

With the release of the newest wave of **touch-enabled devices**, such as the latest [Nokia 5800 XpressMusic](#) device and the forthcoming [Nokia N97](#) device, applications can benefit from **new ways of touch-based interactions**. A typical example of these interactions is represented by **gestures**: simple finger/stylus actions that **allow the user to perform a specific task, without the need for precise interaction** (for example, identifying and pressing a specific button).

This article will explain how to implement **basic touch gestures** in Adobe [Flash Lite](#). Specifically, we'll see how to **detect both horizontal (left-to-right and right-to-left) and vertical (up-to-down and down-to-up) gestures**.

To see the video of a simple **Flash Lite photo viewer** that uses gestures to go from a photo to the next/previous photo, go to [Flash Lite gestures video](#).

## Contents

- [1 Source code](#)
  - ◆ [1.1 Step 1. Detect touch events](#)
  - ◆ [1.2 Step 2. Identify a gesture](#)
  - ◆ [1.3 Step 3. Handling the detected gesture](#)
  - ◆ [1.4 Further development](#)
- [2 Download source code](#)
- [3 Related resources](#)
- [4 Approach 2](#)
  - ◆ [4.1 Gesture logic](#)
  - ◆ [4.2 Source code](#)
    - ◇ [4.2.1 Define variables](#)
    - ◇ [4.2.2 Add a `MouseListener`](#)
    - ◇ [4.2.3 Identify the gesture](#)
    - ◇ [4.2.4 Handling the gesture](#)
  - ◆ [4.3 Further Enhancements](#)
  - ◆ [4.4 Download source code](#)

## Source code

### Step 1. Detect touch events

The first thing to do is to make it possible to **detect and handle touch-based events**. This can be done by implementing a `MouseListener` and its `onMouseDown()` and `onMouseUp()` methods. In the code that follows, **4 Number variables** will also be defined to *hold the x and y coordinates* associated with the mouse up and

## How\_to\_detect\_touch\_gestures\_in\_Flash\_Lite

down events. These will be **used to detect if the touch interaction was actually a gesture** or not.

So, in a **new and empty FLA**, add this code on the first frame:

```
var startX:Number;
var startY:Number;
var endX:Number;
var endY:Number;

var gesturesListener:Object = new Object();

gesturesListener.onMouseDown = function()
{
    _startX._xmouse;
    _startY._ymouse;
}
gesturesListener.onMouseUp = function()
{
    _endX._xmouse;
    _endY._ymouse;

    checkGesture
}
Mouse.addListener(gesturesListener);
```

The *onMouseDown()* function simply **sets the starting coordinates values**, while *onMouseUp()* also calls the *checkGesture()* function, defined below, that will **check if the touch interaction was actually a gesture**.

## Step 2. Identify a gesture

Before trying to identify a gesture, let's define some variables used to identify and define specific gesture types:

```
// minimum length of an horizontal gesture
var MIN_H_GESTURE:Number = Stage.width / 3;
// minimum length of a vertical gesture
var MIN_V_GESTURE:Number = Stage.height / 3;

// flags for each kind of gesture
var UP_TO_DOWN:Number = 1;
var DOWN_TO_UP:Number = 2;
var LEFT_TO_RIGHT:Number = 4;
var RIGHT_TO_LEFT:Number = 8;
```

The *MIN\_H\_GESTURE* and *MIN\_V\_GESTURE* variables define the **minimum horizontal and vertical distances** that must exist between the *onMouseDown()* and the *onMouseUp()* events to have a horizontal or vertical gesture, respectively.

Now, the *checkGesture()* function can be implemented by **getting the horizontal and vertical length of the touch interaction** and **comparing them with the minimum distances** defined above.

```
function checkGesture()
{
var xDelta:Number = endX - startX;
var yDelta:Number = endY - startY;
```

## How\_to\_detect\_touch\_gestures\_in\_Flash\_Lite

```
var gesture:Number = 0;

if(xDelta > MIN_H_GESTURE)
    |= LEFT_TO_RIGHT;
else if(xDelta < - MIN_H_GESTURE)
    |= RIGHT_TO_LEFT;

if(yDelta > MIN_V_GESTURE)
    |= UP_TO_DOWN;
else if(yDelta < - MIN_V_GESTURE)
    |= DOWN_TO_UP;

if(gesture > 0)
    handleGesture(gesture);
}
```

### Step 3. Handling the detected gesture

Once identified as a gesture, it's necessary to actually handle the gesture in some way. To do this, the *handleGesture()* function must be implemented so that it will check the passed argument to find out which is the specific identified gesture.

