都地 奈津美

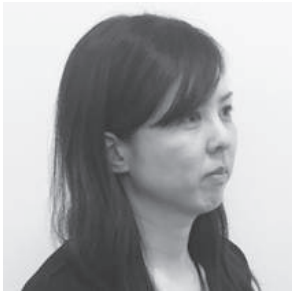
Delphi/400

株式会社ミガロ.

システム事業部 システム1課

[Delphi/400]

Windowsタブレット向けVCLアプリケーション作成テクニック



略歴
1989年8月19日生まれ
2012年3月 関西学院大学 理工学部卒業
2012年4月 株式会社ミガロ. 入社
2012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受託開発とシステム保守を担当している。開発スキルの向上を目指し、日々精進している。

1. はじめに
2. UI／UXデザインを考慮した入力機能
2-1. アプリケーションの画面設計の違い
2-2. 日付のカレンダー入力
2-3. 時間の時計入力
2-4. 動作比較
3. 画面サイズに合わせた項目の自動調整
3-1. 複数端末でのアプリケーション使用時の課題
3-2. 項目の位置・サイズの自動調整
4. さいごに

1. はじめに

近年、軽量で持ち運びやすいことやペーパーレス化の観点から、PCの代わりにタブレット端末を使用して業務を行うことが増えている。タブレット端末を導入する際、PCと同様のソフトウェアを利用できることや、PCと同じ操作感で使えることから、Windows OSを搭載したタブレットを選定する場合も少なくないだろう。

Windowsタブレット向けアプリケーションの場合、VCLフレームワークを使用することで、これまでのPC向けの開発と同様の方法でアプリケーションを開発できる。

しかしタブレットはPCに比べ、画面のサイズが小さいことや、マウス操作ではなくタッチ操作が主体となることなどから、PCとは違ったUI（ユーザーインターフェースの略で、ユーザーがPCとやり取りする際の入力や表示方法などの仕組みのこと）や、UX（ユーザーエクスペリエンスの略で、ユーザーがサービスを利用する際に体験したことや感じたこと）を考慮する必要がある。

そこで本稿では、Windowsタブレット向けVCLアプリケーションの作成テクニックとして、2.で日付と時間の入力方法について紹介する。次に3.では、複数端末でアプリケーションを使用する場合に、1つの画面設計から各端末サイズを基準に、項目の位置・サイズ自動調整する方法について紹介する。

2. UI／UXデザインを考慮した入力機能

2-1. アプリケーションの画面設計の違い

ここでは、タブレット向けアプリケーションを作成する際の入力方法について紹介する。

まず本題に入る前に、PC向けアプリケーションとタブレット向けアプリケーションの画面設計の違いについて説明する。

PCではマウス操作が主体となるのに対し、タブレットではタッチペンや指でのタッチ操作が主体となる。タッチ操作の場合、マウス操作とは異なり、狙ったポイントとはずれた位置をタッチしてしまうことがある。

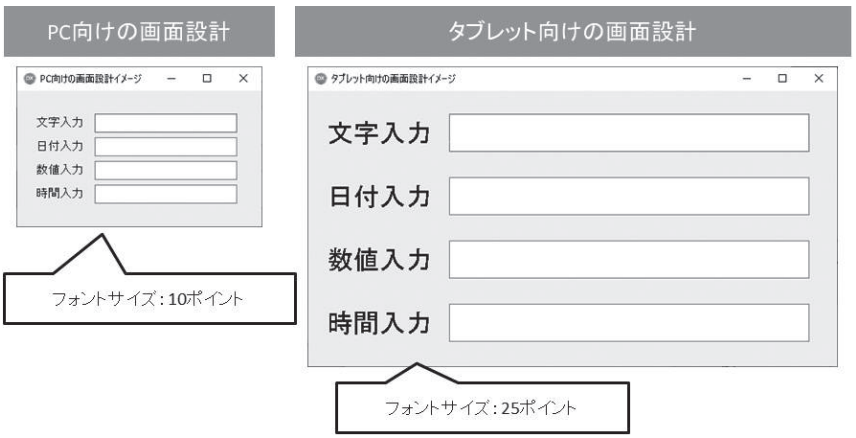
そのため入力項目やボタンのサイズ、フォントサイズは、PC向けアプリケーションに比べて大きく設計する必要がある。さらにタッチミスを考慮し、項目間の余白を広めにとることも必要である。【図1】

またタブレットは片手で操作する可能性が高いことから、キーボードでの入力をできるだけ少なくし、コンボボックスやチェックボックスなど直感的に入力できるよう工夫する必要がある。

今回は項目の入力方法について注目し、その中でも日付や時間を入力する方法について説明していく。日付や時間をソフトウェアキーボードで入力するには、ソフトウェアキーボードを起動したり、手入力で1文字ずつ入力する必要があり、入力に時間がかかる。

そこでカレンダーや時計を使用して、直感的かつ簡単に日付や時間を入力する

図1　PCとタブレットの画面設計の違い



＜タブレットアプリ設計時のポイント＞
①項目自体のサイズやフォントサイズを大きめに設計する
②項目間の余白を広めにとる

図2　日付のカレンダー入力

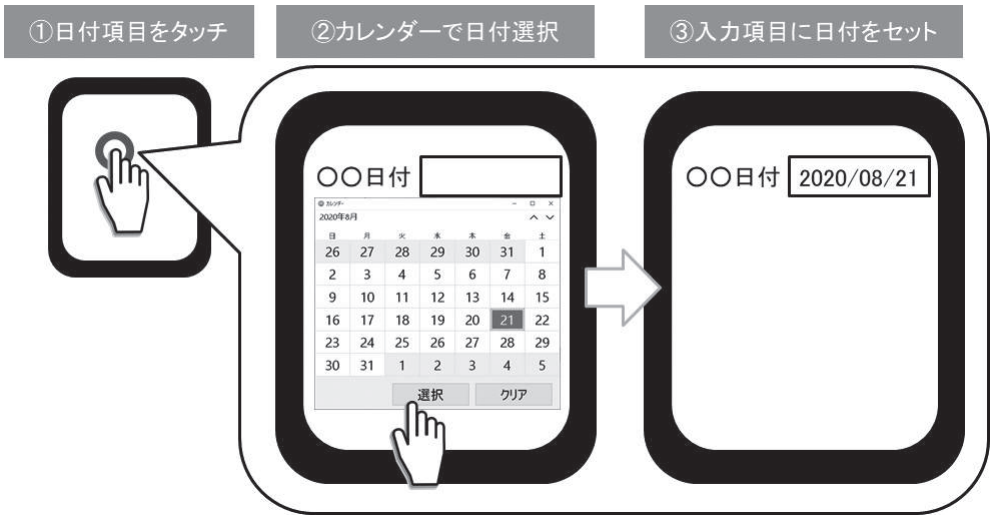
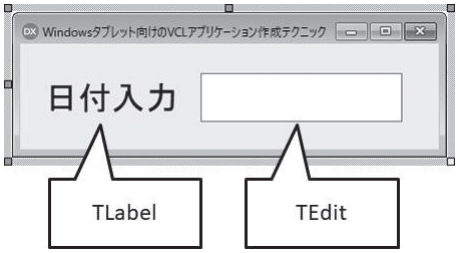


図3　コンポーネントの配置



方法について紹介する。カスタムソフトウェアキーボードを使用した文字入力や、テンキーを使用した数値入力については、2015 年発行のミガロ .テクニカルレポート 2015 にある『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』に紹介されているので、こちらを参考にしていただきたい。

なお本章で作成しているプログラムは、Delphi/400 10.2Tokyo を使用している。

2-2. 日付のカレンダー入力

日付入力が必要な項目に対し、カレンダーを使用した入力方法の実装手順について説明する【図 2】。まずは、メインの入力画面について実装していく。

(1) コンポーネントの配置

日付入力を行うための TEdit、並びに TLabel を画面に配置する。【図 3】

(2) プロパティの設定

日付入力用の TEdit は、キーボードでの入力を禁止し、カレンダーでの入力のみを許可するため、ReadOnly プロパティに True を設定する。

次にカレンダー画面について、メイン画面とは別に新規フォームを作成し実装していく。

(1) コンポーネントの配置

カレンダー表示用の TCalendarView、日付選択用、選択値クリア用の TButton を配置する。【図 4】

(2) 処理の実装

①カレンダー画面の onShow イベントの実装

メイン画面の日付入力 Edit 用のプロパティを宣言し【ソース 1】、フォームの onCreate イベントを【ソース 2】のように実装する。メイン画面より日付入力 Edit をプロパティとして受け取り、カレンダー画面の onShow イベントで日付入力 Edit の値をカレンダーにセット、日付入力 Edit がブランクの場合はシステム日付をセットする。

