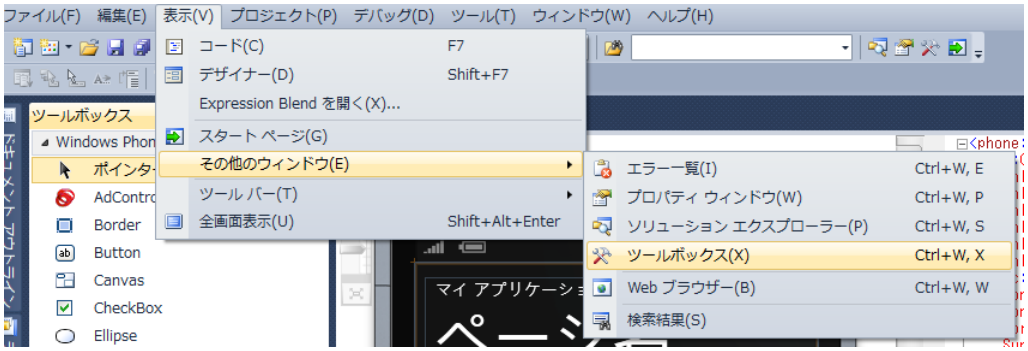
■お絵かきソフトを作ろう！

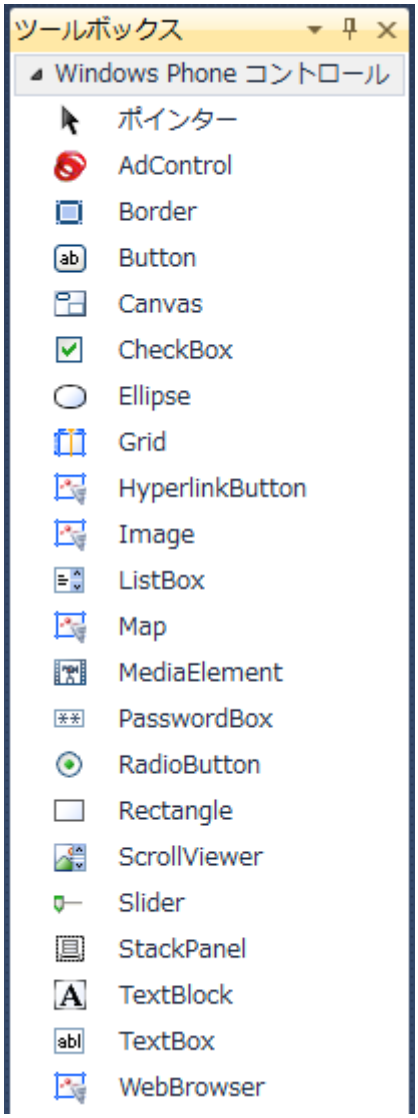
Windows Phone は、パソコンとは異なり、キーボードを持っていません。その代り、マウスよりも直感的に操作できるタッチパネルを持っています。Windows Phone アプリケーションの仕組みや開発環境はパソコンの Windows 環境とほぼ同じですが、アプリケーションを開発する際には、ユーザーインターフェースの違いを意識して作る必要があるでしょう。
さて今回は、Windows Phone の特徴的なタッチパネルのインターフェースを活かした「お絵かきアプリ」を作ってみたいと思います。Windows Phone でのお絵かきアプリは、まるで手帳にペンで絵を描くのと同じような感覚で利用することができます。Windows Phone の画面上に自由に描画ができるようになれば、お絵かきアプリに限らず、手書きのメモ帳など様々なアプリに応用することができるでしょう。

■プロジェクトを作ってみよう！
まずは、Windows Phone のプロジェクトを作ってみましょう。
プロジェクトの作り方は、[Hello World] のコンテンツを参照してください。
今回は、アプリケーション名を［Oekaki］にしましょう。

◆ツールボックスを出そう！
まずは、開発をするのに便利なウィンドウを出しましょう。
[表示]メニューから[その他のウィンドウ][ツールボックス]を選択します。



ツールボックス ウィンドウが開き、Windows Phone コントロールが表示されたと思います。



このコントロールを Windows Phone のデザイン画面に貼り付けることで、簡単に XAML を作成することができます。

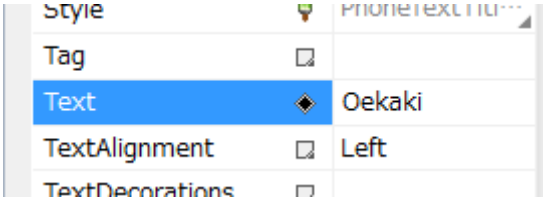
◆タイトルを変更しよう！

現在は、デザイン上で[マイ アプリケーション][ページ名] と書かれています。これを変更してみましょう。

まず、プロパティウィンドウを出します。[表示]メニューから[その他のウィンドウ][プロパティウィンドウ] を選択します。

次に、[ページ名] と書かれたところをクリックします。（または、ドキュメントアウトラインで[TextBlock(PageTitle)] をクリックします。）すると、プロパティウィンドウで、テキストブロックのプロパティが変更できるようになります。

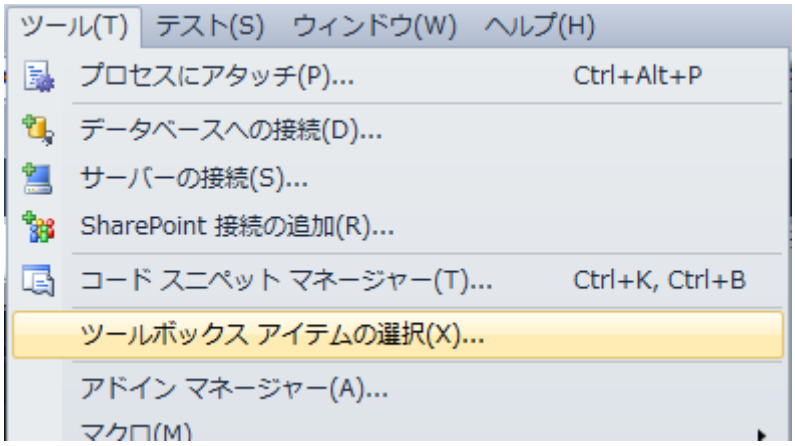
ここで、[テキスト]というプロパティがありますので、これを[Oekaki] に変更します。



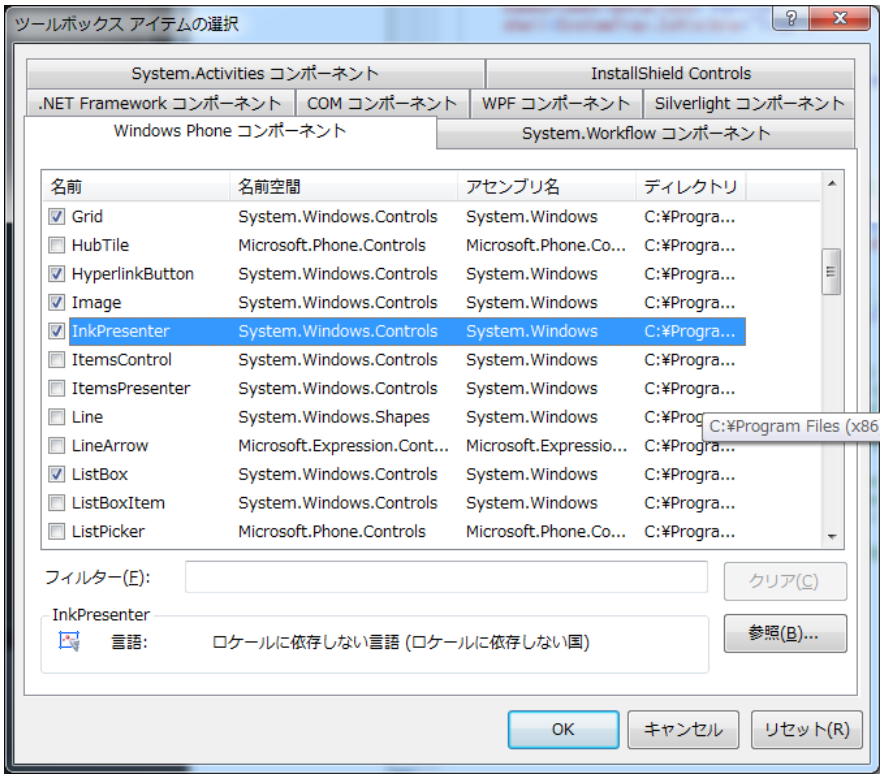
同様に[マイアプリケーション]と書かれた部分も[Windows Phone Sample] に変更します。
これでタイトルの変更は完了です。実行すると以下のような画面が表示されます。

■インクプレゼンター（InkPresenter）コントロールを表示させよう！
お絵かきアプリを作るためには、インクプレゼンター（InkPresenter） というコントロールを使うと便利です。このコントロールは、ペンで描画した軌跡を一筆単位で保存していくことができます。ただし、Visual Studio では、最初はこのコントロールが表示されていないので、まずはツールボックスにインクプレゼンター（InkPresenter）コントロールを登録しましょう。

