Handling touch input in Windows* 8 Applications

With the growing number of devices supporting touch, handling the touch interaction method in your applications is more and more important.

Windows* 8 standard controls and templates already handle touch perfectly, so if you only use them, you don't really need to know what is under the hood.

But if you have a desktop application or if you want to build some custom controls (for a game, for example), then you need to know how to handle touch properly.

Applications that worked on Windows 7 will survive:

- If your application only handles mouse events, touch events will fall back to these (you will only miss the hover event)
- If your application handles Windows 7 touch events (WM_TOUCH / WM_GESTURE APIs) that's fine, these APIs are still available.

In both cases, you should consider improving your application by using the new Windows 8 input APIs, as they will help you provide consistency with other applications and a better overall user experience.

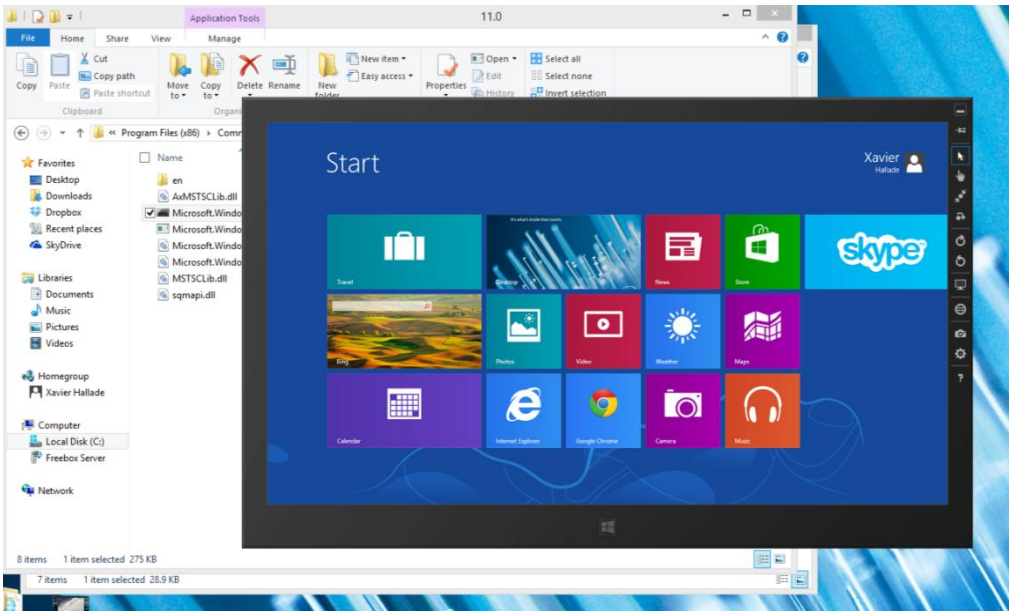
If your application was not designed to handle touch, you may also have to make modifications to the user interface for it to be more touch friendly: larger controls, less clutter, etc. You can find more advice on touch-compatible interface design here: Designing for Ultrabook™ Devices and Touch-enabled Desktop Applications

Developing for touch without a touch-enabled device

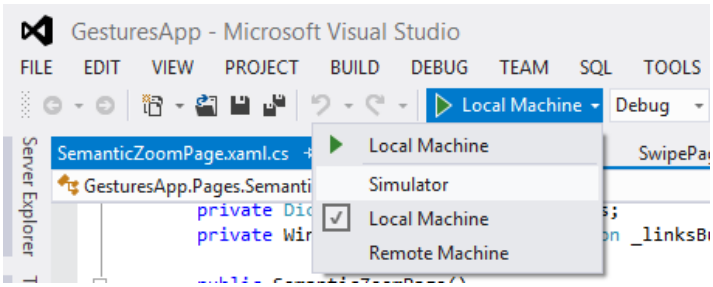
If you don't have any touch-enabled devices, you can still try your desktop or Windows Store application inside the Windows 8 simulator shipped with Visual Studio* 2012 on Windows 8.

