<https://source.android.com/devices/input/touch-devices.html#tools-and-tool-types>

Touch Devices

Android supports a variety of touch screens and touch pads, including stylus-based digitizer tablets.

Touch screens are touch devices that are associated with a display such that the user has the impression of directly manipulating items on screen.

Touch pads are touch devices that are not associated with a display such as a digitizer tablet. Touch pads are typically used for pointing or for absolute indirect positioning or gesture-based control of a user interface.

Touch devices may have buttons whose functions are similar to mouse buttons.

Touch devices can sometimes be manipulated using a variety of different tools such as fingers or a stylus depending on the underlying touch sensor technology.

Touch devices are sometimes used to implement virtual keys. For example, on some Android devices, the touch screen sensor area extends beyond the edge of the display and serves dual purpose as part of a touch sensitive key pad.

Due to the great variety of touch devices, Android relies on a large number of configuration properties to describe the characteristics and desired behavior of each device.

Touch Device Classification

An input device is classified as a multi-touch device if both of the following conditions hold:

The input device reports the presence of the ABS\_MT\_POSITION\_X and ABS\_MT\_POSITION\_Y absolute axes.

The input device does not have any gamepad buttons. This condition resolves an ambiguity with certain gamepads that report axes with codes that overlaps those of the MT axes.

An input device is classified as a single-touch device if both of the following conditions hold:

The input device is not classified as a multi-touch device. An input device is either classified as a single-touch device or as a multi-touch device, never both.

The input device reports the presence of the ABS\_X and ABS\_Y absolute axes, and the presence of the BTN\_TOUCH key code.

Once an input device has been classified as a touch device, the presence of virtual keys is determined by attempting to load the virtual key map file for the device. If a virtual key map is available, then the key layout file for the device is also loaded.

Refer to the section below about the location and format of virtual key map files.

Next, the system loads the input device configuration file for the touch device.

All built-in touch devices should have input device configuration files. If no input device configuration file is present, the system will choose a default configuration that is appropriate for typical general-purpose touch peripherals such as external USB or Bluetooth HID touch screens or touch pads. These defaults are not designed for built-in touch screens and will most likely result in incorrect behavior.

After the input device configuration loaded, the system will classify the input device as a touch screen, touch pad or pointer device.

A touch screen device is used for direct manipulation of objects on the screen. Since the user is directly touching the screen, the system does not require any additional affordances to indicate the objects being manipulated.

A touch pad device is used to provide absolute positioning information to an application about touches on a given sensor area. It may be useful for digitizer tablets.

A pointer device is used for indirect manipulation of objects on the screen using a cursor. Fingers are interpreted as multi-touch pointer gestures. Other tools, such as styluses, are interpreted using absolute positions.

See Indirect Multi-touch Pointer Gestures for more information.

The following rules are used to classify the input device as a touch screen, touch pad or pointer device.

If the touch.deviceType property is set, then the device type will be set as indicated.

If the input device reports the presence of the INPUT\_PROP\_DIRECT input property (via the EVIOCGPROP ioctl), then the device type will be set to touch screen. This condition assumes that direct input touch devices are attached to a display that is also connected.

If the input device reports the presence of the INPUT\_PROP\_POINTER input property (via the EVIOCGPROP ioctl), then the device type will be set to pointer.

If the input device reports the presence of the REL\_X or REL\_Y relative axes, then the device type will be set to touch pad. This condition resolves an ambiguity for input devices that consist of both a mouse and a touch pad. In this case, the touch pad will not be used to control the pointer because the mouse already controls it.

Otherwise, the device type will be set to pointer. This default ensures that touch pads that have not been designated any other special purpose will serve to control the pointer.

Buttons

Buttons are optional controls that may be used by applications to perform additional functions. Buttons on touch devices behave similarly to mouse buttons and are mainly of use with pointer type touch devices or with a stylus.

The following buttons are supported:

BTN\_LEFT: mapped to MotionEvent.BUTTON\_PRIMARY.

BTN\_RIGHT: mapped to MotionEvent.BUTTON\_SECONDARY.

BTN\_MIDDLE: mapped to MotionEvent.BUTTON\_MIDDLE.

BTN\_BACK and BTN\_SIDE: mapped to MotionEvent.BUTTON\_BACK. Pressing this button also synthesizes a key press with the key code KeyEvent.KEYCODE\_BACK.

BTN\_FORWARD and BTN\_EXTRA: mapped to MotionEvent.BUTTON\_FORWARD. Pressing this button also synthesizes a key press with the key code KeyEvent.KEYCODE\_FORWARD.

BTN\_STYLUS: mapped to MotionEvent.BUTTON\_SECONDARY.

BTN\_STYLUS2: mapped to MotionEvent.BUTTON\_TERTIARY.

Tools and Tool Types

A tool is a finger, stylus or other apparatus that is used to interact with the touch device. Some touch devices can distinguish between different types of tools.

Elsewhere in Android, as in the MotionEvent API, a tool is often referred to as a pointer.

The following tool types are supported:

BTN\_TOOL\_FINGER and MT\_TOOL\_FINGER: mapped to MotionEvent.TOOL\_TYPE\_FINGER.