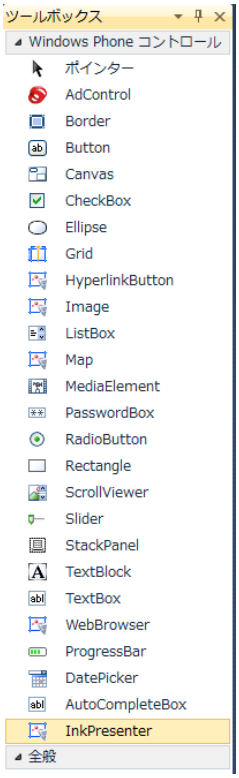
[ツール]メニューから[ツールボックス アイテムの選択]を選択します。



[ツールボックス アイテムの選択]ウィンドウが表示されたら、InkPresenter にチェックします。

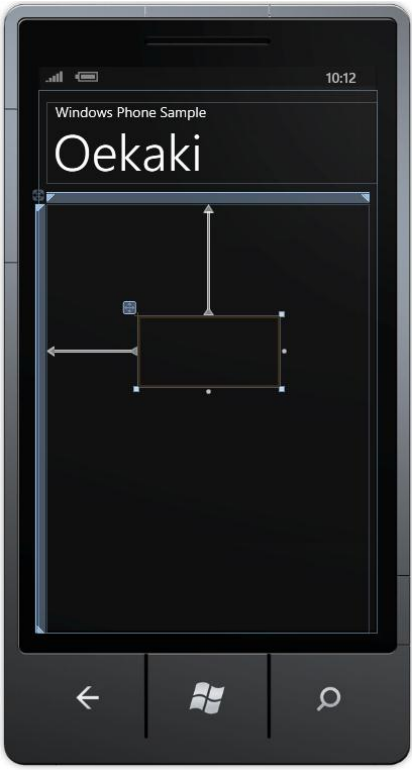


OK ボタンをクリックするとツールボックスにインクプレゼンター（InkPresenter)コントロールが追加されます。



■インクプレゼンターコントロールを貼り付けよう！

では、インクプレゼンター（InkPresenter）コントロールをドラッグ＆ドロップしてデザイン画面に貼り付けます。以下のような画面になります。



ここで、インクプレゼンター（InkPresenter）コントロールを画面一杯に広げます。
マウスを使って大きく引き延ばしてもいいのですが、こういう場合は、XAML のコードを直接書き換えたほうが早いです。
現在、インクプレゼンター（InkPresenter）コントロール部分のコードは以下のようになっています。

```
<InkPresenter Height="100" HorizontalAlignment="Left"
Margin="129,159,0,0" Name="inkPresenter1" VerticalAlignment="Top"
Width="200" />
```

ここで、Nameはそのまま残しておきましょう。他は削除して構わないのですが、追加でBackgroundプロパティを記述します。このプロパティは背景色を指定することができるのですが、ここでは、Transparentを設定します。インクプレゼンター（InkPreasenter）コントロールでは、タッチのイベントを取得するためにはBackgroundプロパティを追加しないといけないのです。Transparentにすると背景色を指定しない設定になりますので、初期の黒のままになります。ここで、Background="DarkGreen" のように深緑を設定すると、黒板のように見えますね。ここでは、Transparentを使っていますが、色を設定しても構いません。
修正した行は以下のようになります。

```
<InkPresenter Name="inkPresenter1" Background="Transparent" />
```

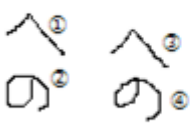
ここでデザイン画面を見るとインクプレゼンター（InkPresenter)コントロールが画面一杯に広がっているのが分ります。

■ベクトルグラフィックスについて

インクプレゼンター（InkPresenter)コントロールでは、ベクトルグラフィックスを扱うことができます。ベクトルグラフィックスとは、画像を点や線の集合として表現したグラフィックのことです。一筆ごとに線の座標を記録していくため、描画の手順を再現することもできます。

ここで、一筆の単位を Stroke と呼びます。インクプレゼンター（InkPresenter）コントロールでは、Stroke を束ねた Strokes というデータを保持します。

例えば、「へのへの」の画像があった場合、インクプレゼンターの内部では、以下のように４つの Stroke データが格納されます。

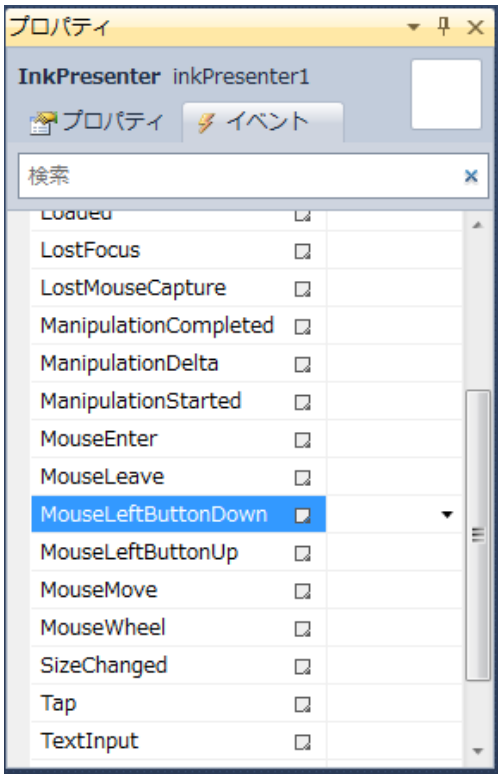
コントロール	Strokes	Stroke
InkPresenter		

■お絵かきができるようにしよう！

前述のとおり、インクプレゼンターコントロールは、一筆ごとに Stroke を記録していきます。よって、以下のような流れでプログラムを記述するとよいでしょう。

1. もしタッチパネルがペンや指で押下されたら、新規に Stroke を作成し InkPresenter に登録する。（以後、再度押下されるまでのタッチについては、この Stroke に記録していきます。）
2.もし、タッチパネル上でペンや指が動いたら（つまり、押した状態で移動したら）現在の Stroke にその移動座標などを記録する

ここで、１の動作は、MouseLeftButtonDown、２の動作は、MouseMove というイベントで記述します。
では、やってみましょう。
先ほど貼り付けたインクプレゼンター（InkPresenter)コントロールをクリックして、プロパティウィンドウを出します。上部にあるタブを[イベント]に切り替え、[MouseLeftButtonDown]をダブルクリックします。



ここで、初めて C#の画面が開きました。最初にインクプレゼンター（InkPresenter）の機能を利用するためプログラム冒頭の using 文に以下の網掛けの行を追加します。

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using System.Windows.Ink;
```

次に、一筆の記録をするための Stroke 変数と、MouseLeftButtonDown の処理を追加します。
以下のように記述します。

```
    private Stroke currentStroke;

    private void inkPresenter1_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
    {
        currentStroke = new Stroke(e.StylusDevice.GetStylusPoints(inkPresenter1));
        currentStroke.DrawingAttributes.Color = Colors.White;
        currentStroke.DrawingAttributes.Width = 5;
        currentStroke.DrawingAttributes.Height = 5;
        inkPresenter1.Strokes.Add(currentStroke);
    }
```

このイベントでは、タッチパネルがタッチされた際の座標情報が MouseButtonEventArgs e の部分で渡されます。

以下の行では、タッチされた座標を使って、新規にStrokeを生成しています。

```
        currentStroke = new Stroke(e.StylusDevice.GetStylusPoints(inkPresenter1));
```

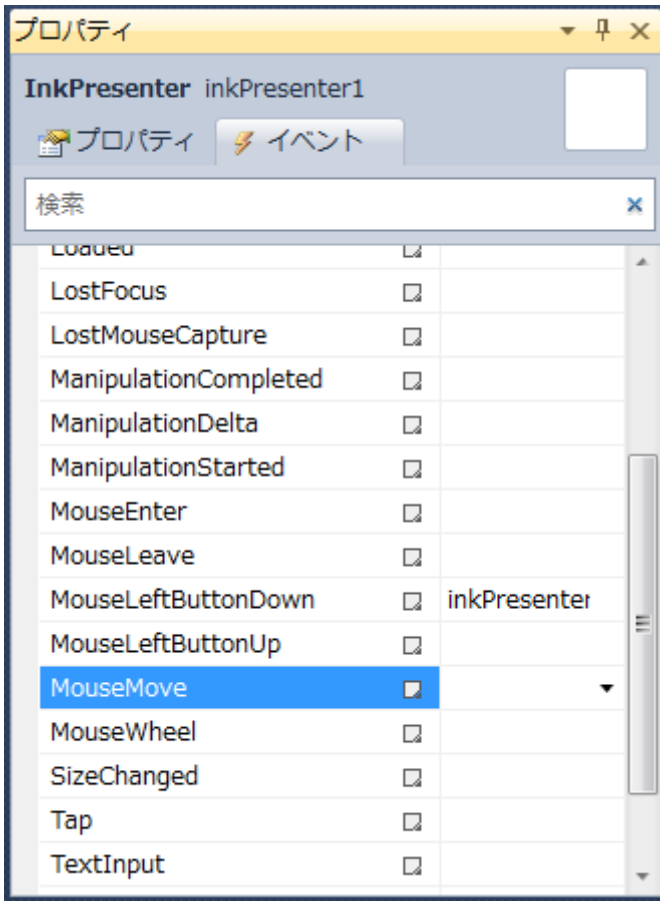
以下はペンの色や太さを設定しています。ここでは、色を白色、太さを５としています。

