Visual Studio2008 C#で
FoxProに似た数値入力を行う

Object Oriented XBASE Forum
Nobuyuki Ichioka
掲載開始日:2009年2月5日

xBASE言語製品が市場から撤退するにつれ、FoxProや他のxBASE言語ツールでアプリケーション開発を行っていらした方にとって、どの開発言語を選択するかということが緊急の課題になっていることと存じます。

C#は、FoxProの言語体系と親和性が高く、移植にあたってもスムーズに移行しやすいと思います。移植にあたり業務等のロジックは、それほど問題が無いと考えられますが、最も違う部分がマンマシンインターフェース部分です。テキストの入力とグリッドの表示周りが相当な工夫を要する部分です。これからのご説明がVisual Studio 2008 C#で開発を行うためのヒントになると幸いです。

製品版をお持ちでなくても、Visual Studio Express Editionという無償配布のバージョンでもこのご説明を実際に試すことが可能です。また、製品版への優待UPグレードも開始されました。この機会にVisual Studio製品を試してみてください。

FoxProは、TextBoxで入力を実施する場合、多彩な入力用のMaskが提供されていると同時に↑↓カーソルキーやEnterキーでの移動がサポートされています。Visual StudioではmaskedTextBoxというコントロールが用意されていますが、この機能だけでは、満足な入力が行えません。いろいろなユーティリティが外販されていますが、簡単なプログラムで擬似的な入力動作が可能となります。自前の入力処理に挑戦してみましょう。

今回このPDFでご説明する内容はTextBox(maskedTextBox)入力についてのみとなります。初歩的に理解しやすいよう最低限のしくみのご説明になります。よってエラートラップ等を施しておりませんので、業務にご利用になる場合は十分ご配慮願います。

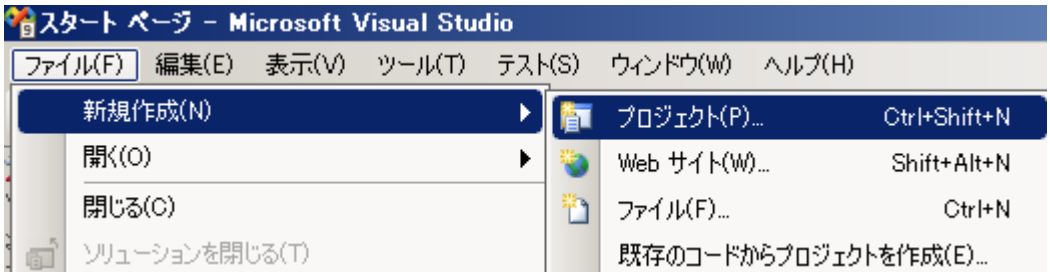
また、ユーザーコントロール化する手法については関数へのアプローチ方法が異なります。あらかじめご理解、ご了承のほどお願い申し上げます。

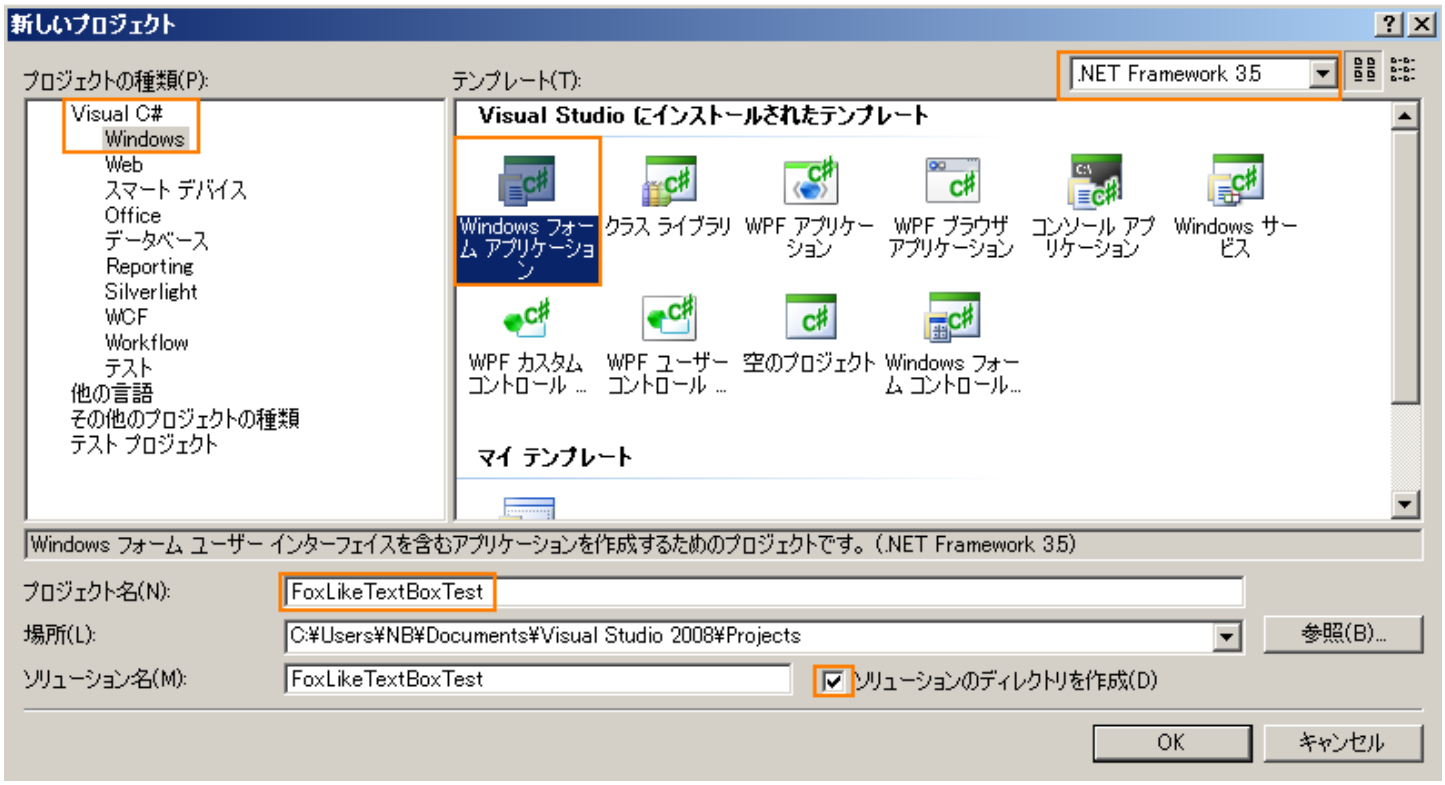
ご説明には Visual Studio Professional Edition SP1を使用しました。

