* 1. 

Hands-On Lab

Multitouch Gestures with MFC - Native

* 1. Lab version: 1.0.0
  2. Last updated: 12/23/2010

**CONTENTS**

[Overview 3](#_Toc243306797)

[Exercise 1: Build a Multitouch Gesture Application 6](#_Toc243306798)

[Task 1 – Create the MFC application using the Application Wizard 6](#_Toc243306799)

[Task 2 – Test the Existence and Readiness of Multitouch Hardware 11](#_Toc243306800)

[Task 3 – Add the Drawing Object Source and Header Files to the Project, and Draw the Rectangle 13](#_Toc243306801)

[Task 4 – Touch Now! 15](#_Toc243306802)

[Task 5 – There Is a Bug! 18](#_Toc243306803)

[Summary 20](#_Toc243306804)

Overview

* 1. Windows 7 gives users the ability to manage applications with the touch of their fingers, without the need of a stylus. This new gesture capability allows multiple input events to occur at the same time from different pointing locations, and it enables complex scenarios, such as managing applications with ten fingers or with multiple simultaneous users. However, to pull this off, we have to adapt our application's user interface and behavior to support this new input model.

# Objectives

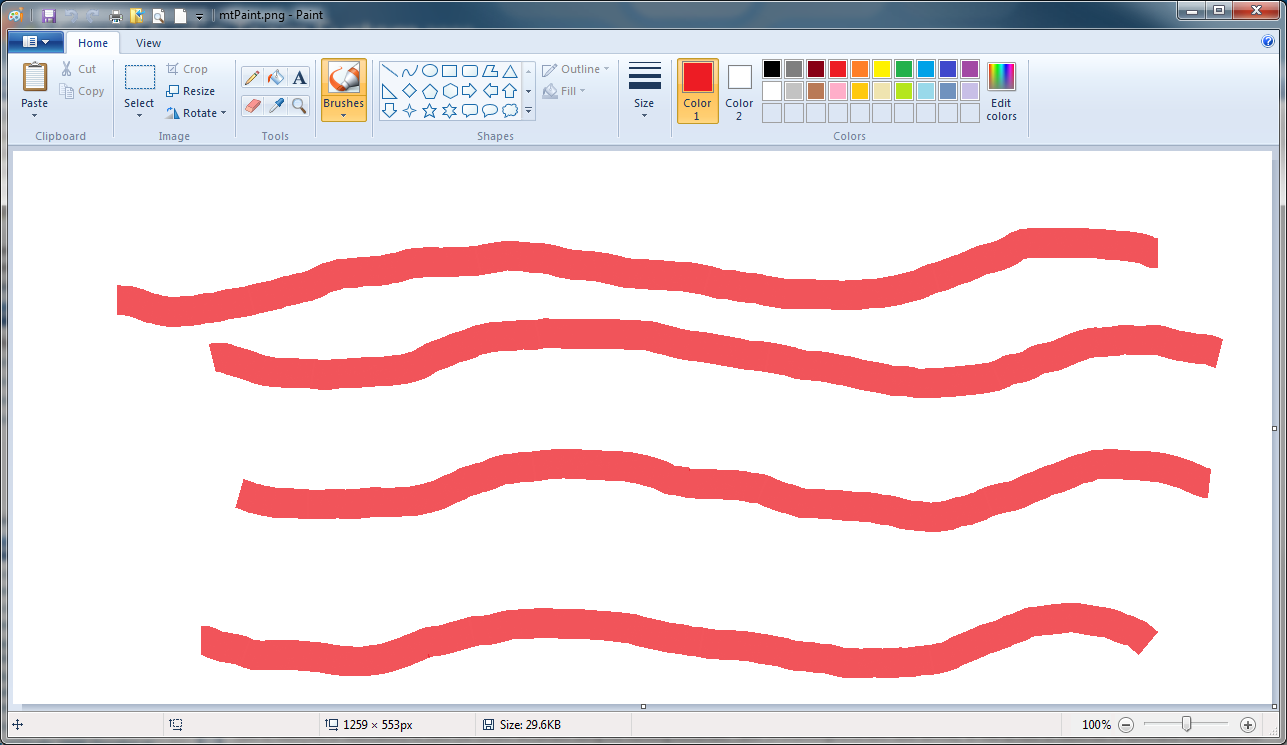
* 1. In this Hands-On Lab, you will learn how to manage gesture events, including:
  + Understanding the implications of manipulating an object with gesture events
  + Checking for Multitouch hardware existence and readiness
  + Extracting information from the gesture Windows Message