BTN\_TOOL\_PEN and MT\_TOOL\_PEN: mapped to MotionEvent.TOOL\_TYPE\_STYLUS.

BTN\_TOOL\_RUBBER: mapped to MotionEvent.TOOL\_TYPE\_ERASER.

BTN\_TOOL\_BRUSH: mapped to MotionEvent.TOOL\_TYPE\_STYLUS.

BTN\_TOOL\_PENCIL: mapped to MotionEvent.TOOL\_TYPE\_STYLUS.

BTN\_TOOL\_AIRBRUSH: mapped to MotionEvent.TOOL\_TYPE\_STYLUS.

BTN\_TOOL\_MOUSE: mapped to MotionEvent.TOOL\_TYPE\_MOUSE.

BTN\_TOOL\_LENS: mapped to MotionEvent.TOOL\_TYPE\_MOUSE.

BTN\_TOOL\_DOUBLETAP, BTN\_TOOL\_TRIPLETAP, and BTN\_TOOL\_QUADTAP: mapped to MotionEvent.TOOL\_TYPE\_FINGER.

Hovering vs. Touching Tools

Tools can either be in contact with the touch device or in range and hovering above it. Not all touch devices are able to sense the presence of a tool hovering above the touch device. Those that do, such as RF-based stylus digitizers, can often detect when the tool is within a limited range of the digitizer.

The InputReader component takes care to distinguish touching tools from hovering tools. Likewise, touching tools and hovering tools are reported to applications in different ways.

Touching tools are reported to applications as touch events using MotionEvent.ACTION\_DOWN, MotionEvent.ACTION\_MOVE, MotionEvent.ACTION\_DOWN, MotionEvent.ACTION\_POINTER\_DOWN and MotionEvent.ACTION\_POINTER\_UP.

Hovering tools are reported to applications as generic motion events using MotionEvent.ACTION\_HOVER\_ENTER, MotionEvent.ACTION\_HOVER\_MOVE and MotionEvent.ACTION\_HOVER\_EXIT.

Touch Device Driver Requirements

Touch device drivers should only register axes and key codes for the axes and buttons that they actually support. Registering excess axes or key codes may confuse the device classification algorithm or cause the system to incorrectly detect the capabilities of the device.

For example, if the device reports the BTN\_TOUCH key code, the system will assume that BTN\_TOUCH will always be used to indicate whether the tool is actually touching the screen or is merely in range and hovering.

Single-touch devices use the following Linux input events:

ABS\_X: (REQUIRED) Reports the X coordinate of the tool.

ABS\_Y: (REQUIRED) Reports the Y coordinate of the tool.

ABS\_PRESSURE: (optional) Reports the physical pressure applied to the tip of the tool or the signal strength of the touch contact.

ABS\_TOOL\_WIDTH: (optional) Reports the cross-sectional area or width of the touch contact or of the tool itself.

ABS\_DISTANCE: (optional) Reports the distance of the tool from the surface of the touch device.

ABS\_TILT\_X: (optional) Reports the tilt of the tool from the surface of the touch device along the X axis.

ABS\_TILT\_Y: (optional) Reports the tilt of the tool from the surface of the touch device along the Y axis.

BTN\_TOUCH: (REQUIRED) Indicates whether the tool is touching the device.

BTN\_LEFT, BTN\_RIGHT, BTN\_MIDDLE, BTN\_BACK, BTN\_SIDE, BTN\_FORWARD, BTN\_EXTRA, BTN\_STYLUS, BTN\_STYLUS2: (optional) Reports button states.

BTN\_TOOL\_FINGER, BTN\_TOOL\_PEN, BTN\_TOOL\_RUBBER, BTN\_TOOL\_BRUSH, BTN\_TOOL\_PENCIL, BTN\_TOOL\_AIRBRUSH, BTN\_TOOL\_MOUSE, BTN\_TOOL\_LENS, BTN\_TOOL\_DOUBLETAP, BTN\_TOOL\_TRIPLETAP, BTN\_TOOL\_QUADTAP: (optional) Reports the tool type.

Multi-touch devices use the following Linux input events:

ABS\_MT\_POSITION\_X: (REQUIRED) Reports the X coordinate of the tool.

ABS\_MT\_POSITION\_Y: (REQUIRED) Reports the Y coordinate of the tool.

ABS\_MT\_PRESSURE: (optional) Reports the physical pressure applied to the tip of the tool or the signal strength of the touch contact.

ABS\_MT\_TOUCH\_MAJOR: (optional) Reports the cross-sectional area of the touch contact, or the length of the longer dimension of the touch contact.

ABS\_MT\_TOUCH\_MINOR: (optional) Reports the length of the shorter dimension of the touch contact. This axis should not be used if ABS\_MT\_TOUCH\_MAJOR is reporting an area measurement.

ABS\_MT\_WIDTH\_MAJOR: (optional) Reports the cross-sectional area of the tool itself, or the length of the longer dimension of the tool itself. This axis should not be used if the dimensions of the tool itself are unknown.

