Touch Developer Guide for Ultra Mobile Devices

Touch Developer Guide

Revision 2.0

January, 2014

Revision History

Revision	Revision History	Date
1.0	First Release	April 9, 2013
1.1	Update for Win 8.1 APIs	August 2013
2.0	Change Title, include Adaptive All in One (Adaptive AIO) considerations	January 2014

Contents

Figures

Tables

Abstract

This Guide contains information about the APIs that application developers need to use when they are developing apps targeted for Ultra Mobile Devices (PCs, Ultrabook™ devices, 2-in-1s, tablets) and Adaptive All in Ones, which have touch screens and multiple usages, such as a monitor or a multi-user tablet that may be easily moved to communal locations. This guide will cover the interfaces required for Windows* (7 through 8.1) as well as for web apps. It will also describe what the common user expectations for gestures are and provide guidance for developing satisfying touch interfaces. Lastly, some tips on optimizing touch-enabled apps will be provided.

Introduction

Now that there are many *touch-capable*, Ultra Mobile Devices available, it is important for software developers to create apps that are not only designed for touch input but also adapt to the different layouts that apply to various screen sizes and positions (portrait vs. landscape). The following list provides basic design considerations for touch-enabled apps.

- *Touch interfaces imply bigger targets*. For 96 dots per inch screens, the most frequently used controls should be at least 40X40 pixels, big enough to be uniquely indicated by a fingertip. For higher resolution screens, use the minimum of a ½ inch square. There should be a minimum of 10 pixels (1/8 of an inch) for padding between the touch targets.
- *Provide immediate feedback.* Elements that are interactive should react when touched, either by changing color, size, or by moving. When the app provides smooth, responsive visual feedback while panning, zooming, and rotating, it feels highly interactive.
- *Moveable content follows finger*. When a user drags an element, it should *follow the user's finger* when moving.
- Interactions should be reversible or at least cancellable.
- *Allow multi-touch interactions.* Touch interactions should not change based on the number of fingers that are touching the screen.
- *Allow both touch and mouse support*. The user may be working from a system that does not have touch capabilities.
- *Don't rely on hover.* While browsers will fake hover on tap, if there is an underlying link, the hover state does not stay long enough for the user to see since the tap action will also fire the link.
- *The app adapts to the device/environment that the user chooses.* This means providing layouts with the correct resolution for every likely method/device for which the user will be able to run the app.
- Optimize performance that results in a highly responsive app. Long delays when interacting with touch elements cause frustration. Ensure that complex processing runs in the background when and where possible.

While the basic design principles for touch apply to all form factors, designing touch interfaces for Adaptive All in One (Adaptive AIO) devices brings further considerations. Apps that are well-suited for the Adaptive AIO may have a multi-user component, for example, games where there is more than one player. Consider the implications of multiple users interacting with your app at the same time. The touch APIs for Windows and web apps can track more than one touch and/or drag at a time, but you still must design your app to behave sensibly under those conditions.

Even if you support multiple touch and/or drag events occurring at the same time, being able to differentiate more than one user is a very difficult task. If you need this capability, the simplest way to distinguish multiple users is to partition the display where each player is assigned their own portion during the game or activity.

For more information on writing touch-enabled apps, refer to the MSDN article, *Touch Interaction Design. Note: Please refer to the MSDN Terms of Use for licensing details.*

Hardware Requirements

Consumers have a variety of touch-capable devices to choose from, ranging from smartphones, tablets, 2-in-1s, PCs, to Adaptive AIOs. Developers are faced with the challenge of developing apps that feel natural for each form-factor. In general, touch-enabled apps should be designed to run on any of their targeted devices while taking full advantage of the touch capabilities of each. Devices that are capable of at least 10 simultaneous points represent the high-end of touch-capability and should be the design point for an app's touch interface.

Since 2011, OEMs have built devices with touchscreens based on Intel processors:

- 2nd generation Intel® Core™ processor family (codenamed Sandy Bridge).
- 3rd generation Intel® Core™ processor family (codenamed Ivy Bridge).
- 4th generation Intel® Core™ processor family (codenamed Haswell).

With so many Ultra Mobile Devices available to consumers, it has become more crucial than ever to develop apps that are touch-enabled for any form-factor for which they are targeted.