That tool can be found in "C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Simulator\11.0\ Microsoft.Windows.Simulator.exe."

You can launch any application inside it, like from your regular session:



Windows 8 Store apps can also directly be launched inside the simulator from Visual Studio:

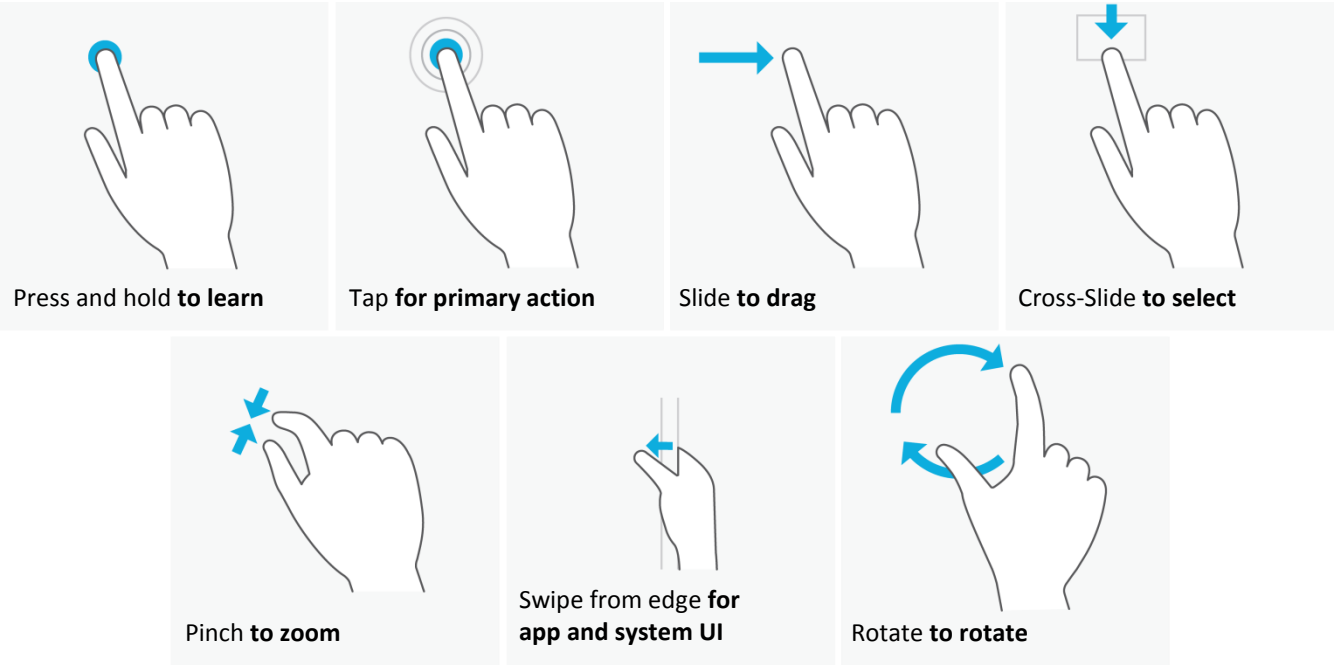


Then you can choose the interaction mode on the right side of the simulator. The default is mouse, but the other three are the interesting ones: single touch, pinch to zoom, and rotation modes.

Dual touch modes work using mouse left click + wheel to zoom or rotate.

Windows 8 OS touch interactions

To provide a good user experience, your app needs to at least be consistent with OS-wide gestures:

