



## Touchscreen Toolkit for LabVIEW Help

June 2014

The Touchscreen Toolkit for LabVIEW Toolkit provides two touch interfaces: a LabVIEW interface to the Windows Touch API and to Touch-Base Ltd's Universal Pointer Device Driver (UPDD).

The Windows Touch/Gesture (WTG) interface does not require any additional drivers and relies on the built-in HID drivers in Windows to properly interface with the touchscreen hardware. The LabVIEW toolkit installs a hook in Windows so that the [WM\\_TOUCH](#) and [WM\\_GESTURE](#) messages are intercepted before being received by the LabVIEW window that initialized the driver. Once intercepted, the touches can be interpreted and filtered and messages are sent back to the LabVIEW application via a user event in the same manner that is done with the UPDD driver. The WTG interface is compatible with both Windows 7 and Windows 8 via the [WM\\_TOUCH](#) and [WM\\_GESTURE](#) messages. Currently, the WTG interface does not expose the Windows 8 only [WM\\_POINTER](#) message. Since Windows 8 reports touch messages directly to the windows process instead of the message queue, the mouse pointer cannot be disabled in Windows 8.

The UPDD interface requires a separate driver (sold separately) which supports multiple Operating Systems and many different touchscreens. This is required for integration with NI standalone chassis, such as the cDAQ-9139. Refer to <http://www.touch-base.com/> or contact [sales@aledyne.com](mailto:sales@aledyne.com). Touch-Base is a leading developer and supplier of touchscreen and pointer device drivers. The Universal Pointer Device Driver is in use on many thousands of devices worldwide. This API provides a LabVIEW wrapper around the UPDD driver and a set of LabVIEW vi's in order to initialize the driver and receive events and touch data when any touchscreen connected is touched.

The Touchscreen Toolkit provides the following features:

- Direct integration with the UPDD driver from Touch-Base AND the Windows 7 Touch API
- User event registration and triggering when a touch occurs on any configured touch screen
- Multiple touch screens can be used simultaneously
- Swipe, zoom, rotate, and pan gestures are recognized and reported with multi-touch hardware
- Multiple touch points (up to 16) are reported with multi-touch hardware
- Ability to disable mouse pointer so touches do not control mouse movement
- Touch coordinates are reported from one or more screens when mouse pointer is disabled
- Receive Windows gesture messages as reported by [WM\\_TOUCH](#) and [WM\\_GESTURE](#)

Please visit <http://touch-base.com/> to purchase and download the UPDD driver for your hardware if using the UPDD APIs in the toolkit. The downloaded UPDD driver will include a TBApi.dll and a UPDD device specific Dll which must both be copied into the directory of this toolkit (<LabVIEW>\vi.lib\Aledyne Engineering\Touchscreen Toolkit). Note that this toolkit references the 32-bit version of the driver so if running on a 64-bit OS, TBApi32.dll will need to be used from the driver installation package and renamed to TBApi.dll when copying to the Toolkit directory.

## Where to Start?

### Windows 7 or 8:



If you have a Windows 7 or 8 machine with a built in multi-touch touchscreen, or have an external Windows supported touchscreen, like the MIMO Magic Touch, start with the following examples:

**Raw Touch coordinates:** WTG\_Example.vi demonstrates how the Windows 7 touch (WM\_TOUCH) and gesture (WM\_GESTURE) messages are captured by the Touchscreen toolkit to monitor multi-touch events from Windows 7. This is the simplest example and a good starting point to see how to interface with the API and see how data is reported from the toolkit.

**Touch enabled Graph:** WTG\_GraphExample.vi demonstrates how to use the touchscreen toolkit to manipulate the x and y scales of a waveform graph. This example detects the zoom, rotate, and pan gestures from the toolkit and modifies the scales of the graph control.

**Touch enabled Picture Control:** WTG\_PictureExample.vi demonstrates how to enable the Windows 7 Gesture Engine and messaging built into the WM\_GESTURE message. This example connects to the touch driver, receives user events that are triggered off of WM\_GESTURE and returns gesture data for Zoom and Rotate and uses the data to manipulate a picture control.

**Detection of Multiple Buttons:** WTG\_MultiButtonExample.vi demonstrates how to use the touchscreen toolkit to detect touch on multiple areas of interest on the front panel simultaneously using the Windows 7 Touch API. A typical application would be to detect touch of two buttons simultaneously to ensure operators hands are away from machinery before starting a dangerous piece of machinery.

**Recognizing multiple gestures in an event handler:** WTG\_Gestures\_Example.vi demonstrates how to use the touchscreen toolkit to process touch and gesture coordinates using the Windows Touch API. This example detects the swipe, zoom, rotate, and pan gestures from the toolkit and shows how to use them to switch images in an image control, rotate and zoom an image, zoom a graph, and detect multiple button presses simultaneously.

### Windows Embedded Standard 7 or Windows XP:



If you are integrating touch to a cDAQ-9138/9139, cRIO-9081/9082 running Windows Embedded Standard 7 (WES7) or you are integrating touch to a Windows XP system, you will need to purchase an additional UPDD touch driver specifically for your touch monitor (sold separately). Please contact [sales@aledyne.com](mailto:sales@aledyne.com) for support on the appropriate driver and installation.

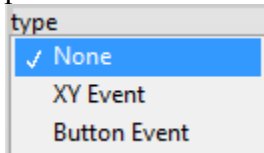
**Raw Touch coordinates:** UPDD\_Example.vi demonstrates how to use the touchscreen toolkit to initialize and monitor events, and get a list of detected devices from the UPDD driver. This is the simplest example and a good starting point to see how to interface with the API and see how data is reported from the toolkit.

**Recognizing multiple gestures in an event handler:** UPDD\_Gestures\_Example.vi demonstrates how to use the touchscreen toolkit to process touch and gesture coordinates using the Windows Touch API. This example detects the swipe, zoom, rotate, and pan gestures from the toolkit and shows how to use them to switch images in an image control, rotate and zoom an image, zoom a graph, and detect multiple button presses simultaneously.

### Touch Data Elements Reported by the Touchscreen Toolkit:

**Type:** Defines if the event reported is an XY event, such as dragging a finger or stylus across the screen, or a Button event, which happens once on a depression or removal of a finger or stylus on the screen.

When a button event occurs, the XY coordinates of the button press or release will also be reported for easy processing. Following a button press, the application will continually receive XY Events with position data until the button is released.



**Tick:** Relative time in ticks that the data was read from the UPDD driver.

**Rawx:** Raw x value of button press/release or drag received from the touch controller normalized to 0-4000 as reported by the UPDD driver. For the WTG API this is the screen coordinate in physical pixels.