ABS\_MT\_WIDTH\_MINOR: (optional) Reports the length of the shorter dimension of the tool itself. This axis should not be used if ABS\_MT\_WIDTH\_MAJOR is reporting an area measurement or if the dimensions of the tool itself are unknown.

ABS\_MT\_ORIENTATION: (optional) Reports the orientation of the tool.

ABS\_MT\_DISTANCE: (optional) Reports the distance of the tool from the surface of the touch device.

ABS\_MT\_TOOL\_TYPE: (optional) Reports the tool type as MT\_TOOL\_FINGER or MT\_TOOL\_PEN.

ABS\_MT\_TRACKING\_ID: (optional) Reports the tracking id of the tool. The tracking id is an arbitrary non-negative integer that is used to identify and track each tool independently when multiple tools are active. For example, when multiple fingers are touching the device, each finger should be assigned a distinct tracking id that is used as long as the finger remains in contact. Tracking ids may be reused when their associated tools move out of range.

ABS\_MT\_SLOT: (optional) Reports the slot id of the tool, when using the Linux multi-touch protocol 'B'. Refer to the Linux multi-touch protocol documentation for more details.

BTN\_TOUCH: (REQUIRED) Indicates whether the tool is touching the device.

BTN\_LEFT, BTN\_RIGHT, BTN\_MIDDLE, BTN\_BACK, BTN\_SIDE, BTN\_FORWARD, BTN\_EXTRA, BTN\_STYLUS, BTN\_STYLUS2: (optional) Reports button states.

BTN\_TOOL\_FINGER, BTN\_TOOL\_PEN, BTN\_TOOL\_RUBBER, BTN\_TOOL\_BRUSH, BTN\_TOOL\_PENCIL, BTN\_TOOL\_AIRBRUSH, BTN\_TOOL\_MOUSE, BTN\_TOOL\_LENS, BTN\_TOOL\_DOUBLETAP, BTN\_TOOL\_TRIPLETAP, BTN\_TOOL\_QUADTAP: (optional) Reports the tool type.

If axes for both the single-touch and multi-touch protocol are defined, then only the multi-touch axes will be used and the single-touch axes will be ignored.

The minimum and maximum values of the ABS\_X, ABS\_Y, ABS\_MT\_POSITION\_X and ABS\_MT\_POSITION\_Y axes define the bounds of the active area of the device in device-specific surface units. In the case of a touch screen, the active area describes the part of the touch device that actually covers the display.

For a touch screen, the system automatically interpolates the reported touch positions in surface units to obtain touch positions in display pixels according to the following calculation:

displayX = (x - minX) \* displayWidth / (maxX - minX + 1)

displayY = (y - minY) \* displayHeight / (maxY - minY + 1)

A touch screen may report touches outside of the reported active area.

Touches that are initiated outside the active area are not delivered to applications but may be used for virtual keys.

Touches that are initiated inside the active area, or that enter and exit the display area are delivered to applications. Consequently, if a touch starts within the bounds of an application and then moves outside of the active area, the application may receive touch events with display coordinates that are negative or beyond the bounds of the display. This is expected behavior.

A touch device should never clamp touch coordinates to the bounds of the active area. If a touch exits the active area, it should be reported as being outside of the active area, or it should not be reported at all.

For example, if the user's finger is touching near the top-left corner of the touch screen, it may report a coordinate of (minX, minY). If the finger continues to move further outside of the active area, the touch screen should either start reporting coordinates with components less than minX and minY, such as (minX - 2, minY - 3), or it should stop reporting the touch altogether. In other words, the touch screen should not be reporting (minX, minY) when the user's finger is really touching outside of the active area.

Clamping touch coordinates to the display edge creates an artificial hard boundary around the edge of the screen which prevents the system from smoothly tracking motions that enter or exit the bounds of the display area.

The values reported by ABS\_PRESSURE or ABS\_MT\_PRESSURE, if they are reported at all, must be non-zero when the tool is touching the device and zero otherwise to indicate that the tool is hovering.

Reporting pressure information is optional but strongly recommended. Applications can use pressure information to implement pressure-sensitive drawing and other effects.

The values reported by ABS\_TOOL\_WIDTH, ABS\_MT\_TOUCH\_MAJOR, ABS\_MT\_TOUCH\_MINOR, ABS\_MT\_WIDTH\_MAJOR, or ABS\_MT\_WIDTH\_MINOR should be non-zero when the tool is touching the device and zero otherwise, but this is not required. For example, the touch device may be able to measure the size of finger touch contacts but not stylus touch contacts.

Reporting size information is optional but strongly recommended. Applications can use pressure information to implement size-sensitive drawing and other effects.

The values reported by ABS\_DISTANCE or ABS\_MT\_DISTANCE should approach zero when the tool is touching the device. The distance may remain non-zero even when the tool is in direct contact. The exact values reported depend on the manner in which the hardware measures distance.

Reporting distance information is optional but recommended for stylus devices.

The values reported by ABS\_TILT\_X and ABS\_TILT\_Y should be zero when the tool is perpendicular to the device. A non-zero tilt is taken as an indication that the tool is held at an incline.