これによりシステム日付、もしくは日付入力 Edit の値がカレンダーにセットされた状態でカレンダー画面が表示される。

②カレンダー画面の選択ボタン onClick イベントの実装

選択ボタンをクリック時に、カレンダーで選択されている日付をプロパティの日付入力 Edit にセットする。選択日付をセット後、カレンダー画面を終了する。【ソース 3】

③カレンダー画面のカレンダー onDblClick イベントの実装

カレンダーをダブルクリック時に、選択ボタンクリック時の処理を呼び出す。【ソース 4】

④カレンダー画面のクリアボタン onClick イベントの実装

クリアボタンをクリック時に、メイン画面の日付入力 Edit の日付をクリアするため、プロパティの日付入力 Edit にブランクをセットする。【ソース 5】

⑤メイン画面の日付 Edit の onEnter イベント

メイン画面の日付入力 Edit にフォーカスがセットされたタイミングで、カレンダー画面を表示するため、onEnter イベントにてカレンダー画面の生成・表示処理を行う。その際、プロパティの受け渡しと、カレンダー画面の表示位置を調整する。【ソース 6】

以上で、カレンダーを使用した日付入力の実装は完了である。

2-3. 時間の時計入力

時間入力が必要な項目に対し、時計を使用した入力方法の実装手順について説明する【図 5】。まずは、メインの入力画面について実装していく。

(1) コンポーネントの配置

2-2 にて作成したサンプルプログラムに対し、時間入力を行うための TEdit、並びに TLabel を画面に配置する。【図 6】

(2) プロパティの設定

日付入力用の TEdit と同様、時間入力用の TEdit の ReadOnly プロパティに True を設定する。

次に時計画面について、メイン画面とは別に新規フォームを作成し実装していく。

(1) コンポーネントの配置

時計表示用の TListBox（時間用、分用の 2 種類）、時間選択用、選択値クリア用の TButton を配置する【図 7】。時計表示用のコンポーネントとして、ドロップダウンで時間を選択できる TTimePicker でも代用が可能である。

今回、TListBox を使用する理由としては、TTimePicker では 0:00 ～ 23:59 のみが入力可能であるのに対し、TListBox は選択内容をソース内で設定するため、作業時間の入力など 24:00 以降の入力も可能となることが挙げられる。

(2) 処理の実装

①時計画面の onShow イベントの実装

メイン画面の時間入力 Edit 用のプロパティを宣言し【ソース 7】、フォームの onCreate イベントを【ソース 8】のように実装する。メイン画面より時間入力 Edit をプロパティとして受け取り、時計画面の onShow イベントで時間入力 Edit の値を時計にセット、時間入力 Edit がブランクの場合はシステム時間をセットする。

これにより、システム時間もしくは時間入力 Edit の値が時計にセットされた状態で時計画面が表示される。

②時計画面の選択ボタン onClick イベントの実装

選択ボタンをクリック時に、時計で選択されている時間をプロパティの時間入力 Edit にセットする。選択時間をセット後、時計画面を終了する。【ソース 9】

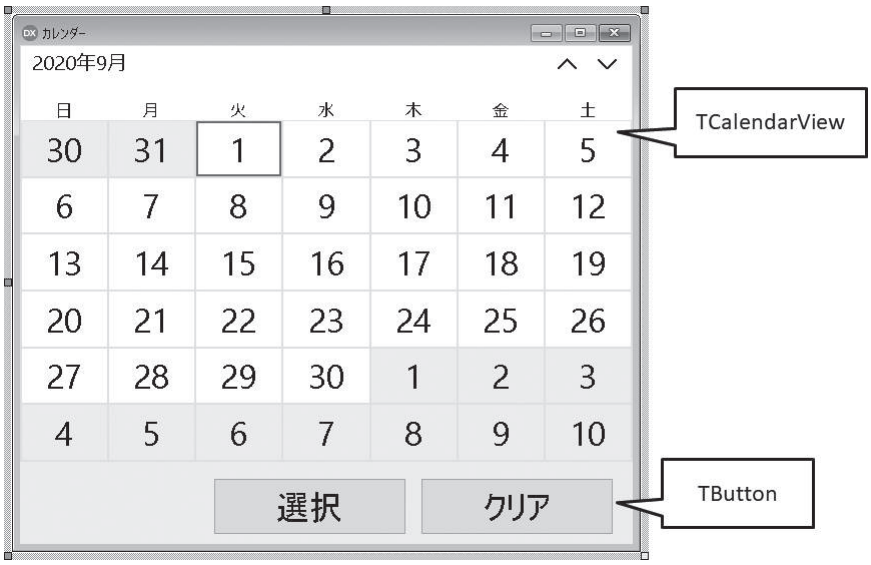
③時計画面のクリアボタン onClick イベントの実装

クリアボタンをクリック時に、メイン画面の時間入力 Edit の時間をクリアするため、プロパティの時間入力 Edit にブランクをセットする。【ソース 10】

④メイン画面の時間 Edit の onEnter イベント

メイン画面の時間入力 Edit にフォーカスがセットされたタイミングで、時計画面を表示するために、onEnter イベントにて時計画面の生成･表示処理を行う。

図4　コンポーネントの配置



ソース1

【カレンダー画面】グローバル変数

```
private
  { Private 宣言 }
  FDateEdit: TEdit;  // 日付入力Edit

public
  { Public 宣言 }
  property DateEdit: TEdit read FDateEdit write FDateEdit;  // 日付入力Edit
end;
```

メイン画面の日付入力Edit用のプロパティ

ソース2

【カレンダー画面】onShowイベント

```
procedure TfrmCalendar.FormShow(Sender: TObject);
var
  sDate: String;  // システム日付
begin
  // システム日付の取得
  sDate := FormatDateTime('YYYY/MM/DD', Now);

  // 遷移元画面の日付＝ブランクの場合、カレンダーにシステム日付をセット
  if (FDateEdit.Text = '') then
  begin
    CalendarView1.Date := StrToDate(sDate);
  end
  // 遷移元画面の日付≠ブランクの場合、カレンダーに遷移元画面の日付をセット
  else
  begin
    CalendarView1.Date := StrToDate(FDateEdit.Text);
  end;
end;
```

メイン画面の日付入力Editの日付 or
システム日付をカレンダーにセット

ソース3

【カレンダー画面】選択ボタンonClickイベント

```
procedure TfrmCalendar.Button1Click(Sender: TObject);
begin
  // 遷移元画面の日付に選択値をセット
  FDateEdit.Text := FormatDateTime('YYYY/MM/DD', CalendarView1.Date);

  // 画面を終了する
  Self.Close;
end;
```

その際、プロパティの受け渡しと、時計画面の表示位置を調整する。【ソース 11】

以上で、時計を使用した時間入力の実装は完了である。

2-4. 動作比較

日付、時間の入力について、本章のサンプルプログラムで入力した場合と、ソフトウェアキーボードより入力した場合を比較して確認してみる。

ソフトウェアキーボードから入力した場合、日付または時間を 1 文字ずつ入力、もしくはクリアする必要がある。また、日付や時間でない値を誤入力する可能性もある。【図 8 ～ 9】

それに対しサンプルプログラムのカレンダー、時計より入力する場合、入力値を選択するため誤入力もなく、簡単に入力・クリアしていると確認できる。【図 10 ～ 12】

またカレンダー、時計での入力のほうが、ソフトウェアキーボードでの入力よりも入力完了までのステップを少なく済ませられる。

3. 画面サイズに合わせた項目の自動調整

3-1. 複数端末でのアプリケーション使用時の課題

Windows タブレット向け VCL アプリケーションを作成する場合、Windows OS であれば特定機種のタブレット端末だけではなく、複数機種のタブレット端末や PC でもアプリケーションを使用できる。

ただし複数機種で 1 つのアプリケーションを使用する場合、画面サイズの違いから項目の位置やサイズが課題となるだろう。それぞれの画面サイズにそれほど差がない場合は問題ないが、そうでない場合は使用する端末に合わせてそれぞれの画面を設計する必要がある。また使用する機種が増えた場合、その画面に合わせて新たに画面を設計せねばならない。【図 13】