Visual Studioを立ち上げ新規プロジェクトを作る

それでは、VS2008を立ち上げてみましょう。

[ファイル]-[新規作成]から[プロジェクト]を選択します。【新しいプロジェクト】というダイアログが開きます。





【新しいプロジェクト】が開きましたら、プロジェクトの種類-Visual C#-Windowsを選択し、テンプレートのWindowsフォームアプリケーションを選択、そして下段のプロジェクト名に今回は、FoxLikeTextBoxTestと記入してください。”□ソリューションのディレクトリを作成する”にチェックが入っているか、右肩のリストボックスの内容が.NET Framework 3.5になっていることを確認して[OK]ボタンを押してください。

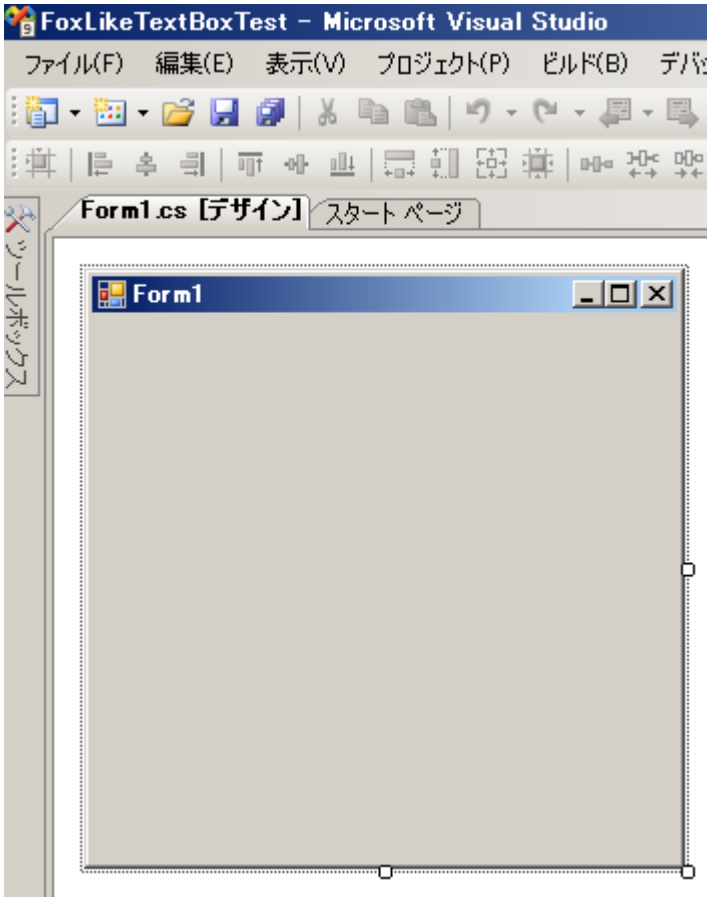
右の図のようなForm1と表示されたダイアログが作成されます。

ダイアログの大きさを変更したい場合は、マウスで四隅の白い”□”をドラッグしたり、辺の中央の”□”をドラッグして動かします。

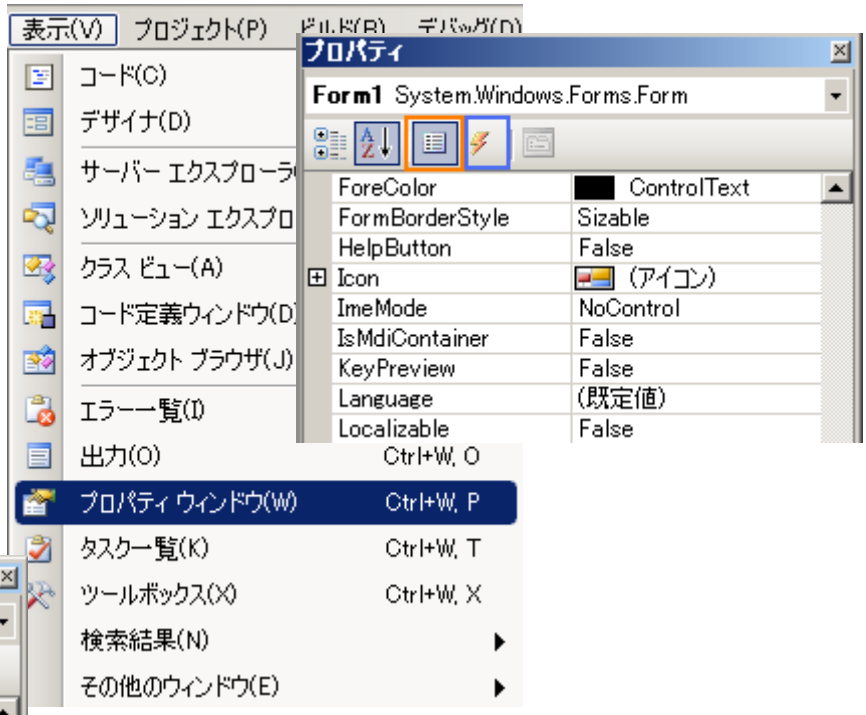
また、プロパティダイアログのSizeプロパティで直接 WidthとHeightを変更することが出来ます。

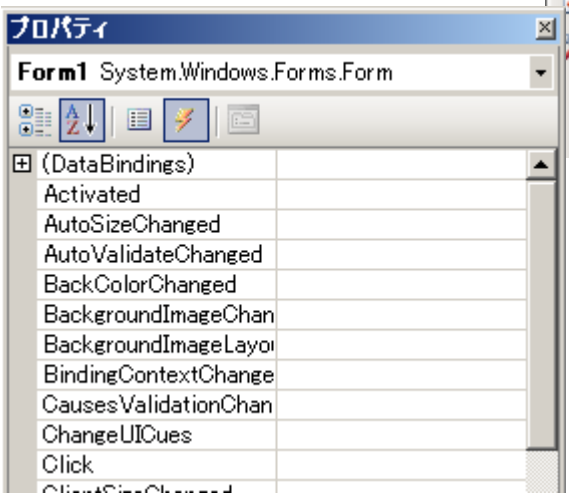
いろいろなウインドウが開きますが誤って閉めて見失った場合は、メニューの[表示]を選択してみてください。

また設計フォーム等のオブジェクトの上でマウス右クリックを行うことでも必要なものを見つけることが出来ます。



プロパティダイアログの上のほうに(右図の)赤枠と青枠で示したアイコンのボタンがあります。ここは、プロパティとイベント入力の表示切り替えボタンとなっています。今回のプログラム作成において使用しますので試しに押してみて確認をしておきましょう。