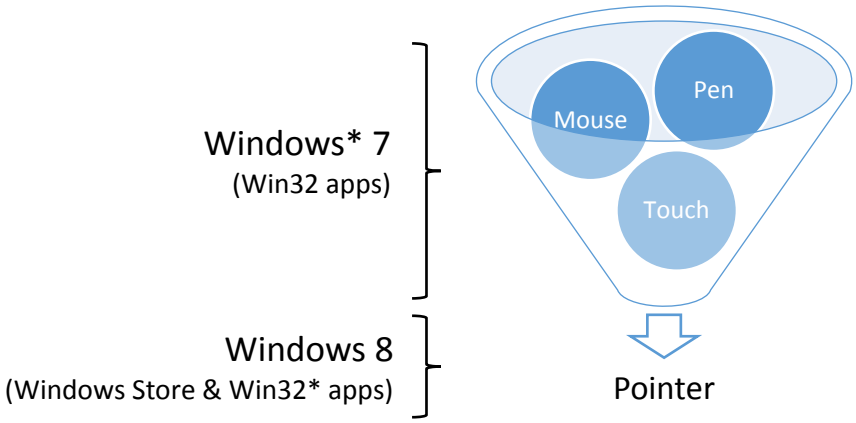


source: //build/ 2011 - APP-391T

Pointer input concept

Overview

Handling more than one input method in applications can be tedious. Fortunately, Microsoft introduced the unified “Pointer” input with Windows 8:



The Pointer input abstracts/includes the Mouse/Pen/Touch input methods. Coding for Pointers allows you to code only once to handle all these input methods.

Pointer events are the most basic thing you will be able to get. You can get these on any Windows 8 XAML UI Element as well as on ICoreWindow. On the HTML5 side, these events are available, but with slightly different names, prefixed with MS.: MSPointerDown, MSPointerMove, MSPointerUp.

The Win32* equivalents of pointer events are WM_POINTERXXX messages, you can receive these in your Win32 window callback function. For Win32 applications, WM_POINTERXXX messages don't include mouse messages by default. You first need to call EnableMouseInPointer(true) to get the completely unified pointer messages.

Pointer Events (XAML / JS / Win32)

- PointerPressed / MSPointerDown / WM_POINTERDOWN
- PointerMoved / MSPointerMove / WM_POINTERUPDATE
- PointerReleased / MSPointerUp / WM_POINTERUP
- PointerEntered / MSPointerHover / WM_POINTERENTER: hover/over state started.
- PointerExited / MSPointerOut / WM_POINTERLEAVE: hover/over state ended.
- PointerCanceled / MSPointerCancel /: Pointer lost (touch canceled by pen coming into range, too many contacts, user log off…).
- PointerCaptureLost / MSLostPointerCapture / WM_POINTERCAPTURECHANGED
- PointerWheelChanged / - / WM_POINTER(H)WHEEL: Standard or horizontal wheel state changed of at least one detent since last event.

Higher level objects like XAML UI elements directly provide Gesture and Manipulation events:

Gesture Events

- Tapped
- DoubleTapped
- RightTapped: Long Tap, equivalent to a right click
- Holding / MSGestureHold: Fired before Pointer release.

Manipulation Events

- ManipulationStarting
- ManipulationStarted
- ManipulationDelta
- ManipulationCompleted
- ManipulationInertiaStarting

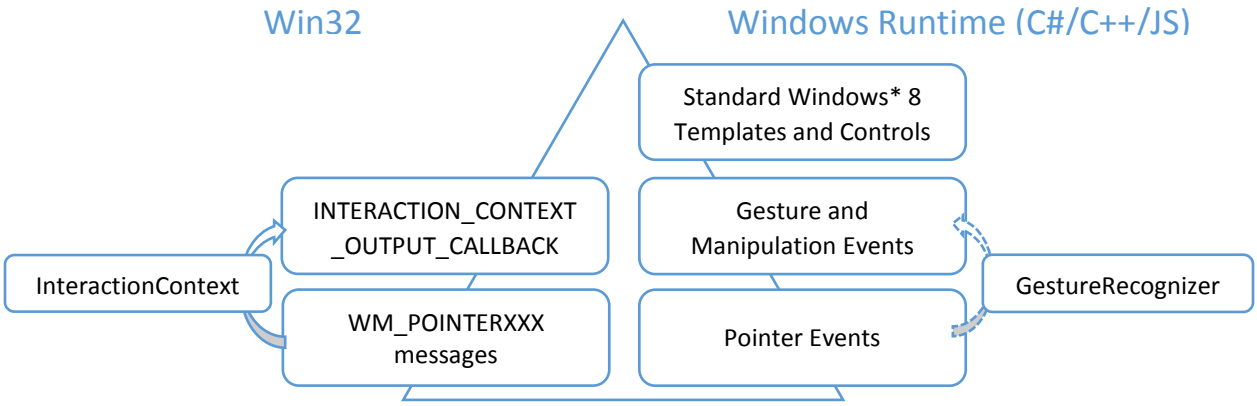
Manipulation events can provide information on zooming, rotating, and panning, and they also provide inertia. They can be configured using ManipulationMode to toggle inertia or allow only some interactions/add constraints (like rails for translation on X/Y).

