Smart Button Project

A project to add more functionality to a button

Auteur: J.A. Kok

Website: www.hzns.nl

content

Introcution

Building an application I encountered a lack of space for buttons to manage the applications functionality. This was the trigger to a new project the "Smart Button Project" (SBP). In this project, the surface of a button is divided into zones. Each of these zones can be associated with functionality.

This document is divided into theoretical and a practical part. In the first part is a mathematical view on the idea of the "smart button". This information is in principle development environment independent. The second part is a practical implementation in a Visual Studio Basic.Net environment (Visual Basic.Net 2015 Express version). In annex 1 to this document you will find the integral VB.Net source code.

This document is not a cookbook (follow the recipe and you always fix it), but more an idea book with hints and maybe some tricks.

The Theory

Analysis

A closer look shows that there are two aspects to be worked out, namely “how I divide the surface of the button in zones” and “how do I know in which zone is clicked”. The coordinate system, which is used for locations on the screen, is the instrument for both creating zones and establishing place of the "mouse click".

Coordinate system

Each development environment uses a coordinate system for describing graphical elements on a screen (screen), in which the origin usually in the left top corner is located with a positive X axis from left to right and positive Y axis from top to bottom. Angles are defined in degrees, with zero degrees is straight up and the count clockwise. Tis is different from the Cartesian coordinate system (see also “Considerations when working with sine, cosine and degrees”), that we are used to from the mathematics.

Usually a point in the system is marked with an X- and a Y-coordinate. However, it is also possible to identify a point by the direction and the distance to another (reference) point. In such a case we call it a polar coordinate.

Graphical elements often use their own "internal" coordinate system, usually with its origin in the upper left corner of the element. Please note that is not always the case. When working with containers (such as a form in Microsoft Visual Studio) you must take the size of the boarders and title bar in account.

when working with containers (such as a form)

Where is the mouse

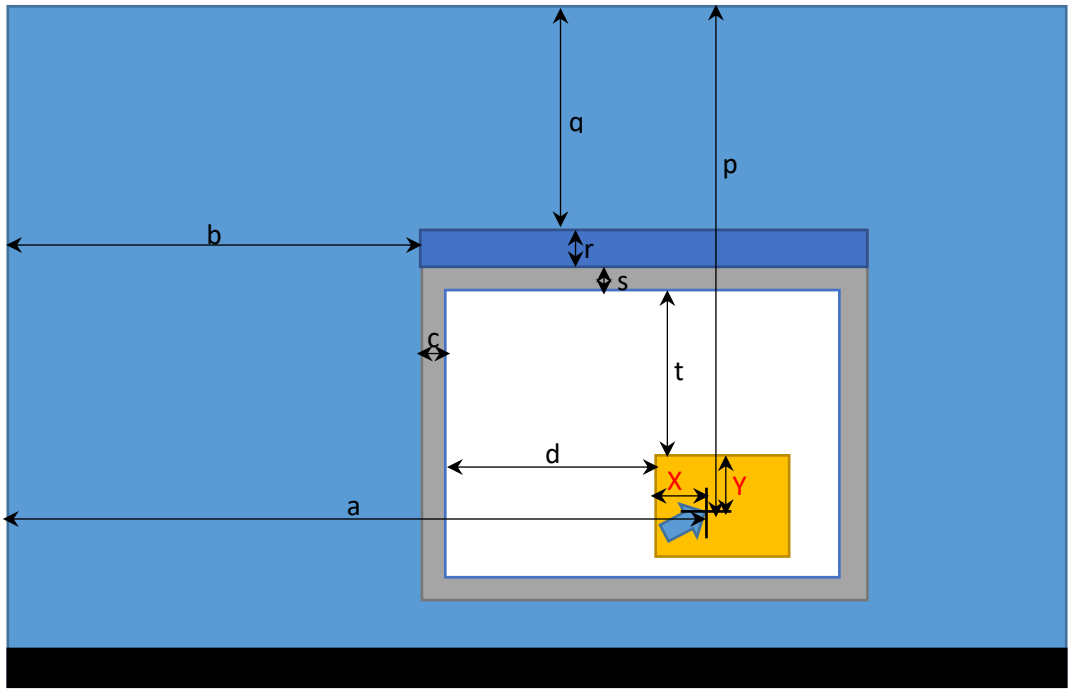
Introduction

The key question in this section is: "where the surface of the button is clicked with the mouse?" Although it seems simple to state where the mouse was at the moment of the “click”, turns out to be a road with pitfalls. The first concerns the fact that mouse position is always compared to the upper-left corner (coordinate (0,0)) of the screen. The second is the fact that each graphic element on our screen has its own internal coordinate system. The third concerns the fact that some graphic

elements are a "container" for other graphic elements, where the coordinate (0,0) of the "container" coordinate system not always exactly in the upper-left corner. And a form is such a "container".

Explanation

For the explanation we use the following diagram. It consists of a screen (light blue), a form (blue (framework bar), gray (framework border), white (workspace)), a button (yellow) and a axes cross (place where with the mouse is clicked). The requested coordinate is the (X, Y) of the mouse click compared to the origin of the button.



For the *X* (*MouseOnButton*$_X$) we can draw up the following equation:

X = a-b-c-d.

The value *a* corresponds to the *x* of the mouse click compared to the origin of the screen (*MouseOnScreen*$_X$), *b* correspond to *x* of the form compared to the origin of the screen (*Form*$_X$), *c* matches with the width of the border of the form (*Framework*$_{BORDER}$) and *d* matches with the *x* compared to the origin of the workspace (*Button*$_X$). The values of *MouseOnScreen*$_X$, *Form*$_X$, and *Button*$_X$ are on-demand variables. *Framework*$_{BORDER}$ isn't always available and must be calculated. The calculation follows further on in this document. The work out if the formula:

MouseOnButton$_X$ = *MouseOnScreen*$_X$ - *Form*$_X$ - *Framework*$_{BORDER}$- *Button*$_X$

For the *Y* (*MouseOnButton*$_Y$) we can draw up a similar equation:

Y = p - q - r - s - t.

The value *p* corresponds to the *y* of the mouse click compared to origin of the screen (*MouseOnScreen*$_Y$), of the form *q* match *y* compared to the origin of the screen (*Form*$_Y$), *r* corresponds to the height of the title bar of the form (*Framework*$_{BAR}$), *s* corresponds to the width of the border of the form (*Framework*$_{BORDER}$) and *t* corresponds to *y* of the workspace of the form compared to the button (*Button*$_Y$). The values of *MouseOnScreen*$_Y$, *Form*$_Y$, and *Button*$_Y$ are once again demand variables. *Framework*$_{BAR}$ and *Framework*$_{BORDER}$ aren't always available and must be calculated. The calculation further on in this document. The work out if the formula:

MouseOnButton$_Y$ = *MouseOnScreen*$_Y$ - *Form*$_Y$ - *Framework*$_{BAR}$ - *Framework*$_{BORDER}$- *Button*$_{Y.}$

The width of the border (*Framework*$_{BORDER}$) can be calculated from the width of the form (*Form*$_{WIDTH}$) and the width of the work space of the form (*Form*$_{INNERWIDTH}$). These two latter variables are usually callable.