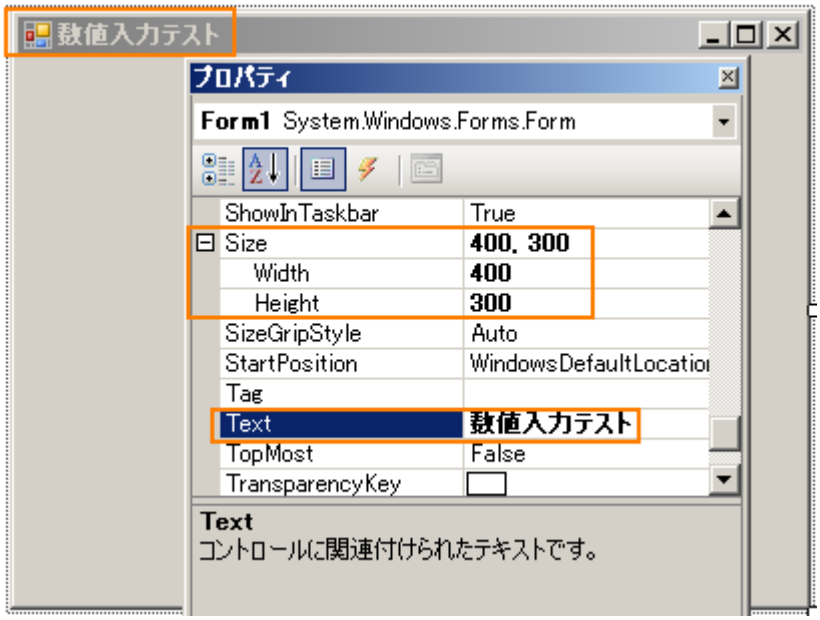


左図でイベントボタンを押した例を示します。
それぞれのコントロール(オブジェクト)で属性の違うプロパティやイベントが表示されます。
プロパティやイベントの各項目名称の上でマウス右クリックを行うとリセットやコマンド説明等の選択が出来ます。

ウインドウのサイズを変える

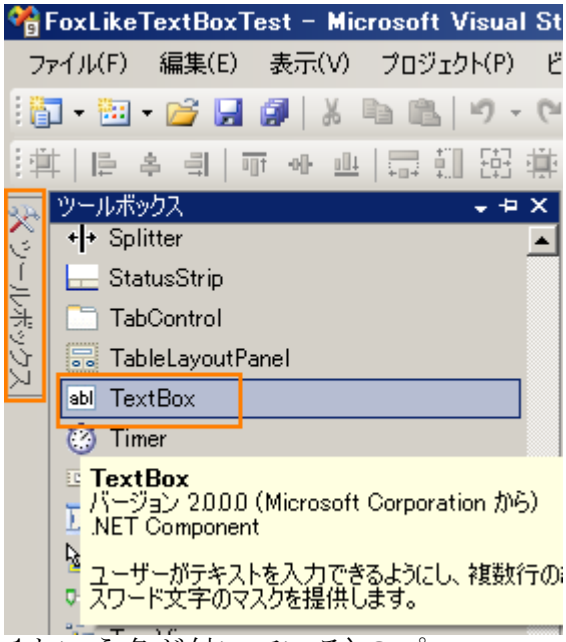
それではアプリケーションの記述を始めましょう。まずは、レイアウトを行い易くするためにウインドウの横幅を多少広げ 横400 縦300にします。

プロパティダイアログのプロパティで[+]Sizeを探してください
[+]をマウスでクリックするとWidthとHeightが展開します。
Widthに400　Heightは300と入力します。
Form1の大きさが変更されます。
また、Textのところに「数値入力テスト」と記入します。
ダイアログのタイトルが変更されます。

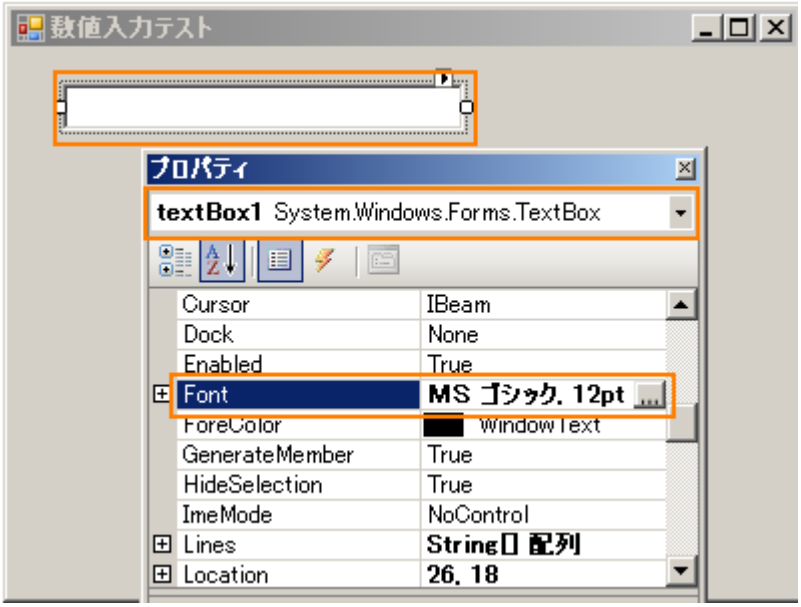


ツールボックスからコントロールを選択する

ダイアログに必要なパーツを設置してみましょう。
単純な入力を行う TextBoxと入力制御が出来るmaskedTextBox、そして念のためボタンを置いてみます。↑↓カーソルキーやリターンキーでの入力フィールドの移動を確認するため TextBoxとmaskedTextBoxは複数置くことにしましょう。



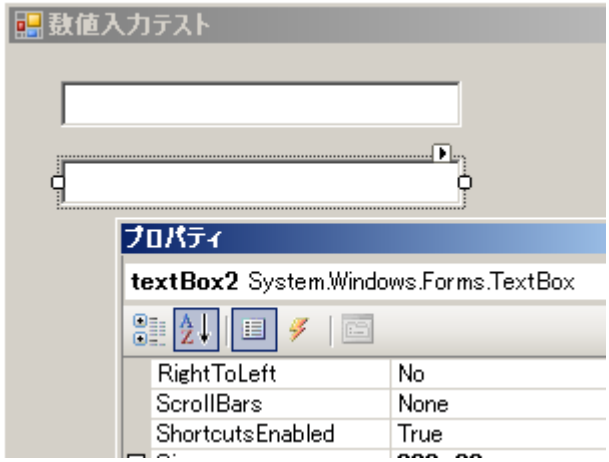
IDE左側面にあるツールボックスタブをクリックしてツールボックスを引き出します。
ツールボックスが見つからない場合は前述のメニュー表示からツールボックスを選択します。
ツールボックスの中からTextBoxを探してワンクリックします。見つからない場合は、ツールボックスの上段にある”すべてのWindowsフォーム”をクリックしてみてください。
TextBoxをワンクリックした後、設計中のダイアログの上で右クリックするとツールボックスが隠れます。(隠れないときは自動で隠れるにしておく) ダイアログの適当なところで左クリックでTextBoxを設置します。大きさはとりあえずWidthを200にしておきます。TextBox(自動でtextBox1という名が付いている)のプロパティでFontをMSゴシック,12ptに設定します。右端のボタンを押すとFont設定ダイアログが開きます。