The tilt angles along the X and Y axes are assumed to be specified in degrees from perpendicular. The center point (perfectly perpendicular) is given by (max + min) / 2 for each axis. Values smaller than the center point represent a tilt up or to the left, values larger than the center point represent a tilt down or to the right.

The InputReader converts the X and Y tilt components into a perpendicular tilt angle ranging from 0 to PI / 2 radians and a planar orientation angle ranging from -PI to PI radians. This representation results in a description of orientation that is compatible with what is used to describe finger touches.

Reporting tilt information is optional but recommended for stylus devices.

If the tool type is reported by ABS\_MT\_TOOL\_TYPE, it will supersede any tool type information reported by BTN\_TOOL\_\*. If no tool type information is available at all, the tool type defaults to MotionEvent.TOOL\_TYPE\_FINGER.

A tool is determined to be active based on the following conditions:

When using the single-touch protocol, the tool is active if BTN\_TOUCH, or BTN\_TOOL\_\* is 1.

This condition implies that the InputReader needs to have at least some information about the nature of the tool, either whether it is touching, or at least its tool type. If no information is available, then the tool is assumed to be inactive (out of range).

When using the multi-touch protocol 'A', the tool is active whenever it appears in the most recent sync report. When the tool stops appearing in sync reports, it ceases to exist.

When using the multi-touch protocol 'B', the tool is active as long as it has an active slot. When the slot it cleared, the tool ceases to exist.

A tool is determined to be hovering based on the following conditions:

If the tool is BTN\_TOOL\_MOUSE or BTN\_TOOL\_LENS, then the tool is not hovering, even if either of the following conditions are true.

If the tool is active and the driver reports pressure information, and the reported pressure is zero, then the tool is hovering.

If the tool is active and the driver supports the BTN\_TOUCH key code and BTN\_TOUCH has a value of zero, then the tool is hovering.

The InputReader supports both multi-touch protocol 'A' and 'B'. New drivers should use the 'B' protocol but either will work.

As of Android Ice Cream Sandwich 4.0, touch screen drivers may need to be changed to comply with the Linux input protocol specification.

The following changes may be required:

When a tool becomes inactive (finger goes "up"), it should stop appearing in subsequent multi-touch sync reports. When all tools become inactive (all fingers go "up"), the driver should send an empty sync report packet, such as SYN\_MT\_REPORT followed by SYN\_REPORT.

Previous versions of Android expected "up" events to be reported by sending a pressure value of 0. The old behavior was incompatible with the Linux input protocol specification and is no longer supported.

Physical pressure or signal strength information should be reported using ABS\_MT\_PRESSURE.

Previous versions of Android retrieved pressure information from ABS\_MT\_TOUCH\_MAJOR. The old behavior was incompatible with the Linux input protocol specification and is no longer supported.

Touch size information should be reported using ABS\_MT\_TOUCH\_MAJOR.

Previous versions of Android retrieved size information from ABS\_MT\_TOOL\_MAJOR. The old behavior was incompatible with the Linux input protocol specification and is no longer supported.

Touch device drivers no longer need Android-specific customizations. By relying on the standard Linux input protocol, Android can support a wider variety of touch peripherals, such as external HID multi-touch touch screens, using unmodified drivers.

Touch Device Operation

The following is a brief summary of the touch device operation on Android.

The EventHub reads raw events from the evdev driver.

The InputReader consumes the raw events and updates internal state about the position and other characteristics of each tool. It also tracks button states.

If the BACK or FORWARD buttons were pressed or released, the InputReader notifies the InputDispatcher about the key event.

The InputReader determines whether a virtual key press occurred. If so, it notifies the InputDispatcher about the key event.

The InputReader determines whether the touch was initiated within the bounds of the display. If so, it notifies the InputDispatcher about the touch event.

If there are no touching tools but there is at least one hovering tool, the InputReader notifies the InputDispatcher about the hover event.

If the touch device type is pointer, the InputReader performs pointer gesture detection, moves the pointer and spots accordingly and notifies the InputDispatcher about the pointer event.

The InputDispatcher uses the WindowManagerPolicy to determine whether the events should be dispatched and whether they should wake the device. Then, the InputDispatcher delivers the events to the appropriate applications.

Touch Device Configuration

Touch device behavior is determined by the device's axes, buttons, input properties, input device configuration, virtual key map and key layout.

Refer to the following sections for more details about the files that participate in keyboard configuration:

Input Device Configuration Files

Virtual Key Map Files

Properties

The system relies on many input device configuration properties to configure and calibrate touch device behavior.

One reason for this is that the device drivers for touch devices often report the characteristics of touches using device-specific units.

For example, many touch devices measure the touch contact area using an internal device-specific scale, such as the total number of sensor nodes that were triggered by the touch. This raw size value would not be meaningful applications because they would need to know about the physical size and other characteristics of the touch device sensor nodes.

The system uses calibration parameters encoded in input device configuration files to decode, transform, and normalize the values reported by the touch device into a simpler standard representation that applications can understand.

Documentation Conventions

For documentation purposes, we will use the following conventions to describe the values used by the system during the calibration process.