Framework$_{BORDER}$ = ½ * (*Form*$_{WIDTH}$ - *Form*$_{INNERWIDTH}$)

The height of the bar (*Framework*$_{BAR}$) can be calculated in a similar way, although you to take the value of the *Framework*$_{BORDER}$ into account.

Framework$_{BAR}$ = *Framework*$_{HEIGHT}$ - *Form*$_{INNERHEIGHT}$ - 2 * *Form*$_{BORDER}$

In these calculations is assumed that the borders around the form have the same width. Also, there is no account taken of other properties)like the edges) of a form.

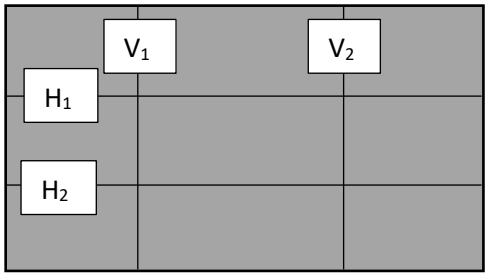
Zones

Introduction

The possibilities to organize into zones is infinite, as long as the boundaries between the zones can described mathematically. In practice, a grid, a circular and/or a segment (cake point) format are good workable options.

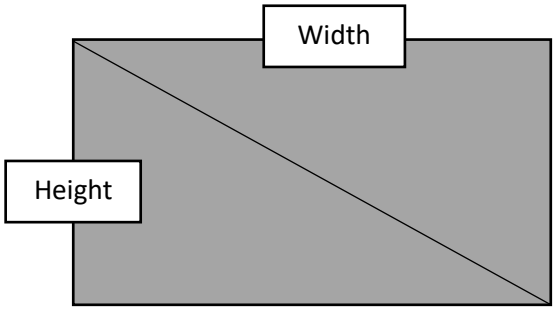
Grid layout

Horizontal and/or vertical boundaries. This is the simplest layout. The mathematical description of the boundaries are horizontal (Y has a fixed value) and vertical lines (X has a fixed value).



In the example to the right is the conditions for a mouse click in the middle box: V_1 <= X_{mouse} <= V_2 and H_1 <= Y_{mouse} <= H_2

Sloped or curved borders. For a grid layout with sloped and/or curved boundaries first the mathematical formula of this boundary must be established. As an example: for diagonal from top left to bottom right is the formula *Y = H(eight) / (W(idth) * X)*. This formula can be reformulated to *0 = (H/B) * X-Y*. The right part of this formula we can use as a kind of test. For a random point (for example the location of the mouse click), we can substitute all the values. This *T(est)* value can have three states: positive, 0 or negative. *T > 0* means the random point lies above the boundary, *T = 0* the random point lies at the boundary and *T < 0* the random point lies below the boundary.

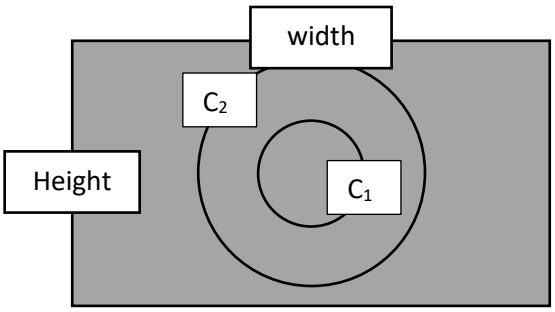


For the mathematical purists: the width of the surface should never have the value 0 (since it is in the denominator). In practice, however, this will not be a problem, a button has always a width. The method also works for curved borders, however the finding the right mathematical formula is more difficult.

Finally: Test your formula in advance. Check in which area your T(est) value has which value (+ , 0 , -). This conceptual phase is the best moment to customize the logic for your implementation.

Circular layout

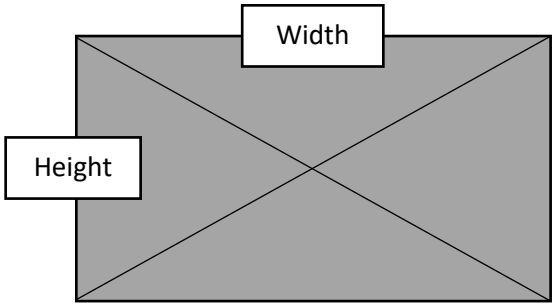
A circular format is a zoning based on the distance of the mouse click and a chosen reference point, whether or not located within the surface. In the example to the right the reference point is the center of the surface ($X_{RefPoint}$ = ½ * *Width*, $Y_{RefPoint}$ = ½ * *Height*).



The distance between the reference point and the mouse click can easily be calculated by using the Pythagorean theorem: $D(istance) = ((X_{Mouse} - X_{RefPoint})^2 + (Y_{Mouse} - Y_{RefPoint})^2)^{½}$. The condition for the area between the circles C1 and C2 is: $Distance\ C_1 < Distance\ Mouse < Distance\ C_2$. The method described above is also usable for an elliptical zone ring, but is mathematically more challenging!

Segment layout

A segment layout is a zoning based on angle Between the mouse click and a reference point. In the example to the right the reference point is the center of the surface ($X_{RefPoint}$ = ½ * *Width*, $Y_{RefPoint}$ = ½ * *Height*). The boundaries of the segments are formed by the diagonals of the surface. By comparing the angle of the mouse click to the center and the angles of these boundaries you can determine in which segment was clicked with the mouse.



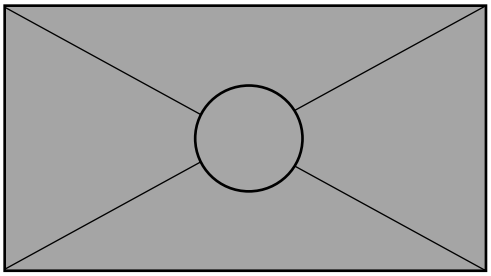
The condition for the right segment is: $Angle_{TopRight} < Angle_{Mouse} < Angle_{BottomRight}$.

Corners can not only defined in degrees but also with their sine and cosine (easier to calculate). To do this, first calculate the sine and cosine the upper left corner as reference values. Based on these values the conditions for the segments are:

Top segment:	$Sine_{Mouse} < Sine_{TopLeft}$
Right segment:	$Cosine_{Mouse} < -Cosine_{TopLeft}$
Bottom segment:	$Sine_{Mouse} > -Sine_{TopLeft}$
Left segment:	$Cosine_{Mouse} > Cosine_{TopLeft}$

Conbinations

It is also possible to make combinations of zone layouts. An example on the right. In such case a smart order of the conditions is important to avoid double hits. In the example First test for circle zone using the distance and second the segment zones.