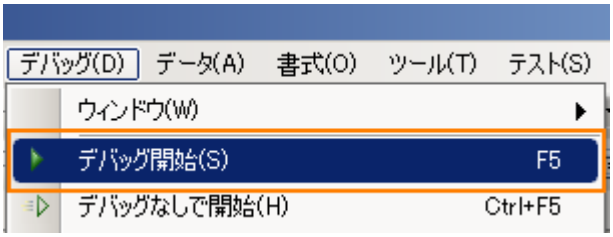


ここでFontを設定しなおすとtextBox1の高さが自動的に変わります。コントロールのサイズを手動で変更したい場合は、マウスやコントロールを選択状態にしてShift+↑↓←→カーソルで大きさを変更することが出来ます。
また、textBox1のレイアウトの右肩にある三角を押すと、複数行入力用のテキストボックスに変更されます。今回は一行入力として利用します。
フォントの設定が終了しましたら、このTextBoxをもう1つ増設します。textBox1を選択状態にして、Ctrl+Cでコピーし Ctrl+Vで貼り付けてください。自動的にフォントやサイズ属性を複写したtextBox2が作成されます。

デバッグを行ってみる

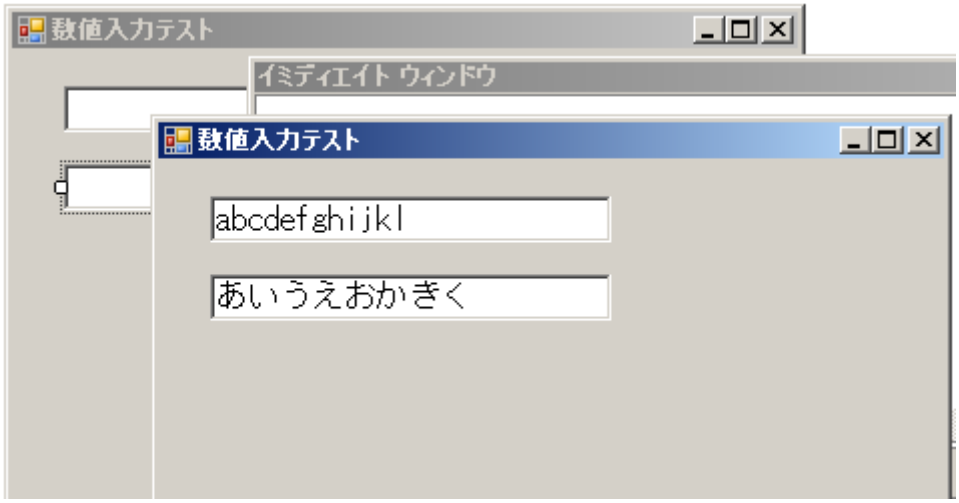
右図のようにTextBoxが2つ設置完了したら試しにダイアログを実行してみましょう。

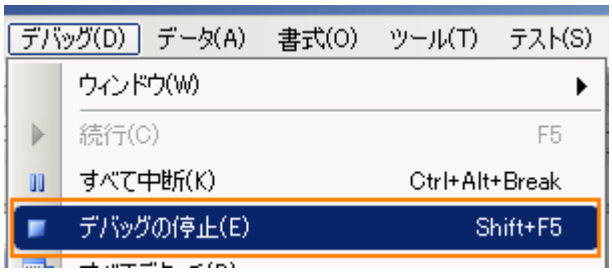




メニューまたはツールバーから[デバック開始]を選択します。

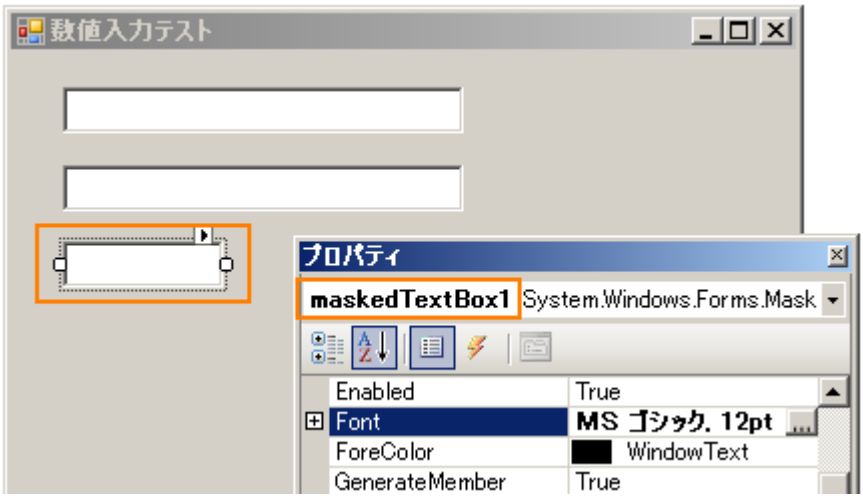
右図のようにダイアログが実行されます。
試しに文字を入力してみましょう。IMEの設定等はプロパティで出来ます。
いろいろと試してTextBoxの動作や色を変えて動きをつかんでみましょう。
ダイアログは動作中のダイアログ自身の右肩クローズボタンで閉めるか、メニューのデバッグの停止を選択することで終了することが出来ます。



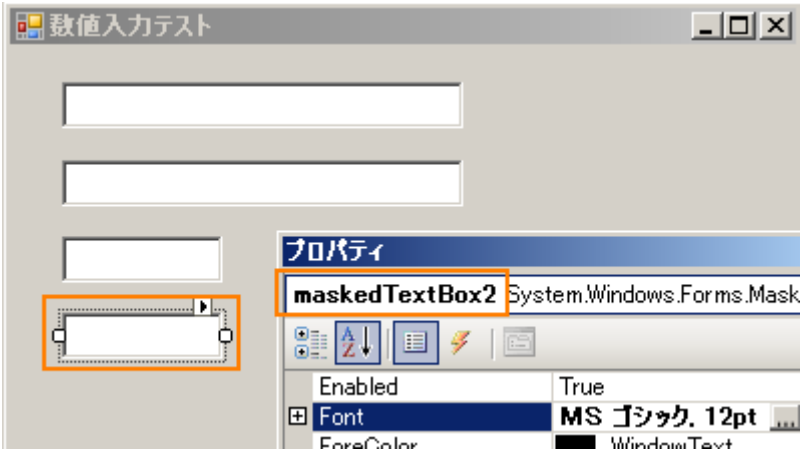


MaskedTextBoxを設置してみる

それでは、単純なTextBoxではなく、書式制御のできるmaskedTextBoxを追加で設置してみましょう。TextBoxのときと同様に[ツールボックス]からmaskedTextBoxを選択してダイアログ上に設置します。サイズはWidth80としておきましょう。FontをMSゴシック,12ptに変更します。
さらにtextBox1をコピーしてTextBox2を作成したのと同様にmaskedTextBox1を選択しCtrl+C⇒Ctrl+Vで MaskedTextBox2を作成してください。