Raw Axis Values

The following expressions denote the raw values reported by the touch device driver as EV\_ABS events.

raw.x

The value of the ABS\_X or ABS\_MT\_POSITION\_X axis.

raw.y

The value of the ABS\_Y or ABS\_MT\_POSITION\_Y axis.

raw.pressure

The value of the ABS\_PRESSURE or ABS\_MT\_PRESSURE axis, or 0 if not available.

raw.touchMajor

The value of the ABS\_MT\_TOUCH\_MAJOR axis, or 0 if not available.

raw.touchMinor

The value of the ABS\_MT\_TOUCH\_MINOR axis, or raw.touchMajor if not available.

raw.toolMajor

The value of the ABS\_TOOL\_WIDTH or ABS\_MT\_WIDTH\_MAJOR axis, or 0 if not available.

raw.toolMinor

The value of the ABS\_MT\_WIDTH\_MINOR axis, or raw.toolMajor if not available.

raw.orientation

The value of the ABS\_MT\_ORIENTATION axis, or 0 if not available.

raw.distance

The value of the ABS\_DISTANCE or ABS\_MT\_DISTANCE axis, or 0 if not available.

raw.tiltX

The value of the ABS\_TILT\_X axis, or 0 if not available.

raw.tiltY

The value of the ABS\_TILT\_Y axis, or 0 if not available.

Raw Axis Ranges

The following expressions denote the bounds of raw values. They are obtained by calling EVIOCGABS ioctl for each axis.

raw.\*.min

The inclusive minimum value of the raw axis.

raw.\*.max

The inclusive maximum value of the raw axis.

raw.\*.range

Equivalent to raw.\*.max - raw.\*.min.

raw.\*.fuzz

The accuracy of the raw axis. eg. fuzz = 1 implies values are accurate to +/- 1 unit.

raw.width

The inclusive width of the touch area, equivalent to raw.x.range + 1.

raw.height

The inclusive height of the touch area, equivalent to raw.y.range + 1.

Output Ranges

The following expressions denote the characteristics of the output coordinate system. The system uses linear interpolation to translate touch position information from the surface units used by the touch device into the output units that will be reported to applications such as display pixels.

output.width

The output width. For touch screens (associated with a display), this is the display width in pixels. For touch pads (not associated with a display), the output width equals raw.width, indicating that no interpolation will be performed.

output.height

The output height. For touch screens (associated with a display), this is the display height in pixels. For touch pads (not associated with a display), the output height equals raw.height, indicating that no interpolation will be performed.

output.diag

The diagonal length of the output coordinate system, equivalent to sqrt(output.width ^2 + output.height ^2).

Basic Configuration

The touch input mapper uses many configuration properties in the input device configuration file to specify calibration values. The following table describes some general purpose configuration properties. All other properties are described in the following sections along with the fields they are used to calibrate.

touch.deviceType

Definition: touch.deviceType = touchScreen | touchPad | pointer | default

Specifies the touch device type.

If the value is touchScreen, the touch device is a touch screen associated with a display.

If the value is touchPad, the touch device is a touch pad not associated with a display.

If the value is pointer, the touch device is a touch pad not associated with a display, and its motions are used for indirect multi-touch pointer gestures.

If the value is default, the system automatically detects the device type according to the classification algorithm.

Refer to the Classification section for more details about how the device type influences the behavior of the touch device.

Prior to Honeycomb, all touch devices were assumed to be touch screens.

touch.orientationAware

Definition: touch.orientationAware = 0 | 1

Specifies whether the touch device should react to display orientation changes.

If the value is 1, touch positions reported by the touch device are rotated whenever the display orientation changes.

If the value is 0, touch positions reported by the touch device are immune to display orientation changes.

The default value is 1 if the device is a touch screen, 0 otherwise.

The system distinguishes between internal and external touch screens and displays. An orientation aware internal touch screen is rotated based on the orientation of the internal display. An orientation aware external touch screen is rotated based on the orientation of the external display.

Orientation awareness is used to support rotation of touch screens on devices like the Nexus One. For example, when the device is rotated clockwise 90 degrees from its natural orientation, the absolute positions of touches are remapped such that a touch in the top-left corner of the touch screen's absolute coordinate system is reported as a touch in the top-left corner of the display's rotated coordinate system. This is done so that touches are reported with the same coordinate system that applications use to draw their visual elements.

Prior to Honeycomb, all touch devices were assumed to be orientation aware.

touch.gestureMode

Definition: touch.gestureMode = pointer | spots | default

Specifies the presentation mode for pointer gestures. This configuration property is only relevant when the touch device is of type pointer.

If the value is pointer, the touch pad gestures are presented by way of a cursor similar to a mouse pointer.

If the value is spots, the touch pad gestures are presented by an anchor that represents the centroid of the gesture and a set of circular spots that represent the position of individual fingers.

The default value is pointer when the INPUT\_PROP\_SEMI\_MT input property is set, or spots otherwise.

X and Y Fields

The X and Y fields provide positional information for the center of the contact area.