In Windows 8 Store apps using HTML5/JavaScript*, you can use a WinRT GestureRecognizer to get these events.

If you want your application to also work inside IE10 (i.e., without WinJS), you can use an MSGesture object. It will trigger these events that are equivalent to manipulation events: MSGestureStart, MSGestureEnd, MSInertiaStart, and MSManipulationStateChanged and these gesture events: MSGestureTap, MSGestureHold.

Note: Manipulation events are also available for C#/WPF 4.x desktop apps like they were under Windows 7, without ManipulationMode.

Windows 8 APIs overview



If the object you are dealing with doesn't trigger gesture events, you can send the pointer events to a GestureRecognizer. The GestureRecognizer will trigger selected gestures and manipulations events as well as **dragging** and **cross-sliding events**.

InteractionContext is the Win32 equivalent of the GestureRecognizer from the Windows Runtime API. The Interaction Context object triggers INTERACTION_CONTEXT_OUTPUT_CALLBACK associated with the different gestures/manipulations.

Something else you can integrate in your touch-enabled application is an InkRecognizer. It allows you to recognize handwriting from desktop apps as well as from Windows 8 Store apps.

You can also inject touch events from desktop applications using the Touch Injection API.

Summary of ways to integrate touch within your application

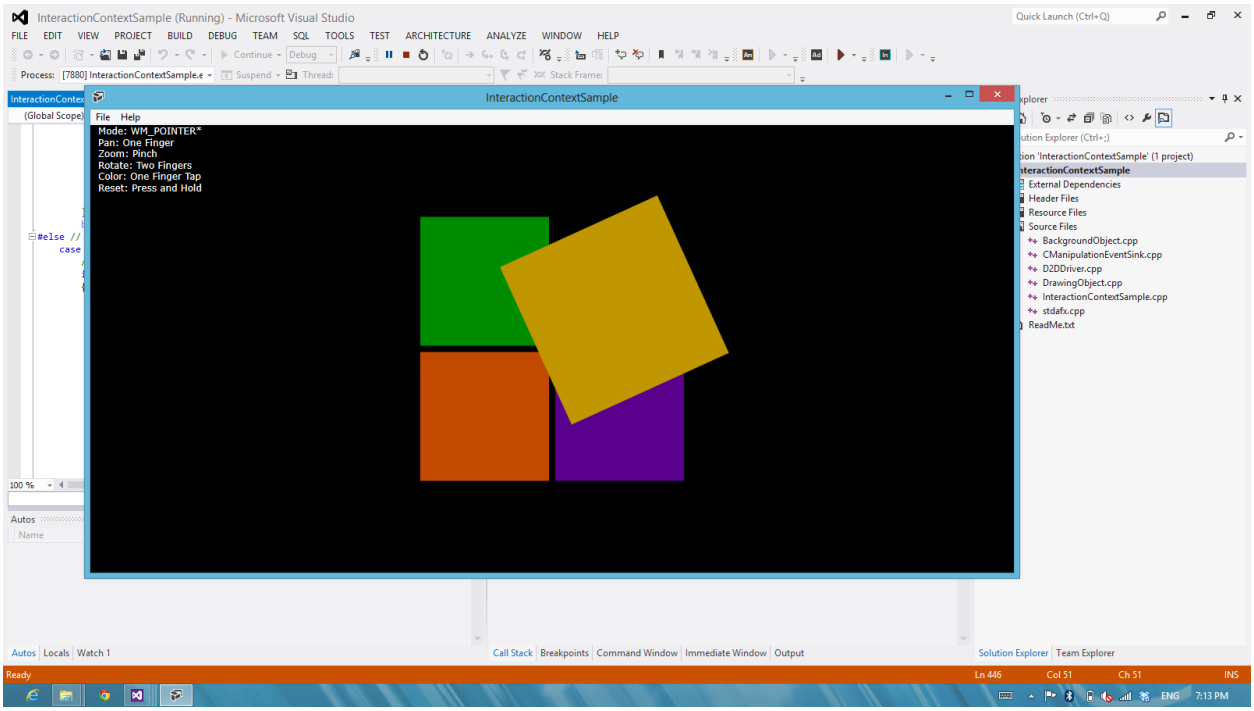
Here is a wrap-up of the ways to handle touch input depending on the objects / application types you are dealing with:

Application Type	Object	Method
Win32*	Window	WM_POINTERXXX WM_TOUCHHITTESTING messages
JS/HTML (Windows* Store or IE10)	HTML elements	MSPointerXXX events MSGestureXXX events (needs a

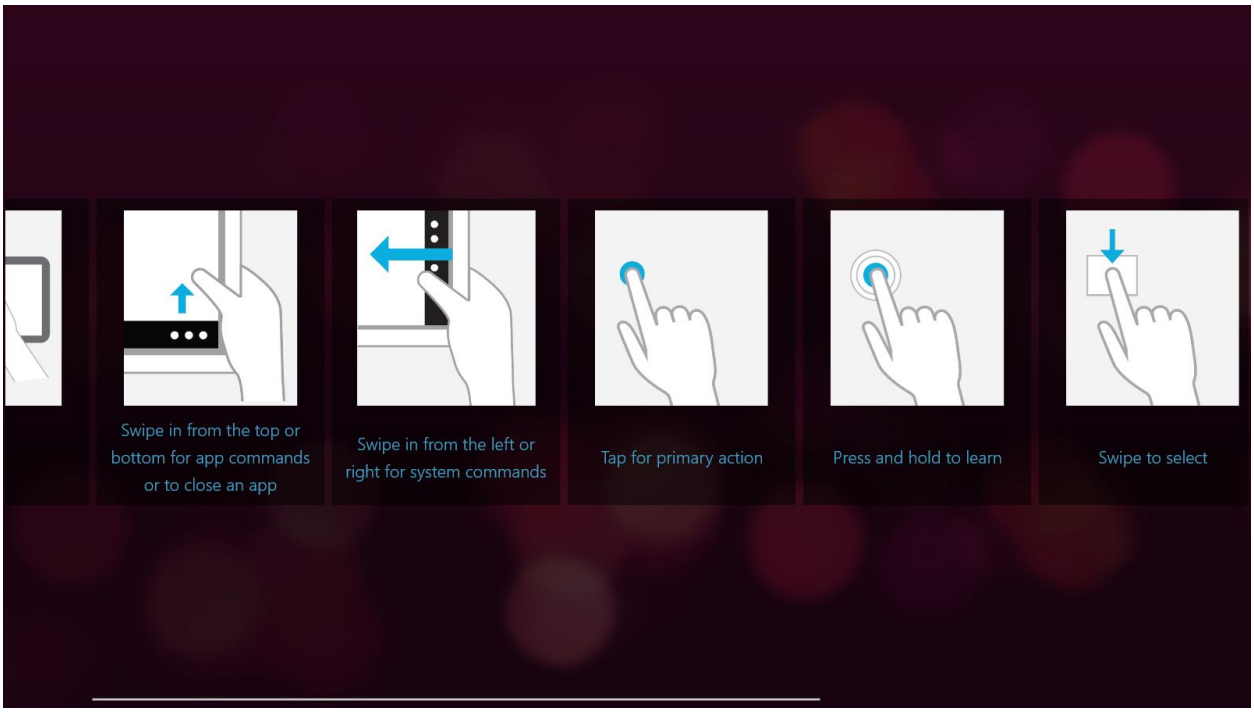
		MSGesture object to be triggered)
Windows Store – C#/C++	ICoreWindow	PointerXXX TouchHitTesting events
Windows Store - XAML	UIElement	PointerXXX ManipulationXXX XXXTapped events
Windows Store – XAML	Control	OnPointerXXX(…) OnManipulationXXX(…) OnXXXTapped(…) delegates
Windows Store (XAML & JS/HTML)	ListView, FlipView, ScrollViewer, etc.	Seamless