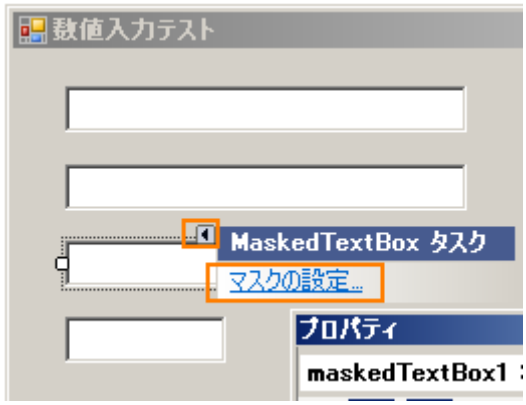


再度デバッグを行ってみる

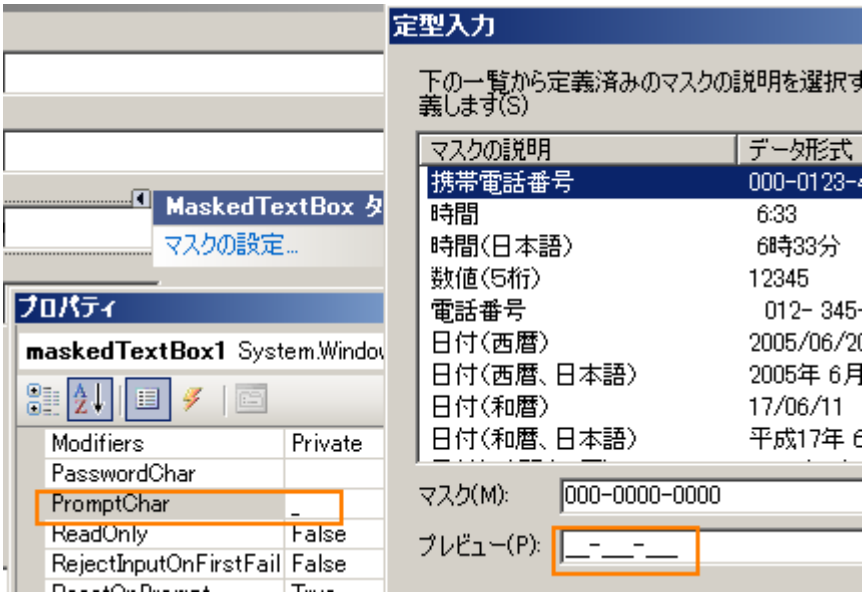


右図のように2つのTextBoxと2つのmaskedTextBoxの設置が完了したら試しにダイアログを実行してみましょう。
当然のことながら入力フィールドの移動は、TABでしか移動出来ません。maskedTextBoxには制御の指示を設定していないので単純なTextBoxとなんら変わらない入力動作であることも確認できます。

MaskedTextBoxのMASKプロパティを設定してみる

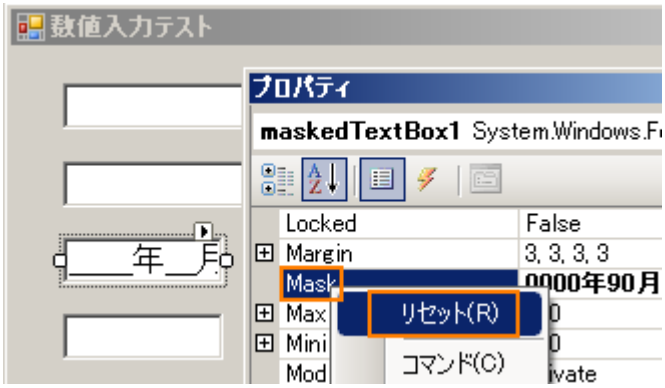


maskedTextBoxに書式制御を設定してみましょう。MaskedTextBox1を選択して右肩にある三角を押すとメニューが表示されます。[マスクの設定]を選択してください。【定型入力】というダイアログが表示されますので、いくつか選択をしてさらにデバッグを選択して、その動作を確認してみましょう。
これで使いやすいし十分だと思われた場合は、この先のご説明はもう必要が無いかもしれません(笑)。
入力フィールドで右図のプレビューにも表示されているアンダースコアが邪魔という場合は、プロパティのPromptCharのアンダースコアを半角のスペース1文字で置き換えてみて下さい。ヌルにはしないで、ホワイトスペースを入れるようにします。この程度の貧弱な入力制御ですと、FoxProをご利用の方ですと非常にストレスになるのではないかと存じます。



特に数値入力をお試し頂きたいのですが、貨幣数値入力については、入力時カンマ無しで入力し入力完了後カンマ区切りで表示されるような 例:[入力時] -123456 ⇒ [入力完了時] -123,456 で表示されるような方法が簡単に設定できません。郵便番号等はこのままの吊るしを使えば大丈夫ですが、数値入力については、もう我慢が出来ない(爆笑)。では、ここにどのような工夫をすると良いのでしょう。以前は1文字入力づつキー値を取ってカーソルの移動制御まで考えたことがありました。これですとキー入力だけで膨大なプログラム量になってしまいます。そこでお手軽で手抜きなプログラミングを考えてみました。