**Rawy:** Raw y value of button press/release or drag received from the touch controller normalized to 0-4000 as reported by the UPDD driver. For the WTG API this is the screen coordinate in physical pixels.

**Calx:** The corresponding calibrated x value as reported by the UPDD driver. For the WTG API this is mostly unused except for in Zoom/Rotate mode it provides the center point of the two touches so that the application can zoom about the center point.

**Caly:** The corresponding calibrated y value as reported by the UPDD driver. For the WTG API this is mostly unused except for in Zoom/Rotate mode it provides the center point of the two touches so that the application can zoom about the center point.

**Left:** The state of the left button where 0 is released and 1 is pressed. The left button is used for touches. **This data is only valid for Button Events.**

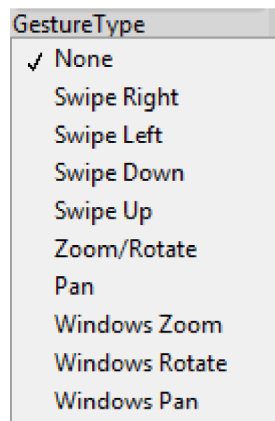
**Right:** The state of the right button where 0 is released and 1 is pressed. **This data is only valid for Button Events.**

**Timed:** Reserved for Future Use.

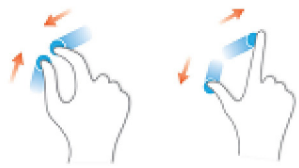
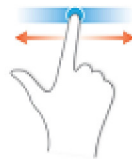
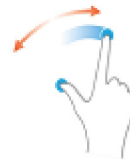
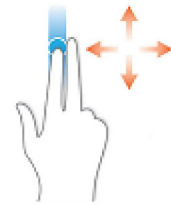
**DeviceHandle:** Used to identify the monitor/device that was pressed in a multi-monitor system. For the UPDD API, this should be used with UPDD\_GetDeviceIndex.vi to identify which monitor is reporting the data. For the WTG API, this is a unique identifier for the monitor that reported the touch event. In a single monitor environment, this is not needed.

**Stylus:** This is the pen/stylus number in a multi-touch application. If the monitor being used supports multi-touch, up to 16 independent pens will be reported based on the capabilities of the hardware. This can be used to create custom gestures using a capacitive touchscreen monitor. In a resistive touch application that only supports one coordinate, this will always be 0.

**GestureType:** The toolkit has built in gesture recognition for swiping, zooming, rotating, and panning. Swipe only requires single touch monitors but zoom, rotate, and pan will only work in multi-touch environments. Zoom and rotate are reported as one gesture but with independent values because zoom and rotate are interpreted as one gesture when using two fingers on a display. The application can monitor either Zoom or Rotate if it is only desired to perform one of the functions. The Windows Zoom, Windows Rotate, and Windows Pan are gesture events triggered by the Windows [WM\\_GESTURE](#) message when Windows specific gestures are enabled using the WTG API. If Pan and Zoom/Rotate are enabled, when the pan threshold is exceeded the Pan gesture will be cancelled and the gesture engine will transition to the Zoom/Rotate gesture if its zoom threshold is exceeded. If the distance between the two touch points is then reduced lower than the zoom threshold, the gesture will be canceled and if the distance is less than the pan threshold, the gesture engine will transition back to the Pan gesture.





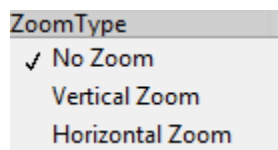
**Zoom In/Out****Swipe****Rotate****Pan**

**Swipe:** Reports the difference in X movement for left to right and right to left swipe gestures and the difference in Y movement for down to up and up to down gestures. This is not reported until the gesture threshold is passed. The threshold can be customized using `UPDD_SetGestureThreshold.vi` or `WTG_SetGestureThreshold.vi`.

**Zoom:** Reports the zoom scale in a multi-touch environment when pinching and expanding motions are detected on the monitor. This is reported as a percentage of the initial spacing between the two touch points for the Aledyne gesture. So if two touch points are initially spaced 1 inch apart and are moved apart to 2 inches, a value of 200% will be reported. The threshold distance to which both fingers must be apart from one another before considering the gesture a zoom can be configured using `UPDD_SetGestureThreshold.vi` or `WTG_SetGestureThreshold.vi`. If a zoom gesture is being reported and the fingers move together again within the set threshold, the zoom gesture will be cancelled. If pan is enabled it will go back to pan mode. For `WM_GESTURE` (Windows Zoom), the zoom quantity is reported as a distance in pixels between the two touch points. The quantity is not converted to a percentage of the initial spacing and, if this is required, it must be done in the user's application. Refer to `WTG_PictureExample.vi` for an example.

**Rotate:** When one finger is rotated around the second finger which remains a pivot point, rotation is detected. The angle of rotation reported correlates to the rotation angle of one finger with respect to the other. The maximum reported angles are -90 to 90 degrees. If the rotation is not sensitive enough in the application, this number can simply be scaled or multiplied by some factor to make the rotation more sensitive to movement of the fingers on the screen.

**ZoomType:** Returns if the zoom orientation is horizontal or vertical based on touch positioning. If the relative angle is greater than 45 degrees from the horizontal axis, a vertical zoom type is returned.



**PanXDelta:** Reports the difference in X movement for left to right and right to left pan gestures. For right to left gestures this will be reported as a negative value. This is not reported until the gesture threshold is passed. The threshold can be customized using `UPDD_SetGestureThreshold.vi` or `WTG_SetGestureThreshold.vi`. If a pan is started and then the fingers are moved apart to commence a zoom gesture, once the pan threshold is exceeded, the gesture engine will automatically cancel the pan and will commence the zoom.

**PanYDelta:** Reports the difference in Y movement for down to up and up to pan gestures. For down to up gestures this will be reported as a negative value. This is not reported until the gesture threshold is

passed. The threshold can be customized using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi. If a pan is started and then the fingers are moved apart to commence a zoom gesture, once the pan threshold is exceeded, the gesture engine will automatically cancel the pan and will commence the zoom.

Refer to "<LabVIEW>\examples\Aledyne Engineering\Touchscreen Toolkit\" for examples of how to interface to the UPDD and WTG driver.

To provide feedback or request support, contact us [here](#).

© 2012–2014 Aledyne Engineering. All rights reserved.

---



## Windows Touch/Gestures Interface Help

June 2014

Part of the Touchscreen Toolkit provides a LabVIEW interface to the Windows 7 touch API. Since Windows 7 already provides support for many touchscreen monitors over a standard HID interface, the WTG APIs of the toolkit provide a means of hooking into the Windows 7 Window Touch ([WM\\_TOUCH](#)) and gesture ([WM\\_GESTURE](#)) messages. The WTG interface is compatible with both Windows 7 and Windows 8 via the [WM\\_TOUCH](#) and [WM\\_GESTURE](#) messages. Currently, the WTG interface does not expose the Windows 8 only [WM\\_POINTER](#) message. Since Windows 8 reports touch messages directly to the windows process instead of the message queue, the mouse pointer cannot be disabled in Windows 8.