Hints for the implementation

The boundaries. It is good to think in advance about what to do with the points on a Boundary between zones, to include these points to which zone. Suppose you have a button with a width of 200 pixels and you have the button divided equally into a left and a right zone. The boundary will be on 100 pixels. You choose to include the boundary in left zone. The X-coordinate of the most left

pixel will be 0 and the most right 100. If we start to count the number of actual pixels the left zone is 101 pixels width. For the right zone only 99 pixels remains and so we have an uneven distribution. The better solution is to include the boundary in the right one. This results in two equal width zones. The conditions for these “better” case are:

Left zone : $X_{Mouse} < X_{Boundary}$

Right zone : $X_{Mouse} >= X_{Boundary}$

The cause of this "phenomenon" is that counting pixels starts with 0, both with the X (horizontal) and Y (vertical). As a rule include the boundary to the right and/or bottom zone.

Create your own functions. When working with segments and circles it can be useful to work with own built functions. You do the thinking once and using it many times. I myself use the following function:

Input: X_{Point}, Y_{Point}, $X_{RefPoint}$, $Y_{RefPoint}$

Output: $Angle_{(Point, RefPoint)}$, $Distance_{(Point, RefPoint)}$, $Cosinus_{(Point, RefPoint)}$, $Sinus_{(Point, RefPoint)}$

Mathematicians among you will notice a wink to the vector geometry. A practical implementation for VB.Net can be found in the practical part of this document.

The practice with Visual Baisc.NET (version 2015 Express)

All code examples were written in Visual studio Basic.Net 2015 Express. This is a free of charge version that Microsoft makes available, however, with restrictions for commercial use. This code is laced with comment lines. These are on the one hand reminders to myself (to avoid the question, "what/why I have done this?") and, on the other hand, to make the code accessible to other people. The code examples in this document ca be copied into VB.Net. The colors of the code are the same used in the development environment of Visual Studio.

In this practical part first some preliminary aspects will be worked out followed by methods to describe zones (including source code examples). At the end you will find three full implementable examples. The source code and the executable files can be found on site www.hzns.nl.

Coordinates

When working with pixel coordinates it may be helpful to introduce a new type of variable. Two important characteristics are: the pixel coordinates are always integers and they are normally smaller than 32000 (and 16 bit). The VB.NET solution looks so like this:

```
'Variable structure for handling pixel-coordinates
Structure Coordinate
    Dim X As Short
    Dim Y As Short
End Structure
```

Older versions of Visual Basic use another syntax for `Structure` and `Short`. Use instead `Short` and `Integer`.

In addition to " ordinary " coordinates polar coordinates can be used. These have the following structure (Arc (angle), Radius (distance), Cosinus (cosine) and Sinus(sine)):

```
'Variable structure for handling pixel-polar coordinates
Structure PolarCoordinate
    Dim Arc As Double
    Dim Radius As Double
    Dim Cosinus As Double
    Dim Sinus As Double
End Structure
```

Unlike the ordinary coordinates are "broken" values possible. At trigonometric functions is, in calculations with 90 degree or multiples increments, The cosine or sine have very small values. The variable type `Double` is recommended.

Mouse positon

The ultimate goal is to find the coordinate of the mouse click relative to the upper-right corner of the button. For this we use the diagram as given in the theory. The button name is `SmartButton`.

Mouse position relative to the screen

The mouse position relative to the upper-right corner of the screen can be captured with the function `MousePosition`. These are the values a (*MouseOnScreen*$_X$) and p (*MouseOnScreen*$_Y$).

```
'Mouse position on screen
 Dim ScreenMouse As Coordinate
 ScreenMouse.X = MousePosition.X
 ScreenMouse.Y = MousePosition.Y
```

Form position relative to the screen

The Forms X and Y coordinate ((values b (*Form*$_X$) and q (*Form*$_Y$) in the diagram) are a properties of the Form.

```
'Form topleftcorner on screen
 Dim FormTopLeftCorner As Coordinate
 FormTopLeftCorner.X = Me.Left
 FormTopLeftCorner.Y = Me.Top
```

Size of the forms framework

The next step is to determine the dimensions of the framework of the form. First we define a variable. This is the width of the `Border` And the height of the title `Bar`. (Based on the presumption the border width is on all sides the same.)

```
'Variable structure for handling the bar and border size of a container
Structure FrameSize
    Dim Border As Short
    Dim Bar As Short
End Structure
```

I can recommend two methods. The first is a more general and the second there is one specifically for working with forms. Both are described in the structure of a function.

For the first method we need four parameters in the function. This concerns:
- the forms outer width (on demand value as `<ContainerName>.Width`)
- the forms outer height (on demand value as `<ContainerName>.Height`)
- the forms inner width (on demand value as `<ContainerName>.ClientSize.Width`)
- the forms inner height (on demand value as `<ContainerName>.ClientSize.Height`)

```
'Function calculates size of the border and titlebar for rectangular
 objects(framework)
 Function ObjectFrame(OuterWidth As Integer, OuterHeight As Integer,
     InnerWidth As Integer, InnerHeight As Integer) As FrameSize
    'Internal function variable
     Dim Result As FrameSize

    'Calculating border and bar size (=Result)
     Result.Border = (OuterWidth - InnerWidth) / 2
```

```
        Result.Bar = OuterHeight - InnerHeight - 2 * Result.Border

       'Returning function value
        Return Result
    End Function
```

Used in an example:

```
   'Variable on module level
    Dim FormFrame As Framesize

   'Forms border- and barsize are calculated by using the function ObjectFrame
    FormFrame = ObjectFrame(Me.Width, Me.Height, Me.ClientSize.width, _
                Me.ClientSize.Height)
```

In the second method the form itself is used as parameter:

```
   'Function calculates border- and barsize for forms
    Function Frame(Element As Form) As FrameSize
       'Internal function variable
        Dim Result As FrameSize

       'Calculating border and bar size (=Result)
        Result.Border = (Element.Width - Element.ClientSize.Width) / 2
        Result.Bar = Element.Height - Element.ClientSize.Height - 2 * Result.Border

       'Returning function value
        Return Result
    End Function
```

Used in an example:

```
   'Variable on module level
    Dim FormFrame As Framesize

   'Forms border- and barsize are calculated by using the function Frame
    FormFrame = Frame(Me)
```

Both functions return the same result, namely `FormFrame.Border` as the values *c* and *r* (*Formulier*$_{KADER}$) and `FormFrame.Bar` for the value *s* (*Formulier*$_{BALK}$).