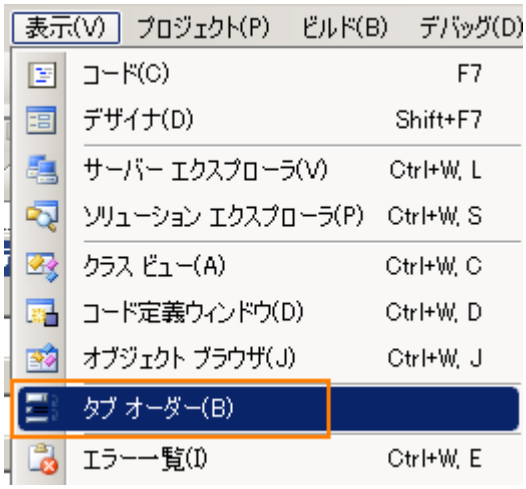
↑↓Enterキーで移動できるようにする

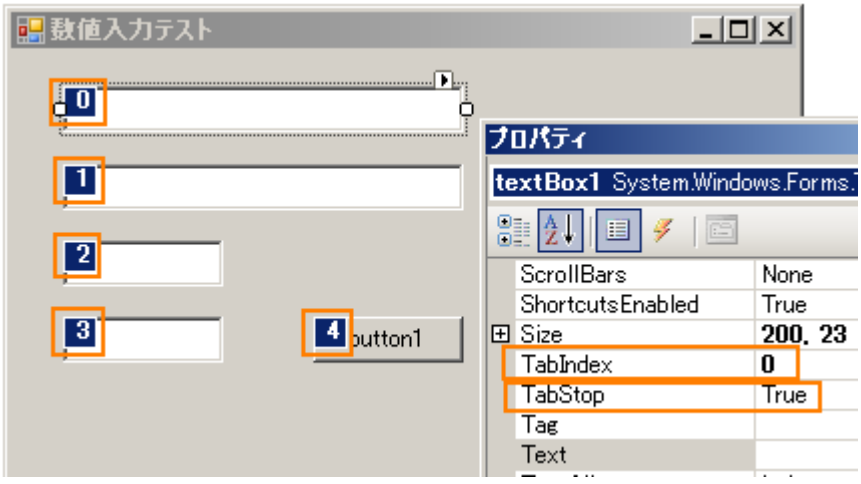


それでは、まず最初に↑↓カーソルキーとEnterキーでコントロール間を移動できるようにしてみます。バーコードリーダー等を併用している入力フィールドでCRコードを検知したりするような場面でも応用が利きます。プログラムの前に試運転を行ったMASK部分を元にもどしておきましょう。
プロパティのMASK行でマウス右クリックをして[リセット]を選択します。
MaskedTextBox2も忘れず実行しておきます。
Tabキーに依存しない入力フィールド間の動きを確認するために ダイアログにダミーで Button コントロールを1つ設置します。ツールボックスからButtonを選択し右図のように適当なところに設置します。

移動はタブ循環と同じ順序でおこなうようになります。Tabの循環が現在どのようになっているか、確認してみましょう。

メニューの[表示]-[タブオーダー]を選択してください。





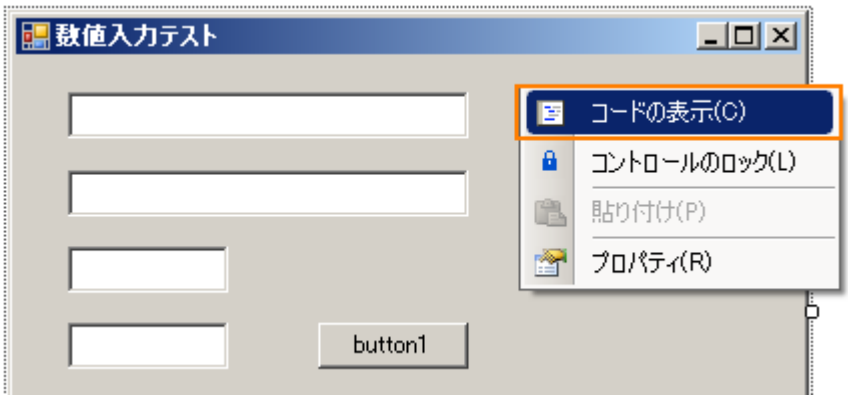
タブオーダーの表示を実行すると[0]からTab循環の順が表示されます。Tab循環はこの状態でマウスでタブ番号をクリックすることにより変更することも出来ますし、プロパティウインドウでTabIndexの値を変更することでも順番を入れ替えられます。Tab循環からコントロールをはずしたい場合は、TabStopプロパティをFalseに設定します。
タブオーダーは変更の必要が無いときは、表示しないように再度メニューのタブオーダーをクリックしておきます。今回は、右上図のようにtextBox1が[0]からの順にしておいてください。

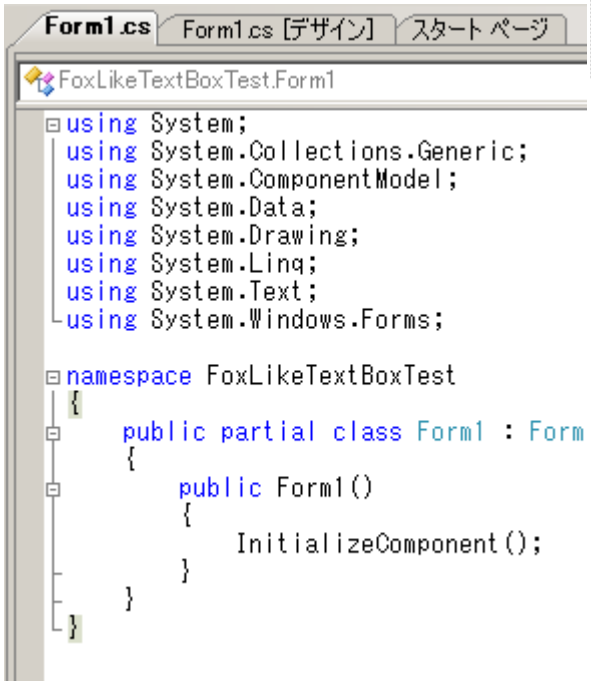
コードを書くためにエディタを開く

GUIでのコントロール配置が出来たところで、↑↓カーソルキーとEnterキーを取得するコードを実際に書いてみます。

※ コントロールのTab循環をトレースするプログラムはコントロールがPanelオブジェクトの上でグループ化されたようなGUIデザインの場合動作しません。ご注意ください。

設計中のダイアログのコントロールの無い所でマウス右クリックから[コードの表示]を選択します。





左図のようなコードを入力できるプログラムエディタが表示されます。種々のフォームを設計を行うようになると、このコードウインドウを見失ってしまうことも多々ありますが、設計フォームの上で右クリックによりこのフォームのプログラムエディタを表示することが出来ます。
それ以外に[表示]-[ソリューションエクスプローラ]から探し当てることも可能です。プログラムは public partial class Form1 : Formより外側に書かないようにします。なを、using を記述する場合はこの範囲外となります。では実際にプログラムを書いてみましょう。