The rest of this Developer Guide assumes that the target platform is a touch-capable system. Software designed for these devices can be adapted to other touch-enabled devices that either run the same OS or run in the web browser.

Operating Environments for Ultra Mobile Devices

Designing apps today requires careful considerations of what environment customers most often use as well as which environment an app is best suited for. Whether an app is targeted for Windows 7 or Windows 8 Desktop or as a Windows* Store app, the developer needs to understand which interfaces are applicable.

Windows Store apps must use the WinRT APIs. If an app is to run in the Windows 8+ Desktop environment, there are two choices: the legacy APIs from Windows 7 or the new Windows 8+ APIs for touch. These interfaces will be discussed further in the sections below. Other options exist for developing web apps. Touch interfaces available for web browsers are also discussed below.

Touch Interactions: Pointer, Gesture, and Manipulation

There are varying levels of interpretation of touch input. Pointer events are the most basic because they represent individual points of touch contact. Gesture and Manipulation events are built on that foundation. Gesture events provide an easy way to capture simple tap-and-hold gestures. Manipulation events are for touch interactions that use physical gestures to emulate physical manipulation of UI elements. Manipulation events provide a more natural experience when the user interacts with UI elements on the screen. The available touch interfaces have varying levels of support for these three levels of interpretation.

Read more about Gestures, Manipulations and Interactions on MSDN

Pointer Interactions

A pointer event is a single, unique input or "contact" from an input device such as a mouse, stylus, single finger, or multiple fingers. When a contact is made, the system creates a pointer when it is first detected and then it is destroyed when the pointer leaves the detection range or is canceled. In the case of multi-touch input, each contact is a unique pointer. Table 1 shows the interfaces for retrieving basic pointer events that are available to mobile devices running Windows 7 and Windows 8+.

Table 1. Pointer APIs available for Touch-Enabled Devices

OS Compatibility	Touch Interface	Remarks
Windows* 7 (Desktop)	WM_TOUCH	• Also compatible with the Windows 8+ Desktop environment. • Maximum number of simultaneous touches limited by hardware. • No built-in gesture recognition. • Must call RegisterTouchWindow since WM_TOUCH messages are not sent by default.
Windows 8+ only (Desktop)	WM_POINTER	• Applicable only to the Windows 8+ Desktop environment. • By default Windows 8+ animations and interaction feedback is generated and available for further processing.
Windows Store app	PointerPoint	• Applicable only for Windows Store apps.

Touch Interfaces available for Windows 7 and Windows 8+:

- WM_TOUCH (messages)
- WM_POINTER (messages)
- PointerPoint (class)
- RegisterTouchWindow (function)

Refer to Guidelines for common user interactions on MSDN.

Windows 7 and Windows 8 Desktop Touch Interface: WM_TOUCH

The WM_TOUCH message can be used to indicate that one or more pointers, such as a finger or pen, have made contact on the screen.

Sample Code:

- Intel® Developer Zone: Touch for Windows Desktop (This sample includes custom Gesture Recognition.)

Guidance:

- MSDN: Touch Interaction Design
- MSDN: Windows 7 Touch Input Programming Guide

Windows 8 and 8.1 Desktop Touch Interface: WM_POINTER

The WM_POINTER messages are part of the Direct Manipulation APIs and are specific to the Windows 8+ Desktop. This interface can be used to capture individual touch pointers as well as Gestures and Manipulations. The WM_POINTER messages will be discussed further in the section on *Manipulation and Gesture Interactions*.

Reference on MSDN: Direct Manipulation APIs

Windows 8.1 Desktop adds direct manipulation support using the touchpad: Prior to Windows 8.1, device support in Windows assumed that the only sources of user input events were from a keyboard, mouse, touchscreen, or pen. Touchpads (present on many laptop designs) were treated as a mouse source. With Windows 8.1, the APIs have been expanded to treat touchpads as a distinct input method. For more information, refer to Touchpad interactions on MSDN.

- IMDT_TOUCHPAD (enum value for INPUT_MESSAGE_DEVICE_TYPE)

- POINTER_DEVICE_TYPE_TOUCH_PAD (enum value for POINTER_DEVICE_TYPE)
- PT_TOUCHPAD (enum value for POINTER_INPUT_TYPE)

There is also a new API: GetPointerInputTransform that is related to touch processing. This API is used for retrieving one or more 4X4 matrices when transforming the screen coordinates from a pointer input message to client coordinates, effectively inverting the mapping from the input device to screen coordinates.