# System Requirements

* 1. You must have the following items to complete this lab:
  + Microsoft Visual Studio 2010
  + Windows 7
  + A multitouch hardware device ☺
  1. **Introduction**

To create a Multitouch driven application you should design your application’s user interface with touch capabilities in mind. For example, the controls you use should be relatively large to support a finger rather than a mouse or stylus. Beyond that you have 3 different approaches you can consider:

1. Rely upon touch capabilities already a part of existing Win32 controls. For example, Win32 controls already support some touch capabilities such as scrolling without any extra coding in your application. Go ahead and try to scroll the document you are reading using your fingers.
2. Read and handle low-level touch events and create your own sophisticated touch handling interface. This is the approach used by complex multiple sliders that users can operate simultaneously or by applications like “Piano” or MS Paint. For example, run MS Paint, select a drawing tool from the gallery and draw with four of your fingers (if the hardware permits):

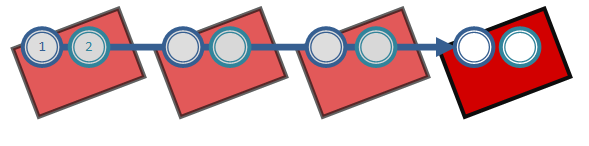


1. Use the new Multitouch gestures to get touch events to your applications to support custom actions like zoom, rotate, and translate without the need to read and manipulate raw touch events. This is the approach used in this Hands-On Lab.

# About the Multitouch Gesture Demo

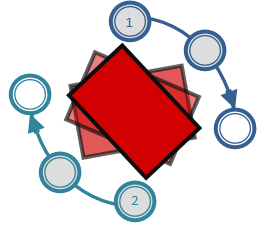
The Multitouch Gesture demo presents a simple window containing a rectangle that can be modified using Multitouch gestures. Inside the ***Sources\MFC\_GestureDemoSource*** folder, you can find a ***Starter*** folder that contains files you will need during this lab and a finished version of the lab which is located in the ***Final*** folder.

This application responds to Multitouch gesture input by interacting with a painted rectangle. The rectangle responds to the following gestures:

* 1. **Translate**
  2. To translate the image,place one or two fingers in the application window and drag in the direction you want. Make sure to leave a little space between the two fingers so that the touch interface sees them as separate contact points.

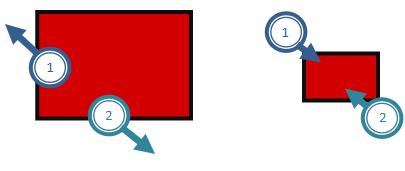
**Rotate**

Touch the rectangle with two fingers and turn fingers in a circle**.**

* 1. 

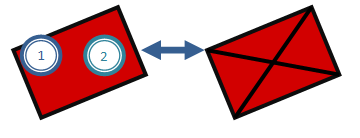
**Zoom**

Touch two fingers and move them farther apart or closer together.

* 1. 

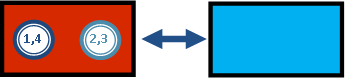
**Two Finger Tap**

Tap once with both fingers to toggle diagonal lines on or off within the red rectangle.

* 1. 