そこで 1 つのアプリケーションを複数機種で使用する場合、それぞれの機種の画面サイズに合わせて項目の位置やサイズ等を自動調整することで、使用する機種ごとに画面を設計する必要がなくなる。

3-2. 項目の位置・サイズの自動調整

項目の位置・サイズを自動調整するプログラムを作成していく。今回のサンプルプログラムでは、メインで使用する端末を画面解像度 1800 × 1200 のタブレット端末で縦向き使用とし、タブレット端末以外に画面解像度 2560 × 1440 の PC を使用すると想定する。

(1) 画面サイズの設定

画面設計については、メインで使用するタブレット端末の画面を基準に設計していく。そのため、タブレット端末の画面解像度に合わせて画面サイズを Height プロパティ = 1800、Width プロパティ = 1200 とする。

フォームの BorderStyle プロパティが bsSizeable（デフォルト値）の場合、画面サイズが一定サイズ以上大きく、もしくは小さくできないため、bsSizeable 以外に設定する必要がある。今回は bsSingle に設定する。【図 14】

今回はメイン端末をタブレット端末と想定しているため、タブレット端末を基準として画面サイズを設定したが、PC がメイン端末の場合、PC を基準に画面サイズを設定する。

開発環境により、1800 × 1200 では画面サイズが大きく画面設計しにくい場合、縦横比が画面解像度と同様（今回の場合は 3 : 2）となるように調整する。

後続の処理にて画面サイズを調整するため、画面解像度と同サイズでなくても縦横比が崩れてさえいなければサイズ調整しても問題ない。今回のサンプルプログラムでは、900 × 600 にサイズ調整している。

(2) コンポーネントの配置

今回のサンプルプログラムでは、商品検索の画面を想定してコンポーネントを配置する。画面上部に TPanel、TPanel 上に検索条件用の TComboBox、TEdit、検索ボタン用の TButton 並びに TLabel、画面下部に検索結果表示用の TStringGrid を配置する。【図 15】

各コンポーネントは、ParentFont プロパティを False に設定しておく。ParentFont プロパティが True の場合、フォームのフォントサイズを自動調整する際に、各コンポーネントのサイズがフォントサイズに合わせて自動で変更されてしまう。

今回はメイン端末をタブレット端末と想定しているため、2-1 に記載した項目のサイズ、フォントサイズを大きく設計し、項目間の余白を広めにとっている。

(3) 処理の実装

①画面の onCreate イベントの実装

画面起動時に、画面項目のサイズ・位置を調整する【ソース 12】。サイズ・位置調整については、private 宣言部に function GetScale（AForm: TForm）: Currency、procedure MagnificationFrm（AForm: TForm; AScale: Currency）を追加し、追加した独自の関数・手続き内でサイズ・位置を調整する。処理内容の詳細は、以下の②～④に記載する。

②サイズ調整に必要な倍率の算出

まず端末の画面サイズと、アプリケーションの画面サイズをもとに、サイズ調整に使用する倍率を算出する。倍率については、端末の画面の高さを基準に算出し、算出した倍率で調整したアプリケーション画面の幅が端末の画面の幅を超える場合、再度端末の画面の幅を基準に倍率を算出する。【ソース 13】

③画面サイズに合わせた項目サイズ・位置の調整

次に、②で算出した倍率をもとに、画面サイズ、フォームのフォントサイズを調整する【ソース 14】。コンポーネントごとのサイズ、位置調整については、次の④に記載する。

④コンポーネントのサイズ・位置の調整

次にフォームの ComponentCount を取得し、フォーム上に配置されているすべてのコンポーネントに対してサイズ・位置を調整する。【ソース 15】

④-1. TEdit、TComboBox、TButton、TLabel の場合

②で算出した倍率をもとにフォントサイズ、項目のサイズ、項目の位置を調整する【ソース 16】。サンプルソースでは TEdit の場合のみを記載しているが、TComboBox、TButton、TLabel につ

ソース4

【カレンダー画面】カレンダーonDblClickイベント

```
procedure TfrmCalendar.CalendarView1DblClick(Sender: TObject);
begin
  // 選択ボタン押下時処理
  Button1Click(Button1);
end;
```

ソース5

【カレンダー画面】クリアボタンonClickイベント

```
procedure TfrmCalendar.Button2Click(Sender: TObject);
begin
  // 遷移元画面の日付をクリア
  FDateEdit.Text := '';

  // 画面を終了する
  Self.Close;
end;
```

ソース6

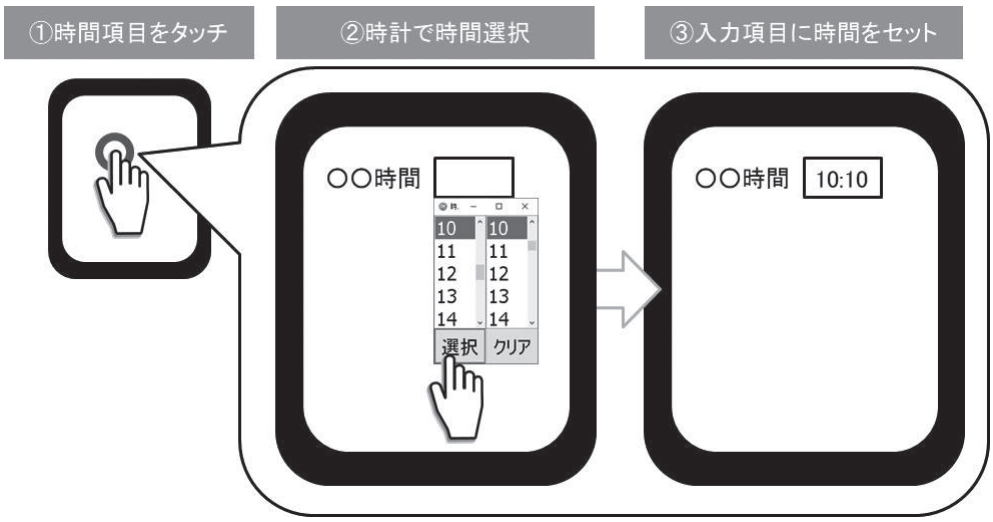
【メイン画面】日付EditのonEnterイベント

```
procedure TForm4.edtCalendarEnter(Sender: TObject);
begin
  // カレンダーフォームの起動
  frmCalendar := TfrmCalendar.Create(Self);

  // 日付入力Editのプロパティ
  frmCalendar.DateEdit := edtCalendar;

  // Left位置調整
  frmCalendar.Left := Self.Left                          // 画面のLeft位置
                    + edtCalendar.Left;                  // EditのLeft位置

  // Top位置調整
  frmCalendar.Top := GetSystemMetrics(SM_CYCAPTION)      // タイトルの高さ
                   + Self.Top                            // 画面のTop位置
                   + edtCalendar.Top                     // EditのTop位置
                   + edtCalendar.Height;                 // Editの高さ

  // カレンダーフォームの表示
  frmCalendar.Show;
end;
```

カレンダー画面のプロパティに日付入力Editをセットする

カレンダー画面の表示位置の調整

図5　時間の時計入力



いても同様に設定する。

④-2. TPanel の場合

④-1 と同様、フォントサイズ、項目のサイズ、位置を調整する。TPanel の場合、Align プロパティ = alTop、alBottom、alClient の際、項目幅は画面幅と同値になるため、Width プロパティの設定は不要である。

同様に、Align プロパティ = alRight、alLeft、alClient の場合、項目の高さは画面の高さと同値になるため、Height プロパティの設定は不要である。

さらに、Align プロパティ = alNone、alCustom 以外の場合、項目の位置は画面の空き領域の左端・上端になるため、Left プロパティ、Top プロパティの設定は不要である。【ソース 17】

④-3. TStringGrid の場合

④-1 と同様、フォントサイズ、項目のサイズ、位置を調整する。TStringGrid の場合、項目のサイズ調整のみでなく、明細の行列情報のサイズ調整も必要である。

設定が必要なプロパティは DefaultColWidth プロパティ、DefaultRowHeight プロパティ、ColWidths プロパティ、RowHeights プロパティであり、基本的にはフォントサイズ等と同様に調整する。

ただし、ここで 1 つ注意点がある。ColWidths プロパティと RowHeights プロパティは、DefaultColWidth プロパティと DefaultRowHeight プロパティの変更値に合わせて自動設定される。そのため、いったん値を内部保持したあと、再セットし直す必要がある。【ソース 18】