Windows* Modern UI Touch Interface: PointerPoint

The PointerPoint class is part of the Windows Runtime environment and is compatible only with Windows Store apps. It provides basic properties for the input pointer associated with a single mouse, stylus, or touch contact. MSDN has sample code that can help developers get started working with the PointerPoint interface.

Sample Code on MSDN: Input: XAML user input events

Note: Please refer to the MSDN Terms of Use for licensing

Manipulation and Gesture Interactions

Gesture events are used to handle static-finger interactions such as tapping and press-and-hold. Double-tap and right-tap are derived from these basic gestures:

- **Gestures:** the physical act or motion performed on or by the input device that can be one or more fingers, a stylus, or a mouse.
- **Manipulation**: the immediate, ongoing response an object has to a gesture. For example, the slide gesture causes an object to move in some way.
- **Interactions**: how a manipulation is interpreted and the command or action that results from the manipulation. For example, both the slide and swipe gestures are similar but the results vary according to whether a distance threshold is exceeded.

Table 2. Basic gestures used for manipulation interactions

Gesture	Type	Description
Press and Hold	Static Gesture	A single contact is detected and does not move. Press and hold causes detailed information or teaching visuals to be displayed without a commitment to an action.
Tap	Static Gesture	One finger touches the screen and lifts up immediately.
Turn	Manipulation Gesture	Two or more fingers touch the screen and move in a clockwise or counter-clockwise direction.
Slide	Manipulation Gesture	One or more fingers touch the screen and move in the same direction (also called Panning)
Swipe	Manipulation	One or more fingers touch the screen and move a short distance in the same

	Gesture	direction.
Pinch	Manipulation Gesture	Two or more fingers touch the screen and move closer together.
Stretch	Manipulation Gesture	Two or more fingers touch the screen and move further apart.

Table 3. Gesture Interfaces available for Touch-Enabled Devices

OS Compatibility	GESTURE Interface	Remarks
Windows* 7 Windows 8+ (Desktop)	WM_TOUCH + IManipulationProcessor	• This combination gives the developer the functionality of the WM_POINTER API that is available only to Windows 8/8.1 desktop. • Maximum Touch points dictated by hardware.
Windows 7 Windows 8+ (Desktop)	WM_GESTURE + GESTUREINFO structure	• Maximum of two simultaneous touch points • No simultaneous gestures • If the app requires more complex manipulations than what is available from the WM_GESTURE message, a custom gesture recognizer needs to be written using the WM_TOUCH interface.
Windows 8+ (Desktop)	WM_POINTER	• Gesture interactions result from the use of the Direct Manipulation APIs, which take in a stream of the pointer input messages.
Windows Modern UI	PointerPoint	• Gesture interactions result from the use of GestureRecognizer, which takes the output from PointerPoint.
Windows 8.1	GetPointerInputTransform	• New Direct Manipulation API. Applicable to interactions with the touchpad.

References on MSDN:

- WM_TOUCH + IManipulationProcessor
- WM_GESTURE + GESTUREINFO
- WM_POINTER + Direct Manipulation APIS
- GetPointerInputTransform + Direct Manipulation APIS
- PointerPoint + GestureRecognizer

Table 4. Standard expected interactions and consequences for touch interactions

Interactions	**Description**
Press and Hold to learn	Causes detailed information or teaching visuals to be displayed.

Tap for primary action	Invokes a primary action, for example launching an application or executing a command.
Slide to pan	Used primarily for panning interactions but can also be used for moving, drawing, or writing. Can also be used to target small, densely packed elements by scrubbing (sliding the finger over related object such as radio buttons).
Swipe to select, command, and move	Sliding the finger a short distance, perpendicular to the panning direction, selects objects in a list or grid.
Pinch and stretch to zoom	Not only used for resizing, this interaction also enables jumping to the beginning, end, or anywhere within the content with Semantic Zoom. A SemanticZoom control results in a zoomed out view for showing groups of items and quick ways to go back to them.
Turn to rotate	Rotating with two or more fingers causes an object to rotate.
Swipe from edge for app commands	App commands are revealed by swiping from the bottom or top edge of the screen.
Swipe from edge for system commends	Swiping from the right edge of the screen shows the “charms” that are used for system commands. Swiping from the left edge results in cycling through currently running apps and sliding from the top edge toward the bottom of the screen closes the app. Sliding from the top edge down and to the left or right edge snaps the current app to that side of the screen.