Sample codes

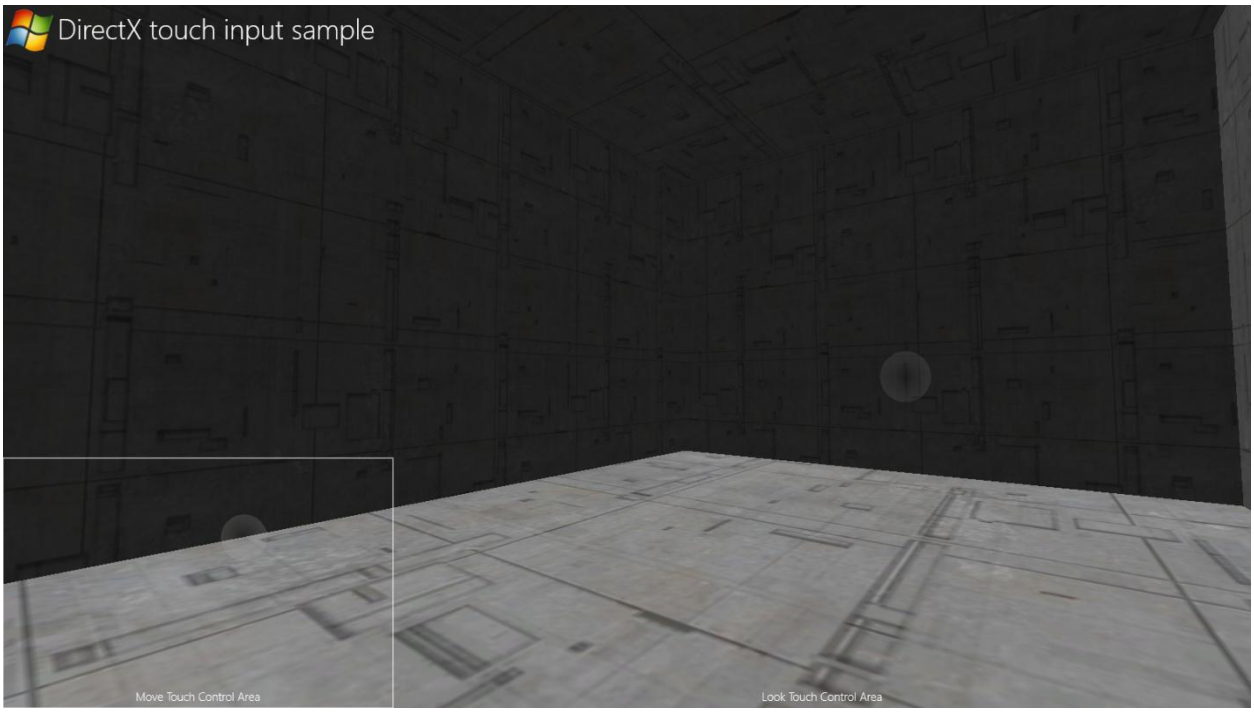
Win32* touch/Interaction Context sample – Intel



Windows* 8 gestures C#/JS sample – MSDN



DirectX* Touch input C++ sample – MSDN



InkRecognizer JS sample – MSDN

Notices

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.