**Finger Roll**

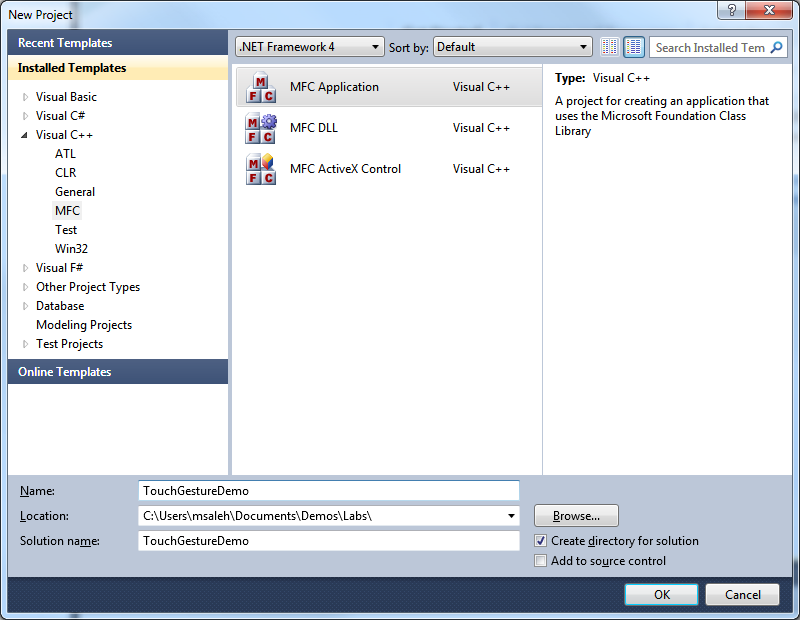
Press and hold one finger, then tap with the other and then remove the first one to change the color of the rectangle.

* 1. 