以上で、項目の位置・サイズの自動調整の実装は完了である。

上記プログラムを実行した場合と、自動調整を行わずにそのままの状態で表示した場合とを比較して確認してみよう。

自動調整を行わないアプリケーションを PC で実行時、表示スケール（Windows 10 のディスプレイの設定における拡大縮小の設定）100％の場合、アプリケーションが PC 画面内に収まって表示される。

しかし今回の PC の解像度では、表示スケール 200％がデフォルト設定（推奨）されており、その場合アプリケーションの一部が PC 画面内に表示されない。【図 16】

またタブレット端末で実行時、画面解像度の 1/2 のサイズで設計したため、タブレット画面に対して小さく表示される。【図 17】

それに対してサンプルプログラムの項目の場合、端末の画面サイズに対してアプリケーションのサイズ、各項目のサイズ、位置、フォントサイズを自動調整できていることが確認できる。

また PC の場合、自動調整前は表示スケールごとにアプリケーションのサイズが異なっていたが、自動調整により表示スケールに依存せず自動調整できている。【図 18 ～ 19】

4. さいごに

本稿では、Windows タブレット向け VCL アプリケーションを作成する際の作成テクニックについて紹介してきた。

PC の代わりにタブレット端末をビジネスで活用する場面が増えたことで、Windows タブレット向けアプリケーションを開発する局面も増えているだろう。VCL フレームワークを選択することで、これまでと同様の手法での開発が可能となる反面、本稿で取り上げたような PC とは操作方法が異なる点や、PC とタブレット端末双方の使用を考慮することなど、UI ／ UX を考慮して、いかにユーザーが見やすく、使いやすいアプリケーションを作成するかが重要になる。

Windows タブレット向け VCL アプリケーションを作成時に、本稿で紹介した開発テクニックを役立てていただければ幸いである。

M

図6　コンポーネントの配置

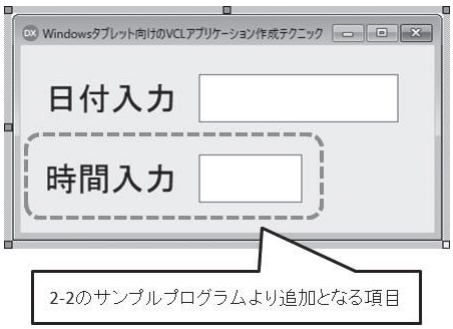
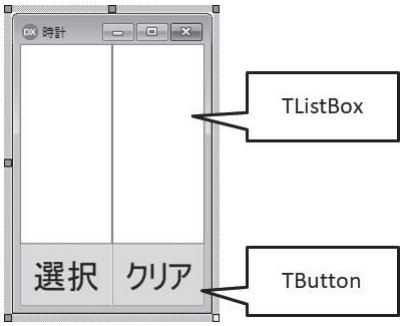


図7　コンポーネントの配置



ソース7

【時間画面】グローバル変数

```
private
  { Private 宣言 }
  FTimeEdit: TEdit;   // 時間入力Edit

public
  { Public 宣言 }
  property TimeEdit: TEdit read FTimeEdit write FTimeEdit;   // 時間入力Edit
end;
```

メイン画面の時間入力Edit用のプロパティ

ソース8

【時間画面】onShowイベント

```
procedure TfrmTime.FormShow(Sender: TObject);
var
  i: Integer;
  sTime: String;  // システム時間
begin
  // 初期化
  ListBox1.Items.Clear;  // 時間
  ListBox2.Items.Clear;  // 分

  // 時間のセット
  for i := 0 to 23 do
  begin
    ListBox1.Items.Add(FormatFloat('00', i));
  end;

  // 分のセット
  for i := 0 to 59 do
  begin
    ListBox2.Items.Add(FormatFloat('00', i));
  end;

  // システム時間の取得
  sTime := FormatDateTime('HH:MM', Now);

  // 遷移元画面の時間＝ブランクの場合、時間にシステム時間をセット
  if (FTimeEdit.Text = '') then
  begin
    ListBox1.ItemIndex := ListBox1.Items.IndexOf(Copy(sTime, 1, 2));
    ListBox1.TopIndex  := ListBox1.ItemIndex;
    ListBox2.ItemIndex := ListBox2.Items.IndexOf(Copy(sTime, 4, 2));
    ListBox2.TopIndex  := ListBox2.ItemIndex;
  end
  // 遷移元画面の時間≠ブランクの場合、時間に遷移元画面の時間をセット
  else
  begin
    ListBox1.ItemIndex := ListBox1.Items.IndexOf(Copy(FTimeEdit.Text, 1, 2));
    ListBox1.TopIndex  := ListBox1.ItemIndex;
    ListBox2.ItemIndex := ListBox2.Items.IndexOf(Copy(FTimeEdit.Text, 4, 2));
    ListBox2.TopIndex  := ListBox2.ItemIndex;
  end;
end;
```

時間と分のTListBoxにそれぞれの選択値をセット

メイン画面の時間入力Editの時間
orシステム時間を時計にセット

ソース9

【時計画面】選択ボタンonClickイベント

```
procedure TfrmTime.Button1Click(Sender: TObject);
begin
  // 選択時間をセット
  FTimeEdit.Text := ListBox1.Items[ListBox1.ItemIndex]   // 時間
                  + ':'
                  + ListBox2.Items[ListBox2.ItemIndex];  // 分

  // 画面を終了する
  Self.Close;
end;
```

ソース10

【時計画面】クリアボタンonClickイベント

```
procedure TfrmTime.Button2Click(Sender: TObject);
begin
  // 遷移元画面の時間をクリア
  FTimeEdit.Text := '';

  // 画面を終了する
  Self.Close;
end;
```

ソース11

【メイン画面】時間EditのonEnterイベント

```
procedure TForm4.edtTimeEnter(Sender: TObject);
begin
  // 時間フォームの起動
  frmTime := TfrmTime.Create(Self);

  // 時間入力Editのプロパティ
  frmTime.TimeEdit := edtTime;

  // Left位置調整
  frmTime.Left := Self.Left                          // 画面のLeft位置
                + edtTime.Left;                      // EditのLeft位置

  // Top位置調整
  frmTime.Top := GetSystemMetrics(SM_CYCAPTION)      // タイトルの高さ
               + Self.Top                            // 画面のTop位置
               + edtTime.Top                         // EditのTop位置
               + edtTime.Height;                     // Editの高さ

  // 時間フォームの表示
  frmTime.Show;
end;
```