Position of the Button related to the workspace of the form

The buttons top left corner (values *d* (*Button*$_X$) and *t* (*Button*$_Y$) in diagram) is a property of the button. They are variable on demand. They are defined related to the workspace (`ClientSize`).

```
   'Position topleftcorner of button based on clientsize form
   'SmartButton is a button on the form
    Dim ButtonTopLeftCorner As Coordinate
    ButtonTopLeftCorner.X = SmartButton.Left
    ButtonTopLeftCorner.Y = SmartButton.Top
```

The end result

The Workout the formula from the theory looks like:

```
   'Position mouse on SmartButton
    Dim ButtonMouse As Coordinate
   'Calculate relative position mouse on the SmartButton
    ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X – _
                    FormFrame.Border - ButtonTopLeftCorner.X
    ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - _
                    FormFrame.Border - FormFrame.Bar - ButtonTopLeftCorner.Y
```

Raster zones

Simpel raster

In a simple raster layout the zones are separated by vertical and horizontal (boundaries) lines. Vertical zone boundaries have a "fixed" X-coordinate. In code may look like this:

```
'Zone left side of the border
 ButtonMouse.X < VerticalZoneBoundary
'zone right side of the border
 ButtonMouse.X > VerticalZoneBoundary
```

The boundary was included in de right zone (as mention in the theory).

For the horizontal borders it is more or less the same (with a “fixed” Y coordinate):

```
'Zone above the border
 ButtonMouse.Y < HorizontalZoneBorder
'Zone below the border
 ButtonMouse.Y > HorizontalZoneBorder
```

The boundary was included in de Bottom zone (as mention in the theory).

In the second example you will find work out for a raster with both horizontal and vertical boundaries.

Sloped lines or mathematical formulas as boundary

If as boundary is sloped or is described with a mathematical formula we must use the test value (`TestValue`) method. As an example a simple line as boundary with the formula *Y = X*. When we transform this formula (as in the theory) in the test value structure we get the following formula: *Test value = X – Y.* The code example for this case is:

```
'Internal variable
 Dim TestValue As Short

'Calculating TestValue
 TestValue = MouseButton.X - MouseButton.Y

'Determinating above, on or below the border
 If TestValue > 0 then
     MsgBox ("Above border",,)
 ElseIf TestValue  = 0 then
     MsgBox ("On border",,)
 Else
     MsgBox ("Below border",,)
 EndIf
```

Creating circular zones

In mathematics a circle is the set of points that are at a given distance from a given point, the center. A distance between two points (for example, the center of a circle and the place where is clicked with the mouse) can be calculated using the Pythagorean theorem. In this part of this document we will call the center of the circle the reference point and the distance to that reference point the radius. The code for the calculation of the Radius could look like:

```
Function CreateRadius(Point As Coordinate,  Referencepoint As Coordinate) As Double
   'Internal function variable
    Dim Result As Double
```

```
    'Calculating Radius (= Result)
     Result = ((Point.X - Referencepoint.X) ^ 2 + _
         (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5

    'Returning function value
     Return Result
End Function
```

The decision logic might look like this:

```
    'Variable on module level
     Dim MouseClick As Coordinate
     Dim Center As Coordinate
     Dim RadiusBoundary As Short
     Dim RadiusMouse As Double

    'Calculating Radius
     RadiusMouse = CreateRadius(MouseClick, Center)

    'Determinating in-, on or outside the circle
     If RadiusMouse > RadiusBoundary Then
         MsgBox("Outside circle",,)
     ElseIf RadiusMouse = RadiusBoundary Then
         MsgBox("On circle",,)
     Else
         MsgBox("Inside circle",,)
     End If
```

In the first example, further on in this document is a circular zone around the center of the button used.

Creating segment zones

Segments can be described as a pie wedges whose boundaries be determines by the angle to a certain fixed point (reference point). By comparing the angle of the mouse click to the reference point with the boundary angles we can determine whether the mouse click was in within or outside the segment.

Angles can be described in different ways. This document describes two methods: one using sine/cosine and another using degree.

Using sine and cosine

The sine and the cosine in mathematics are described using a right triangle. The sine as "opposite side" divided by the "hypotenuse" and the cosine as "adjacent side" divided by the "hypotenuse".

In a practical implementation the "hypotenuse" can be translated as the distance between the center of the segments (*Referencepoint*) and a specific point (for example, the mouse click (*Mouseclick*)), the "adjacent side" the difference in X-coordinates (with sign) between the two points ($Mouseclick_X$ - $Referencepoint_X$) and the "opposite side" the difference in Y-coordinate (with sign) between the two points ($Mouseclick_Y$ - $Referencepoint_Y$). In code, the following functions can be used:

```
Function CreateSinus(Point As Coordinate, Referencepoint As Coordinate) As Double
    'Internal function variable variables
     Dim Result As Double
     Dim Radius As Double

    'Calculating Radius and Sinus (= result)
     Radius = ((Point.X - Referencepoint.X) ^ 2 + _
         (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
```

```
    Result = (Point.Y - Referencepoint.Y) / Radius

   'Returning function value
    Return Result
End Function

Function CreateCosinus(Point As Coordinate, Referencepoint As Coordinate) As Double
   'Internal function variable variables
    Dim Result As Double
    Dim Radius As Double

   'Calculating Radius and Cosinus (= result)
    Radius = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Result = (Point.X - Referencepoint.X) / Radius

   'Returning function value
    Return Result
End Function
```

The decision logic for a circle with six equal segments might look like this:

```
   'Variable on module level
    Dim ButtonMouse As Coordinate   'Location of mouse click on button
    Dim Center As Coordinate        'Location of center of segments

   'Calculating Cosinus and Sinus
    Cosinus = CreateCosinus(ButtonMouse, Center)
    Sinus = CreateSinus(ButtonMouse, Center)

   'Determinating in which segment was clicked (six segments)
    Select Case Cosinus
        Case >= 0
            If Sinus < -0.5 Then
                MsgBox("top right",,)
            ElseIf Sinus >= -0.5 And Sinus < 0.5 Then
                MsgBox("middle right",,)
            ElseIf Sinus >= 0.5 Then
                MsgBox("bottom right",,)
            End If
        Case < 0
            If Sinus < -0.5 Then
                MsgBox("top left",,)
            ElseIf Sinus >= -0.5 And Sinus < 0.5 Then
                MsgBox("middle left",,)
            ElseIf Sinus >= 0.5 Then
                MsgBox("bottom left",,)
            End If
    End Select
```

In the first example sine and cosine are used to identify the correct zone.

Using degrees

With a given sine and cosine of an angle you can make a conversion to degrees. The VB.NET functions generate an angle in radians. By dividing by 2 Pi and multiplying by 360thes value is converted to degree's: Value $_{\text{in Degree}}$ = Value $_{\text{in Radians}}$ * 360 / 2 Pi (is same as Value $_{\text{in Radians}}$ * 180 / Pi used in de functions).