Calculation

The calculation is straightforward: positional information from the touch driver is linearly interpolated to the output coordinate system.

xScale = output.width / raw.width

yScale = output.height / raw.height

If not orientation aware or screen rotation is 0 degrees:

output.x = (raw.x - raw.x.min) \* xScale

output.y = (raw.y - raw.y.min) \* yScale

Else If rotation is 90 degrees:

output.x = (raw.y - raw.y.min) \* yScale

output.y = (raw.x.max - raw.x) \* xScale

Else If rotation is 180 degrees:

output.x = (raw.x.max - raw.x) \* xScale

output.y = (raw.y.max - raw.y) \* yScale

Else If rotation is 270 degrees:

output.x = (raw.y.max - raw.y) \* yScale

output.y = (raw.x - raw.x.min) \* xScale

End If

TouchMajor, TouchMinor, ToolMajor, ToolMinor, Size Fields

The TouchMajor and TouchMinor fields describe the approximate dimensions of the contact area in output units (pixels).

The ToolMajor and ToolMinor fields describe the approximate dimensions of the tool itself in output units (pixels).

The Size field describes the normalized size of the touch relative to the largest possible touch that the touch device can sense. The smallest possible normalized size is 0.0 (no contact, or it is unmeasurable), and the largest possible normalized size is 1.0 (sensor area is saturated).

When both the approximate length and breadth can be measured, then the TouchMajor field specifies the longer dimension and the TouchMinor field specifies the shorter dimension of the contact area. When only the approximate diameter of the contact area can be measured, then the TouchMajor and TouchMinor fields will be equal.

Likewise, the ToolMajor field specifies the longer dimension and the ToolMinor field specifies the shorter dimension of the tool's cross-sectional area.

If the touch size is unavailable but the tool size is available, then the tool size will be set equal to the touch size. Conversely, if the tool size is unavailable but the touch size is available, then the touch size will be set equal to the tool size.

Touch devices measure or report the touch size and tool size in various ways. The current implementation supports three different kinds of measurements: diameter, area, and geometric bounding box in surface units.

touch.size.calibration

Definition: touch.size.calibration = none | geometric | diameter | area | default

Specifies the kind of measurement used by the touch driver to report the touch size and tool size.

If the value is none, the size is set to zero.

If the value is geometric, the size is assumed to be specified in the same surface units as the position, so it is scaled in the same manner.

If the value is diameter, the size is assumed to be proportional to the diameter (width) of the touch or tool.

If the value is area, the size is assumed to be proportional to the area of the touch or tool.

If the value is default, the system uses the geometric calibration if the raw.touchMajor or raw.toolMajor axis is available, otherwise it uses the none calibration.

touch.size.scale

Definition: touch.size.scale = <a non-negative floating point number>

Specifies a constant scale factor used in the calibration.

The default value is 1.0.

touch.size.bias

Definition: touch.size.bias = <a non-negative floating point number>

Specifies a constant bias value used in the calibration.

The default value is 0.0.

touch.size.isSummed

Definition: touch.size.isSummed = 0 | 1

Specifies whether the size is reported as the sum of the sizes of all active contacts, or is reported individually for each contact.

If the value is 1, the reported size will be divided by the number of contacts prior to use.

If the value is 0, the reported size will be used as is.

The default value is 0.

Some touch devices, particularly "Semi-MT" devices cannot distinguish the individual dimensions of multiple contacts so they report a size measurement that represents their total area or width. This property should only be set to 1 for such devices. If in doubt, set this value to 0.

Calculation

The calculation of the TouchMajor, TouchMinor, ToolMajor, ToolMinor and Size fields depends on the specified calibration parameters.

If raw.touchMajor and raw.toolMajor are available:

touchMajor = raw.touchMajor

touchMinor = raw.touchMinor

toolMajor = raw.toolMajor

toolMinor = raw.toolMinor

Else If raw.touchMajor is available:

toolMajor = touchMajor = raw.touchMajor

toolMinor = touchMinor = raw.touchMinor

Else If raw.toolMajor is available:

touchMajor = toolMajor = raw.toolMajor

touchMinor = toolMinor = raw.toolMinor

Else

touchMajor = toolMajor = 0

touchMinor = toolMinor = 0

size = 0

End If

size = avg(touchMajor, touchMinor)

If touch.size.isSummed == 1:

touchMajor = touchMajor / numberOfActiveContacts

touchMinor = touchMinor / numberOfActiveContacts

toolMajor = toolMajor / numberOfActiveContacts

toolMinor = toolMinor / numberOfActiveContacts

size = size / numberOfActiveContacts

End If

If touch.size.calibration == "none":

touchMajor = toolMajor = 0

touchMinor = toolMinor = 0

size = 0

Else If touch.size.calibration == "geometric":

outputScale = average(output.width / raw.width, output.height / raw.height)

touchMajor = touchMajor \* outputScale

touchMinor = touchMinor \* outputScale

toolMajor = toolMajor \* outputScale

toolMinor = toolMinor \* outputScale

Else If touch.size.calibration == "area":

touchMajor = sqrt(touchMajor)

touchMinor = touchMajor

toolMajor = sqrt(toolMajor)

toolMinor = toolMajor

Else If touch.size.calibration == "diameter":

touchMinor = touchMajor

toolMinor = toolMajor

End If

If touchMajor != 0:

output.touchMajor = touchMajor \* touch.size.scale + touch.size.bias

Else

output.touchMajor = 0

End If

If touchMinor != 0:

output.touchMinor = touchMinor \* touch.size.scale + touch.size.bias

Else

output.touchMinor = 0

End If

If toolMajor != 0:

output.toolMajor = toolMajor \* touch.size.scale + touch.size.bias

Else

output.toolMajor = 0

End If

If toolMinor != 0:

output.toolMinor = toolMinor \* touch.size.scale + touch.size.bias

Else

output.toolMinor = 0

End If

output.size = size

Pressure Field

The Pressure field describes the approximate physical pressure applied to the touch device as a normalized value between 0.0 (no touch) and 1.0 (full force).