記述位置は、public Form1()

```
{
    ・・・
}
```

------ この位置から書き始めます。-----

実際のプログラムは次項にて

キーを取得し次のコントロールへ移動する中核コードを書く

↑↓カーソルキーとEnterキーを取得する中核部分を最初に共通プロシージャとして作成します。

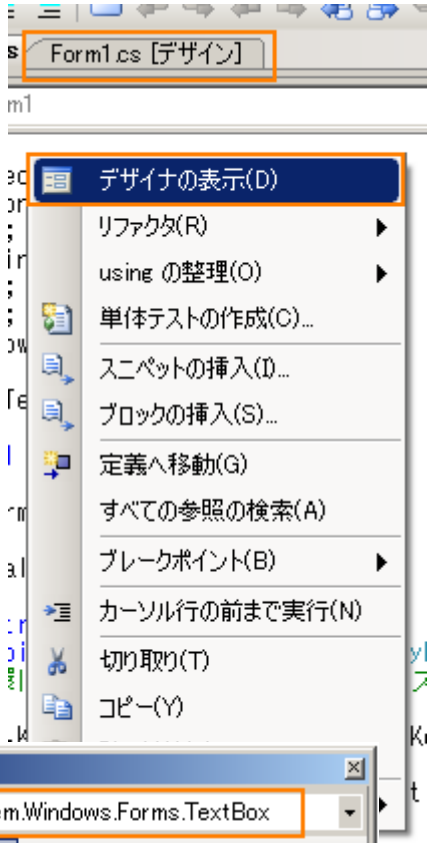
```
private string sPopup = "";
private void NextControlMoveService(KeyEventArgs e,bool isESCtrap)
// TAB循環にEnterと↑↓によるフォーカス移動を追加 KeyDown Methodから呼ばれる
{
    if (e.KeyCode == Keys.Return || e.KeyCode == Keys.Down)
    {
        this.SelectNextControl(this.ActiveControl, true, true, false, true);
    }
    else if (e.KeyCode == Keys.Up)
    {
        this.SelectNextControl(this.ActiveControl, false, true, false, true);
    }
    else if (isESCtrap && e.KeyCode == Keys.Escape)
    {
        {
            this.ActiveControl.Text = sPopup;
        }
    }

}
```

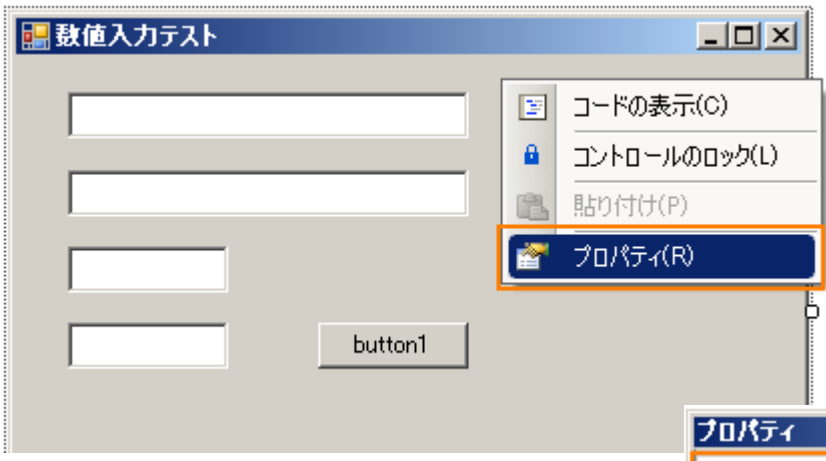
中核部分のコードは上記のようにとても簡単です。各コントロールのKeyDownメソッドからこのプロシージャを呼び出しキー値(KeyEventArgs e 経由)をチェックしEnterと↓キーであればTab循環で次の位置に設定されているコントロールへフォーカスが移動し、↑キーであれば前方のTab循環に設定されているコントロールへフォーカスが移るようになっています。string sPopupは数値入力キーの入力で[ESC]キーを押すことで入力前の値を戻すための変数です。入力値を戻すかどうかは bool isESCtrapが真(True)であれば戻し、偽(False)であれば何もしません。

```
namespace FoxLikeTextBoxTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private string sPopup = "";
        private void NextControlMoveService(KeyEventArgs e, bool isESCtrap)
        // TAB循環にRETURNと↑↓によるフォーカス移動を追加 KeyDown Methodから呼ばれる
        {
            if (e.KeyCode == Keys.Return || e.KeyCode == Keys.Down)
            {
                this.SelectNextControl(this.ActiveControl, true, true, false, true);
            }
            else if (e.KeyCode == Keys.Up)
            {
                this.SelectNextControl(this.ActiveControl, false, true, false, true);
            }
            else if (isESCtrap && e.KeyCode == Keys.Escape)
            {
                {
                    this.ActiveControl.Text = sPopup;
                }
            }
        }
    }
}
```

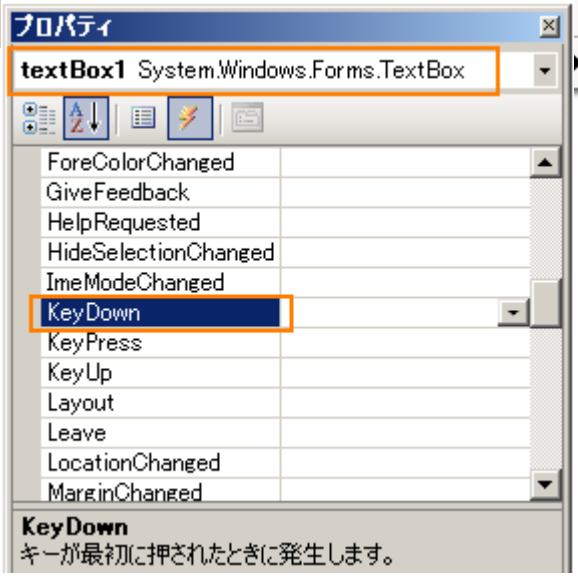
各コントロールのキーダウンイベントから呼び出すようにする

各コントロールのKeyDownメソッド(イベント)からこのプロシージャを呼び出すようにプログラムします。いったんフォームのデザイン画面にもどる必要があります。デザイン画面にもどるには、コードを書いているエディタ白紙部分でマウス右クリックで[デザイナの表示]を選択するか、Form1.cs[デザイン]タブを押します。(右図参照)
デザイン画面でプロパティウインドウを開けます。
マウス右クリックでプロパティを選択すると簡単。





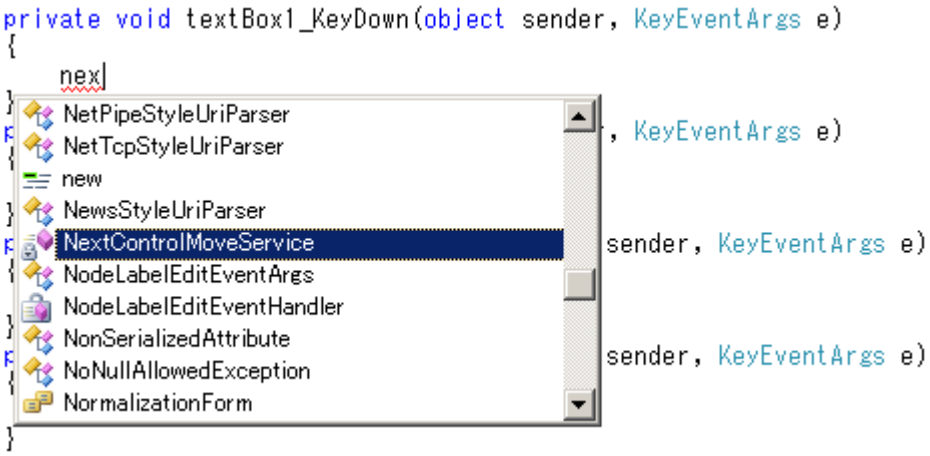
右図のようにtextBox1を選択してイベントボタンを押して各イベントを表示させます。
KeyDownメソッドを探してください。
KeyDownの文字列の上でマウスをダブルクリックします。するとコードエディタが開きtextBox1_KeyDownというイベントに対応するプロシージャが作成されています。
ここでは何も記入せず、またデザイン画面に戻りTextBox2 maskedTextBox1 maskedTextBox2のKeyDownメソッドを同様にダブルクリックしてそれぞれのプロシージャを作成します。下記のような感じに自動生成されていると思います。




```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{

}
private void textBox2_KeyDown(object sender, KeyEventArgs e)
{

}
private void maskedTextBox1_KeyDown(object sender, KeyEventArgs e)
{

}
private void maskedTextBox2_KeyDown(object sender, KeyEventArgs e)
{

}
```