In this example, the *handleGesture()* function simply traces the kind of identified gesture(s).

```
function handleGesture(gestureFlags:Number)
{
    if(gestureFlags & LEFT_TO_RIGHT)
        trace("left to right gesture");
    if(gestureFlags & RIGHT_TO_LEFT)
        trace("right to left gesture");
    if(gestureFlags & UP_TO_DOWN)
        trace("up to down gesture");
    if(gestureFlags & DOWN_TO_UP)
        trace("down to up gesture");
}
```

### Further development

A number of improvements can be made to the code presented in this article. It is possible, for example, to do the following:

- **Detect diagonal gestures.** By checking the appropriate **gesture flags**, it is already possible to **identify mixed diagonal gestures**. So, basically, this can be accomplished by extending the *handleGesture()* method by a bit.
- Define a **maximum time** to perform a gesture. This means to basically accept a gesture only if the time elapsed to perform it is below a certain limit; it could be useful in some scenarios, to avoid detecting fake gestures.

## Download source code

To download the full source code used in this article:

- [Handling gestures in Flash Lite source code](#)

## Related resources

- [Touch input on Forum Nokia Flash Lite Developer's Library](#)
- [Using basic touch gestures in Symbian C++ on Forum Nokia Wiki](#)

## Approach 2

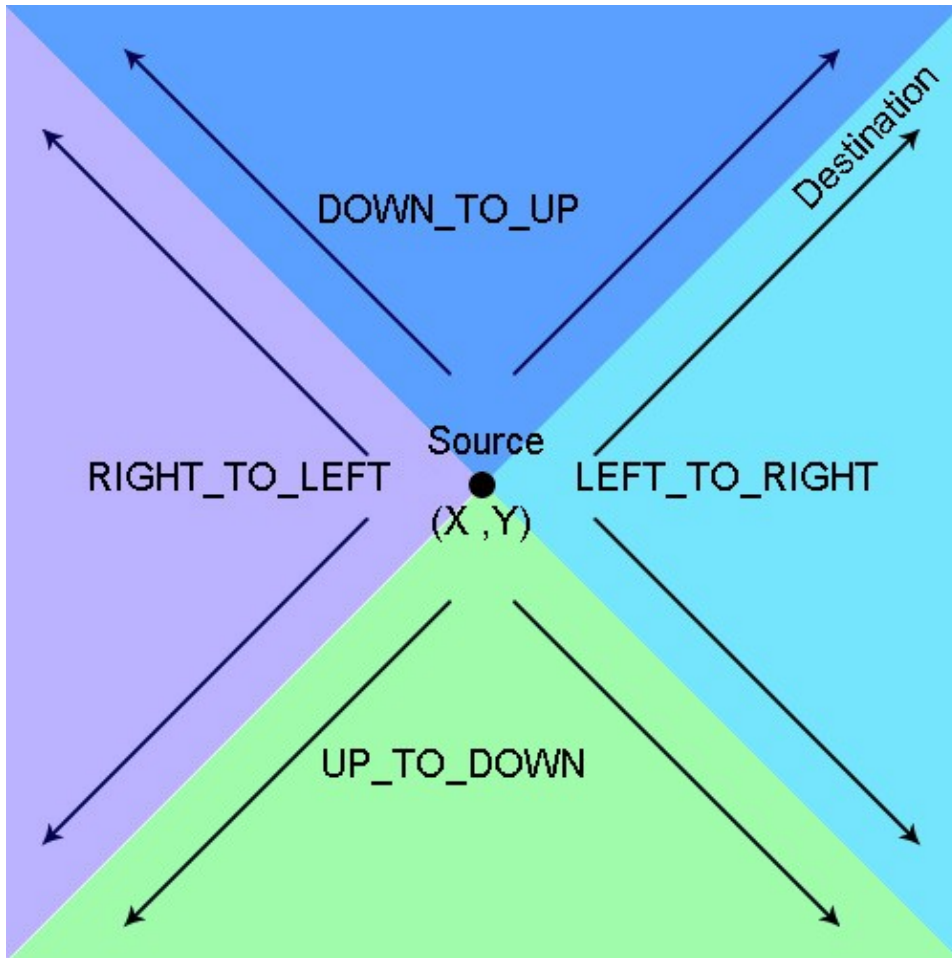
Flash Lite-supported touch-based interaction is available in Nokia's latest S60 5th Edition devices, making these interactions/gestures much more user-friendly and widely acceptable. However, there are a few flaws in the working example that can be overcome with minor changes in the logic. Here is a **practical implementation** of the gestures that can be used in Flash Lite applications.

## Gesture logic

The implementation approach described is absolutely fine but, as mentioned, there are some flaws in the working example.