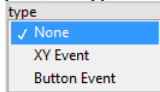
The WTG API provides the following features:

- Direct integration with Windows 7 touch API
- User event registration and triggering when a touch occurs on any connected touch screen
- Multiple touch screens can be used simultaneously
- Custom swipe, zoom, rotate, and pan gestures are recognized and reported with multi-touch hardware if [WM\\_TOUCH](#) is enabled
- Ability to disable mouse pointer so touches do not trigger left/right button events
- The standard Windows 7 [WM\\_GESTURE](#) messages can be hooked including zoom, rotate, and pan.

Please visit the Microsoft MSDN pages for above for details on the Windows touch API.

### Touch Data Elements Reported by the Toolkit:

**Type:** Defines if the event reported is an XY event, such as dragging a finger or stylus across the screen, or a Button event, which happens once on a depression or removal of a finger or stylus on the screen. When a button event occurs, the XY coordinates of the button press or release will also be reported for easy processing. Following a button press, the application will continually receive XY Events with position data until the button is released.



**Tick:** Relative time in ticks that the data was read from the UPDD driver.

**Rawx:** Raw x value of button press/release or drag received from the touch controller normalized to 0-4000 as reported by the UPDD driver. For the WTG API this is the screen coordinate in physical pixels.

**Rawy:** Raw y value of button press/release or drag received from the touch controller normalized to 0-4000 as reported by the UPDD driver. For the WTG API this is the screen coordinate in physical pixels.

**Calx:** The corresponding calibrated x value as reported by the UPDD driver. For the WTG API this is mostly unused except for in Zoom/Rotate mode it provides the center point of the two touches so that the application can zoom about the center point.

**Caly:** The corresponding calibrated y value as reported by the UPDD driver. For the WTG API this is mostly unused except for in Zoom/Rotate mode it provides the center point of the two touches so that the application can zoom about the center point.

**Left:** The state of the left button where 0 is released and 1 is pressed. The left button is used for touches. **This data is only valid for Button Events.**

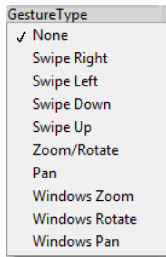
**Right:** The state of the right button where 0 is released and 1 is pressed. **This data is only valid for Button Events.**

**Timed:** Reserved for Future Use.

**DeviceHandle:** Used to identify the monitor/device that was pressed in a multi-monitor system. For the UPDD API, this should be used with UPDD\_GetDeviceIndex.vi to identify which monitor is reporting the data. For the WTG API, this is a unique identifier for the monitor that reported the touch event. In a single monitor environment, this is not needed.

**Stylus:** This is the pen/stylus number in a multi-touch application. If the monitor being used supports multi-touch, up to 16 independent pens will be reported based on the capabilities of the hardware. This can be used to create custom gestures using a capacitive touchscreen monitor. In a resistive touch application that only supports one coordinate, this will always be 0.

**GestureType:** The toolkit has built in gesture recognition for swiping, zooming, rotating, and panning. Swipe only requires single touch monitors but zoom, rotate, and pan will only work in multi-touch environments. Zoom and rotate are reported as one gesture but with independent values because zoom and rotate are interpreted as one gesture when using two fingers on a display. The application can monitor either Zoom or Rotate if it is only desired to perform one of the functions. The Windows Zoom, Windows Rotate, and Windows Pan are gesture events triggered by the Windows [WM\\_GESTURE](#) message when Windows specific gestures are enabled using the WTG API. If Pan and Zoom/Rotate are enabled, when the pan threshold is exceeded the Pan gesture will be cancelled and the gesture engine will transition to the Zoom/Rotate gesture if its zoom threshold is exceeded. If the distance between the two touch points is then reduced lower than the zoom threshold, the gesture will be canceled and if the distance is less than the pan threshold, the gesture engine will transition back to the Pan gesture.

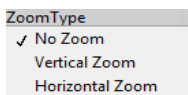
**Zoom In/Out****Swipe****Rotate****Pan**

**Swipe:** Reports the difference in X movement for left to right and right to left swipe gestures and the difference in Y movement for down to up and up to down gestures. This is not reported until the gesture threshold is passed. The threshold can be customized using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi.

**Zoom:** Reports the zoom scale in a multi-touch environment when pinching and expanding motions are detected on the monitor. This is reported as a percentage of the initial spacing between the two touch points for the Aledyne gesture. So if two touch points are initially spaced 1 inch apart and are moved apart to 2 inches, a value of 200% will be reported. The threshold distance to which both fingers must be apart from one another before considering the gesture a zoom can be configured using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi. If a zoom gesture is being reported and the fingers move together again within the set threshold, the zoom gesture will be cancelled. If pan is enabled it will go back to pan mode. For WM\_GESTURE (Windows Zoom), the zoom quantity is reported as a distance in pixels between the two touch points. The quantity is not converted to a percentage of the initial spacing and, if this is required, it must be done in the user's application. Refer to WTG\_PictureExample.vi for an example.

**Rotate:** When one finger is rotated around the second finger which remains a pivot point, rotation is detected. The angle of rotation reported correlates to the rotation angle of one finger with respect to the other. The maximum reported angles are -90 to 90 degrees. If the rotation is not sensitive enough in the application, this number can simply be scaled or multiplied by some factor to make the rotation more sensitive to movement of the fingers on the screen.

**ZoomType:** Returns if the zoom orientation is horizontal or vertical based on touch positioning. If the relative angle is greater than 45 degrees from the horizontal axis, a vertical zoom type is returned.



**PanXDelta:** Reports the difference in X movement for left to right and right to left pan gestures. For right to left gestures this will be reported as a negative value. This is not reported until the gesture threshold is passed. The threshold can be customized using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi. If a pan is started and then the fingers are moved apart to commence a zoom gesture, once the pan threshold is exceeded, the gesture engine will automatically cancel the pan and will commence the zoom.

**PanYDelta:** Reports the difference in Y movement for down to up and up to pan gestures. For down to up gestures this will be reported as a negative value. This is not reported until the gesture threshold is passed. The threshold can be customized using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi. If a pan is started and then the fingers are moved apart to commence a zoom gesture, once the pan threshold is exceeded, the gesture engine will automatically cancel the pan and will commence the zoom.

Refer to "<LabVIEW>\examples\Aledyne Engineering\Touchscreen Toolkit\" for examples of how to interface to the UPDD and WTG driver.