```
        currentStroke.DrawingAttributes.Color = Colors.White;
        currentStroke.DrawingAttributes.Width = 5;
        currentStroke.DrawingAttributes.Height = 5;
```

最後にインクプレゼンター（InkPresenter）の Strokes プロパティに今作成した Stroke を追加します。

```
        inkPresenter1.Strokes.Add(currentStroke);
```

続いて MouseMove イベントを追加しましょう。上部のタブで[MainPage.xaml]をクリックし、デザイン画面に戻します。
先ほどと同様に、プロパティウィンドウで、今度は、[MouseMove]をダブルクリックします。



また C#の画面に切り替わります。
ここでは、タッチが移動したときに MouseLeftButtonDown のイベントで作成した Stroke にペンの軌跡を追加しましょう。
プログラムは、以下のように記述します。

```
private void inkPresenter1_MouseMove(object sender, MouseEventArgs e)
{
    currentStroke.StylusPoints.Add(e.StylusDevice.GetStylusPoints(inkPresenter1));
}
```

これでお絵かきをするプログラムは完成です。プログラムを実行してみましょう。

画面をタッチすると絵が描けることが確認できましたか？

■アプリケーションバー（ApplicationBar）を追加しよう！

ここからは、いくつかの機能を追加していきたいと思います。

今回追加する機能は、以下の４つです。

１．ペンと消しゴムを切り替える機能

２．画像を消去する機能
３．画像を保存する機能
４．背景画像を読み込む機能

これらの機能の呼び出しには、ボタンを配置してもよいのですが、ここでは、アプリケーションバー(ApplicationBar)を使ってみます。
アプリケーションバー(ApplicationBar) は、Windows Phone 固有の機能で、画面下部に最大４つのボタンとスライドしてメニューを表示する機能があります。
実は、アプリケーションバー(ApplicationBar) は、新規にプロジェクトを作った際に簡単なサンプルが付いています。
上部にある MainPage.xaml タブ をクリックして XAML のコードを表示してみましょう。
XAML のコードの最後の方に以下のようなプログラムが書かれていると思います。

```
    <!--ApplicationBar の使用法を示すサンプル コード-->
    <!--<phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
            <shell:ApplicationBarIconButton IconUri="/Images/appbar_button1.png"
Text="Button 1"/>
            <shell:ApplicationBarIconButton IconUri="/Images/appbar_button2.png"
Text="Button 2"/>
            <shell:ApplicationBar.MenuItems>
                <shell:ApplicationBarMenuItem Text="MenuItem 1"/>
                <shell:ApplicationBarMenuItem Text="MenuItem 2"/>
            </shell:ApplicationBar.MenuItems>
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>-->
```

では、このサンプルプログラムの動きを見てみましょう。上記コードで網掛けをした部分がコメントですので、これを削除します。
できましたら、プログラムを実行してみましょう。
以下のように画面下部のボタンが表示されるはずです。

ここで、右下の[・・・]をクリックしてみましょう。

今度はメニューが表示されました。専用に作られている画面ですので、見た目もすっきりしていて分りやすいです。また他のアプリケーションでも共通で使われるので、直感的に利用できます。

ここでボタンを見ると×になっています。これは、xaml のコードの以下の部分でアイコン画像が設定されているのですが、指定したフォルダにアイコンファイルが存在していない

からです。