Table 5. Basic gestures defined for touchpads – Windows* 8.1

Gesture	**Description**
Hover to learn	Allows the user to hover over an element to get more detailed information or teaching visuals without a commitment to action.
Single finger tap for primary action	Invokes the primary action (such as launching an app).
Two finger tap to right-click	Tapping with two fingers simultaneously displays the app bar with global commands or on an element to select it and display the app bar with contextual commands.
Two finger slide to pan	Used primarily for panning interactions.
Pinch and stretch to zoom	Used to resize and for semantic zooming.
Single finger press and slide to rearrange	Drag and drops an element.
Single finger press and slide to select text	Allows pressing from within selectable text and sliding to select it. Double-tap is used to select a word.
Single finger press and slide to	• Swiping from the right edge of the screen: reveals the charms exposing system

select text – Edges for system commands	commands • Left and right click zone: emulates left and right mouse buttons.

Interpreting Manipulation and Gesture Interactions for Windows 7 Desktop

The IManipulationProcessor interface can be used in conjunction with the WM_TOUCH API to provide a way to add translation, rotation, scaling, and inertia to UI objects. This combination provides functionality similar to the gesture recognizing features of WM_POINTER. Once the Manipulation Processor is enabled, manipulation starts as soon as a touch gesture is initiated.

Sample Code:

- MSDN: Manipulation and Inertia Sample
  Note: Please refer to the MSDN Terms of Use for licensing details.

WM_GESTURE messages have a structure called GESTUREINFO that is available for the interpretation of gestures and manipulations. The MSDN web page for WM_GESTURE shows an example of how to obtain gesture-specific information using the GESTUREINFO structure.

Note that WM_GESTURE has limitations, such as the maximum number of simultaneous touch inputs is only two and it does not support simultaneous gestures. For apps that require more capability but still need to support Windows 7 desktop, use the WM_TOUCH interface and either write a custom gesture recognizer, as detailed in the section *Custom Gesture Recognition* below, or use the Manipulation Processor interface with WM_TOUCH.

Sample Code on Intel Developer Zone (WM_GESTURE API + GESTUREINFO: Sample Application: Touch for Desktop

For more information on writing touch-enabled apps, refer to the MSDN article: *Touch Interaction Design*.

Handling Manipulation and Gesture Interactions for Windows 8+ Desktop Apps

Applications targeted only for the Windows 8 Desktop can use the Direct Manipulation APIs (WM_POINTER messages). The pointer messages are passed to an internal Interaction Context object that performs recognition on the manipulation without the need to implement a custom gesture recognizer. There is a callback infrastructure where all interactions involving tracked contacts are managed.

Direct Manipulation is designed to handle both manipulation and gesture interactions and supports two models for processing input:

1. *Automatic/Independent*: Window messages are automatically intercepted by Direct Manipulation on the delegate thread and handled without running application code, making it *independent* of the application.
2. *Manual/Dependent*: Window messages are received by the window procedure running in the UI thread, which then calls Direct Manipulation to process the message, making it *dependent* on the application.

Gestures can be captured by initializing Direct Manipulation and preparing the system for input processing.

Refer to the Quickstart: Direct Manipulation on MSDN for an outline of the API calls required to accomplish typical tasks when working with Direct Manipulation.

Handling Manipulation and Gesture Interactions for Windows 8 Store Apps

The GestureRecognizer API is used to handle pointer input to process manipulation and gesture events. Each object returned by the PointerPoint method is used to feed pointer data to the GestureRecognizer. The gesture recognizer listens for and handles the pointer input and processes the static gesture events. For an example of how to create a GestureRecognizer object and then enable manipulation gesture events on that object see the MSDN GestureRecognizer web page (referenced below.)

References on MSDN:

- GestureRecognizer APIs
- PointerPoint method

Sample Code on Intel Developer Zone: Sample Application: Touch for Windows* Store.