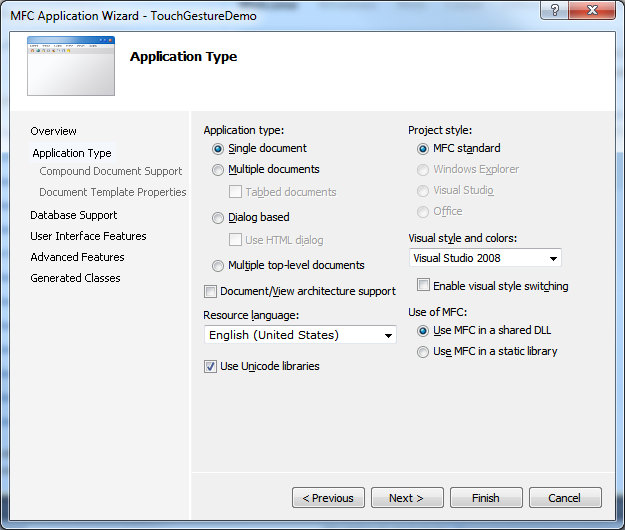
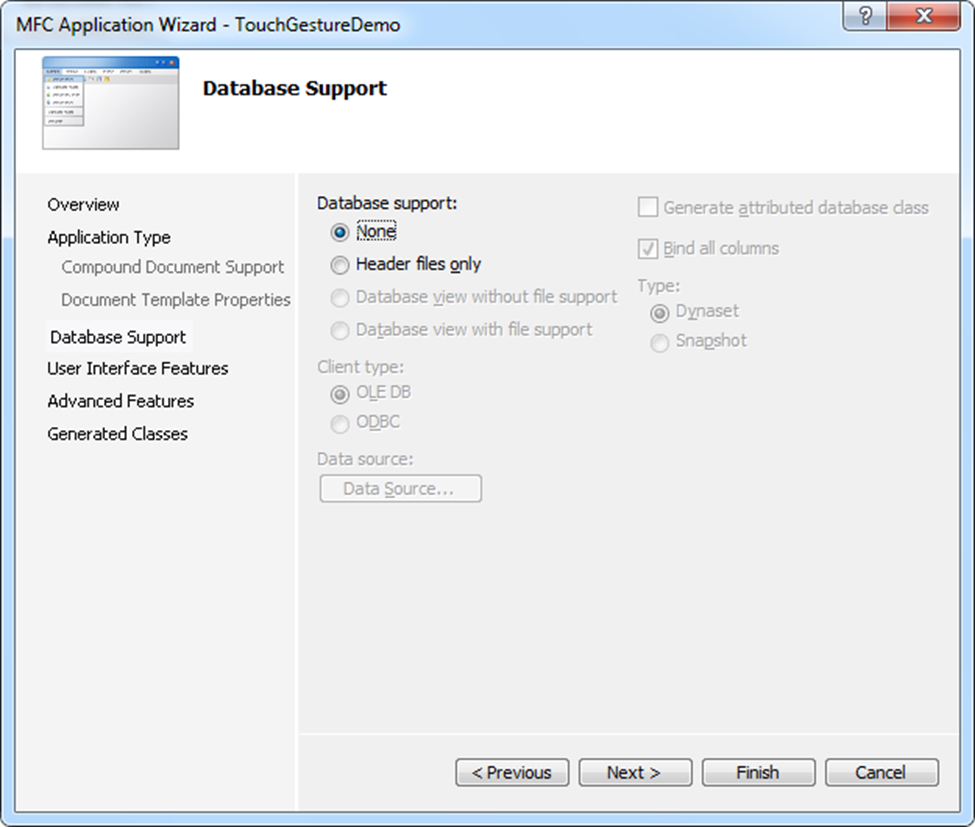
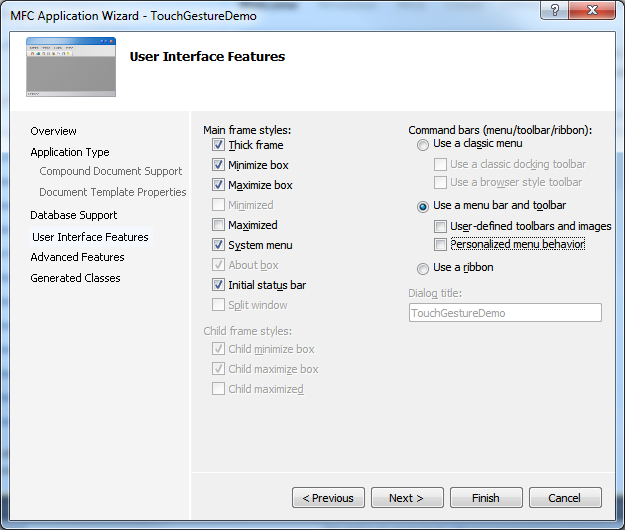
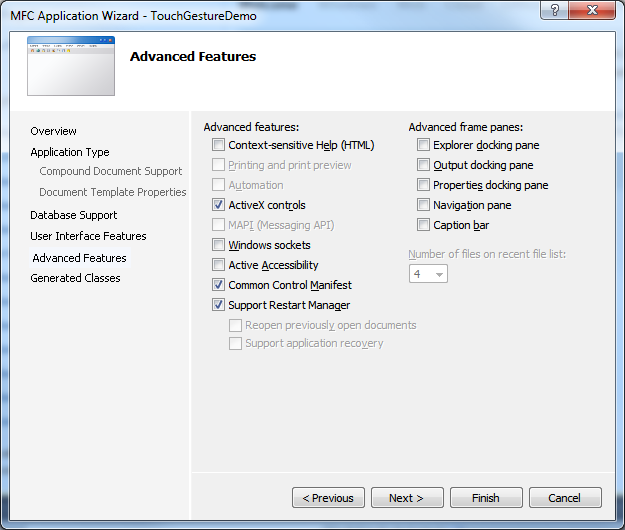
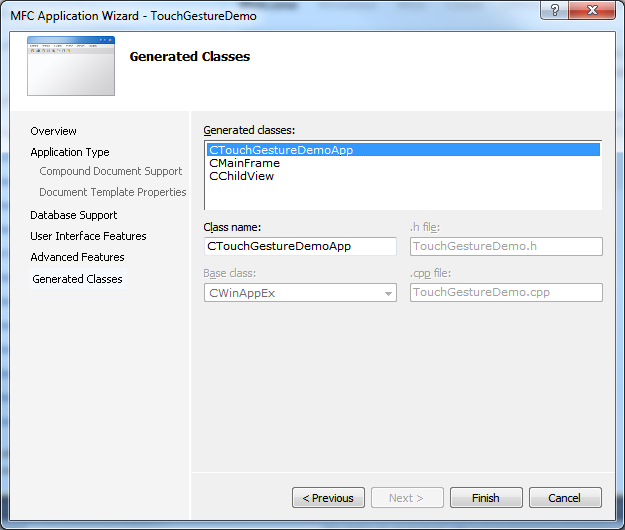
Exercise 1: Build a Multitouch Gesture Application