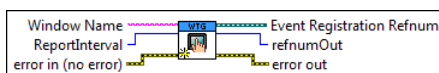
To provide feedback or request support, contact us [here](#).

© 2012–2014 Aledyne Engineering. All rights reserved.

## WTG\_Initialize VI

**Installed With:** LabVIEW

Creates a user event and sends the reference to the WTG driver for registration of the callback. Also sets a Windows hook to receive WM\_TOUCH and WM\_GESTURE messages that are directed to LabVIEW from Windows. The Window that is specified is registered to receive touch messages from. If no window name is specified, the highest caller in the call chain is used for the touch enabled window. By default RegisterTouchWindow() is called which will redirect WM\_TOUCH messages but will stop WM\_GESTURE messages. To enable Windows 7 gestures, this can be done through WTG\_GestureFilter.vi. By default with WM\_TOUCH enabled, the Aledyne gesture engine is enabled. The interval at which user events are reported can also be specified by **ReportInterval**. If this is set to 1, every received Windows event will be passed to LabVIEW. If this is 2, then every other event will be passed to LabVIEW. This can be increased to reduce the amount of interrupts generated by the driver. By default this is set to 1.



**Window Name** define the window to register for touches. If no window name is specified, the highest caller in the call chain is used for the touch enabled window. If multiple windows need to be enabled for touch, it is recommended that each window Initializes, then Closes its connection to the touch API before another window

registers for touch since only one window at a time can be registered for touch.

**U32I** **ReportInterval** defines how often to send touch messages from the driver up to the LabVIEW user event. By default this is 1 so that every message is sent. Messages are sent about every 5-10ms. This can be increased so that LabVIEW is not interrupted too often. The maximum is 10.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ETF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**I32I** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ABC** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ETF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**I32I** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ABC** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**U8** **Event Registration Refnum** is user event registration that should be wired to an events structure's event registration terminal.

**U8** **refnumOut** is the reference to the driver instance.

## WTG\_MousePointerDisable VI

Installed With: LabVIEW

When using the Windows 7 touch API VIs, this will force discarding of mouse button events for touch related messages that are sent to the owning window. The discarded messages include WM\_LBUTTONDOWN, WM\_LBUTTONUP, WM\_RBUTTONDOWN, and WM\_RBUTTONUP.



**U8** **refnum** is the reference to the driver instance.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ETF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**I32I** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ABC** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ETF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**I32I** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ABC** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**U8** **refnumOut** is the reference to the UPDD instance.

## WTG\_MousePointerEnable VI

Installed With: LabVIEW

When using the Windows 7 touch API VIs, this will re-enable mouse button events for touch related messages that are sent to the owning window.



**U8** **refnum** is the reference to the driver instance.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ETF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**I32I** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ABC** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**ETF**

**ETF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning)

gives more information about the error displayed.

**I32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

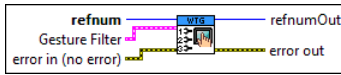
**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**U8** **refnumOut** is the reference to the UPDD instance.

## WTG\_GestureFilter VI

Installed With: LabVIEW

Calls into the Windows Touch/Gesture driver and enables/disables gesture features. If Windows 7 Gestures are enabled, all other custom gesture types are disabled and the Windows WM\_GESTURE messages are reported to the application. This also disables the WM\_TOUCH messages the Windows reports so the application will stop receiving touch coordinate data. If Windows 7 Gestures are disabled, the Aledyne custom gesture engine becomes enabled and multi-touch data is reported.



### Gesture Filter

**TF** **W7** defines if Windows 7 Gestures are enabled. If true, Windows 7 WM\_GESTURE messages are received and WM\_TOUCH and the Aledyne gesture engine is disabled. This setting does not do anything for the UPDD instance. Windows 7 gestures are automatically disabled and this setting is ignored for UPDD. This can only be enabled for the WTG instance.

**TF** **Swipe** defines if the Aledyne swipe gesture is enabled. This can only be enabled if W7 is disabled.

**TF** **Zoom/Rotate** defines if the Aledyne zoom/rotate gesture is enabled. This can only be enabled if W7 is disabled.

**TF** **Pan** defines if the Aledyne pan gesture is enabled. This can only be enabled if W7 is disabled.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**E** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**I32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**E** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**I32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**U8** **refnum** is the reference to the driver instance.

**U8** **refnumOut** is the reference to the driver instance.

## WTG\_SetGestureThreshold VI

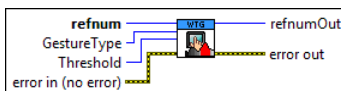
Installed With: LabVIEW

Sets the thresholds of gestures for finer tuning of gesture response. Currently, the only gestures that allow fine tuning are the Aledyne Swipe and Pan. These thresholds are defined in pixels as reported by the WM\_TOUCH coordinates.

For Swipe, when a single finger is swiped across the screen, once it has moved greater than the specified distance in pixels, the API will start registering the gesture. The default for the Swipe threshold is 100 pixels.

For Pan, when two fingers are moved across the screen, and the two fingers start with a distance apart that is less than the specified distance in pixels, the API will start registering the gesture. The default for the Pan threshold is 100 pixels.

For Zoom, when a zoom is started by holding two fingers close together, the zoom percentage will stay at 100% until the distance the fingers are moved apart is greater than the specified distance in pixels. This prevents the start zoom to be really small when a zoom is started causing a large zoom percentage for a small movement apart. The default for the Zoom threshold is 100 pixels.



**U8** **refnum** is the reference to the driver instance.

**E** **GestureType** is the type of gesture to be configured. Currently, the only gesture thresholds that can be configured are the Aledyne swipe and pan.

**I32** **Threshold** is the threshold to set for the gesture specified. The default for swipe is 100 pixels and the default for pan is 100 pixels.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**E** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



**E32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**EFF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**E32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

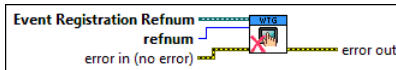
**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**UB** **refnumOut** is the reference to the UPDD instance.

## WTG\_Close VI

Installed With: LabVIEW

Unregisters the event and closes the connection to WTG driver. Also unhooks the Windows hook for monitoring WM messages.



**UB** **Event Registration Refnum**

**UB** **refnum** is the reference to the driver instance created by UPDD\_Initialize.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**EFF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**E32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**EFF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

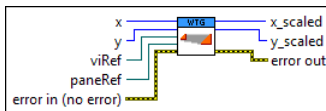
**E32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## WTG\_Scale VI

Installed With: LabVIEW

Scales the input x-y coordinates received from the windows touch driver to map to the coordinates of a front panel VI and pane. Note that the top-left coordinate of the touch panel will be mapped to the top-left coordinate of the VI pane referenced.