時計画面のプロパティに時間入力Editをセットする

時計画面の
表示位置の調整

図8　ソフトウェアキーボードを使用した日付入力

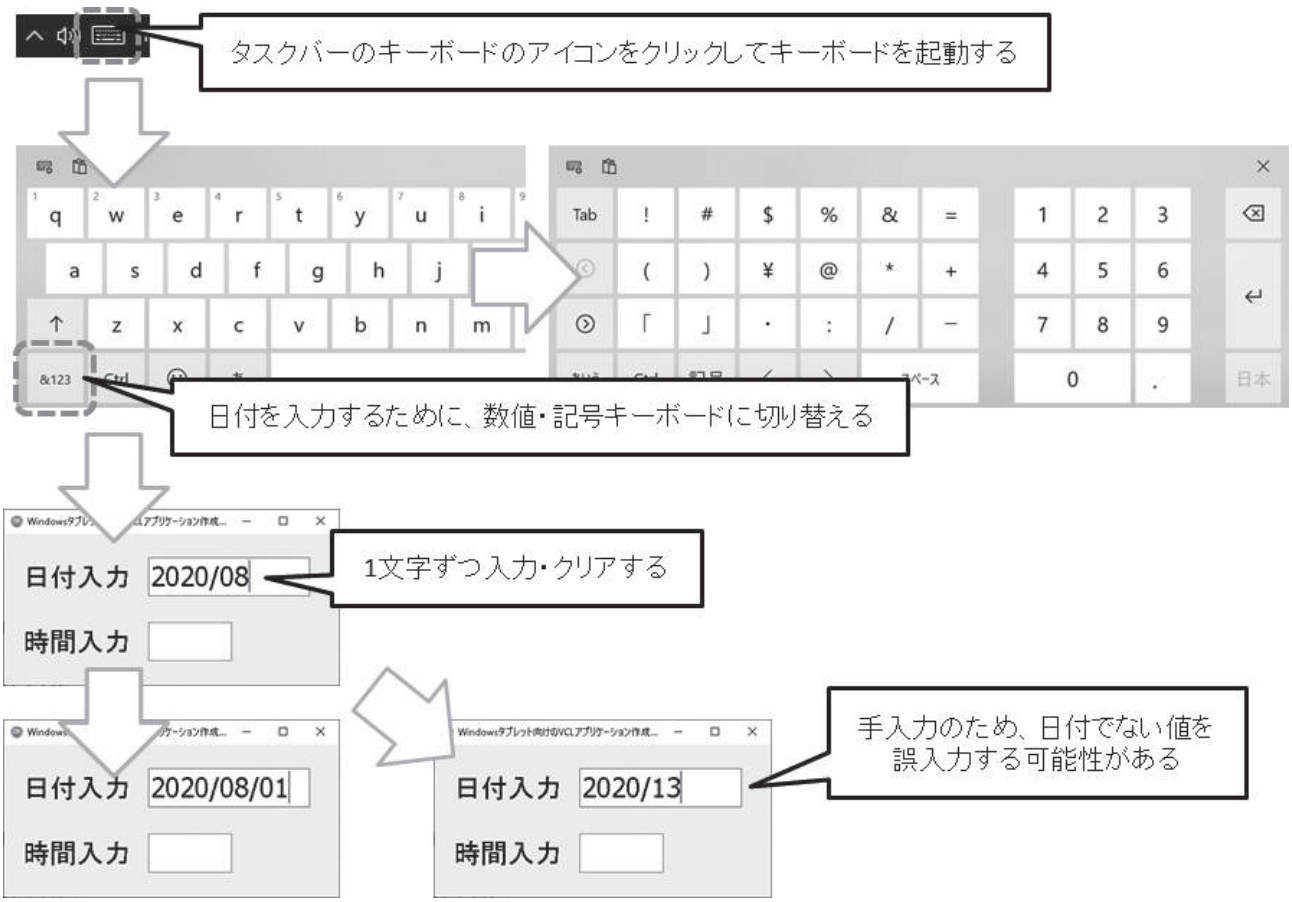


図9　ソフトウェアキーボードを使用した時間入力

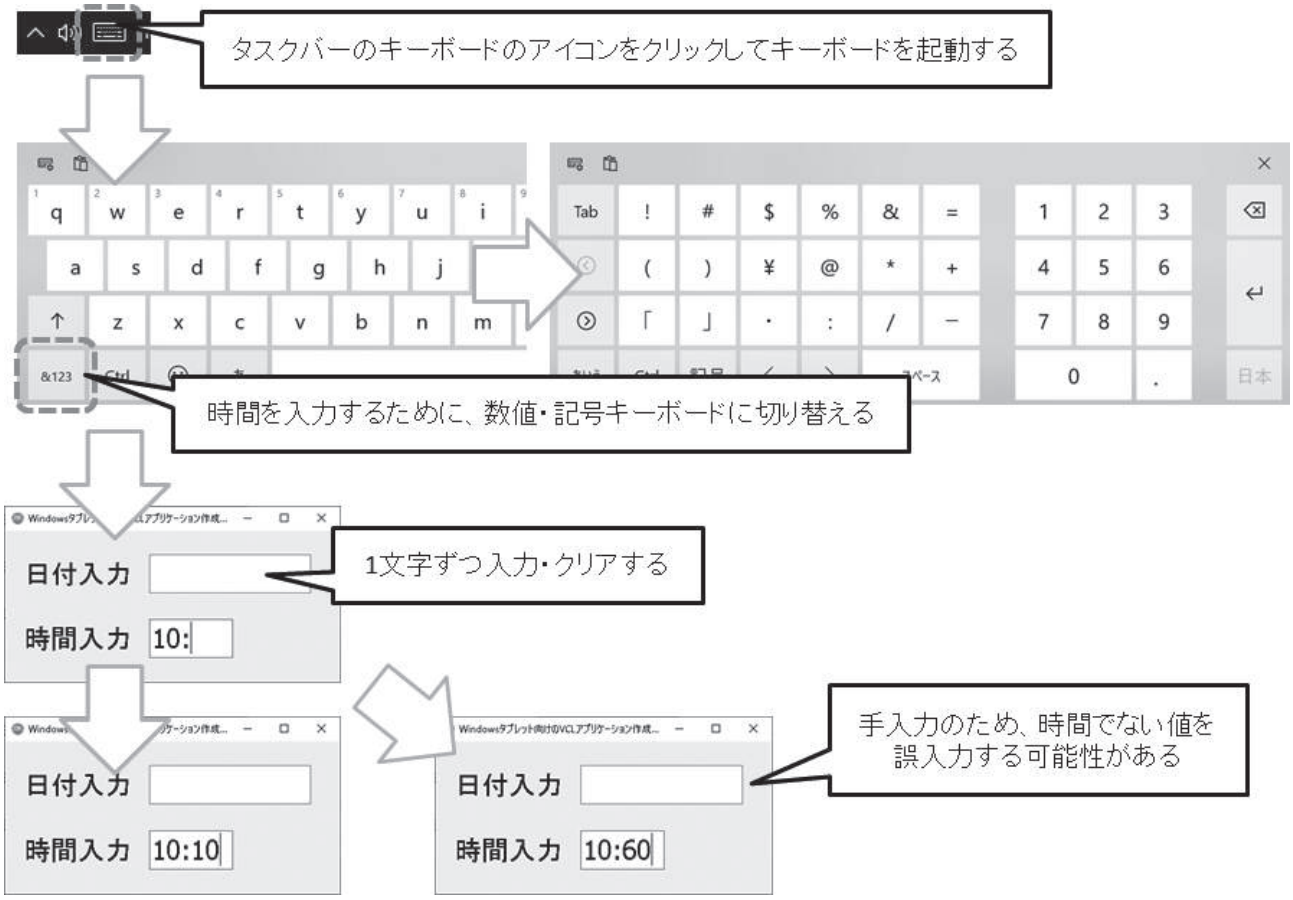


図10　カレンダー画面を使用した日付入力①

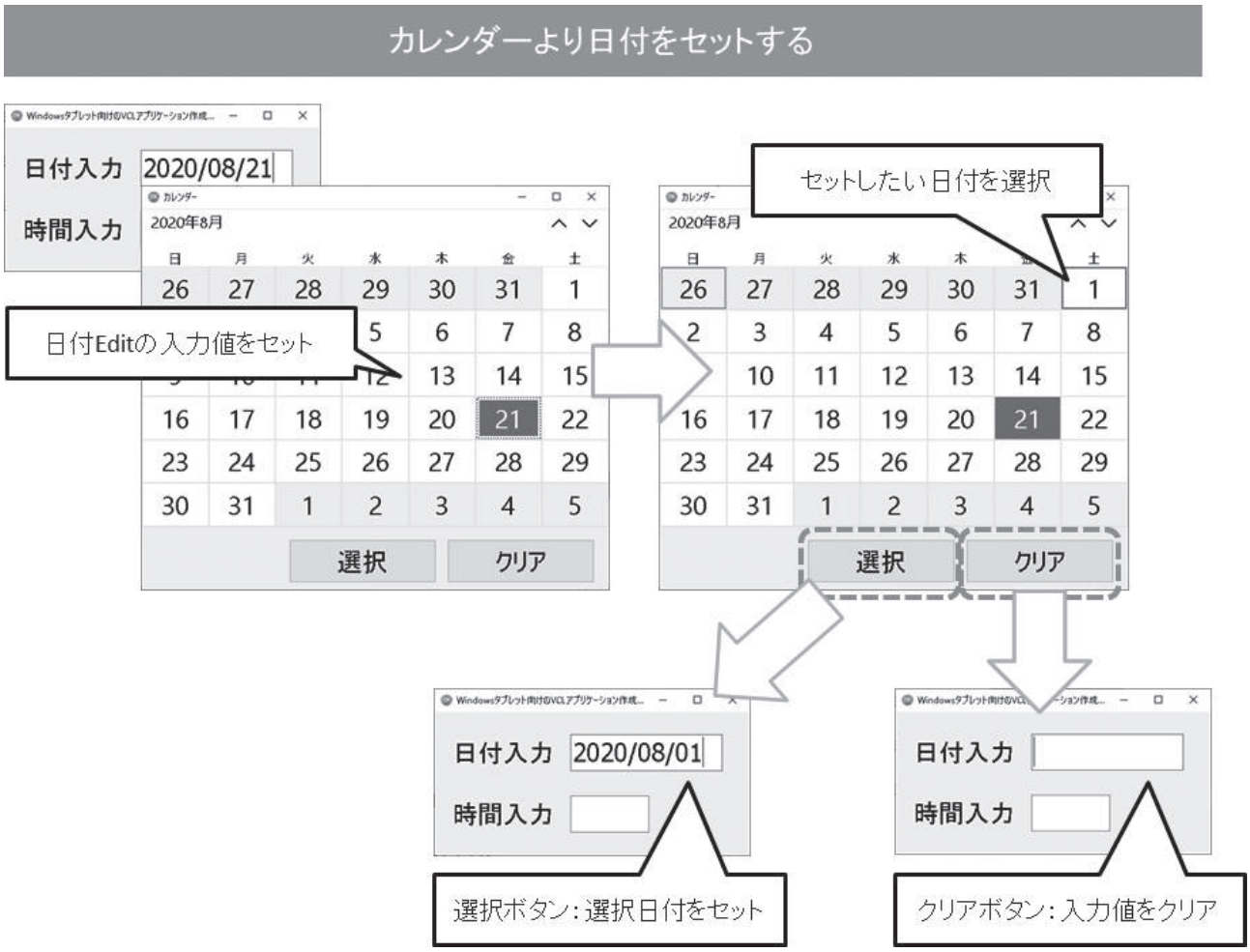


図11　カレンダー画面を使用した日付入力②

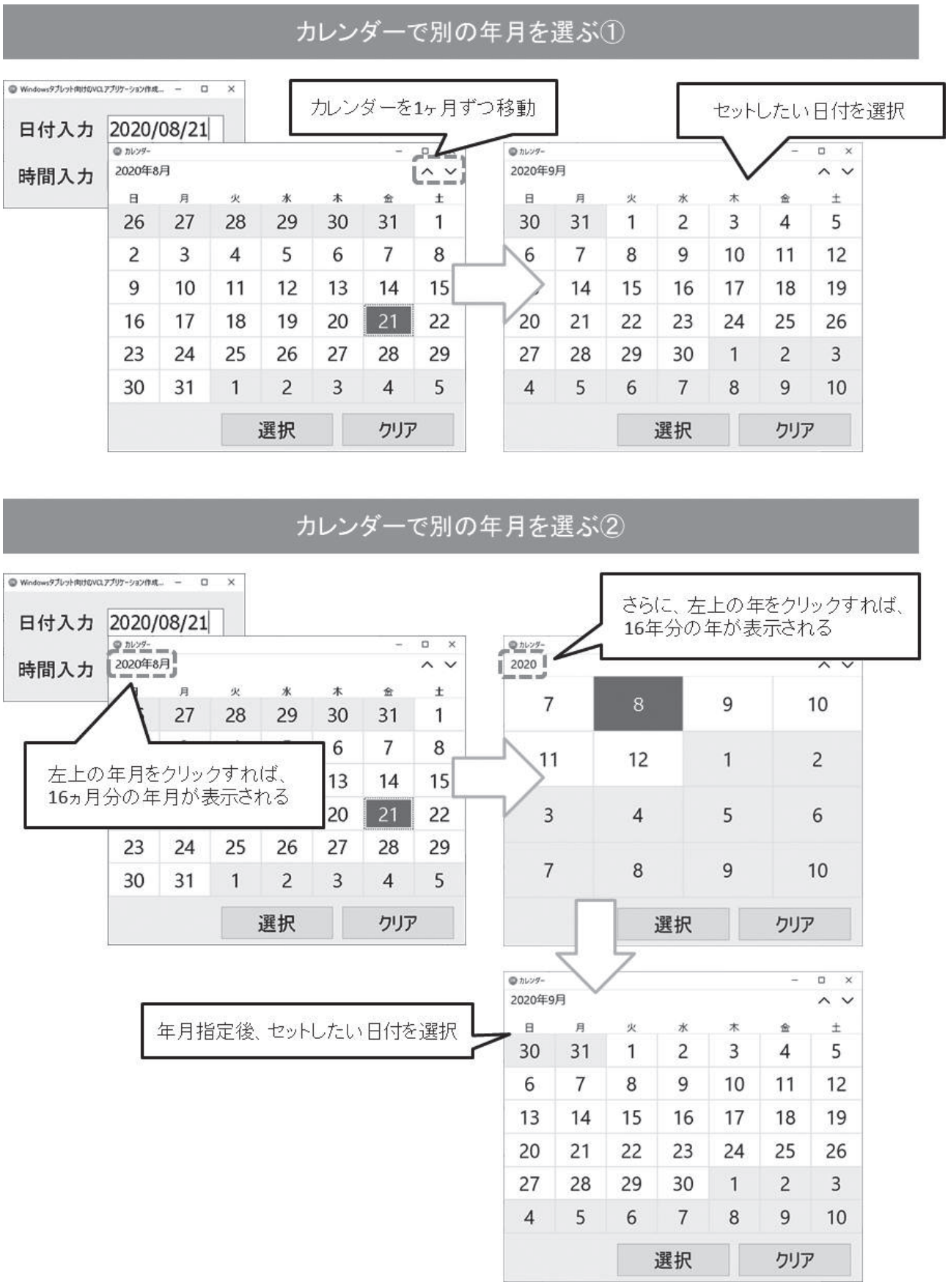


図12　時計画面を使用した時間入力

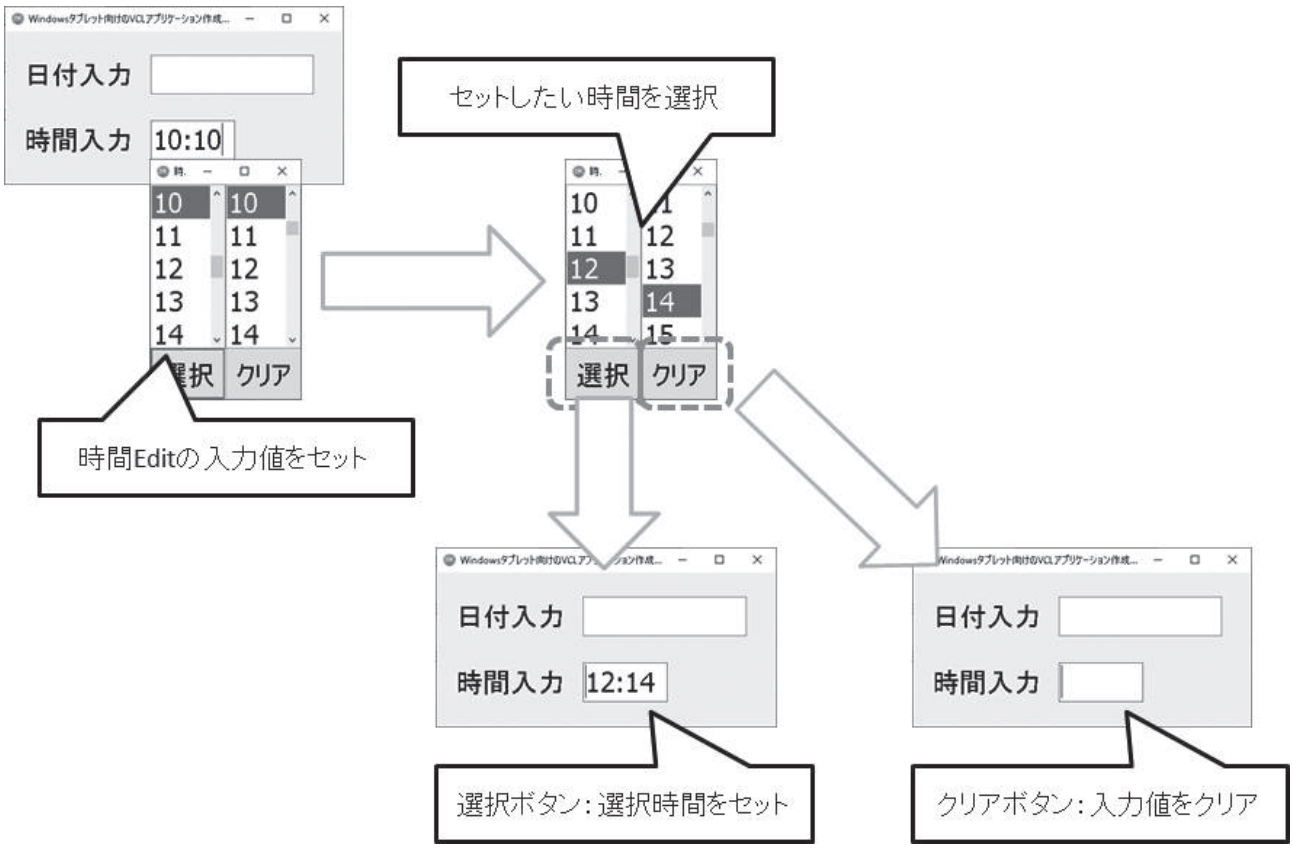


図13　アプリケーションを複数で使用する場合の画面設計

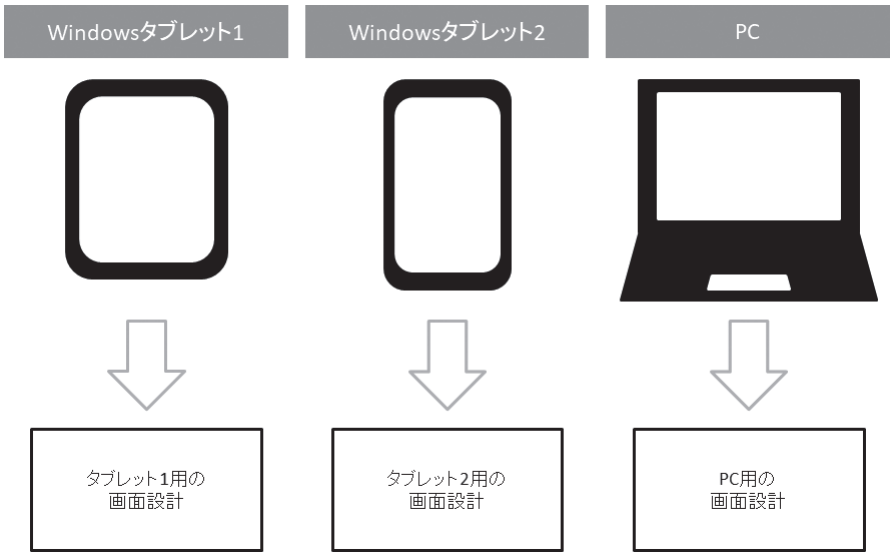


図14　画面サイズの設定

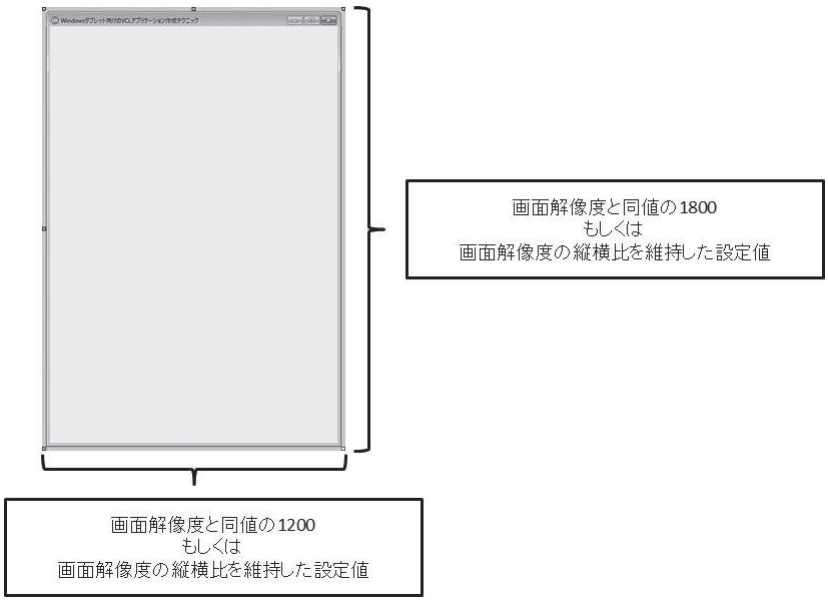
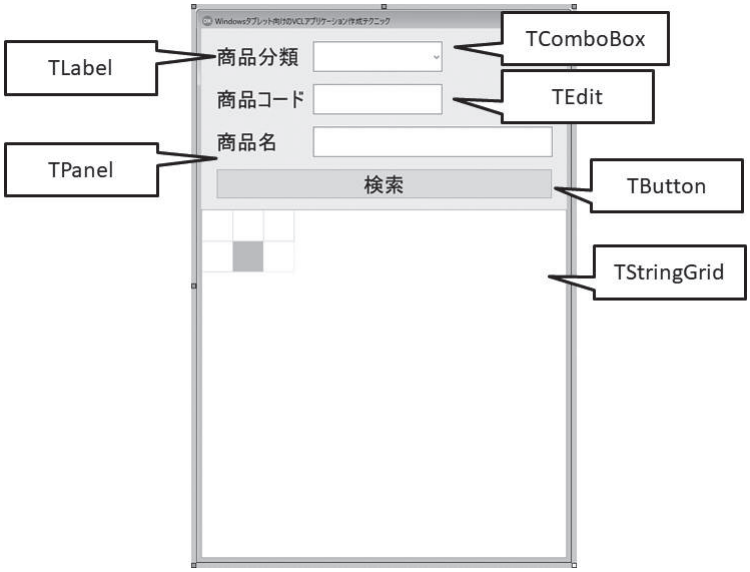


図15　コンポーネントの配置



ソース12

画面のonCreateイベント

```
procedure TForm1.FormCreate(Sender: TObject);
var
  cScale: Currency;  // 倍率
begin
  // 倍率を取得
  cScale := GetScale(Self);

  // 画面等倍処理
  MagnificationFrm(Self, cScale);
end;
```

ソース13

ソース14

ソース13

サイズ調整に必要な倍率の算出

```
function TForm1.GetScale(AForm: TForm): Currency;
var