各コントロールのキーダウンイベントに記述

各コントロールのKeyDownメソッド(イベント)にNextControlMoveServiceを書き込んでいきます。コードエディターに移り、textBox1_KeyDownに nexと書き始めてください、するとインテリセンスウインドウが開き候補となるものを表示してくれます。NextControlMoveServiceのところへカーソルキーで移動して 次に (をタイプするだけで　NextControlMoveService(　と自動的に書かれます。楽チンですね。



NextControlMoveService(e, false)； と書いてください。
NextControlMoveService(e, false)；一行を領域指定してCtrl+Cでコピーします。
残りのそれぞれのKeyDownメソッドに Ctrl+Vで貼り付けます。

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    NextControlMoveService( e, false);
}
private void textBox2_KeyDown(object sender, KeyEventArgs e)
{
    NextControlMoveService(e, false);
}
private void maskedTextBox1_KeyDown(object sender, KeyEventArgs e)
{
    NextControlMoveService(e, false);
}
private void maskedTextBox2_KeyDown(object sender, KeyEventArgs e)
{
    NextControlMoveService(e, false);
}
```

これで、とりあえず動くかどうか確かめてみましょう。念のためメニューの[ファイル]-[すべて保存]を実行しておきましょう。これは習慣にすると良いです。デバックで実行して↑↓カーソルキーで動作を確認してみましょう。Enterキーはボタンでトラップされます。↑↓キーはボタンをスルーすることが確認できましたでしょうか。また、試しにButtonのTabStopプロパティFalseに設定して同様の動作を確認してみてください。
次項から数値入力用のサービスを追加していきます。

maskedTextBoxに数値入力制御を付加するプログラムを記述

maskedTextBoxに数値入力を行えるようプログラムを付加します。

```
private void numericTextBox_SetEnterProp(MaskedTextBox oTextBox,string cMask)
{
    oTextBox.Text = oTextBox.Text.Replace(",", "");
    sPopup = oTextBox.Text; // ESCで戻すデータ
    oTextBox.Mask = cMask;
    oTextBox.Select();
    oTextBox.SelectionStart = 1;
    oTextBox.SelectAll();
}

private void numericTextBox_SetLeaveProp(MaskedTextBox oTextBox, string
cMask, double nMinVale, double nMaxValue)
{
    string sNum = oTextBox.Text;
    oTextBox.Mask = "";
    if (!String.IsNullOrEmpty(sNum))
    {
        Double nNum = Double.Parse(sNum);
        if ( nNum >= nMinVale && nNum <= nMaxValue )
        {
            oTextBox.Text = nNum.ToString(cMask);}
        else
        {
            MessageBox.Show("数値が範囲外です");
            oTextBox.Text = "";
            oTextBox.Focus();
        }
    }
}
```

上記のコードをNextControlMoveServiceの次あたりに追加で書き込みます。
numericTextBox_SetEnterPropはmaskedTextBoxのEnterイベントから呼ばれます。
パラメータは maskedTextBox名,”マスク設定”　です。
numericTextBox_SetLeavePropはmaskedTextBoxのLeaveイベントから呼ばれます。
パラメータは maskedTextBox名,ToStringの変換書式,入力最小値,入力最大値)
一応Doubleになっているので小数点での制御も可能となります。ここは利用環境に応じて変更してください。

記入が終わりましたら、デザイン画面へ戻ってください。maskedTextBox1とmaskedTextBox2の両コントロールについて、プロパティダイアログで Maskをリセットして PromptCharを半角スペースに置き換え、ImeModeをDisableに、TextAlignをRightに、TextMaskFormatをExcludePromptAndLiteralsにします。

KeyDownの時と同様にプロパティダイアログでイベントボタンを押し、Enterイベント、Leaveイベント、MouseClickイベントでダブルクリックします。両コントロールで同じものを作成してください。

maskedTextBoxの各イベントにプログラムを記述

maskedTextBoxのイベントは下記のように自動生成されていると思います。

```
private void maskedTextBox1_Enter(object sender, EventArgs e)
{

}
private void maskedTextBox1_Leave(object sender, EventArgs e)
{

}
private void maskedTextBox1_MouseClick(object sender, MouseEventArgs e)
{

}
private void maskedTextBox2_Enter(object sender, EventArgs e)
{

}
private void maskedTextBox2_Leave(object sender, EventArgs e)
{

}
private void maskedTextBox2_MouseClick(object sender, MouseEventArgs e)
{

}
```

それぞれに下記のように記述します。

```
private void maskedTextBox1_Enter(object sender, EventArgs e)
{
    numericTextBox_SetEnterProp(maskedTextBox1, "#999999");
}

private void maskedTextBox1_MouseClick(object sender, MouseEventArgs e)
{
    numericTextBox_SetEnterProp(maskedTextBox1, "#999999");
}

private void maskedTextBox1_Leave(object sender, EventArgs e)
{
    numericTextBox_SetLeaveProp(maskedTextBox1, "#,#", 0, 150000);
}
```

MaskedTextBox2にも同様にコピー貼り付けで記入してしまいましょう。
この場合第一パラメータが maskedTextBox2 になるよう気をつけましょう。
次に既に書いたコードに手を加えます。

```
private void maskedTextBox1_KeyDown(object sender, KeyEventArgs e)
{
    NextControlMoveService(e, false⇒true);
}
private void maskedTextBox2_KeyDown(object sender, KeyEventArgs e)
{
    NextControlMoveService(e, falsen⇒true);
}
```

上記のように NextControlMoveServiceの第二パラメータをtrueにします。これはESCキーでいったん既入力データを戻すようにする設定です。これでデバックして動作を確認してみましょう。　　以上で説明を終わります。