A zero pressure indicates that the tool is hovering.

touch.pressure.calibration

Definition: touch.pressure.calibration = none | physical | amplitude | default

Specifies the kind of measurement used by the touch driver to report the pressure.

If the value is none, the pressure is unknown so it is set to 1.0 when touching and 0.0 when hovering.

If the value is physical, the pressure axis is assumed to measure the actual physical intensity of pressure applied to the touch pad.

If the value is amplitude, the pressure axis is assumed to measure the signal amplitude, which is related to the size of the contact and the pressure applied.

If the value is default, the system uses the physical calibration if the pressure axis available, otherwise uses none.

touch.pressure.scale

Definition: touch.pressure.scale = <a non-negative floating point number>

Specifies a constant scale factor used in the calibration.

The default value is 1.0 / raw.pressure.max.

Calculation

The calculation of the Pressure field depends on the specified calibration parameters.

If touch.pressure.calibration == "physical" or "amplitude":

output.pressure = raw.pressure \* touch.pressure.scale

Else

If hovering:

output.pressure = 0

Else

output.pressure = 1

End If

End If

Orientation and Tilt Fields

The Orientation field describes the orientation of the touch and tool as an angular measurement. An orientation of 0 indicates that the major axis is oriented vertically, -PI/2 indicates that the major axis is oriented to the left, PI/2 indicates that the major axis is oriented to the right. When a stylus tool is present, the orientation range may be described in a full circle range from -PI or PI.

The Tilt field describes the inclination of the tool as an angular measurement. A tilt of 0 indicates that the tool is perpendicular to the surface. A tilt of PI/2 indicates that the tool is flat on the surface.

touch.orientation.calibration

Definition: touch.orientation.calibration = none | interpolated | vector | default

Specifies the kind of measurement used by the touch driver to report the orientation.

If the value is none, the orientation is unknown so it is set to 0.

If the value is interpolated, the orientation is linearly interpolated such that a raw value of raw.orientation.min maps to -PI/2 and a raw value of raw.orientation.max maps to PI/2. The center value of (raw.orientation.min + raw.orientation.max) / 2 maps to 0.

If the value is vector, the orientation is interpreted as a packed vector consisiting of two signed 4-bit fields. This representation is used on Atmel Object Based Protocol parts. When decoded, the vector yields an orientation angle and confidence magnitude. The confidence magnitude is used to scale the size information, unless it is geometric.

If the value is default, the system uses the interpolated calibration if the orientation axis available, otherwise uses none.

Calculation

The calculation of the Orientation and Tilt fields depends on the specified calibration parameters and available input.

If touch.tiltX and touch.tiltY are available:

tiltXCenter = average(raw.tiltX.min, raw.tiltX.max)

tiltYCenter = average(raw.tiltY.min, raw.tiltY.max)

tiltXAngle = (raw.tiltX - tiltXCenter) \* PI / 180

tiltYAngle = (raw.tiltY - tiltYCenter) \* PI / 180

output.orientation = atan2(-sin(tiltXAngle), sinf(tiltYAngle))

output.tilt = acos(cos(tiltXAngle) \* cos(tiltYAngle))

Else If touch.orientation.calibration == "interpolated":

center = average(raw.orientation.min, raw.orientation.max)

output.orientation = PI / (raw.orientation.max - raw.orientation.min)

output.tilt = 0

Else If touch.orientation.calibration == "vector":

c1 = (raw.orientation & 0xF0) >> 4

c2 = raw.orientation & 0x0F

If c1 != 0 or c2 != 0:

If c1 >= 8 Then c1 = c1 - 16

If c2 >= 8 Then c2 = c2 - 16

angle = atan2(c1, c2) / 2

confidence = sqrt(c1\*c1 + c2\*c2)

output.orientation = angle

If touch.size.calibration == "diameter" or "area":

scale = 1.0 + confidence / 16

output.touchMajor \*= scale

output.touchMinor /= scale

output.toolMajor \*= scale

output.toolMinor /= scale

End If

Else

output.orientation = 0

End If

output.tilt = 0

Else

output.orientation = 0

output.tilt = 0

End If

If orientation aware:

If screen rotation is 90 degrees:

output.orientation = output.orientation - PI / 2

Else If screen rotation is 270 degrees:

output.orientation = output.orientation + PI / 2

End If

End If

Distance Field

The Distance field describes the distance between the tool and the touch device surface. A value of 0.0 indicates direct contact and larger values indicate increasing distance from the surface.

touch.distance.calibration

Definition: touch.distance.calibration = none | scaled | default

Specifies the kind of measurement used by the touch driver to report the distance.