Task 1 – Create the MFC application using the Application Wizard

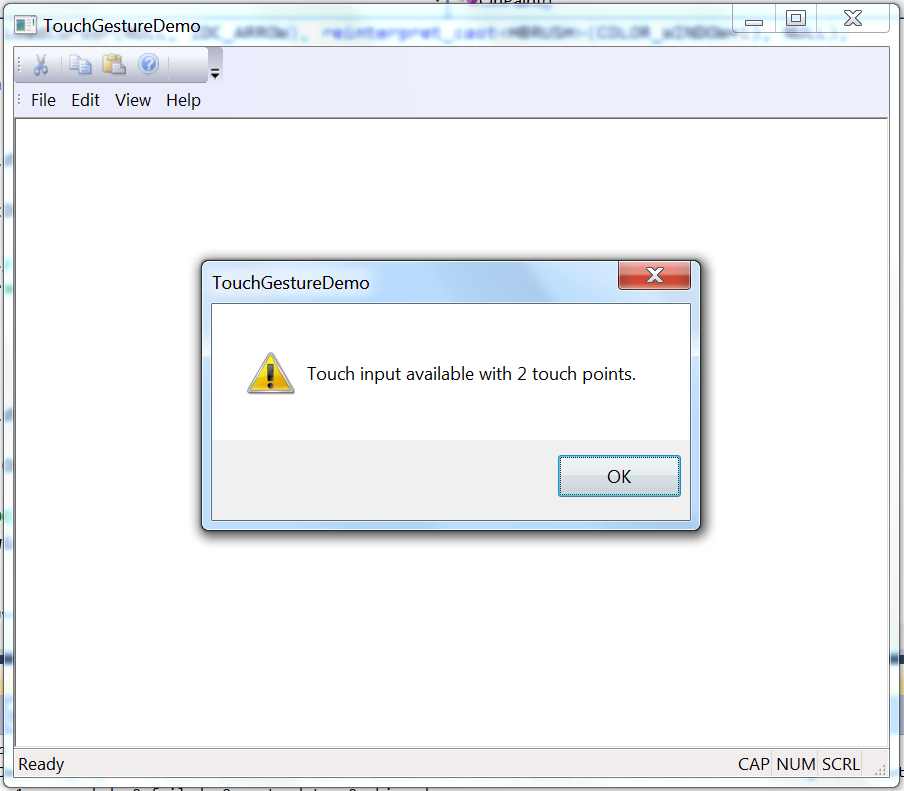
* 1. Start Visual Studio 2010.
  2. Create a new MFC application project with the name ***TouchGestureDemo*** and press **OK**.



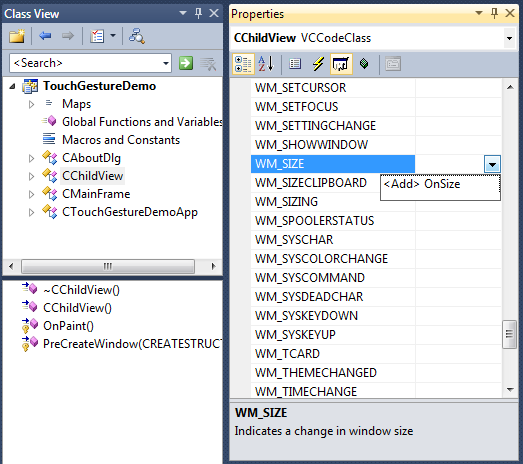
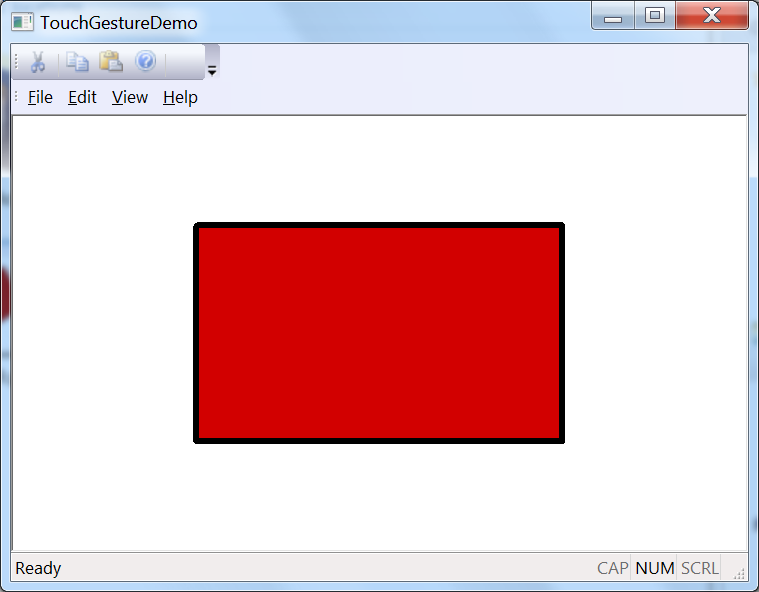
* 1. In the Application Type dialog of the MFC Application Wizard, select Single Document. To keep the application simple, select (or deselect) the options in the various dialogs of the MFC Application Wizard based on the screenshots below and then press **Next >** after each dialog.