```
<shell:ApplicationBarIconButton IconUri="/Images/appbar_button1.png" Text="Button 1"/>
<shell:ApplicationBarIconButton IconUri="/Images/appbar_button2.png" Text="Button 2"/>
```

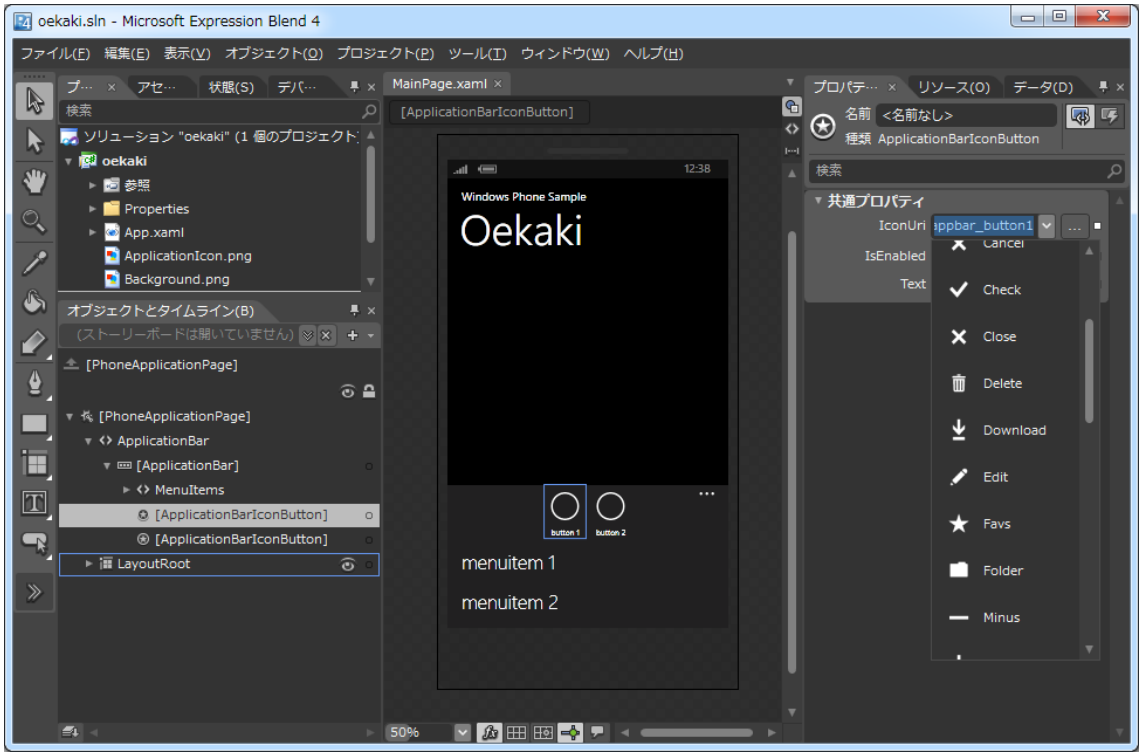
今回のサンプルプログラムでは、開発ツールに付属したアイコンファイルと、こちらで用意したアイコンファイルをコピーして実装しますが、アプリケーションバー(ApplicationBar) を使ったアプリケーションを作成する際には、Expression Blend を使うのも便利です。

Expession Blend ではコレクションエディターで用意されているアイコンをそのまま利用することができます。以下は参考までに、Expression Blend での設定画面になります。

Expression Blend を起動するには、[表示]メニューから[Expression Blend を開く]を選択します。

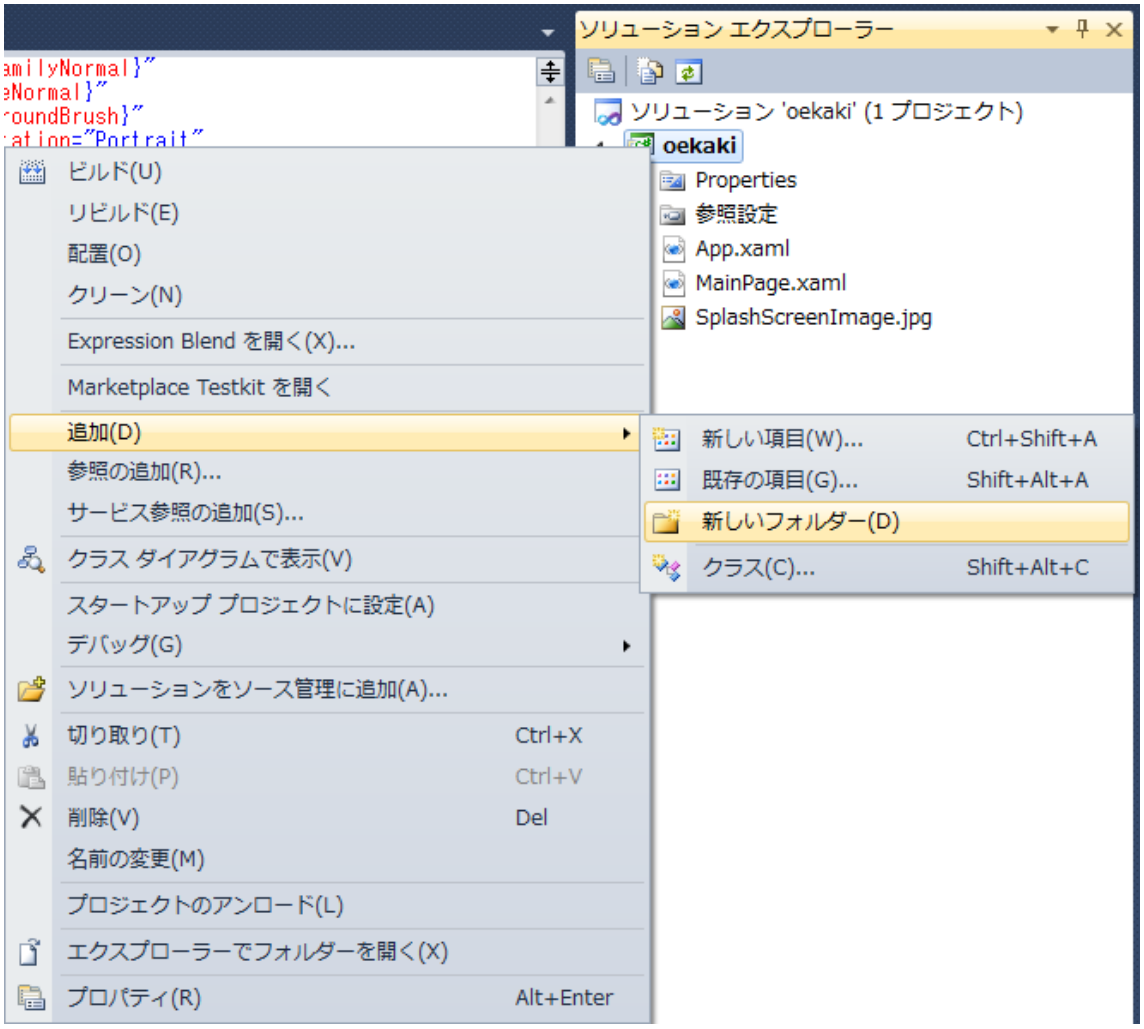
左側のオブジェクトとタイムラインのところで、アプリケーションバー(ApplicationBar)の中の ApplicationBarIconButton を選択して、右側の共通プロパティの IconUri でアイコンを選択することができます。

今回は、Expression Blend は使いません。



では、今回のプログラムに合わせてアプリケーションバー(ApplicationBar)を修正してみましょう。

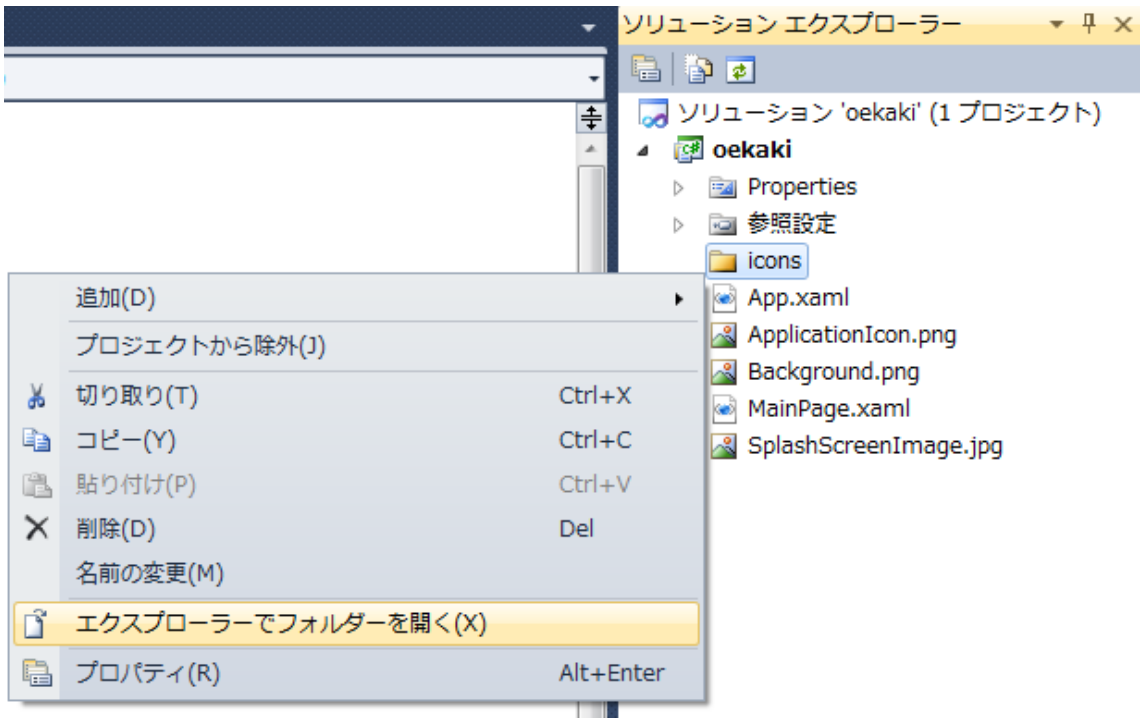
まずは、ボタンアイコンを保存するフォルダを作りましょう。
ソリューションエクスプローラーで、プロジェクト名[oekaki]のところを右クリックして、[追加][新しいフォルダー]を選択します。



ここでフォルダ名は、icons とします。

次に、ボタンに必要なアイコン画像をこのフォルダにコピーしましょう。
icons のフォルダは、標準の場合以下になります。
C:¥Users¥ユーザー名¥Documents¥visual studio 2010¥Projects¥oekaki¥oekaki¥icons

icons フォルダを右クリックして、[エクスプローラで開く]を選択すると簡単です。



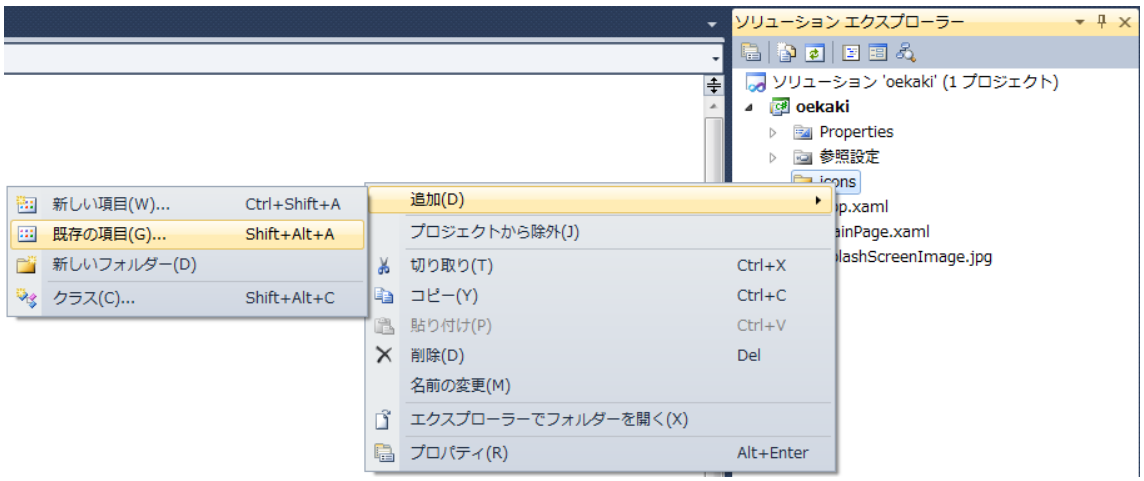
ここにコピーするアイコンファイルは、5 本あります。まず、appbar.erase.rest.png だけ、ここからダウンロードしてから icons フォルダにコピーしてください。

残りの 4 本は、以下です。

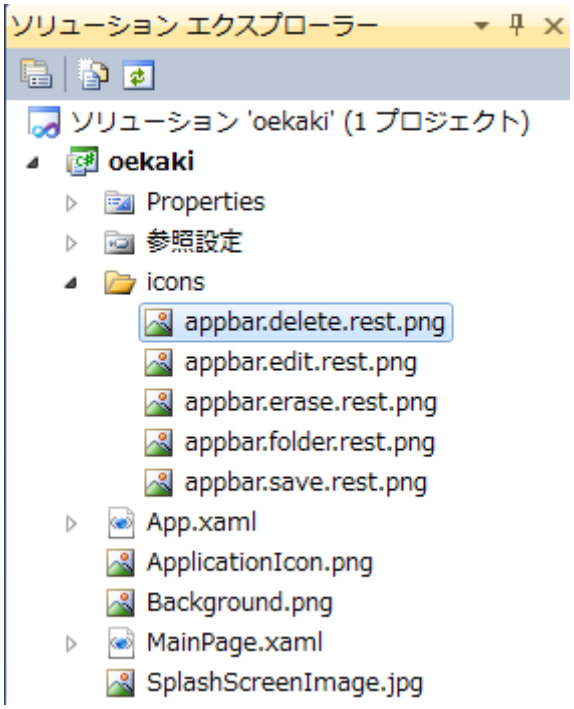
appbar.edit.rest.png
appbar.folder.rest.png
appbar.save.rest.png
appbar.delete.rest.png

これらのファイルは、Windows のバージョンによって格納されている場所が異なります。以下のいずれかに入っていますので、ご利用の Windows バージョンを確認の上、4 つのアイコンファイルを icons フォルダにコピーしてください。
64bit：C:¥Program Files (x86)¥Microsoft SDKs¥Windows Phone¥v7.1¥Icons¥dark
32bit：C:¥Program Files¥Microsoft SDKs¥Windows Phone¥v7.1¥Icons¥dark

続いて、icons に先ほどコピーしたファイルをプロジェクトに追加していきます。
Icons フォルダで右クリックして、[追加][既存の項目]を選択します。

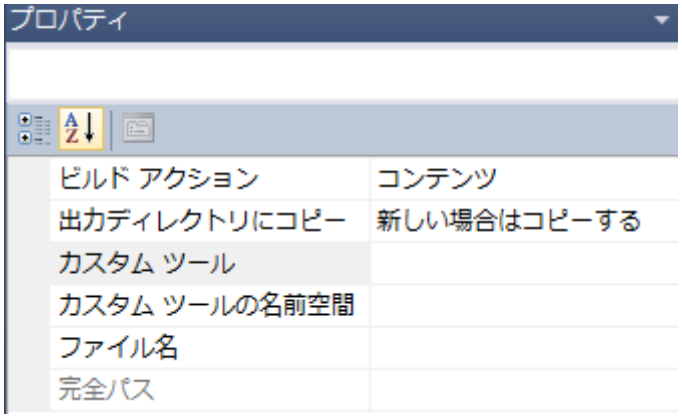


ここで、先ほどコピーしたアイコンファイルを５つとも追加してください。追加が終わるとソリューションエクスプローラーは以下のようになります。



次に追加したアイコンファイルを全て選択し、プロパティを以下のように変更します。
全て指定するには、先頭（appbar.edit.rest.png）をクリックし、SHIFT キーを押しながら最後（appbar.save.rest.png）をクリックします。
もしプロパティウィンドウが表示されていない場合は、[表示]メニューの[プロパティ ウィンドウ]を選択して表示してください。

ビルドアクション：コンテンツ
出力ディレクトリにコピー：新しい場合は、コピーする



これで、アイコンファイルの設定は完了です。

今度は xaml コードを修正しましょう。プログラムの実行を停止して、xaml コードを以下のように修正します。

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True" x:Name="AppBar" >
        <shell:ApplicationBarIconButton IconUri="/icons/appbar.erase.rest.png" Text="Pen"
x:Name="PenButtton" />
        <shell:ApplicationBarIconButton IconUri="/icons/appbar.delete.rest.png"
Text="Clear" x:Name="ClearButton" />
        <shell:ApplicationBarIconButton IconUri="/icons/appbar.save.rest.png" Text="Save"
x:Name="SaveButton" />
        <shell:ApplicationBarIconButton IconUri="/icons/appbar.folder.rest.png"
Text="Load" x:Name="LoadButton" />
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

ここまでできたら一度プログラムを実行しましょう。

以下のようにアイコンが書き換わりましたか？

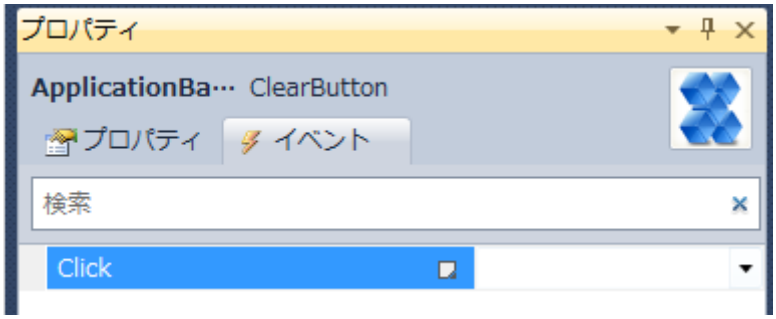
■画面消去機能を追加しよう！

ApplicationBar の二番目のボタンが画面消去です。このボタンが押されたときに画面を消去するプログラムを記述します。

MainPage.xaml を表示した状態で、以下の網掛けの行をクリックします。