**UB** **viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.

**U32** **x** is the physical x coordinate reported by the Windows Touch API.

**U32** **y** is the physical y coordinate reported by the Windows Touch API.

**UB** **paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**EFF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**E32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**EFF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**E32** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

displayed.

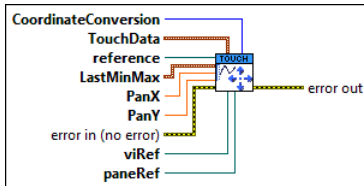
**x\_scaled** is the scaled x LabVIEW panel coordinate.

**y\_scaled** is the scaled y LabVIEW panel coordinate.

## PanGraph VI

Installed With: LabVIEW

Controls panning of a graph by scaling the min and max ranges of the x and y scales relative to the physical x/y coordinates provided to the input. If the x/y coordinates input map to a graph coordinate outside the current range, the graph will not be panned. Panning will only be executed if the touches are within the bounds of the graph control. **CoordinateConversion** identifies if the caller is using Windows Touch (WTG) or the UPDD driver as this determines how to map the physical coordinates to LabVIEW pane coordinates.



**TouchData** is the current touch data reported by the WTG or UPDD API. This is used to determine if the current touch is within the bounds of the graph control.



**reference** to a graph control used to change the scales programatically for panning.

**LastMinMax** is the last locked in graph range used as input to modify the scales based on **PanX** and **PanY**. This should remain the same from the beginning of a touch pan to the end of a touch pan and is used as a reference for modifying the scales.



The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



**PanX** is a scaling factor to apply to the x scale minimum and maximum. The difference between the max and min will be scaled by this amount.



**PanY** is a scaling factor to apply to the y scale minimum and maximum. The difference between the max and min will be scaled by this amount.



**viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.



**paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.

**CoordinateConversion** is used to identify if the Windows Touch (WTG) or UPDD API is being used. This is used to determine the proper mapping of physical coordinates to panel coordinates. The UPDD driver reports touch coordinates normalized to a scale of 0-4000 whereas the WTG driver reports touch coordinates relative to physical pixels.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

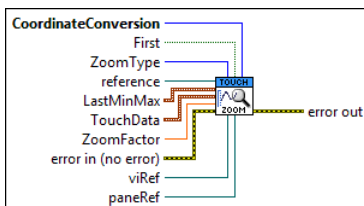
The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## GraphZoomAtPoint VI

Installed With: LabVIEW

Controls zooming of a graph by scaling the min and max ranges of the x and y scales relative to the physical x/y coordinates provided to the input. If the x/y coordinates input map to a graph coordinate outside the current range, the graph will not be zoomed. Zooming will only be executed if the touches are within the bounds of the graph control. Zoom is executed about the input x/y coordinate of **TouchData**. If **First** is true, the center point is used from calx/caly of **TouchData** and is stored for following calls to activate zooming about a center point. **CoordinateConversion** identifies if the caller is using Windows Touch (WTG) or the UPDD driver as this determines how to map the physical coordinates to LabVIEW pane coordinates.



**reference** to a graph control used to change the scales programatically.



**viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.

**paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.

**ZoomType** defines either an x axis or y axis zoom.

**LastMinMax** is the last locked in graph range used as input to modify the scales based on **PanX** and **PanY**. This should remain the same from the beginning of a touch pan to the end of a touch pan and is used as a reference for modifying the scales.

**ZoomFactor** is the zoom amount as a percentage where 100 equates to 100% of the image size.

**First** should be set to TRUE on first touch. This forces the touch coordinate to be stored internally to this VI in order to control zooming about the point defined by calx and caly of **TouchData**.

**CoordinateConversion** is used to identify if the Windows Touch (WTG) or UPDD API is being used. This is used to determine the proper mapping of physical coordinates to panel coordinates. The UPDD driver reports touch coordinates normalized to a scale of 0-4000 whereas the WTG driver reports touch coordinates relative to physical pixels.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**TouchData** is the current touch data reported by the WTG or UPDD API. This is used to determine if the current touch is within the bounds of the graph control.

Also, calx and caly are used to determine the first point of touch to control the point where zoom is activated on a graph control.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

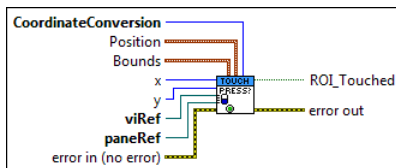
The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## ObjectTouched VI

Installed With: LabVIEW

Determines if the input **x** and **y** coordinates are within the object bounds defined in the **ROI\_Bounds** and **ROI\_Position** inputs. The Bounds and Position properties of a front panel's object that needs to be monitored for a touch can be wired to the **ROI\_Bounds** and **ROI\_Position** inputs. A reference to the front panel's vi and pane must also be wired to convert display coordinates to panel coordinates. **CoordinateConversion** identifies if the caller is using Windows Touch (WTG) or the UPDD driver as this determines how to map the physical coordinates to LabVIEW pane coordinates.



**y** is the physical y coordinate reported by either the WTG or UPDD driver. This VI scales the coordinate based on **CoordinateConversion**.

### Bounds

The **Width** of the front panel item to check for touch.

The **Height** of the front panel item to check for touch.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

### Position

The **Left** coordinate in LabVIEW front panel coordinates of the front panel item to check for touch.


The **Top** coordinate in LabVIEW front panel coordinates of the front panel item to check for touch.


**CoordinateConversion** is used to identify if the Windows Touch (WTG) or UPDD API is being used. This is used to determine the proper mapping of physical coordinates to panel coordinates. The UPDD driver reports touch coordinates normalized to a scale of 0-4000 whereas the WTG driver reports touch coordinates relative to physical pixels.


The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 **paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.

 **viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.

 **x** is the physical x coordinate reported by either the WTG or UPDD driver. This VI scales the coordinate based on **CoordinateConversion**.