- If the user drags over **both X and Y** coordinates with a different **source and destination**, the result is that **TWO** touch gestures are detected. For example, if we start at (10 , 10) and end at (70 , 80) we get **TWO** gestures ? **LEFT\_TO\_RIGHT** and **UP\_TO\_DOWN**. Ideally, there should be **only one gesture**.
- Touch-gesture detection is based on the **assumption** that when the user drags from **right to left** or vice versa, the **Y coordinate is going to be the same**, but actually if you use your finger you may end up with a **different Y value**, which is **slightly less/more** than **source Y**. However, even as an end user you want it to be clearly perceived as a ?**LEFT\_TO\_RIGHT**? interaction. The same is true for Up and Down detection.

To correct this situation, **we will modify the basics/assumptions about the interaction direction**. In the image below, we will divide the space into **four quadrants** (StartX , StartY being the reference point) and treat each quadrant for each interaction direction.



Starting from the **reference point** (StartX , StartY), if the user releases a finger/stylus in the **First quadrant**, we will treat it as a **LEFT\_TO\_RIGHT** interaction.

## Source code

### Define variables

First, we will define some variables required for the execution. There are two important variables, namely **?nMin\_Gesture\_Horizontal?** and **?nMin\_Gesture\_Vertical?**, which contain the **minimum displacement** in respective directions.

```
var nStartX:Number;
var nStartY:Number;
var nEndX:Number;
var nEndY:Number;
var double_angle;
var nMin_Gesture_Horizontal:Number = 30;
var nMin_Gesture_Vertical:Number = 30;
```

## Add a MouseListener

To access the mouse interaction we need to add a **MouseListener** and two methods, *onMouseUp()* and *onMouseDown()*. Accordingly, we will create a mouseListener object and define the *onMouseUp* and *onMouseDown* methods.

```
var oMouseListener:Object = new Object();
oMouseListener.onMouseDown = function() {
    nStartX;
    nStartY;
};
oMouseListener.onMouseUp = function() {
    nEndX;
    nEndY;
    fnDetectGesture
};
Mouse.addListener(oMouseListener);
```

## Identify the gesture

In order to detect the gesture, a function *fnDetectGesture* is called. In this function we are implementing the logic described above. We access the **Source** and the **Destination X, Y** coordinates and treat **them as two ends of a line**. Using the Math formula, we calculate the **angle formed** by this line. If the angle happens to be within the range of **-45 to 45**, it is treated as a **LEFT\_TO\_RIGHT** interaction. Similarly, if the angle is within the range of **-45 to -135**, it is treated as a **DOWN\_TO\_UP** interaction. In the case of **-135 to -179**, the interaction is **RIGHT\_TO\_LEFT**. In the case of **45 to 135**, the interaction is treated as **UP\_TO\_DOWN**.

```
function fnDetectGesture() {
var nDiffX:Number = nEndX-nStartX;
var nDiffY:Number = nEndY-nStartY;
    double_angle=Math.atan2(nEndY-nStartY, nEndX-nStartX)*180/Math.PI;
if (double_angle == 0) {
if (nDiffX>=nMin_Gesture_Horizontal) {
        ("LEFT_TO_RIGHT")Detected
}
} else if (double_angle == -90) {
if (nDiffY<=-nMin_Gesture_Vertical) {
        ("DOWN_TO_UP")Detected
}
} else if (double_angle == 180) {
if (nDiffX<=-nMin_Gesture_Horizontal) {
        ("RIGHT_TO_LEFT")Detected
}
} else if (double_angle == 90) {
if (nDiffY>=nMin_Gesture_Vertical) {
        ("UP_TO_DOWN")Detected
}
} else if (double_angle>=(-45) && double_angle<45) {
if (nDiffX>=nMin_Gesture_Horizontal) {
        ("LEFT_TO_RIGHT")Detected
}
} else if (double_angle<(-45) && double_angle>=(-135)) {
if (nDiffY<=-nMin_Gesture_Vertical) {
        ("DOWN_TO_UP")Detected
}
} else if ((double_angle<(-135) && double_angle>=(-179)) || (double_angle>=(135) && double_angle<
```

## How\_to\_detect\_touch\_gestures\_in\_Flash\_Lite

```
if (nDiffX<=-nMin_Gesture_Horizontal) {
    ("RIGHTTOLEFTGestureDetected")
}
} else if (double_angle<(135) && double_angle>=(45)) {
if (nDiffY>=nMin_Gesture_Vertical) {
    ("UP_TO_DOWN GestureDetected")
}
}
}
```

