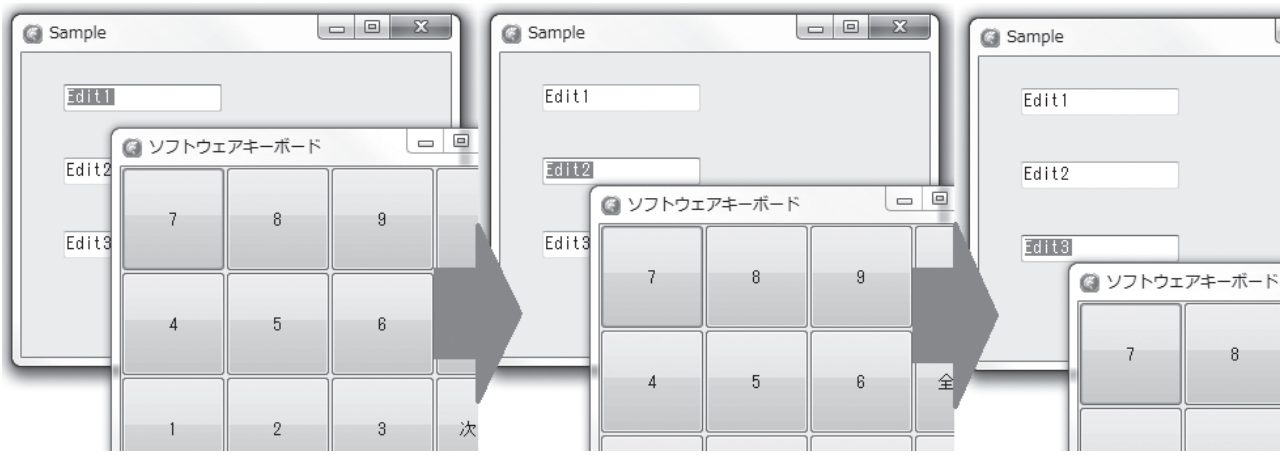

Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
4
5
6
1
2
3
次
全

図19


ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目
Sample
Edit1
Edit2
Edit3