```
    <phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True" x:Name="AppBar" >
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.erase.rest.png"
Text="Pen" x:Name="PenButtton"  />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.delete.rest.png"
Text="Clear" x:Name="ClearButton" />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.save.rest.png"
Text="Save" x:Name="SaveButton"  />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.folder.rest.png"
Text="Load" x:Name="LoadButton" />
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>
```

ここでプロパティウィンドウを見ると、ClearButton のプロパティが表示されているはずです。イベントタブを選択してください。



[Click]をダブルクリックして、画面消去ボタンの Click イベントを追加します。
C#の画面に切り替わったら、以下の網掛けの行をプログラムに追加しましょう。

```
    private void ClearButton_Click(object sender, EventArgs e)
    {
        inkPresenter1.Strokes.Clear();
    }
```

ここでは、インクプレゼンター（InkPresenter）の Strokes クラスで Clear()メソッドを呼出しています。Clear メソッドは、Strokes の保持する全ての Stroke を消去します。全ての筆跡を削除したので、画面も全て消えることになります。

プログラムを実行して実際に描画した線画が消去できることを確認してみてください。

■ペンと消しゴムの切り替え処理を追加しよう！

今度は、消しゴム処理を追加してみます。まずは、アプリケーションバー(ApplicationBar)の消しゴムボタンをクリックしたら、ペンと消しゴムが切り替わるようにしてみます。

先ほど消去ボタンのイベントを追加したのと同様に、今度は消しゴムボタンをクリックしたときのイベントを追加します。

MainPage.xaml の画面に切り替え、網掛けの行をクリックしたあとプロパティウィンドウを確認します。

```
    <phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True" x:Name="AppBar" >
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.erase.rest.png"
Text="Pen" x:Name="PenButttton"  />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.delete.rest.png"
Text="Clear" x:Name="ClearButton" Click="ClearButton_Click" />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.save.rest.png"
Text="Save" x:Name="SaveButton"  />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.folder.rest.png"
Text="Load" x:Name="LoadButton" />
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>
```

消去ボタンと同様に Click イベントを追加します。

C#の画面が開いたら以下のようにプログラムを追加します。

```
    public bool penMode = true;
    private void PenButttton_Click(object sender, EventArgs e)
    {
        if (penMode == true)
        {
            penMode = false;
            ApplicationBarIconButton penButton =
(ApplicationBarIconButton)ApplicationBar.Buttons[0];
            penButton.IconUri = new Uri("/icons/appbar.edit.rest.png", UriKind.Relative);
```

```
        }
        else
        {
            penMode = true;
            ApplicationBarIconButton penButton =
(ApplicationBarIconButton)ApplicationBar.Buttons[0];
            penButton.IconUri = new Uri("/icons/appbar.erase.rest.png", UriKind.Relative);

        }
    }
```

一行目で、penMode という変数を定義しています。これは、True の場合は、現在はペンモード、False の場合は、消しゴムモードを意味します。
if 文では、ボタンが押されたとき、もしペンモードの場合は消しゴムモードに切り替え、消しゴムモードの場合は、ペンモードに切り替える処理となっています。ボタンアイコンの画像もここで差し替えています。ボタンのアイコン画像を変更するには、ボタンの IconUri プロパティを変更します。ここで、以下のような記述があります。

```
ApplicationBar.Buttons[0]
```

これは、アプリケーションバー（ApplicationBar）の最初のボタンを表しています。つまり今回の場合は、消しゴム（ペン）ボタンになります。[0] は一番目のボタンということです。もし、画面消去ボタンのアイコンを変更したい場合、2 番目のボタンなので、ApplicationBar.Buttons[1] となります。

■消しゴムモードの処理を追加しよう！
消しゴム処理もペンの処理と同様に、タッチパネルがタッチされたときと、タッチしたまま動かしたときの処理で行います。実際に線を削除する処理は、ここでは、MouseMove のイベントが発生したときだけにします。つまり、タッチして線を擦ると消えるということです。
プログラムを以下のように網掛け部分を変更します。

```
    private void inkPresenter1_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
    {
        if (penMode == true)
        {
            currentStroke = new Stroke(e.StylusDevice.GetStylusPoints(inkPresenter1));
```

```
                currentStroke.DrawingAttributes.Color = Colors.White;
                currentStroke.DrawingAttributes.Width = 5;
                currentStroke.DrawingAttributes.Height = 5;
                inkPresenter1.Strokes.Add(currentStroke);
            }
        }

        private void inkPresenter1_MouseMove(object sender, MouseEventArgs e)
        {
            if (penMode == true)
            {

currentStroke.StylusPoints.Add(e.StylusDevice.GetStylusPoints(inkPresenter1));
            }
            else
            {
                StylusPointCollection erasePoints =
e.StylusDevice.GetStylusPoints(inkPresenter1);
                StrokeCollection hitStrokes = inkPresenter1.Strokes.HitTest(erasePoints);
                foreach (Stroke hitStroke in hitStrokes)
                {
                    inkPresenter1.Strokes.Remove(hitStroke);
                }
            }
        }
```

さて、プログラムを見てみましょう。

MouseLeftButtonDown の処理は簡単です。もしペンモードの場合だけ今まで通りの処理をし、消しゴムモードでは何もしないだけです。

MouseMove では、消しゴムモードのときに処理を追加しています。

一行目には、以下があります。