* + 1. 
    2. 
    3. 
    4. 
  1. Continue clicking **Next** until you finally hit **Finish** on the **Generated Classes** wizard dialog:
     1. 

Task 2 – Test the Existence and Readiness of Multitouch Hardware

* 1. The application that we are building requires touch enabled hardware, so we need to check for this in the application.
  2. In TouchGestureDemo.cpp, add the following code at the end of CTouchGestureDemoApp::InitInstance():
     1. C++
     2. BYTE digitizerStatus = (BYTE) GetSystemMetrics(SM\_DIGITIZER);
     3. if ((digitizerStatus & (0x80 + 0x40)) == 0) //Stack Ready + MultiTouch
     4. {
     5. AfxMessageBox(L"No touch input is currently available.");
     6. return FALSE;
     7. }
     8. BYTE nInputs = (BYTE) GetSystemMetrics(SM\_MAXIMUMTOUCHES);
     9. CString str;
     10. str.Format(L"Touch input available with %d touch points.", nInputs);
     11. AfxMessageBox(str);
     12. return TRUE;
  3. You can see that besides checking for touch availability, we also find out the number of touch inputs that the hardware support.
  4. Build and run.
  5. Depending on the number of touch inputs you have on your machine, you should see output similar to the following screenshot:
     1. 

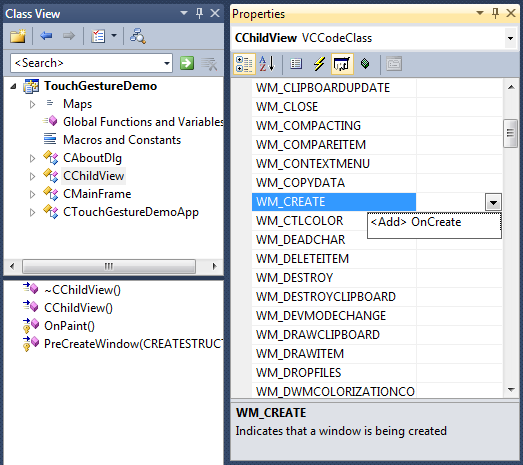
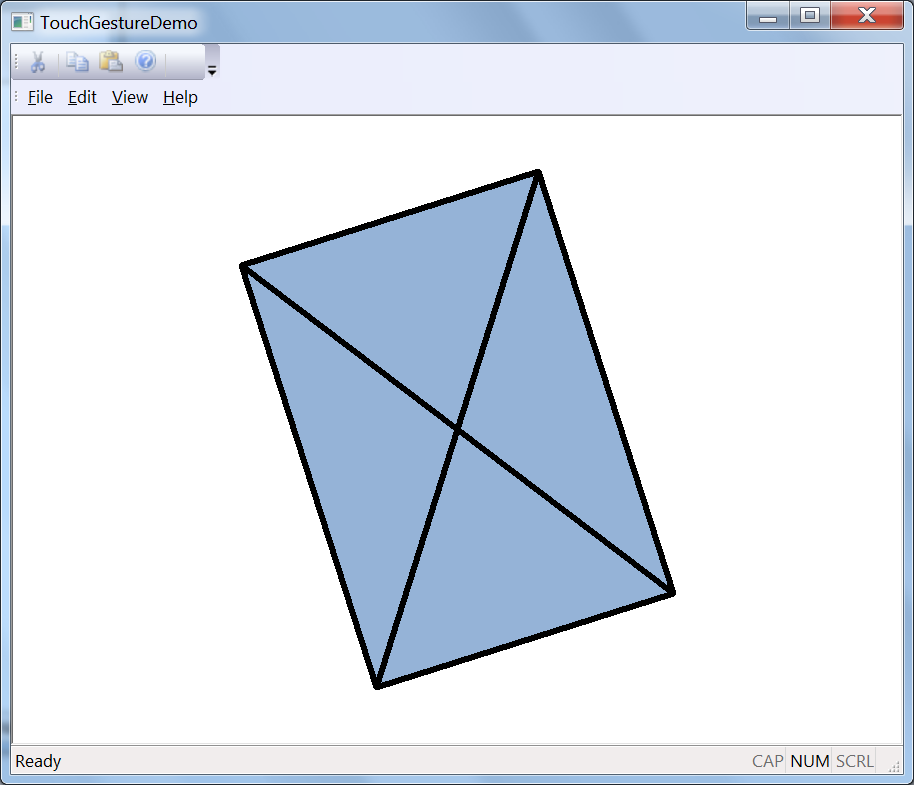
Task 3 – Add the Drawing Object Source and Header Files to the Project, and Draw the Rectangle