Using a given sine and cosine of an angle we can determine the angle in degrees. The VB.Net features for sine and cosine make an angle in radians. This value divided by 2 Pi and multiplying by 360 is this corner converted to degrees (2Pi/360 = Pi/180).

```
Function CreateDegree(Point As Coordinate, Referencepoint As Coordinate) As Double
   'Internal function variable variables
    Dim Result As Double
    Dim Radius As Double
    Dim Sinus As Double
    Dim Cosinus As Double
    Dim TempArc As Double

   'Calculating Radius, Sinus and Cosinus
    Radius = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Sinus = (Point.Y - Referencepoint.Y) / Radius
    Cosinus = (Point.X - Referencepoint.X) / Radius

   'Calculating TempArc in degrees
    TempArc = Math.Acos (Cosinus) * (180 / Math.PI)

   'Determinating the real arc (=Result) based on Sinus and Cosinus in degree
    If Cosinus >= 0 And Sinus < 0 Then
        Result = 90 - TempArc
    EleseIf Cosinus >= 0 And Sinus >= 0 Then
        Result = 90 + TempArc
    EleseIf Cosinus < 0 And Sinus >= 0 Then
        Result = 90 + TempArc
    EleseIf Cosinus < 0 And Sinus < 0 Then
        Result = 450 - TempArc
    End If

   'Returning function value
    Return Result

End Function
```

The decision logic for a circle with six equal segments might look like this:

```
   'Variable on module level
    Dim ButtonMouse As Coordinate   'Location of mouse click on button
    Dim Center As Coordinate        'Location of center of segments

   'Calculating Cosinus and Sinus
    Degree = CreateDegree(ButtonMouse, Center)

   'Determinating in which segment was clicked (six segments)
    If Degree >= 0 And Degree < 60 Then
        MsgBox("top right",,)
    ElseIf Degree >= 60 And Degree < 120 Then
        MsgBox("middle right",,)
    ElseIf Degree >= 120 And Degree < 180 Then
        MsgBox("bottom right",,)
    ElseIf Degree >= 180 And Degree < 240 Then
        MsgBox("bottom left",,)
    ElseIf Degree >= 240 And Degree < 300 Then
        MsgBox("middle left",,)
    ElseIf Degree >= 300 And Degree < 360 Then
        MsgBox("top left",,)
    End If
```

In the third example degrees ar used to identify the correct zone.

Considerations when using sine, cosine and degrees

When using the trigonometric functions sine and cosine be ware they are based on a Cartesian coordinate system: X increases from left to right, Y increases from **bottom to top** and angles **anti-clockwise**. The coordinate system we use on our screen is different: X increases from left to right too, but Y increases from **top to bottom** and angles **clockwise**. When creating the decision logic, we must take differences into account.

A second point of focus is order of the terms in determining the differences. The first term is the specific point and the second the reference point. In this way the sign of the difference in X and the difference in Y correspond to the sign of the coordinate system. Turning the order affects the decision logic.

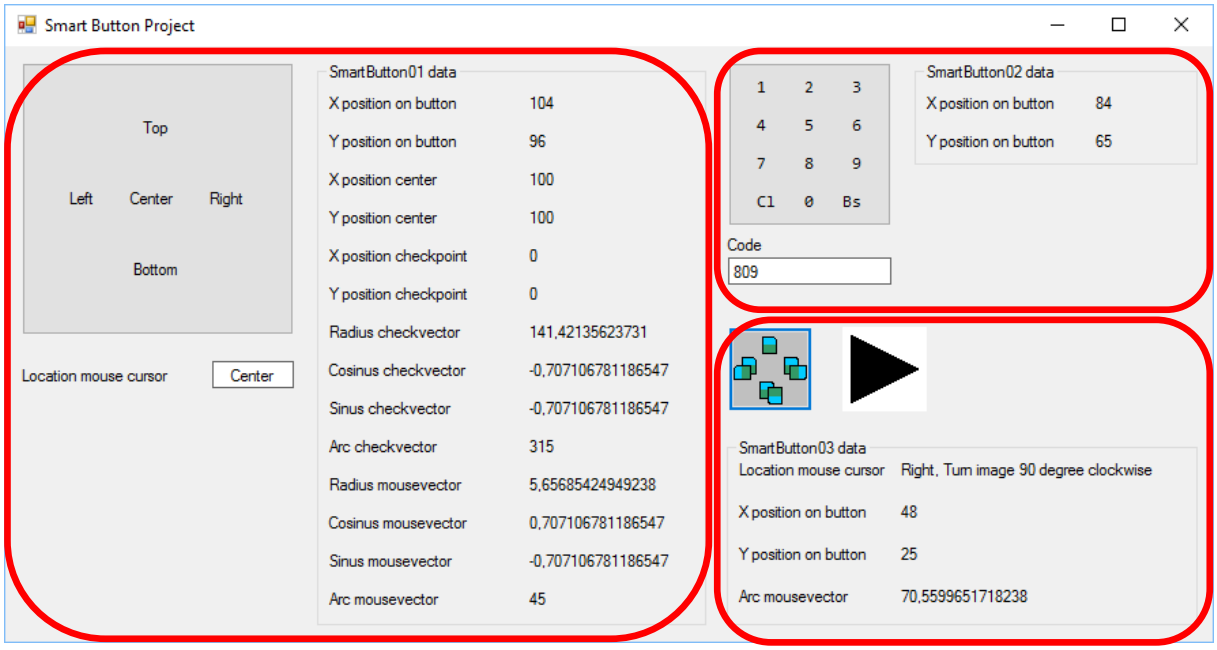
Polar coordinate

In the theory is already indicated that a point can also be described using a polar coordinate, describing a point by giving the distance and direction (angle) relative to another point. In the function CreatePolarCoord is generating a variable with all the features of a polar coordinate. Actually it's a combination of the preceding four functions (`CreateRadius, CreateSinus, CreateCosinus, CreateDegree`).

```
Function CreatePolarCoord(Point As Coordinate, Referencepoint As Coordinate) _
    As PolarCoordinate
   'Internal function variable variables
    Dim TempArc As Double
    Dim Result As PolarCoordinate

   'Calculating Radius, Cosinus, Sinus and Arc
    Result.Radius = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Centerpoint.Y) ^ 2) ^ 0.5
    Result.Cosinus = (Point.X - Referencepoint.X) / Result.Radius
    Result.Sinus = (Point.Y - Referencepoint.Y) / Result.Radius

   'Calculating TempArc in degrees
    TempArc = Math.Acos(Result.Cosinus) * (180 / Math.PI)

    'Determining the real arc (=Result.Arc) based on Sinus and Cosinus in degree
    If Result.Cosinus >= 0 And Result.Sinus < 0 Then
        Result.Arc = 90 - TempArc
    ElseIf Result.Cosinus >= 0 And Result.Sinus >= 0 Then
        Result.Arc = 90 + TempArc
    ElseIf Result.Cosinus < 0 And Result.Sinus >= 0 Then
        Result.Arc = 90 + TempArc
    ElseIf Result.Cosinus < 0 And Result.Sinus < 0 Then
        Result.Arc = 450 - TempArc
    End If

   'Returning function value
    Return Result
```

Examples

This document includes three examples. You will find them in the sample application "Smart Button Project" (SBP.exe). Most of the functionality discussed in this document returns in these examples. `Smartbutton01` (left) is the first example. It was used for developing and describing the functionality. `Smartbutton02` (upper right) is an example of using raster zones. `Smartbutton03` (bottom right) shows you four different (related) functionalities an a 62 by 62 pixels size button.