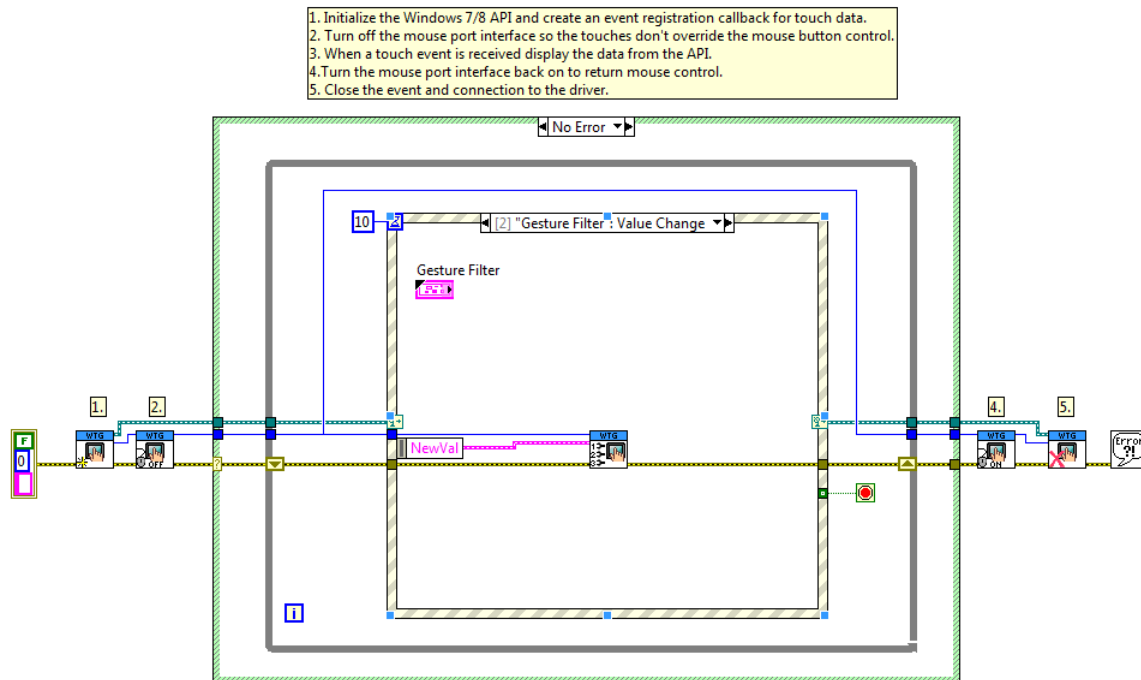
 **ROI\_Touched** is a TRUE if the specified region is touched.

## WTG Example VI

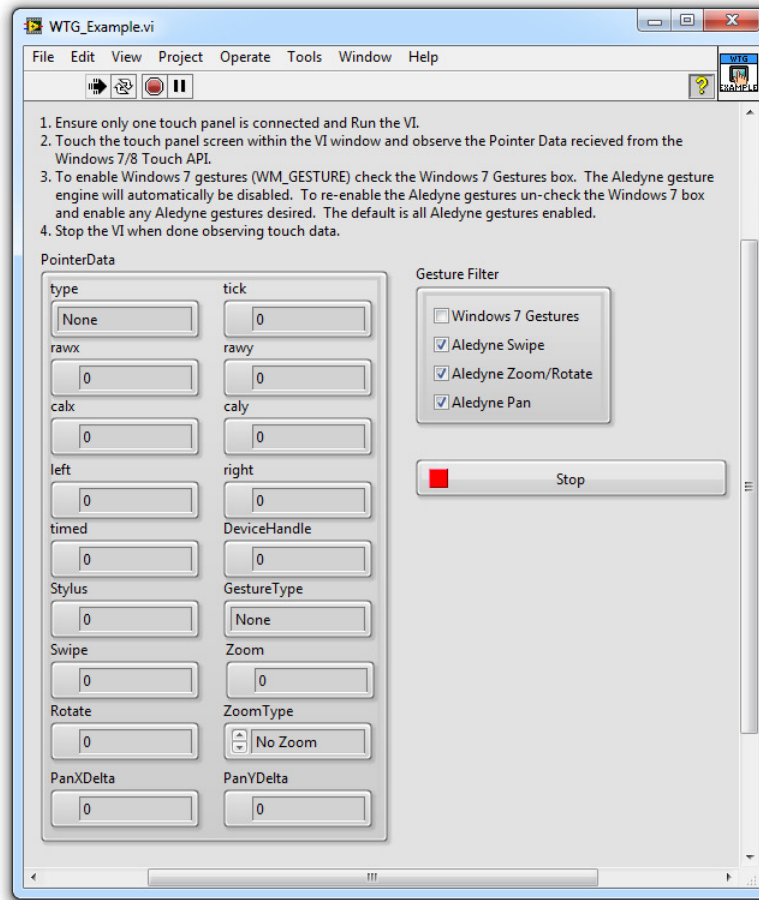
Installed With: LabVIEW

A fully featured example that shows the integration between LabVIEW and the Windows 7 Touch API. This example demonstrates the initialization of the Windows Touch/Gesture (WTG) driver by creating a user event which is then monitored for touch events. If a touch event occurs, the user event will be triggered and the touch coordinates along with the device handle will be returned. The WTG driver will report multi-touch coordinates (from simultaneous presses) if the hardware connected supports multi-touch, such as a capacitive touch monitor. The Windows 7 touch (WM\_TOUCH) and gesture (WM\_GESTURE) messages are captured by and interpreted by the UPDD toolkit to monitor multi-touch events from Windows 7. If Windows 7 gestures are disabled, the Aledyne gesture engine provides additional touch gesture recognition above and beyond Windows 7, which includes specifying direction of swipes, and x/y mode two-finger panning.

Also, the mouse pointer is disabled at the beginning such that a touch presses on the screen will not send mouse down events to the window. You may still see a cursor overlay depending on the touch driver, but mouse events will not be sent to the LabVIEW window. This is useful if it is not desirable that the touchscreen creates mouse presses when touched.



On the front panel, the type of gestures that are enabled can be specified. This must be changed only after running the VI. If Windows 7 Gestures are enabled, position data via WM\_TOUCH will not be sent anymore as the Windows 7 touch API only sends either [WM\\_GESTURE](#) or [WM\\_TOUCH](#), but not both. But Windows 7 gestures can be used to enable the default gesture handler that is part of Windows. If additional gesture functionality is required along with gaining access to the x/y coordinates of each touch point, the Aledyne gesture engine can be used instead. By disabling the Windows 7 Gestures, the Aledyne gesture engine is automatically enabled and individual gestures can be disabled/enabled using the Gesture Filter. After initialization and event registration, whenever a touch occurs the touch data and other driver information will be displayed in PointerData.