Direct Manipulation Support for Windows 8.1 Store Apps

For Windows 8.1, Direct Manipulation has been added for touchpad gestures. Note that the ScrollViewer class will be updated, and as a result, apps will no longer see UIElementPointerWheelChanged events that were available in Windows 8.

References on MSDN:

- Touch interactions
- Quickstart: Identifying pointer devices
- UIElement.PointerEntered event
- Responding to touchpad input

Sample Code on MSDN: Input: Gestures and manipulations with GestureRecognizer sample (Windows 8.1)

Custom Gesture Recognition

When possible, use the built-in gesture recognizers (see Table 3). If the provided gesture and manipulation interfaces do not provide the functionality that is needed, or if the app needs to disambiguate between taps and gestures more rapidly, it may be necessary to write a custom gesture recognition algorithm. If this is the case, users expect an intuitive experience involving direct interaction with the UI elements in the app. It is best to base custom interactions on the standard controls to keep user actions consistent and discoverable. Custom interactions should only be used if there is a clear, well-defined requirement and basic interactions don't support the app's desired functionality. See Table 4 for the list of common and expected interactions and consequences for touch interactions.

Code Sample on Intel Developer Zone (WM_TOUCH with custom gesture recognition): Touch for Windows Desktop

Touch Support in Web Browsers

Touch input is also available to apps running in web browsers, with varying degrees of support depending on the browser. Since web browser capabilities change rapidly, it is generally better to detect supported features instead of specific browsers. Feature detection has proven to be a more effective technique once the determination has been made if it is Internet Explorer* (IE) 11, a browser built on Webkit*, or a different browser that requires support. Feature detection requires less maintenance for the following reasons:

- New browsers get released and existing browsers are often updated. Existing code may not factor in the new browser versions. Updated browsers may support standards and features that were not supported when the browser detection code was designed.

- New devices frequently include new versions of browsers, and so browser detection code must be reviewed continually to support the new browsers. Creating customized implementations for each browser can become extremely complicated.
- Many browsers support the ability to modify the user-agent string, making browser detection difficult to accurately identify.

WebKit powers Apple Safari* and Google Chrome*, and soon Opera will move their browser over to use it. Internet Explorer 10 does not use WebKit; however, both WebKit and IE 10 are built on top of the Document Object Model (DOM) Level 3 Core Specification. To review the standards associated with touch events, refer to the standard's Touch Events Version 1, dated January 2013.

References:

- Webkit
- Document Object Model (DOM) Level 3 Core Specification
- Touch Events Version 1
- The navigator object
- Navigator userAgent property

Note: Please refer to the MSDN Terms of Use for licensing details.

IE 10 has its own touch interfaces that must be called for processing touch events. Use the *navigator* object with the *userAgent* property to determine if the browser supports the desired features. The following example code can be used to determine if the browser is IE.

Usage:
```
<script type="text/JavaScript">
If (navigator.userAgent.indexOf("MSIE")>0)
    {
         // Run custom code for Internet Explorer.
    }
</script>
```
Figure 1. Snippet for determining if browser is Internet Explorer*

Use the *hasFeature* method to determine if specific features are supported in the browser. For example, here is how to determine if a browser supports touch events (this works for IE 10 as well):

```
var touchSupported = document.implementation.hasFeature("touch-events","3.0");
```

Where "touch-events" is the feature that we are checking for and "3.0" is the DOM specification level that we are interested in. An app can then listen for the following touch events*: touchstart*, *touchend, touchmove*, and *touchcancel*.

Reference: hasFeature method

To process touch events using a WebKit-based browser (Chrome, Safari, etc.), simply set up the following three events to cover the main input states:

```
canvas.addEventListener( ‘touchstart’, onTouchStart, false );
canvas.addEventListener( ‘touchmove’, onTouchMove, false);
canvas.addEventListener( ‘touchend’, onTouchEnd, false);
```

For Internet Explorer, reference the MSPointer event instead:

```
canvas.addEventListener( ‘MSPointerDown’, onTouchStart, false );
canvas.addEventListener( ‘MSPointerMove’, onTouchMove, false);
canvas.addEventListener( ‘MSPointerUp’, onTouchEnd, false);
```

Similarly, there are three gesture event listeners: gestureStart, gestureChange, and gestureEnd for non-IE 10 browsers.