The project uses a single form (SPBmain. vb (VB code, annex 1). The VB.Net code consists of 6 sections, namely "Structures for module-level variables and variables", "Functions", "Activities when loading the form" and the functionality for `SmartButton01` to `SmartButton03`. On the image above you see all kinds of data is displayed. These are related to the variables that are used within the different subroutines.

Part 1: Structures for module-level variables and variables

In part 1 will find the variable structures for coordinates, polar coordinates and the framework of a form. In addition, they included two variables used at the module level for `SmartButton02` and `SmartButton03`.

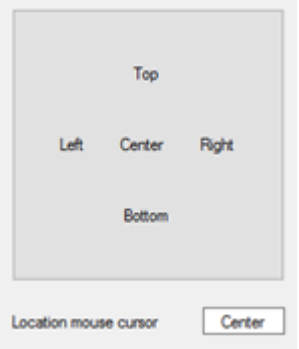
Part 2: Functions

On this spot you will find all the functions described in this document.

Part 3: Activities when loading the form

When the form is opened the text of `SmartButton01` and `SmartButton02` are loaded. Also a copy is maded of the "arrow" image used by `SmartButton03`.

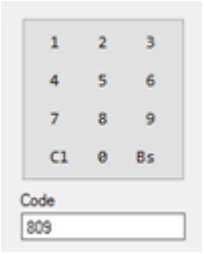
Part 4: Example Smartbutton01



This first example was my first finger exercise of this project. The goal was to achieve a button with 5 zones, namely a circular zone in the middle surrounded by 4 zones which are separated by the diagonals. (see theory combinations). The button is for testing purposes over dimensioned. The first steps of the `SmartButton01_Click` subroutine the location of the mouse click on the button is established. Then the sine, cosine and the distance to the upper left corner (`Checkpoint/Check Vector`) and the

mouse click is determined by the function `CreatePolarCoord`. Then Both angles are used for determining the appropriate zone by comparability. The actual action linked to the mouse click is showing the zone where was clicked.

Part 5: Example Smartbutton02



The second example is a "simulation" of a phone keypad. The button is divides into a raster with twelve zones (layout three zones width, four zones height). Also in this example, the first steps the location of the mouse click on the button is established. Next, the X and Y of the mouse click compared with the boundaries between the zones. The actual action linked to the mouse click is added the clicked character to the test window “Code” below the button. The zones CL and Bs offer the possibility to delete all characters (CL = Clear) or only the last character (Bs = Backspace).

Part 6: Example Smartbutton03

The third example is real prototype for a button I use in another application. The square button consists of 4 zones, separated by the diagonals. (Coherent) functionalities to rotate an image are linked to the zones. (Clockwise from top: Image Restore, 90 degrees to Rotate clockwise, rotate 180 degrees and 90 degrees counter-clockwise rotate). In this prototype you will see the effect of clicking on one of the zones; the arrow turns. Once again are the first steps in this subroutine are aimed to locate the mouse click. Then, using the function `CreatePolarCoord` the angle is established. This angle determine the clicked zone.
For your information: the images Button.bmp and URArrow.bmp wer placed in the "My Project" folder. In the project they were imported into "Project Resource file" and from there linked to the correct PictureBox or Button.

Experiences

- A 30 by 30 pixels zone delivers a usable situation, both for the use of a mouse as for the use of a stick on a touchscreen.
- Test the functionality of all zones, as soon as possible. If necessary with the VB.Net messages-functionality (MsgBox). In this way, you can convince yourselves of the accuracy of the decision logic.
- When building decision logic take in account the differences between the screen coordinate system and the Cartesian system (see “Considerations when using sine, cosine and degrees”)