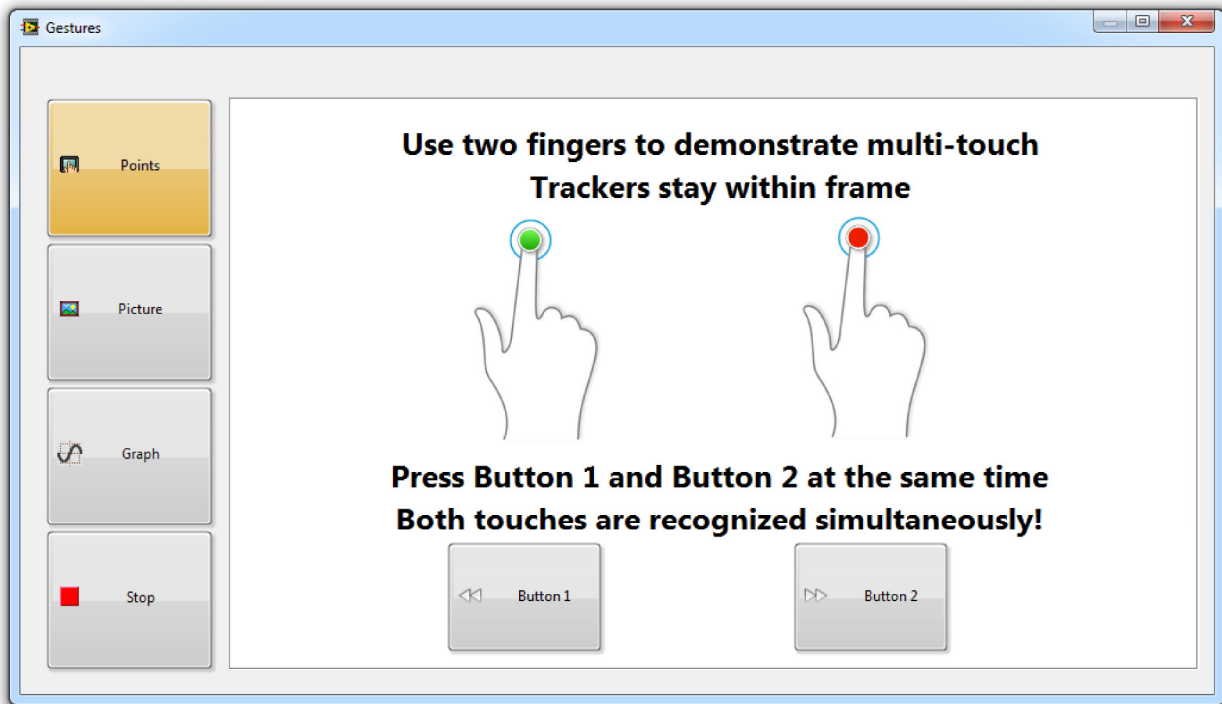


## WTG Gestures Example VI

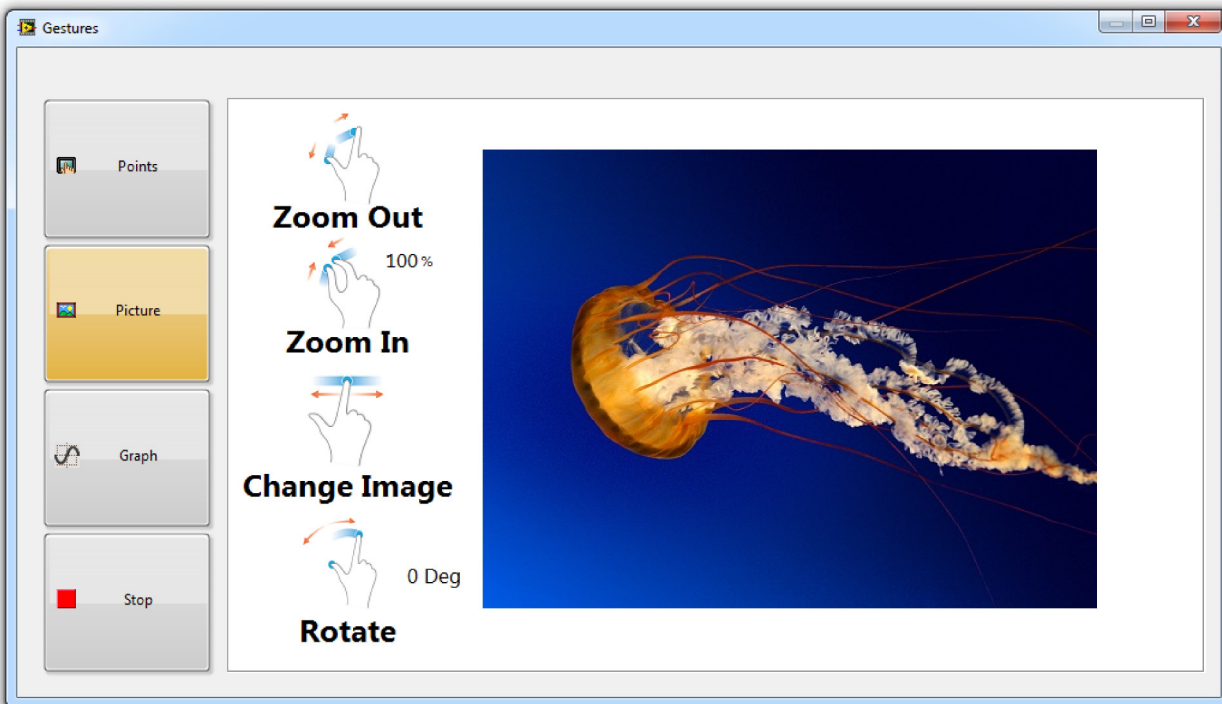
**Installed With:** LabVIEW

A fully featured example that shows how the UPDD toolkit can be used in multi-touch environments to process gestures and detect touches using the Windows 7 Touch API. This example demonstrates detection and dragging of two independent touch points, detecting presses of multiple buttons simultaneously, flipping by swipe, zooming and rotating images in a picture control, and zooming/panning of a waveform graph. This example is different from the UPDD Gesture Example in that it does not require use of the Touch-Base UPDD driver. This fully featured example uses the built in touch features of Windows 7 so an additional driver is not required. This example has been specifically developed for use with a MIMO Magic Touch 10.1" Capacitive Touch monitor and should be maximized on the monitor before running.

Use multiple touches to demonstrate tracking two movements within the touch area. Also, pressing both Button 1 and Button 2 simultaneously show how to detect multiple button presses at the same time.