If the value is none, the distance is unknown so it is set to 0.

If the value is scaled, the reported distance is multiplied by a constant scale factor.

If the value is default, the system uses the scaled calibration if the distance axis available, otherwise uses none.

touch.distance.scale

Definition: touch.distance.scale = <a non-negative floating point number>

Specifies a constant scale factor used in the calibration.

The default value is 1.0.

Calculation

The calculation of the Distance field depends on the specified calibration parameters.

If touch.distance.calibration == "scaled":

output.distance = raw.distance \* touch.distance.scale

Else

output.distance = 0

End If

Example

# Input device configuration file for a touch screen that supports pressure,

# size and orientation. The pressure and size scale factors were obtained

# by measuring the characteristics of the device itself and deriving

# useful approximations based on the resolution of the touch sensor and the

# display.

#

# Note that these parameters are specific to a particular device model.

# Different parameters will need to be used for other devices.

# Basic Parameters

touch.deviceType = touchScreen

touch.orientationAware = 1

# Size

# Based on empirical measurements, we estimate the size of the contact

# using size = sqrt(area) \* 28 + 0.

touch.size.calibration = area

touch.size.scale = 28

touch.size.bias = 0

touch.size.isSummed = 0

# Pressure

# Driver reports signal strength as pressure.

#

# A normal index finger touch typically registers about 80 signal strength

# units although we don't expect these values to be accurate.

touch.pressure.calibration = amplitude

touch.pressure.scale = 0.0125

# Orientation

touch.orientation.calibration = vector

Compatibility Notes

The configuration properties for touch devices changed significantly in Android Ice Cream Sandwich 4.0. All input device configuration files for touch devices must be updated to use the new configuration properties.

Older touch device drivers may also need to be updated.

Virtual Key Map Files

Touch devices are often used to implement virtual keys.

There are several ways of doing this, depending on the capabilities of the touch controller. Some touch controllers can be directly configured to implement soft keys by setting firmware registers. Other times it is desirable to perform the mapping from touch coordinates to key codes in software.

When virtual keys are implemented in software, the kernel must export a virtual key map file called virtualkeys.<devicename> as a board property. For example, if the touch screen device drivers reports its name as "touchyfeely" then the virtual key map file must have the path /sys/board\_properties/virtualkeys.touchyfeely.

A virtual key map file describes the coordinates and Linux key codes of virtual keys on the touch screen.

In addition to the virtual key map file, there must be a corresponding key layout file and key character map file to map the Linux key codes to Android key codes and to specify the type of the keyboard device (usually SPECIAL\_FUNCTION).

Syntax

A virtual key map file is a plain text file consisting of a sequence of virtual key layout descriptions either separated by newlines or by colons.

Comment lines begin with '#' and continue to the end of the line.

Each virtual key is described by 6 colon-delimited components:

0x01: A version code. Must always be 0x01.

<Linux key code>: The Linux key code of the virtual key.

<centerX>: The X pixel coordinate of the center of the virtual key.

<centerY>: The Y pixel coordinate of the center of the virtual key.

<width>: The width of the virtual key in pixels.

<height>: The height of the virtual key in pixels.

All coordinates and sizes are specified in terms of the display coordinate system.

Here is a virtual key map file all written on one line.

# All on one line

0x01:158:55:835:90:55:0x01:139:172:835:125:55:0x01:102:298:835:115:55:0x01:217:412:835:95:55

The same virtual key map file can also be written on multiple lines.

# One key per line

0x01:158:55:835:90:55

0x01:139:172:835:125:55

0x01:102:298:835:115:55

0x01:217:412:835:95:55

In the above example, the touch screen has a resolution of 480x800. Accordingly, all of the virtual keys have a <centerY> coordinate of 835, which is a little bit below the visible area of the touch screen.

The first key has a Linux scan code of 158 (KEY\_BACK), centerX of 55, centerY of 835, width of 90 and height of 55.

Example

Virtual key map file: /sys/board\_properties/virtualkeys.touchyfeely.

0x01:158:55:835:90:55

0x01:139:172:835:125:55

0x01:102:298:835:115:55

0x01:217:412:835:95:55

Key layout file: /system/usr/keylayout/touchyfeely.kl.

key 158 BACK

key 139 MENU

key 102 HOME

key 217 SEARCH

Key character map file: /system/usr/keychars/touchyfeely.kcm.

type SPECIAL\_FUNCTION

Indirect Multi-touch Pointer Gestures

In pointer mode, the system interprets the following gestures:

Single finger tap: click.

Single finger motion: move the pointer.

Single finger motion plus button presses: drag the pointer.

Two finger motion both fingers moving in the same direction: drag the area under the pointer in that direction. The pointer itself does not move.

Two finger motion both fingers moving towards each other or apart in different directions: pan/scale/rotate the area surrounding the pointer. The pointer itself does not move.

Multiple finger motion: freeform gesture.