Download sample code handling DOM pointer events on MSDN: Input DOM pointer event handling sample. *Note: Please refer to the MSDN Terms of Use for licensing details.*

Internet Explorer 10 and its Compatibility with Windows 7

While IE 10 does not use WebKit, it is built on top of the DOM Level 3 Events, HTML5, and Progress Event standards. This section provides information about IE 10 and how it interacts on Windows 7.

Standards:

- DOM Level 3 Events
- HTML5
- Progress Events

Internet Explorer 10 on Windows 7 handles touch and pen input as simulated mouse input for the following Document Object Model (DOM) events:

- MSPointerCancel
- MSPointerDown
- MSPointerMove
- MSPointerOver

- MSPointerUp

IE 10 on Windows 7 will not fire any of the following DOM Gesture events:

- MSGestureChange
- MSGestureEnd
- MSGestureHold
- MSGestureStart
- MSGestureTap
- MSManipulationStateChanged

Table 6. Touch Interfaces for Internet Explorer* 10

Interface	Windows* 7 MSVS 2010	Windows 8 MSVS 2012 (Desktop)	Windows 8 Modern UI	Remarks
MSGESTURE	No	Yes	Yes	• Get high level gestures such as hold, pan, and tap easily without capturing every pointer event individually
MSPOINTER	Yes	YES	Yes	• Part of DOM Object Model (DOM) Core • The *getCurrentPoint* and *getIntermediatePoints* methods retrieve a collection of PointerPoint objects and are available only on Windows 8.

For more information on developing touch-enabled web apps for IE 10 (MSDN): Internet Explorer 10 Guide for Developers

Sample code on MSDN: Input: Manipulations and gestures (JavaScript*)

Note: Please refer to the MSDN Terms of Use for licensing details.

Internet Explorer 11

IE 11 adds the following touch enhancements / APIs:

- Direct manipulation for mouse, keyboard, touchpad, and touch, including hardware-accelerated pan and zoom to all input types.

- Updates to the existing MSPointer APIs to reflect the latest Candidate Recommendation specification. IE 11 supports unprefixed Pointer Events.
- New API, the msZoomTo method. This new method scrolls and/or zooms an element to a specified location and uses animation. Note: msZoomTo is not supported on Windows 7.

For more information on developing touch-enabled web apps for IE 11 (MSDN):
Internet Explorer 11 Guide for Developers

IE 10 and IE 11 also include Cascading Style Sheets with features applicable to Touch. The following table summarizes the controls that enable touch input and gesture recognition for Internet Explorer versions.

Table 7. Scrolling and Zooming Properties for Internet Explorer* Versions 10 and 11

Scrolling/Zooming property for IE 10 and IE 11	**Touchscreen**	**Touchpad**	**Mouse**	**Keyboard**
-ms-scroll-snap-points-x, -ms-scroll-snap-points-y, -ms-scroll-snap-type, -ms-scroll-snap-x, -ms-scroll-snap-y	IE 10+	IE 11	IE 11	IE 11
-ms-content-zoom-chaining, msContentZoomFactor, -ms-content-zooming, -ms-content-zoom-limit, -ms-content-zoom-limit-max, -ms-content-zoom-limit-min, -ms-content-zoom-snap, -ms-content-zoom-snap-points, -ms-content-zoom-snap-type, -ms-scroll-chaining, -ms-scroll-rails	IE 10+	IE 11	--	--
-ms-overflow-style, -ms-scroll-limit, -ms-scroll-limitXMax, -ms-scroll-limitXMin, -ms-scroll-limitYMax, -ms-scroll-limitYMin	IE 10+	IE 10+	IE 10+	IE 10+
-ms-scroll-translation	--	--	IE 10+	--

Identifying Touch Capability

Whether an app is a native app or a web app, the developer will want to add a check for hardware touch capability so that the app can configure its UI appropriately. Use the following methods to test for touch capability.