## Handling the gesture

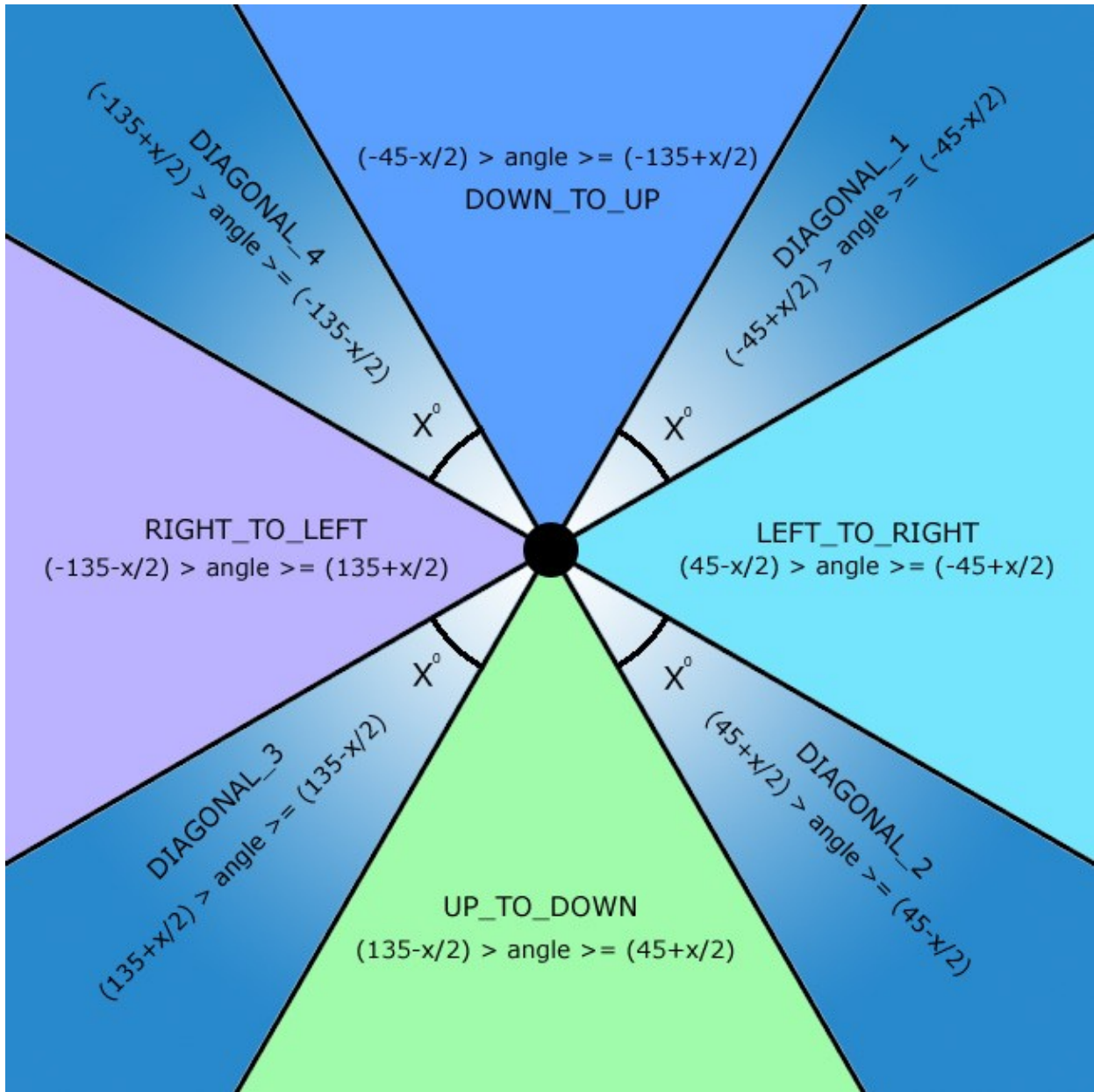
Once the gesture has been detected, the function *fnGestureDetected?* is called and the **interaction direction** is passed as a **parameter**. You can implement the project-specific requirement in this function.

```
function fnGestureDetected(sGestureDirection:String) {
trace("Gesture Direction:: "+sGestureDirection);
    text.txt.sGestureDirection;
//Implement the project logic here;
}
```

## Further Enhancements

- The gesture detection can be **restricted/limited** to certain time period. The gesture interaction should be allowed only if the interaction is completed within a defined time period.
- Even a **threshold displacement** can be set in order to recognise the interaction as a touch gesture. Eg. In order for a Left-to-Right/Right-to-Left gesture a minimum of **20/30(threshold) pixels** can be set. If user releases the screen without travelling this distance, the gesture will be ignored. The same hold for Up-to-Down and Down-to-Up gesture.
- The gesture logic can be extended to support the **diagonal interaction**. We can define an angle of **30/40 degrees** at the middle of each quadrant. The gesture in this region will be treated as diagonal gesture.

The following image will clear the logic for **Diagonal gestures**.



Here **X** is the angle which will be used for diagonal gesture. Once you define the value of **X**, the *fnDetectGesture* function can be changed to accommodate the new diagonal gestures. The image shows the **range of angle** which maps to the respective gesture.

## Download source code

The source code is available at [Handling gestures in Flash Lite:Practical Approach](#).