Annex 1: VB Code of SBPmain.

```
Public Class SBPmain

    'Part 1: Variable structures and variables on module level

    'Variable structure for handling pixel-coordinates
    Structure Coordinate
        Dim X As Short
        Dim Y As Short
    End Structure

    'Variable structure for handling pixel-polar coordinates
    Structure PolarCoordinate
        Dim Arc As Double
        Dim Radius As Double
        Dim Cosinus As Double
        Dim Sinus As Double
    End Structure

    'Variable structure for handling the bar and border size of a container
    Structure FrameSize
        Dim Border As Short
        Dim Bar As Short
    End Structure

    'Variable used by SmartButton02
    Dim SmartText As String

    'Variable (object) used by SmartButton03
    Dim BaseImage As Image

    'Part 2: Functions

    'Function calculates size of the border and titlebar for rectangular objects (framework)
    Function ObjectFrame(OuterWidth As Integer, OuterHeight As Integer,
        InnerWidth As Integer, InnerHeight As Integer) As FrameSize
        'Internal function variable
        Dim Result As FrameSize

        'Calculating border and Bar size (=Result)
        Result.Border = (OuterWidth - InnerWidth) / 2
        Result.Bar = OuterHeight - InnerHeight - 2 * Result.Border

        'Returning function value
        Return Result

    End Function

    'Function calculates border- and barsize for forms
    Function Frame(Element As Form) As FrameSize
        'Internal function variable
        Dim Result As FrameSize

        'Calculating border and Bar size (=Result)
        Result.Border = (Element.Width - Element.ClientSize.Width) / 2
        Result.Bar = Element.Height - Element.ClientSize.Height - 2 * Result.Border

        'Returning function value
        Return Result

    End Function

    'Function calculates the distance (Radius) between two points
    Function CreateRadius(Point As Coordinate, Referencepoint As Coordinate) As Double
        'Internal function variable variables
        Dim Result As Double

        'Calculating Radius (= Result)
        Result = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5

        'Returning function value
        Return Result
```

```
End Function

'Function calculates the sinus between two points
Function CreateSinus(Point As Coordinate, Referencepoint As Coordinate) As Double
    'Internal function variables
    Dim Result As Double
    Dim Radius As Double

    'Calculating Radius and Sinus (= Result)
    Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Result = (Point.Y - Referencepoint.Y) / Radius

    'Returning function value
    Return Result

End Function

'Function calculates the cosinus between two points
Function CreateCosinus(Point As Coordinate, Referencepoint As Coordinate) As Double
    'Internal function variables
    Dim Result As Double
    Dim Radius As Double

    'Calculating Radius and Cosinus (= Result)
    Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Result = (Point.X - Referencepoint.X) / Radius

    'Returning function value
    Return Result

End Function

'Function calculates the arc in degrees between two points
Function CreateDegree(Point As Coordinate, Referencepoint As Coordinate) As Double
    'Internal function variables
    Dim Result As Double
    Dim Radius As Double
    Dim Sinus As Double
    Dim Cosinus As Double
    Dim TempArc As Double

    'Calculating Radius, Sinus and Cosinus
    Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Sinus = (Point.Y - Referencepoint.Y) / Radius
    Cosinus = (Point.X - Referencepoint.X) / Radius

    'Calculating TempArc in degree
    TempArc = Math.Acos(Cosinus) * (180 / Math.PI)

    'Determining the real arc (=Result) based on Sinus and Cosinus in degree
    If Cosinus >= 0 And Sinus < 0 Then
        Result = 90 - TempArc
    ElseIf Cosinus >= 0 And Sinus >= 0 Then
        Result = 90 + TempArc
    ElseIf Cosinus < 0 And Sinus >= 0 Then
        Result = 90 + TempArc
    ElseIf Cosinus < 0 And Sinus < 0 Then
        Result = 450 - TempArc
    End If

    'Returning function value
    Return Result

End Function

'Function calculates polar coordinate between two points
Function CreatePolarCoord(Point As Coordinate, Referencepoint As Coordinate) As PolarCoordinate
    'Internal function variable variables
    Dim TempArc As Double
    Dim Result As PolarCoordinate

    'Calculating Radius, Cosinus, Sinus and Arc
    Result.Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
```

```
        Result.Cosinus = (Point.X - Referencepoint.X) / Result.Radius
        Result.Sinus = (Point.Y - Referencepoint.Y) / Result.Radius

        'Calculating TempArc in degrees
        TempArc = Math.Acos(Result.Cosinus) * (180 / Math.PI)

        'Determining the real arc (=Result.Arc) based on Sinus and Cosinus in degree
        If Result.Cosinus >= 0 And Result.Sinus < 0 Then
            Result.Arc = 90 - TempArc
        ElseIf Result.Cosinus >= 0 And Result.Sinus >= 0 Then
            Result.Arc = 90 + TempArc
        ElseIf Result.Cosinus < 0 And Result.Sinus >= 0 Then
            Result.Arc = 90 + TempArc
        ElseIf Result.Cosinus < 0 And Result.Sinus < 0 Then
            Result.Arc = 450 - TempArc
        End If

        'Returning function value
        Return Result

    End Function


    'Part 3: Form loading activities

    Private Sub SBPmain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'On load the text or image for the buttons is created (Text is horizontal an vertical centered
)
        'Subroutine internal variables
        Dim ButtonText As String
        Dim EndOfLine As String = Chr(13) & Chr(10)

        'Creating the text for SmartButton01
        ButtonText = "Top" & EndOfLine & EndOfLine & EndOfLine & EndOfLine &
            "Left         Center         Right" &
             EndOfLine & EndOfLine & EndOfLine & EndOfLine & "Bottom"
        SmartButton01.Text = ButtonText

        'Creating the text for SmartButton02
        ButtonText = "1    2    3" & EndOfLine & EndOfLine &
            "4    5    6" & EndOfLine & EndOfLine &
            "7    8    9" & EndOfLine & EndOfLine &
            "Cl   0   Bs"
        SmartButton02.Text = ButtonText

        'Image used for SmartButton03
        'Saving a copy of the original picture
        BaseImage = Button03PictureBox.Image.Clone

    End Sub

    'Part 4: Handling clicks on SmartButton01
    Private Sub SmartButton01_Click(sender As Object, e As EventArgs) Handles SmartButton01.Click

        'determination position of mouse click

        'Mouse position on screen
        Dim ScreenMouse As Coordinate
        ScreenMouse.X = MousePosition.X
        ScreenMouse.Y = MousePosition.Y

        'Form topleftcorner on screen
        Dim FormTopLeftCorner As Coordinate
        FormTopLeftCorner.X = Me.Left
        FormTopLeftCorner.Y = Me.Top

        'Forms border- and barsize are calculated by using the function Frame
        Dim FormFrame As FrameSize
        FormFrame = Frame(Me)

        'Position topleftcorner of button based on clientsize form
        Dim ButtonTopLeftCorner As Coordinate
        ButtonTopLeftCorner.X = SmartButton01.Left
        ButtonTopLeftCorner.Y = SmartButton01.Top
```

```
        'Position mouse on SmartButton
        Dim ButtonMouse As Coordinate
        'Calculate relative position mouse on the SmartButton
        ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X - FormFrame.Border - ButtonTopLeftCorner.X
        ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - FormFrame.Border - FormFrame.Bar -
ButtonTopLeftCorner.Y

        'Data to labels on form
        XposMouse01Value.Text = ButtonMouse.X
        YposMouse01Value.Text = ButtonMouse.Y

        'Calculating center of SmartButton01
        Dim Center As Coordinate
        Center.X = SmartButton01.Width / 2
        Center.Y = SmartButton01.Height / 2

        'Data to labels on form
        XposCenterValue.Text = Center.X
        YposCenterValue.Text = Center.Y

        'To discriminate Top/Bottom/Left/Right of the mouseposition the mouse sinus and cosinus
        'will be compared with the sinus and cosinus of the topleftcormer of the button
(CheckPoint/CheckVector)

        'Setting CheckPoint data
        Dim CheckPoint As Coordinate
        Dim CheckVector As PolarCoordinate
        CheckPoint.X = 0
        CheckPoint.Y = 0
        'Calculating data for CheckVector
        CheckVector = CreatePolarCoord(CheckPoint, Center)