```
StylusPointCollection erasePoints = e.StylusDevice.GetStylusPoints(inkPresenter1);
```

ここでは、タッチパネルがタッチされた点の集合を erasePoints に格納しています。

MouseMove の処理の中で一番重要なのは、以下の行です。

```
StrokeCollection hitStrokes = inkPresenter1.Strokes.HitTest(erasePoints);
```

ここでは、インクプレゼンター(InkPresenter)に登録されている全ての Stroke について、

erasePoint というタッチされた座標の集合と交わる部分がある Stroke を HitTest というメソッドを使って抽出しています。
つまり、ここで、hitStrokes は、消しゴムで接触した Stroke(線)の集合ということです。
あとは、foreach で、hitStrokes に抽出された Stroke を消していくだけです。
以下の行では､インクプレゼンターの Strokes の中から交わりのあった線を消しています。

```
inkPresenter1.Strokes.Remove(hitStroke);
```

これで消しゴムモードは完成です。プログラムを起動して実際に線が消せることを確認してみましょう。

左眉と鼻の一部を消してみました。

■消しゴム処理を拡張しよう！
先ほど作成した消しゴム処理では、Stroke 単位での線の消去を行いました。
でも、お絵かきアプリとしては、できれば、線を丸ごと消すのではなく、消しゴムが触れ

た部分だけを消去したいものです。こういったことは実現できるのでしょうか？
最初に説明しましたが、インクプレゼンター（InkPresenter）では、ベクトルグラフィックを扱っているため、ピクセル単位で画像を書き換えるものには向いていません。
しかし、ベクトルグラフィックで、部分的な消去を行うのは、少々面倒ですが不可能ではありません。例えば、左右に伸びる一本の線（Stroke）があり、その中心部分を消去したい場合は、その中心部分を除いた左側と右側の線を 2 本の Stroke に分割して追加登録し、元の線を削除すれば結果として中心部分が消えたことになります。
ただし、これでもピクセル単位で消すことができるわけではありません。
Stroke は、内部にペンの軌跡を座標データで保持しているのですが、例えば素早く描画した場合などは、まばらにデータが保持されるのです。
つまり、この方法で消すことができるのは、あくまで Stroke 内部に持つ座標の単位までということになります。

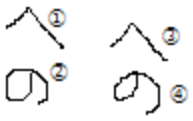
では、実際にプログラムを修正してみましょう。
網掛け部分をプログラムに追加します。

```
    private void inkPresenter1_MouseMove(object sender, MouseEventArgs e)
    {
        if (penMode == true)
        {

currentStroke.StylusPoints.Add(e.StylusDevice.GetStylusPoints(inkPresenter1));
        }
        else
        {
            StylusPointCollection erasePoints =
e.StylusDevice.GetStylusPoints(inkPresenter1);
            StrokeCollection hitStrokes = inkPresenter1.Strokes.HitTest(erasePoints);
            foreach (Stroke hitStroke in hitStrokes)
            {
                Stroke sliceStroke = new Stroke();
                sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
                foreach (StylusPoint sp in hitStroke.StylusPoints)
                {
                    sliceStroke.StylusPoints.Add(sp);
                    if (sliceStroke.HitTest(erasePoints))
```

```
                    {
                        sliceStroke.StylusPoints.RemoveAt(sliceStroke.StylusPoints.Count -
1);
                        sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
                        inkPresenter1.Strokes.Add(sliceStroke);
                        sliceStroke = new Stroke();
                        sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
                    }
                }
                inkPresenter1.Strokes.Add(sliceStroke);
                inkPresenter1.Strokes.Remove(hitStroke);
            }
        }
    }
```

foreach で hitStroke を消去する部分までは同じです。ここでは、単純に hitStroke を消去するのではなく、消しゴムとの接触点で 2 本の Stroke に分割する処理を追加します。
まずは、Stroke と、StyusPoint の関係を見てみましょう。
先ほどの図に StylusPoint を追加しました。

コントロール	Strokes	Stroke	StylusPoints
InkPresenter			

この図で分るように、Stroke は、StylusPoint の集合である StylusPoint を所持しているということになります。
さて、プログラムの流れとしては、hitStroke が格納している StylusPoint を一つずつ取り出し、消しゴムの軌跡の erasePoints と接触しているかを調べることになります。
接触を調べるには、先ほどの HitTest を使うのですが、ここで問題となるのは、HitTest は、Stroke と Stroke の接触を調べることができるのですが、Stroke と StylusPoint の接触は調べられないのです。そこで、StylusPoint を一つずつ取り出して、新規に Stroke を作り、

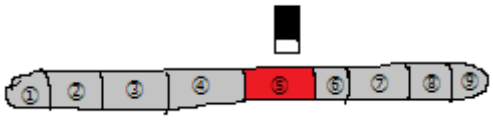
少しずつ Stroke を伸ばしながら接触を調べることにします。

```
Stroke sliceStroke = new Stroke();
sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
```

ここで作っている SliceStroke が StylusPoint を登録するための Stroke です。

```
foreach (StylusPoint sp in hitStroke.StylusPoints)
{
    sliceStroke.StylusPoints.Add(sp);
```

このループで、SliceStroke に StylusPoint を追加しています。
下の図を見てください。



この線は、hitStroke です。hitStroke は①～⑨までの StylusPoint を保持しています。
赤い部分は、erasePoint との交点です。
sliceStroke は、①から順番に StylusPoint を追加して線を伸ばしていきます。

続いて以下のプログラムがあります。

```
if (sliceStroke.HitTest(erasePoints))
{
        sliceStroke.StylusPoints.RemoveAt(sliceStroke.StylusPoints.Count - 1);
        sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
        inkPresenter1.Strokes.Add(sliceStroke);

        sliceStroke = new Stroke();
        sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
}
```

ここでは、SliseStroke と消しゴムの当たった erasePoint との接触を調べています。
接触が無ければ条件を満たさないので素通りします。図の場合⑤の StylusPoint を SliseStroke に追加すると接触することになります。
接触した場合は、接触した StylusPoint（図の⑤の部分）だけを削除し、そこまでの Stroke（図の①～④）をインクプレゼンター（InkPresenter）に追加しています。
更に、以下の行を実行することで次の Stroke の為に sliceStroke を初期化しています。

```
sliceStroke = new Stroke();
```

```
sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
```

これにより、新しい Stroke に残りの StylusPoint（図の⑥～⑨）が追加されます。
ループを抜けると以下の行があります。