  cScale: Currency;
begin
  // 画面の高さを基準に算出
  cScale := Screen.Height / 900;

  // 調整した幅が画面幅を超えている場合、画面の幅を基準に再度算出
  if (Trunc(AForm.Width * cScale) > Screen.Width) then
  begin
    cScale := Screen.Width / 600;
  end;

  // 算出結果を返却
  Result := cScale;
end;
```

ソース14

画面サイズに合わせた項目サイズ・位置の調整

```
procedure TForm1.MagnificationFrm(AForm: TForm; AScale: Currency);
var
  i: Integer;
begin
  // 倍率＝1の場合、処理終了
  if (AScale = 1) then
  begin
    Exit;
  end;

  // 画面大きさ制御
  if (Screen.Width <> AForm.Width) then
  begin
    // 大きさ変更
    for i := 0 to AForm.ComponentCount - 1 do
    begin
      ComponentScale(AForm, AForm.Components[i], AScale);
    end;

    // 画面サイズ
    AForm.Width  := Trunc(AForm.Width  * AScale);
    AForm.Height := Trunc(AForm.Height * AScale);

    // フォントサイズ
    AForm.Font.Size := Trunc(AForm.Font.Size * AScale);
  end;
end;
```

ソース15

ソース15

コンポーネント毎のサイズ・位置の調整①メイン処理

```
procedure TForm1.ComponentScale(AForm: TForm; AComponent: TComponent;
  AScale: Currency);
var
  i: Integer;
  slGridColWidth: TStringList;    // TStringGridの明細列幅内部保持用
  slGridRowHeight: TStringList;   // TStringGridの明細行の高さ内部保持用
begin
  // すでに変更されている場合、処理しない
  if (AComponent.Owner is TForm) and
     (Screen.Width = TForm(AComponent.Owner).Width) then
  begin
    Exit;
  end;
```

＜コンポーネント毎のサイズ・位置の調整＞
ソース16～19

```
end;
```

ソース16

コンポーネント毎のサイズ・位置の調整②TEdit、TComboBox、Tbutton、TLabel

```
// TEdit
if (AComponent is TEdit) then
begin
  with TEdit(AComponent) do
  begin
    // フォントサイズ
    Font.Size := Trunc(Font.Size * AScale);

    // サイズ
    Width  := Trunc(Width  * AScale);  // 幅
    Height := Trunc(Height * AScale);  // 高さ

    // 位置
    Left := Trunc(Left * AScale);  // Left位置
    Top  := Trunc(Top  * AScale);  // Top位置
  end;
end
else
```