* 1. In the Starter folder, you will find two files: DrawingObject.h and DrawingObject.cpp. Copy them to the project folder and use “Add🡪 Existing item…” to add them to the project.
  2. Add an #include DrawingObject.h line near the end of stdafx.h header file, just after #include <afxcontrolbars.h>:
     1. C++
     2. #include "DrawingObject.h"
  3. Add the following *protected* member variable definition to *ChildView.h*. This is the main drawing object:
     1. C++
     2. // The drawing object
     3. CDrawingObject m\_drawingObject;
  4. In addition, add the following *protected* member variables to *ChildView.h*. These are needed to support positioning and sizing of the drawing object:
     1. C++
     2. // Needed for drawing object position and size calculations
     3. double m\_dblZoomRatioStart;
     4. double m\_dblZoomRatioTotal;
     5. CPoint m\_ptCenter;
  5. To draw the rectangle, add the following line in ChildView.cpp at the end of CChildCiew::OnPaint():
     1. C++
     2. m\_drawingObject.Paint(&dc);
  6. When the window is resized, we want to reset the drawing rectangle to its default location and size (making it easy to find it, if it moves outside the client area boundaries), so let’s add a WM\_SIZE message handler.
  7. Switch to the Class View (Select the Class View option under the View Menu) and Select the *CChildView* class.
  8. In the Properties page, go to the **Messages** property sheet (by pressing  icon) and navigate to WM\_SIZE, then add the OnSize() message handler from the drop down box:
     1. 
  9. Inside the CChildView::OnSize() handler (after the TODO comment) append the following code to reset the drawing object:
     1. C++
     2. m\_drawingObject.ResetObject(cx, cy);
  10. Build and run.
  11. After responding to the initial message about the availability of touch inputs, a red rectangle should appear in the middle of the window:
      1. 

Task 4 – Touch Now!