```
inkPresenter1.Strokes.Add(sliceStroke);
inkPresenter1.Strokes.Remove(hitStroke);
```

ここで最後の Stroke（図の⑥～⑨）をインクプレゼンター（InkPresenter）に追加し、不要になった hitStroke を削除しています。
これで、Stroke 単位ではなく、StylusPoint 単位で削除することができるようになりました。

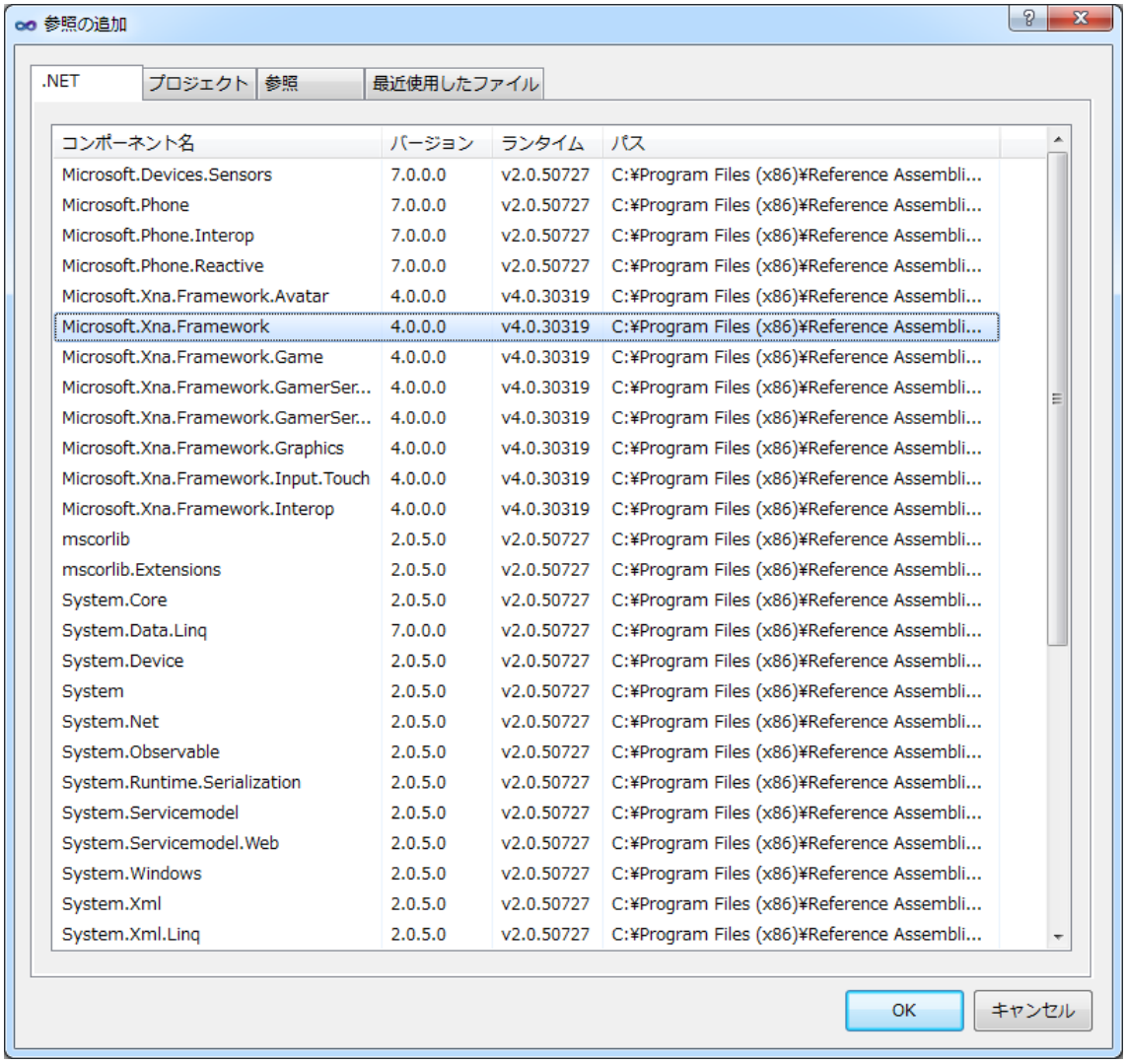
実際にプログラムを実行してみましょう。

ロと眉毛に切れ目を入れてみました。

■ラスタグラフィックスについて
ここまでの画像の描画には、ベクトルグラフィックを使ってきました。このあと、画像の保存を行うプログラムを書くのですが、そこでは、ラスタグラフィックスを利用します。

ラスタグラフィックは JPEG や PNG 形式の画像データのように、線ではなく面（ドットの集合）で扱う形式です。通常ベクトルグラフィックよりも大きなデータサイズになりますが、写真などの画像を表現することができます。Windows Phone ではラスタグラフィックを使う場合でも便利な機能があります。

■画面を保存しよう！
ここでは、描いた画像を保存するプログラムを追加しましょう。
Windows Phone で画像を保存するには、メディアライブラリ（MediaLibrary）を利用します。メディアライブラリを利用するためには、現在のプロジェクトに専用のライブラリを追加しなければいけません。
[プロジェクト]メニューで[参照の追加]を選択します。
[参照の追加]ウィンドウが開きましたら、
Microsoft.XNA.Framework
を選択して OK ボタンを押します。



参照の追加が済みましたら、プログラムの先頭に以下のように網掛け部分を追加します。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
```

```
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using System.Windows.Ink;
using System.Windows.Media.Imaging;
using System.IO;
using Microsoft.Xna.Framework.Media;
using Microsoft.Phone.Tasks;
```

次に、これまでと同様に ApplicationBar の Save ボタンのイベントを追加します。
MainPage.xaml タブをクリックして、デザインページを表示し、以下の網掛けの行をクリックした状態でプロパティウィンドウを確認します。

```
    <phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True" x:Name="AppBar" >
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.erase.rest.png"
Text="Pen" x:Name="PenButtton" Click="PenButtton_Click" />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.delete.rest.png"
Text="Clear" x:Name="ClearButton" Click="ClearButton_Click" />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.save.rest.png"
Text="Save" x:Name="SaveButton"  />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.folder.rest.png"
Text="Load" x:Name="LoadButton" />
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>
```

プロパティウィンドウでイベントタブをクリックし、[Click]をダブルクリックします。
C#の画面になりましたら、以下のようにプログラムを修正しましょう。

```
    private void SaveButton_Click(object sender, EventArgs e)
    {
        WriteableBitmap bmp = new WriteableBitmap(inkPresenter1, null);
        MemoryStream stream = new MemoryStream();
        bmp.SaveJpeg(stream, bmp.PixelWidth, bmp.PixelHeight, 0, 80);
        using (MediaLibrary mediaLib = new MediaLibrary())
        {
            mediaLib.SavePicture("PaintTest", stream.ToArray());
```

```
        }
        MessageBox.Show("保存しました");
    }
```

画像処理を行ったり保存をしたりする場合には、WriteableBitmap を利用します。WriteableBitmap は、インクプレゼンター（InkPresenter）コントロールから直接画像を作成することができます。
以下の行で、bmp にインクプレゼンター（InkPresenter）の画像が抽出できます。

```
        WriteableBitmap bmp = new WriteableBitmap(inkPresenter1, null);
```

続いて、画像データを Jpeg 形式で取り出します。これは、WriteableBitmap の SaveJpeg メソッドを利用します。ここでは、メモリストリーム（MemoryStream）という一時的に用意したメモリ上に保存します。

```
        MemoryStream stream = new MemoryStream();
        bmp.SaveJpeg(stream, bmp.PixelWidth, bmp.PixelHeight, 0, 80);
```

最後に、メモリ上に保存した Jpeg 画像データをメディアライブラリ（MeriaLibrary）に保存します。メディアライブラリに保存された画像を Windows Phone のピクチャーからでも確認できます。

```
        using (MediaLibrary mediaLib = new MediaLibrary())
        {
            mediaLib.SavePicture("PaintTest", stream.ToArray());
        }
```

ここでは、using 文が使われています。このように記述すると、ここでの処理が終わったあと、ファイルのクローズ処理など自動的に行ってくれます。終了処理を自分で記述することも可能ですが、できるだけこういった書き方をする癖を付けることにより終了処理のし忘れといったバグを減らすことができるでしょう。
SavePicture は、メディアライブラリに保存するメソッドですが、ここでは、サンプルプログラムなのでファイル名を常に「PaintTest」という名前で保存しています。実用的なプログラムにするために自分で名前を付けるように修正するのもよいでしょう。

では、プログラムを実行してみましょう。
保存に成功すると「保存しました」と表示されるはずです。

■背景画像を読み込もう！

最後に画像の読み込みのプログラムを作ります。

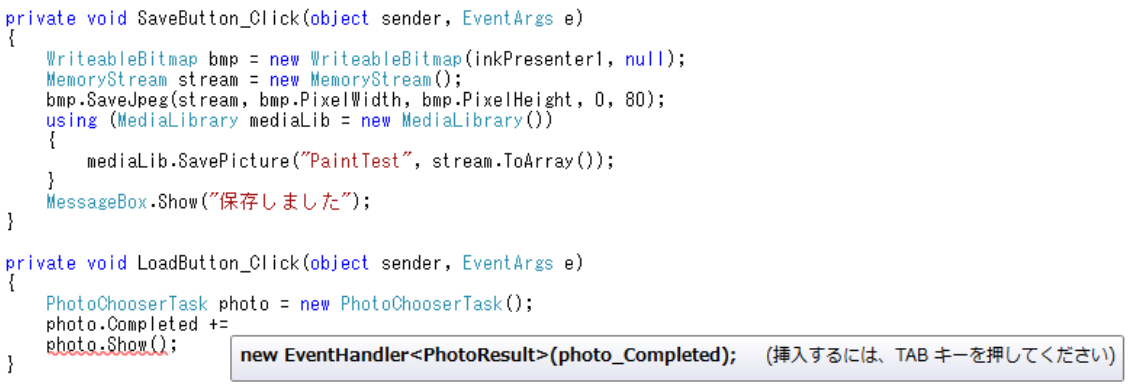
アプリケーションバー（ApplicationBar）の Load ボタンのイベントを追加してください。

手順は今までと一緒です。

[Click]イベントを追加すると、C#の画面が開きます。
プログラムに以下のように追加します。

```
        private void LoadButton_Click(object sender, EventArgs e)
        {
            PhotoChooserTask photo = new PhotoChooserTask();
            photo.Completed +=
            photo.Show();
        }
```

ここで、photo.Completed +=のところで、+= と入力したときにハンドラ挿入の画面が表示されますので、ここで続けて TAB、TAB と 2 回 TAB を押してください。



すると、プログラムが自動的に生成され以下のようになります。

```
        private void LoadButton_Click(object sender, EventArgs e)
        {
            PhotoChooserTask photo = new PhotoChooserTask();
            photo.Completed += new EventHandler<PhotoResult>(photo_Completed);
            photo.Show();
        }

        void photo_Completed(object sender, PhotoResult e)
        {
            throw new NotImplementedException();
        }
```

photo_Completed というハンドラが自動的に生成されましたので、こちらも以下のように修正します。

```
        void photo_Completed(object sender, PhotoResult e)
        {
```

```
            if (e.TaskResult == TaskResult.OK)
            {
                BitmapImage bmp = new BitmapImage();
                bmp.SetSource(e.ChosenPhoto);
                ImageBrush imageBrush = new ImageBrush();
                imageBrush.Stretch = Stretch.UniformToFill;
                imageBrush.ImageSource = bmp;

                inkPresenter1.Background = imageBrush;
            }
        }
```