Windows 7 and Windows 8 Desktop

Apps targeting Windows 7 or Windows 8 Desktop can call *GetSystemMetrics* with **SM_DIGITIZER** as the argument. The following code snippet is part of a Touch sample that can be downloaded from the Intel Developer Zone: *Touch for Windows Desktop*

References:

- Code Sample: Touch for Windows Desktop
- GetSystemMetrics method

Note: Please refer to the MSDN Terms of Use for licensing details.

```
// Check for Touch support
// Get the Touch Capabilities by calling GetSystemMetrics
BYTE digitizerStatus = (BYTE)GetSystemMetrics(SM_DIGITIZER);
// Hardware touch capability (0x80); readiness (0x40)
if ((digitizerStatus & (0x80 + 0x40)) != 0) //Stack Ready + MultiTouch
{
    RegisterTouchWindow(m_pWindow->GetHWnd(), TWF_WANTPALM);
}
```

Figure 2. Windows* 7 Example for identifying touch capability

Note that *GetSystemMetrics* can be used to find out what is the maximum number of touch points that are available:

```
BYTE nInputs = (BYTE)GetSystemMetrics(SM_MAXIMUMTOUCHES);
```

Windows 8 (Windows Store apps)

Determine touch capabilities for Windows Store apps by using the *TouchCapabilities* class. The following code snippet can be found in the code sample on MSDN that demonstrates its use: *Input: Device capabilities sample*.

References:

- Code Sample on MSDN: Input: Device capabilities sample
- TouchCapabilities class

  Note: Please refer to the MSDN Terms of Use for licensing details.

```
void SDKSample::DeviceCaps::Touch::TouchGetSettings_Click(Platform::Object^ sender,
Windows::UI::Xaml::RoutedEventArgs^ e)
```

```
{
    Button^ b = safe_cast<Button^>(sender);
    if (b != nullptr)
    {
        TouchCapabilities^ pTouchCapabilities = ref new TouchCapabilities();
        Platform::String^ Buffer;

        Buffer = "There is " + (pTouchCapabilities->TouchPresent != 0 ? "a" : "no") + "
              digitizer present\n";
        Buffer += "The digitizer supports " + pTouchCapabilities->Contacts.ToString() + "
              contacts\n";

        TouchOutputTextBlock->Text = Buffer;
    }
}
```

Figure 3. Windows*UI Example for Identifying Touch Capability

Web apps

For Internet Explorer, use the *msMaxTouchPoints* property described as follows:

Test for touch capable hardware:

```
If (navigator.msMaxTouchPoints) {…}
```

Test for multi-touch capable hardware:

```
If (navigator.msMaxTouchPoints >1) {…}
```

Get the maximum number of touch points the hardware supports:

```
Var touchPoints = navigator.msMaxTouchPoints;
```

For Chrome and Safari, use the following (same as above but replace *msMaxTouchPoints* with *maxTouchPoints*):

```
var result = navigator.maxTouchPoints;
```

It can be somewhat tricky to test for touch devices generically from web apps. While some functions work well on some browsers, others indicate that touch is present when it is not, i.e., if the browser itself supports touch, it may report that touch is available even if the device is not touch-capable.

Note that MaxTouchPoints will return 0 in IE 10 (Desktop) running on Windows 7.

References:

- msMaxTouchPoints
- maxTouchPoints

Note: Please refer to the MSDN Terms of Use for licensing details.

UI design for Touch-Enabled Devices

Apps designed for touch-enabled devices may need to process gestures such as taps, pans, zooms, etc. Apps that are touch-enabled may do little with the raw pointer data except to pass it to gesture detection.

New applications should be designed with the expectation that touch will be the primary input method. Mouse and stylus support require no additional work; however, software developers should consider several other factors when designing touch-optimized apps.

Table 8. Considerations for Touch-Enabled Apps

Factor	Touch	Mouse/Stylus
Precision	• Contact area for fingertip is much larger than a single x-y coordinate. • The shape of the contact area changes with the movement • There is no mouse cursor to help with targeting	• Mouse/Stylus gives a precise x-y coordinate • Keyboard focus is explicit
Human Anatomy	• Fingertip movements are imprecise • Some areas on the touch surface may be difficult to reach • Objects may be obscured by one or more fingertips	• Straight-line motion with the mouse/stylus are easier to perform • Mouse/Stylus can reach any part of the screen • Indirect input devices do not cause obstruction
Object state	• Touch uses a two-state model. The touch surface is either touched or not. There is no hover state that can trigger additional visual feedback.	• Three states are available: on, off, hover (focus)
Rich interaction	• Multi-touch – multiple input points (fingertips) are available.	• Supports only a single input point.