コンポーネント毎のサイズ・位置の調整③TPanel

```
// TPanel
if (AComponent is TPanel) then
begin
  if (TPanel(AComponent).Align <> alClient) then
  begin
    with TPanel(AComponent) do
    begin
      // フォントサイズ
      Font.Size := Trunc(Font.Size * AScale);

      // 幅
      if not (Align in [alTop, alBottom, alClient]) then
      begin
        Width := Trunc(Width  * AScale);
      end;

      // 高さ
      if not (Align in [alRight, alLeft, alClient]) then
      begin
        Height := Trunc(Height * AScale);
      end;

      // 表示位置を指定していない場合
      if (Align in [alNone, alCustom]) then
      begin
        Left := Trunc(Left * AScale);  // Left位置
        Top  := Trunc(Top  * AScale);  // Top位置
      end;
    end;
  end;
end
else
```

ソース18

コンポーネント毎のサイズ・位置の調整⑤TStringGrid

```
// TStringGrid
if (AComponent is TStringGrid) then
begin
  with TStringGrid(AComponent) do
  begin
    // フォントサイズ
    Font.Size := Trunc(Font.Size * AScale);

    // 明細自体のサイズ
    if (not AutoSize) then
    begin
      Width  := Trunc(Width  * AScale);  // 幅
      Height := Trunc(Height * AScale);  // 高さ
    end;

    // 位置
    Left := Trunc(Left * AScale);  // Left位置
    Top  := Trunc(Top  * AScale);  // Top位置

    // サイズ変更前の状態を内部保持用
    slGridColWidth  := TStringList.Create;  // 幅
    slGridRowHeight := TStringList.Create;  // 高さ

    try
      // 明細のデフォルトサイズを変更すると全て同一値に変更されるため
      // 変更前の幅と高さを内部保持する
      // 明細の列幅
      for i := 0 to ColCount - 1 do
      begin
        slGridColWidth.Add(IntToStr(ColWidths[i]));
      end;

      // 明細の行の高さ
      for i := 0 to RowCount - 1 do
      begin
        slGridRowHeight.Add(IntToStr(RowHeights[i]));
      end;

      // 明細のデフォルトサイズ
      DefaultColWidth  := Trunc(DefaultColWidth  * AScale);  // 幅
      DefaultRowHeight := Trunc(DefaultRowHeight * AScale);  // 高さ

      // 明細の列幅（内部保持値より変換）
      for i := 0 to slGridColWidth.Count - 1 do
      begin
        ColWidths[i] := Trunc(StrToInt(slGridColWidth[i]) * AScale);
      end;

      // 明細の行の高さ（内部保持値より変換）
      for i := 0 to slGridRowHeight.Count - 1 do
      begin
        RowHeights[i] := Trunc(StrToInt(slGridRowHeight[i]) * AScale);
      end;
    finally
      FreeAndNil(slGridColWidth);
      FreeAndNil(slGridRowHeight);
    end;
  end;
end;
```

ColWidthsプロパティ、RowHeightsプロパティの設定値を内部保持

DefaultColWidthプロパティ、DefaultRowHeightプロパティの調整

内部保持値を基にColWidthsプロパティ、RowHeightsプロパティを調整

図16　項目の自動調整を行わない場合（PC）

表示スケール100%

表示スケール200%

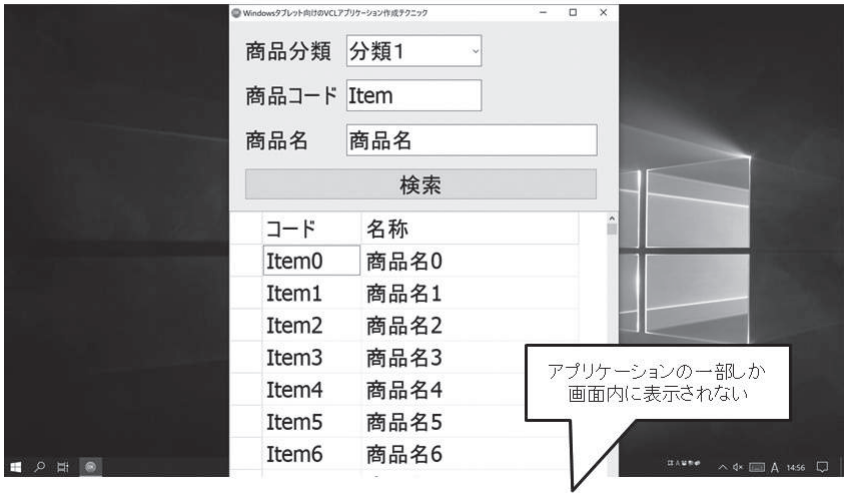


図17　項目の自動調整を行わない場合（タブレット）

図18　項目の自動調整を行った場合（PC）

表示スケール100%

表示スケール200%

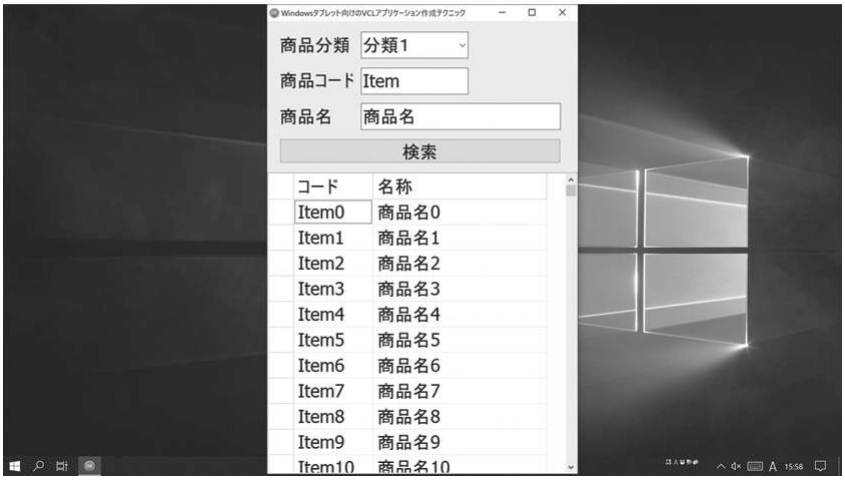
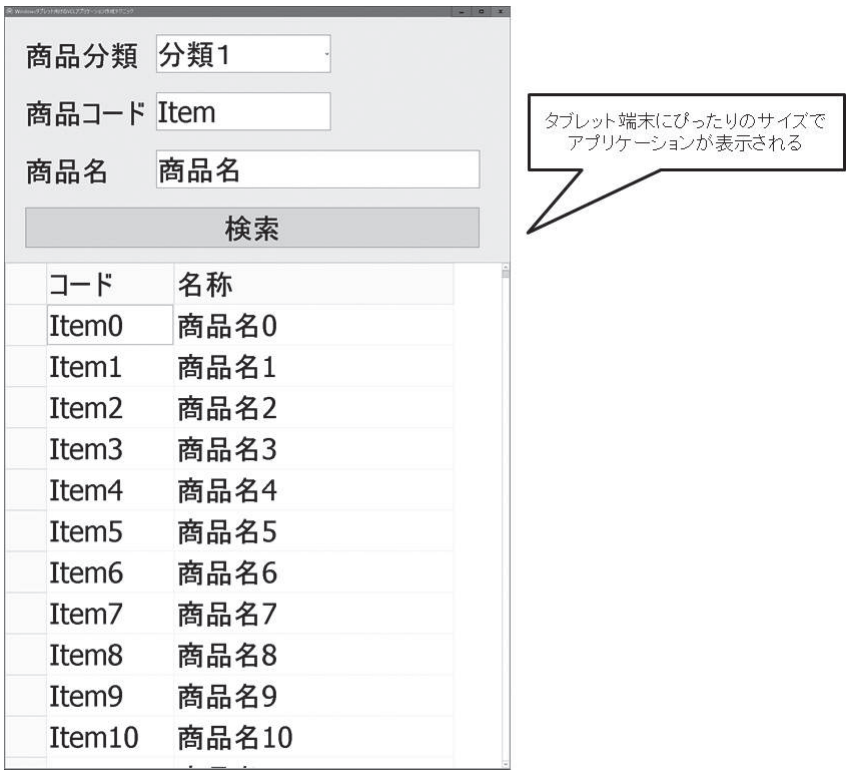


図19　項目の自動調整を行った場合（タブレット）



タブレット端末にぴったりのサイズで
アプリケーションが表示される