では、プログラムを見ていきましょう。

LoadButton_Click イベントでは、PhotoChooserTask を生成しています。このクラスの Show メソッドを呼出すと、画像の選択画面が表示されます。このとき、画像が選択されるのを待たずに、イベントは終了してしまいます。

これにより、画像選択などをしているときに本体のプログラムが止まらないようになります。こういったプログラムの作り方は、Windows Phone ではいろいろなところで利用されます。

ここで、以下の行があります。

```
photo.Completed += new EventHandler<PhotoResult>(photo_Completed);
```

Completed イベントは、画像の選択が完了したときに呼出すイベントハンドラを指定します。

つまり、ここでは画像の選択が完了すると、photo_Comleted のハンドラが呼出されるというわけです。

では、photo_Comleted イベントハンドラを見てみましょう。

以下の if 文では、画像選択が正常に完了したかを判断しています。正しく選択された場合は、TaskResult.OK が戻りますので、読み込み処理を進行することになります。

例えば、画像選択中に←ボタン（戻る）を押した場合は、TaskResult.Cancel になりますので、ここでは何もしないで抜けることになります。

```
            if (e.TaskResult == TaskResult.OK)
            {
```

以下では、選択された画像をインクプレゼンター（InkPresenter）の背景画像に設定しています。

BitmapImage クラスは、画像管理するのに便利なクラスです。

ここでは、SetSource を使って選択された画像を読み込んでいます。

最終的に格納するインクプレゼンター（InkPresenter）の背景画像にするためには、ブラシ（Brush）の形式で設定しなければなりません。そこで、ImageBrush クラスを生成し、そこに BitmapImage の画

像を読み込ませています。
Stretch プロパティでは、UniformToFill を指定しています。この設定は、画像の縦横の比率を崩さず横幅一杯に画面を引き延ばして表示する指定です。（比率は崩れませんが縦にはみ出すことはあります。）
他に比率を崩して、画面内に最大のサイズで納める Fill などの指定ができます。
詳しくは MSDN で ImageBrush のメンバーを確認してみてください。

```
BitmapImage bmp = new BitmapImage();
bmp.SetSource(e.ChosenPhoto);
ImageBrush imageBrush = new ImageBrush();
imageBrush.Stretch = Stretch.UniformToFill;
imageBrush.ImageSource = bmp;

inkPresenter1.Background = imageBrush;
```

では、プログラムを動かしてみましょう。
背景画像ファイルを読み込んで、その画像の上にお絵かきができることが確認できましたか？

■おわりに

このお絵かきアプリでは、Windows Phone の特有の機能の ApplicationBar、ベクトルグラフィックスとラスタグラフィックスの両方の画像の取り扱いなどを学びました。

今回利用した技術は、非常に応用範囲が広い技術ですので、様々なアプリで利用することができるでしょう。このサンプルプログラムを元にいろいろ拡張してみるのもよい学習方

法だと思います。例えば、今回はペンの色が白色でサイズも固定でしたが、色の選択やペンの太さを指定できるようにしてみてはいかがでしょうか？ペンの縁の形状を変更してみるのも面白いでしょう。
MSDN の Code Recipe（http://code.msdn.microsoft.com/）というサイトもご覧になってください。沢山の便利なプログラムが用意されています。是非ご活用ください。

■完成したプログラムの全文
[XAML（MainPage.xaml）のプログラム]

```
<phone:PhoneApplicationPage
    x:Class="oekaki.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="696"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True" DataContext="{Binding}">

    <!--LayoutRoot は、すべてのページ コンテンツが配置されるルート グリッドです-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel は、アプリケーション名とページ タイトルを格納します-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="Windows Phone Sample"
Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="Oekaki" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
```

```
        </StackPanel>

        <!--ContentPanel - 追加コンテンツをここに入力します-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <InkPresenter Name="inkPresenter1" Background="Transparent"
MouseLeftButtonDown="inkPresenter1_MouseLeftButtonDown"
MouseMove="inkPresenter1_MouseMove" />
        </Grid>
    </Grid>

    <!--ApplicationBar の使用法を示すサンプル コード-->
    <phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True" x:Name="AppBar" >
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.erase.rest.png"
Text="Pen" x:Name="PenButtton" Click="PenButtton_Click" />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.delete.rest.png"
Text="Clear" x:Name="ClearButton" Click="ClearButton_Click" />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.save.rest.png"
Text="Save" x:Name="SaveButton" Click="SaveButton_Click" />
            <shell:ApplicationBarIconButton IconUri="/icons/appbar.folder.rest.png"
Text="Load" x:Name="LoadButton" Click="LoadButton_Click" />
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>

</phone:PhoneApplicationPage>
```

[C# (MainPage.xaml.cs) のプログラム]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
```

```
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using System.Windows.Ink;
using System.Windows.Media.Imaging;
using System.IO;
using Microsoft.Xna.Framework.Media;
using Microsoft.Phone.Tasks;
using Microsoft.Phone.Shell;


namespace oekaki
{
    public partial class MainPage : PhoneApplicationPage
    {
        // コンストラクター
        public MainPage()
        {
            InitializeComponent();
        }

        private Stroke currentStroke;

        private void inkPresenter1_MouseLeftButtonDown(object sender, MouseButtonEventArgs
e)
        {
            if (penMode == true)
            {
                currentStroke = new Stroke(e.StylusDevice.GetStylusPoints(inkPresenter1));
                currentStroke.DrawingAttributes.Color = Colors.White;
                currentStroke.DrawingAttributes.Width = 5;
                currentStroke.DrawingAttributes.Height = 5;
                inkPresenter1.Strokes.Add(currentStroke);
            }
        }
```

```
        private void inkPresenter1_MouseMove(object sender, MouseEventArgs e)
        {
            if (penMode == true)
            {

currentStroke.StylusPoints.Add(e.StylusDevice.GetStylusPoints(inkPresenter1));
            }
            else
            {
                StylusPointCollection erasePoints =
e.StylusDevice.GetStylusPoints(inkPresenter1);
                StrokeCollection hitStrokes = inkPresenter1.Strokes.HitTest(erasePoints);

                foreach (Stroke hitStroke in hitStrokes)
                {
                    Stroke sliceStroke = new Stroke();
                    sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;

                    foreach (StylusPoint sp in hitStroke.StylusPoints)
                    {
                        sliceStroke.StylusPoints.Add(sp);

                        if (sliceStroke.HitTest(erasePoints))
                        {

sliceStroke.StylusPoints.RemoveAt(sliceStroke.StylusPoints.Count - 1);
                            sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
                            inkPresenter1.Strokes.Add(sliceStroke);

                            sliceStroke = new Stroke();
                            sliceStroke.DrawingAttributes = hitStroke.DrawingAttributes;
                        }
                    }
                    inkPresenter1.Strokes.Add(sliceStroke);
                    inkPresenter1.Strokes.Remove(hitStroke);
```

```
            }
        }
    }

    private void ClearButton_Click(object sender, EventArgs e)
    {
        inkPresenter1.Strokes.Clear();
    }

    public bool penMode = true;
    private void PenButtton_Click(object sender, EventArgs e)
    {
        if (penMode == true)
        {
            penMode = false;
            ApplicationBarIconButton penButton =
(ApplicationBarIconButton)ApplicationBar.Buttons[0];
            penButton.IconUri = new Uri("/icons/appbar.edit.rest.png",
UriKind.Relative);
        }
        else
        {
            penMode = true;
            ApplicationBarIconButton penButton =
(ApplicationBarIconButton)ApplicationBar.Buttons[0];
            penButton.IconUri = new Uri("/icons/appbar.erase.rest.png",
UriKind.Relative);
        }
    }

    private void SaveButton_Click(object sender, EventArgs e)
    {
        WriteableBitmap bmp = new WriteableBitmap(inkPresenter1, null);
        MemoryStream stream = new MemoryStream();
        bmp.SaveJpeg(stream, bmp.PixelWidth, bmp.PixelHeight, 0, 80);
        using (MediaLibrary mediaLib = new MediaLibrary())
```

```
            {
                mediaLib.SavePicture("PaintTest", stream.ToArray());
            }
            MessageBox.Show("保存しました");
        }

        private void LoadButton_Click(object sender, EventArgs e)
        {
            PhotoChooserTask photo = new PhotoChooserTask();
            photo.Completed += new EventHandler<PhotoResult>(photo_Completed);
            photo.Show();
        }

        void photo_Completed(object sender, PhotoResult e)
        {
            if (e.TaskResult == TaskResult.OK)
            {
                BitmapImage bmp = new BitmapImage();
                bmp.SetSource(e.ChosenPhoto);
                ImageBrush imageBrush = new ImageBrush();
                imageBrush.Stretch = Stretch.UniformToFill;
                imageBrush.ImageSource = bmp;

                inkPresenter1.Background = imageBrush;
            }
        }
    }
}
```