* 1. It’s time to start! By default a touch-enabled system provides WM\_GESTURE messages to the target window. This is somewhat similar to mouse and keyboard messages. The system consumes the low-level touch input events and calculates the resulting gesture for us. MFC provides a number of message handler overrides that can receive each of the gesture types and each returns a Boolean value. If a gesture input is processed by the application, the corresponding override should return TRUE; otherwise it returns FALSE.
  2. Let’s move! Add the following five protected method definitions to *ChildView.h*:
     1. C++
     2. // Overrides
     3. protected:
     5. // Gesture handlers
     6. virtual BOOL OnGestureZoom(CPoint ptCenter, long lDelta);
     7. virtual BOOL OnGesturePan(CPoint ptFrom, CPoint ptTo);
     8. virtual BOOL OnGestureRotate(CPoint ptCenter, double dblAngle);
     9. virtual BOOL OnGesturePressAndTap(CPoint ptFirstFinger, long lDelta);
     10. virtual BOOL OnGestureTwoFingerTap(CPoint ptCenter);
  3. Starting with the Pan Gesture handler, add this code to *ChildView.cpp*:
     1. C++
     2. BOOL CChildView::OnGesturePan(CPoint ptFrom, CPoint ptTo)
     3. {
     4. int dx = ptTo.x - ptFrom.x;
     5. int dy = ptTo.y - ptFrom.y;
     6. if (dx != 0 || dy != 0)
     7. {
     8. m\_drawingObject.Move(dx, dy);
     9. RedrawWindow();
     10. }
     11. return TRUE;
     12. }
  4. Now, the Zoom Gesture implementation, add this code to *ChildView.cpp*:
     1. C++
     2. BOOL CChildView::OnGestureZoom(CPoint ptCenter, long lDelta)
     3. {
     4. if ((m\_pCurrentGestureInfo->dwFlags & GF\_BEGIN) == GF\_BEGIN)
     5. {
     6. m\_dblZoomRatioStart = m\_dblZoomRatioTotal = lDelta;
     7. }
     8. else if (lDelta != 0)
     9. {
     10. m\_dblZoomRatioTotal += lDelta;
     11. double zoomFactor = (double)m\_dblZoomRatioTotal / m\_dblZoomRatioStart;
     13. m\_drawingObject.Zoom(zoomFactor, ptCenter.x, ptCenter.y);
     15. m\_dblZoomRatioStart = m\_dblZoomRatioTotal;
     16. RedrawWindow();
     17. }
     18. return TRUE;
     19. }
  5. Add the Rotate Gesture handler in *ChildView.cpp*:
     1. C++
     2. BOOL CChildView::OnGestureRotate(CPoint ptCenter, double dblAngle)
     3. {
     4. if ((m\_pCurrentGestureInfo->dwFlags & GF\_BEGIN) == GF\_BEGIN)
     5. {
     6. // Make the first center, the rotating one
     7. m\_ptCenter = ptCenter;
     8. }
     9. else if (dblAngle != 0.)
     10. {
     11. m\_drawingObject.Rotate(dblAngle \* PI / 100.0, m\_ptCenter.x, m\_ptCenter.y);
     12. RedrawWindow();
     13. }
     14. return TRUE;
     15. }
  6. Similarly, add the handlers for the “Press and Tap” and “Two Finger Taps” Gestures in *ChildView.cpp*:
     1. C++
     2. BOOL CChildView::OnGesturePressAndTap(CPoint ptFirstFinger, long lDelta)
     3. {
     4. if ((m\_pCurrentGestureInfo->dwFlags & GF\_BEGIN) != 0)
     5. {
     6. m\_drawingObject.ShiftColor();
     7. RedrawWindow();
     8. }
     9. return TRUE;
     10. }
     11. BOOL CChildView::OnGestureTwoFingerTap(CPoint ptCenter)
     12. {
     13. m\_drawingObject.TogleDrawDiagonals();
     14. RedrawWindow();
     15. return TRUE;
     16. }
  7. Build and run.
  8. Try to move the rectangle with two fingers; you can see that it follows your fingers’ movement.
  9. Try to move the rectangle without touching it; instead touch the screen in an empty area of the Window. It moves! We didn’t perform “hit testing” to check if the touch location is inside the rectangles’ boundaries. We respond to any gesture within the whole client area.
  10. Try using the remaining gestures as well: Rotate, Zoom, Two Finger Tap and Finger Roll.

Task 5 – There Is a Bug!

* 1. Try to rotate the rectangle. What happened? By default, a Window receives all gestures except rotation. However, we can configure the touch engine to supply any gestures we want. MFC provides a class that can help, CGestureConfig, which allows you to customize Windows gesture features such as zoom, pan or rotate.
  2. Add this member variable to *ChildView.h*:
     1. C++
     2. // Fields
     3. protected:
     4. // Holds gesture configuration
     5. CGestureConfig m\_gestureConfig;
  3. Switch to the **Class View** and Select the CChildView class. In the Properties page, go to the **Messages** property sheet and navigate to WM\_CREATE, then add the OnCreate() message handler from the drop down box:
     1. 
  4. Inside the CChildView::OnCreate() handler (after the TODO comment) append the following code to enable receiving the rotation gesture:
     1. C++
     2. GetGestureConfig(&m\_gestureConfig);
     3. // Only rotation is not enabled by default
     4. m\_gestureConfig.EnableRotate();
     5. SetGestureConfig(&m\_gestureConfig);
  5. Build and run.
  6. Try to rotate the rectangle. It works! Well done!
  7. Here is a screenshot for the demo in action, after “Rotation”, “Press and Tap” and “Two Finger Taps” gestures have all been applied:
     1. 

Summary

* 1. In this lab, you have learned how to consume touch gesture messages to manipulate an object on the screen through MFC. You’ve seen how to determine the availability of Multitouch enabled hardware and how easy it is to extract gesture message information and act on it.
  2. Enjoy and keep in touch!