        'Data to labels on form
        XposCheckValue.Text = CheckPoint.X
        YposCheckValue.Text = CheckPoint.Y
        RadiusCheckVectorValue.Text = CheckVector.Radius
        CosinusCheckVectorValue.Text = CheckVector.Cosinus
        SinusCheckVectorValue.Text = CheckVector.Sinus
        ArcCheckVectorValue.Text = CheckVector.Arc

        'Calculate PolarCoordinate MouseVector
        Dim MouseVector As PolarCoordinate
        MouseVector = CreatePolarCoord(ButtonMouse, Center)

        'Data to the labels on the form
        RadiusMouseVector01Value.Text = MouseVector.Radius
        CosinusMouseVector01Value.Text = MouseVector.Cosinus
        SinusMouseVector01Value.Text = MouseVector.Sinus
        ArcMouseVector01Value.Text = MouseVector.Arc

        'Creating 5 zones and adding functionality to those zones

        'If Mouse position within a radius of 20% of the CheckVector > Center
        If MouseVector.Radius < CheckVector.Radius / 5 Then
            Button01ResultValue.Text = "Center"

            'Comparing sinus and cosinus values for determination Top/Bottom/Left/Right

        ElseIf MouseVector.Sinus < CheckVector.Sinus Then
            Button01ResultValue.Text = "Top"

        ElseIf MouseVector.Sinus > -CheckVector.Sinus Then
            Button01ResultValue.Text = "Bottom"

        ElseIf MouseVector.Cosinus < -CheckVector.Cosinus Then
            Button01ResultValue.Text = "Left"

        ElseIf MouseVector.Cosinus > CheckVector.Cosinus Then
            Button01ResultValue.Text = "Right"

        Else
            'For the most incredible cases (if cases above don't fit)
            Button01ResultValue.Text = "Nothing"
```

```
        End If

    End Sub

    'Part 5: Handling clicks on SmartButton02
    Private Sub SmartButton02_Click(sender As Object, e As EventArgs) Handles SmartButton02.Click

        'Determination position of mouse click

        'Mouse position on screen
        Dim ScreenMouse As Coordinate
        ScreenMouse.X = MousePosition.X
        ScreenMouse.Y = MousePosition.Y

        'Form topleftcorner on screen
        Dim FormTopLeftCorner As Coordinate
        FormTopLeftCorner.X = Me.Left
        FormTopLeftCorner.Y = Me.Top

        'Forms border- and barsize are calculated by using the function Frame
        Dim FormFrame As FrameSize
        FormFrame = Frame(Me)

        'Position topleftcorner of button based on clientsize form
        Dim ButtonTopLeftCorner As Coordinate
        ButtonTopLeftCorner.X = SmartButton02.Left
        ButtonTopLeftCorner.Y = SmartButton02.Top

        'Position mouse on SmartButton
        Dim ButtonMouse As Coordinate
        'Calculate relative position mouse on the SmartButton
        ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X - FormFrame.Border - ButtonTopLeftCorner.X
        ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - FormFrame.Border - FormFrame.Bar -
ButtonTopLeftCorner.Y

        'Data to labels on form
        XposMouse02Value.Text = ButtonMouse.X
        YposMouse02Value.Text = ButtonMouse.Y

        'Adding functionality to zones on SmartButton02 (adding characters to SmartText)

        'Vertical borders at (X=) 40 and 80
        'Horizontal borders a (Y=) 30, 60 and 90
        Select Case ButtonMouse.X
            Case 0 To 39
                Select Case ButtonMouse.Y
                    Case 0 To 29
                        SmartText = SmartText + "1"
                    Case 30 To 59
                        SmartText = SmartText + "4"
                    Case 60 To 89
                        SmartText = SmartText + "7"
                    Case 90 To 119
                        'Deleting all text
                        SmartText = ""
                End Select
            Case 40 To 79
                Select Case ButtonMouse.Y
                    Case 0 To 29
                        SmartText = SmartText + "2"
                    Case 30 To 59
                        SmartText = SmartText + "5"
                    Case 60 To 89
                        SmartText = SmartText + "8"
                    Case 90 To 119
                        SmartText = SmartText + "0"
                End Select
            Case 80 To 119
                Select Case ButtonMouse.Y
                    Case 0 To 29
                        SmartText = SmartText + "3"
                    Case 30 To 59
                        SmartText = SmartText + "6"
                    Case 60 To 89
```

```
                    SmartText = SmartText + "9"
                Case 90 To 119
                    'Deleting last characters (only if Smart <> "")
                    If SmartText <> "" Then
                        SmartText = Mid(SmartText, 1, Len(SmartText) - 1)
                    End If
            End Select
        End Select

        'Smarttect to label on form
        Button02ResultValue.Text = SmartText
    End Sub

    'Part 6: Handling clicks on SmartButton03

    Private Sub SmartButton03_Click(sender As Object, e As EventArgs) Handles SmartButton03.Click
        'Subroutine internal variable
        Dim TempImage As Bitmap

        'Determinating positioon of mouse click

        'Mouse position on screen
        Dim ScreenMouse As Coordinate
        ScreenMouse.X = MousePosition.X
        ScreenMouse.Y = MousePosition.Y

        'Form topleftcorner on screen
        Dim FormTopLeftCorner As Coordinate
        FormTopLeftCorner.X = Me.Left
        FormTopLeftCorner.Y = Me.Top

        'Forms border- and barsize are calculated by using the function Frame
        Dim FormFrame As FrameSize
        FormFrame = Frame(Me)

        'Position topleftcorner of button based on clientsize form
        Dim ButtonTopLeftCorner As Coordinate
        ButtonTopLeftCorner.X = SmartButton03.Left
        ButtonTopLeftCorner.Y = SmartButton03.Top

        'Position mouse on SmartButton
        Dim ButtonMouse As Coordinate
        'Calculate relative position mouse on the SmartButton
        ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X - FormFrame.Border - ButtonTopLeftCorner.X
        ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - FormFrame.Border - FormFrame.Bar -
ButtonTopLeftCorner.Y

        'Data to labels on form
        XposMouse03Value.Text = ButtonMouse.X
        YposMouse03Value.Text = ButtonMouse.Y

        'Calculating center of SmartButton03
        Dim Center As Coordinate
        Center.X = SmartButton03.Width / 2
        Center.Y = SmartButton03.Height / 2

        'To discreminate Top/Bottom/Left/Right of the mouseposition the mouse Arc will be used

        'Calculate PolarCoordinate MouseVector
        Dim MouseVector As PolarCoordinate
        MouseVector = CreatePolarCoord(ButtonMouse, Center)

        'Data to the label on the form
        ArcMouseVector03Value.Text = MouseVector.Arc

        'Discriminating 4 zones of 90 degree
        If MouseVector.Arc >= 315 Or MouseVector.Arc < 45 Then          'Top zone
            'Data to label on form
            Button03ResultValue.Text = "Top, Reset image"
            'Loading the original picture
            Button03PictureBox.Image = BaseImage.Clone

        ElseIf MouseVector.Arc >= 45 And MouseVector.Arc < 135 Then     'Right zone
            'Data to label on form
```

```
            Button03ResultValue.Text = "Right, Turn image 90 degree clockwise"
            'Turning image 90 degree
            TempImage = CType(Button03PictureBox.Image, Bitmap)
            TempImage.RotateFlip(RotateFlipType.Rotate90FlipNone)
            Button03PictureBox.Image = TempImage

        ElseIf MouseVector.Arc >= 135 And MouseVector.Arc < 225 Then    'Bottom zone
            'Data to label on form
            Button03ResultValue.Text = "Bottom, Turn image 180 degree"
            'Turning image 180 degree
            TempImage = CType(Button03PictureBox.Image, Bitmap)
            TempImage.RotateFlip(RotateFlipType.Rotate180FlipNone)
            Button03PictureBox.Image = TempImage

        ElseIf MouseVector.Arc >= 225 And MouseVector.Arc < 315 Then    'Left zone
            'Data to label on form
            Button03ResultValue.Text = "Left, Turn image 90 degree anti-clockwise"
            'Turning image 270 degree (= -90 degree)
            TempImage = CType(Button03PictureBox.Image, Bitmap)
            TempImage.RotateFlip(RotateFlipType.Rotate270FlipNone)
            Button03PictureBox.Image = TempImage

        End If

    End Sub

End Class
```