Software developers should supply appropriate visual feedback during interactions so that users can recognize, learn, and adapt to how their interactions are interpreted by both the app and the OS. Visual feedback is important for users to let them know if their interactions are successful, so they can improve their sense of control. It can help reduce errors and help users understand the system and input device.

Tips for Building Optimized, Responsive Apps

1. The Click-delay: For web apps, this happens when the click event is delayed on mobile devices causing pages to feel slow or unresponsive. This happens because the browser is trying to decide if the click

was a tap or if it was a double-tap. The solution to this is to use a good fastclick library. Fastclick libraries listen for touchend.

2. Avoid expensive operations in touch handlers. Instead of processing the touches immediately, store them and process them later.
3. Touch handlers can cause "Scroll Jank." This can happen when there is a touch event handler on a page where scrolling is allowed. If the app needs both scrolling and touch events, make the touch area as small as possible.

Resources for Developing Touch Applications

Related Articles on Intel Developer Zone:

1. Comparing Touch Coding Techniques – Windows 8 Desktop Touch Sample
2. Exploring Touch Samples for Windows* 8 apps
3. Touch Code Sample for Windows* 8 Store
4. Touch Code Sample for Windows* 8 Desktop
5. Porting Win32* Apps to Windows* 8 Desktop
6. Real-Time Strategy Game with Touch Screen
7. Virtual Trackpad
8. Mixing Stylus and Touch Input on Windows* 8
9. Implementing multi-user multi-touch scenarios using WPF in Windows* Desktop Apps

Related Articles on MSDN

1. Windows 7 Touch Input Programming Guide
2. Architectural Overview (Windows 7)
3. Troubleshooting Applications
4. Adding Manipulation Support in Unmanaged Code
5. Windows Touch Samples
6. Build Advanced Touch Apps in Windows 8* (Video)
7. Windows 8 SDK
8. Input: Touch hit testing sample
9. Desktop App Development Documentation (Windows)
10. Windows Touch Gestures Overview (Windows)
11. Getting Started with Windows Touch Messages (Windows)
12. Get PointerTouchInfo function (Windows)
13. (MSDN) Internet Explorer 10 Guide for Developers
14. (MSDN) Internet Explorer 11 Guide for Developers
15. (MSDN) Touchpad Interactions
16. (MSDN) Scrolling and Zooming with touch and other inputs
17. (MSDN) Terms of Use

Web Apps

1. Dragging and Scaling and Object (Sample Code)
2. Handling Multi-Touch Gestures
3. W3C Software Notice and License

Videos:

1. Google I/O 2013 – Point, Click, Tap, Touch – Building Multi-Device Web Interfaces

Summary

Developers who want to develop touch-enabled apps, whether they are native or web apps, need to have a clear understanding of which APIs are available to them. This guide covered the interfaces available for the following environments: Windows 7, Windows 8+ Desktop, Windows Modern UI, as well as apps running in web browsers. While Gestures and Manipulations are possible in Windows 7, app developers may find that the Windows 8+ APIs (those targeted for the Desktop and/or for Windows Store apps) offer the best options for automatic gesture recognition.

Windows 8.1 and IE 11 add touchpad APIs and interactions. Many of the existing APIs used for Gestures, Manipulation, Pointers for Windows 8 Desktop, Windows 8 Store Apps, and IE 11 have been updated as well.

Developers who are writing touch-enabled web apps need to check their code for IE 10+ since IE 10+ has its own interface that must be used to process touch, gestures, and manipulation. Other Webkit-based browsers are also based on the DOM Level 3 standards and have touch and gesture event support.

This guide also covered descriptions of common Gesture and Manipulation interactions and provided some guidelines for developing touch-enabled apps.

About the Author

Gael Hofemeier is an Evangelist Application Engineer at Intel Corporation. Her focus is providing technical content that developers writing software for Intel® Architecture need. In addition to writing content, she also moderates the Business Client Forum on the Intel Developer Zone.

See Gael's Blog Author Page

View my profile on Linked in

Notices