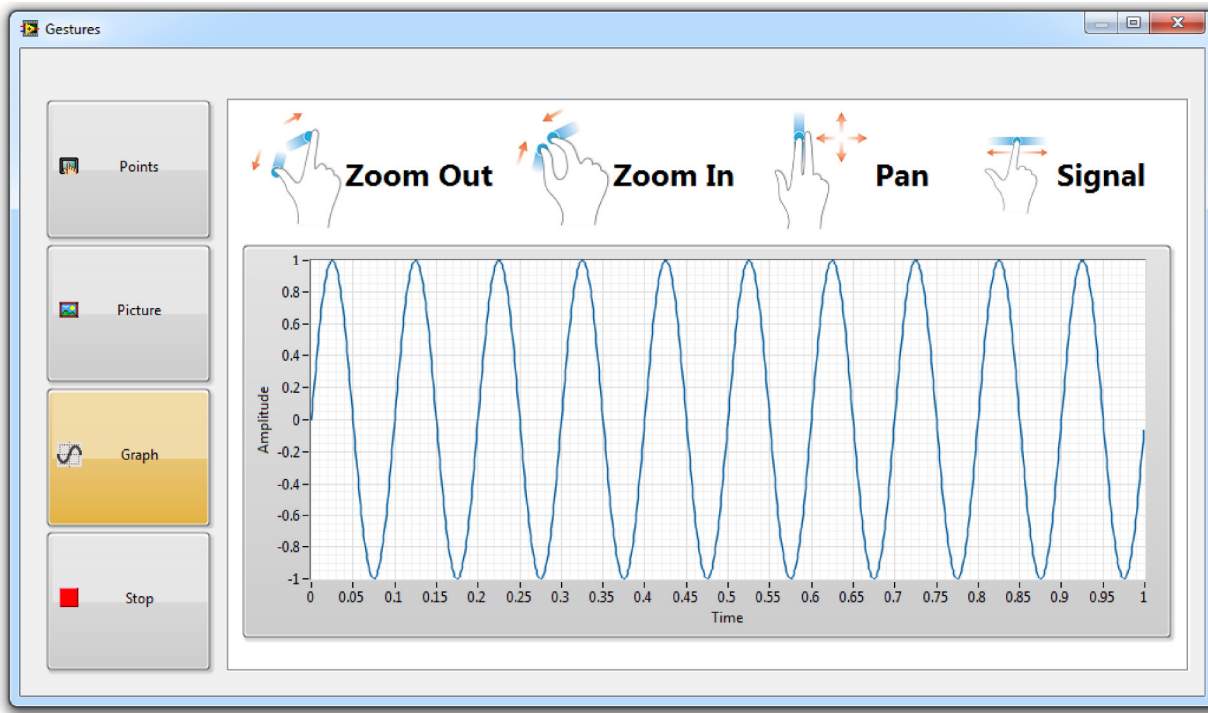


Use a left to right and right to left swiping motion to change the image. Use zoom contract/expand gestures to zoom the picture in and out. Rotate one finger around the other as it remains stationary to rotate the image clockwise and counterclockwise. The image will snap to increments of 90 degrees when the touch points are removed.

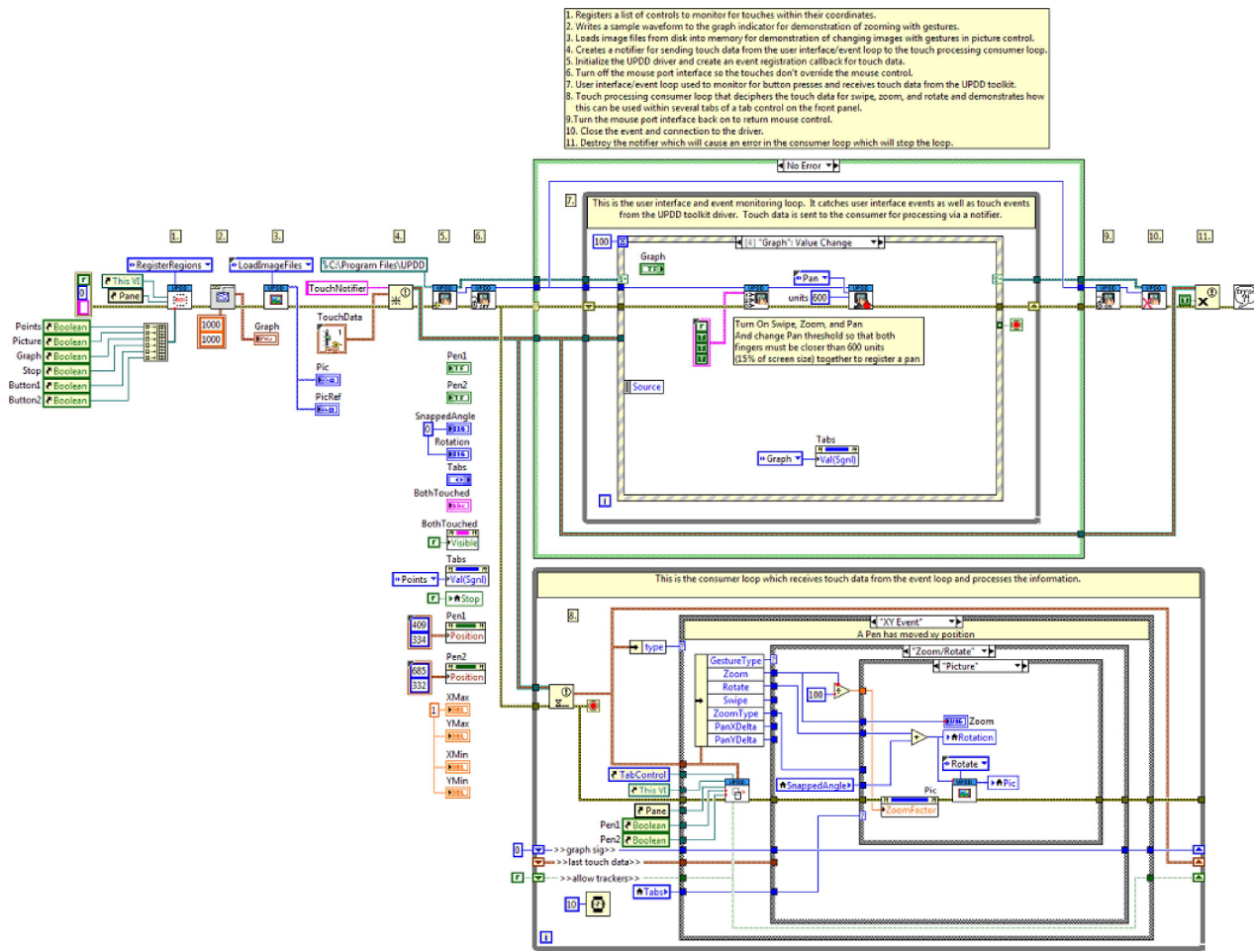


Use horizontal zoom gestures to zoom in and out in the horizontal (time) axis. Use vertical zoom gestures to zoom in and out in the vertical (Amplitude) axis. Place two fingers close together on the graph then move left/right/down/up to pan the graph. Use horizontal swiping movements to change the signal type.





The example uses two functional loops. The top loop is a producer loop that monitors for touch events from the toolkit driver as well as for user events. Touch events are passed to the bottom (consumer loop) using a lossy notifier. Since it is streaming XY data it can be lossy. However, it is important that the button press/release events are always detected. To make sure this happens, the producer loop waits for a listener before sending the event. The consumer loop reads the touch data and determines how to process the gestures to control either the multi-touch trackers, the picture, or graph control.

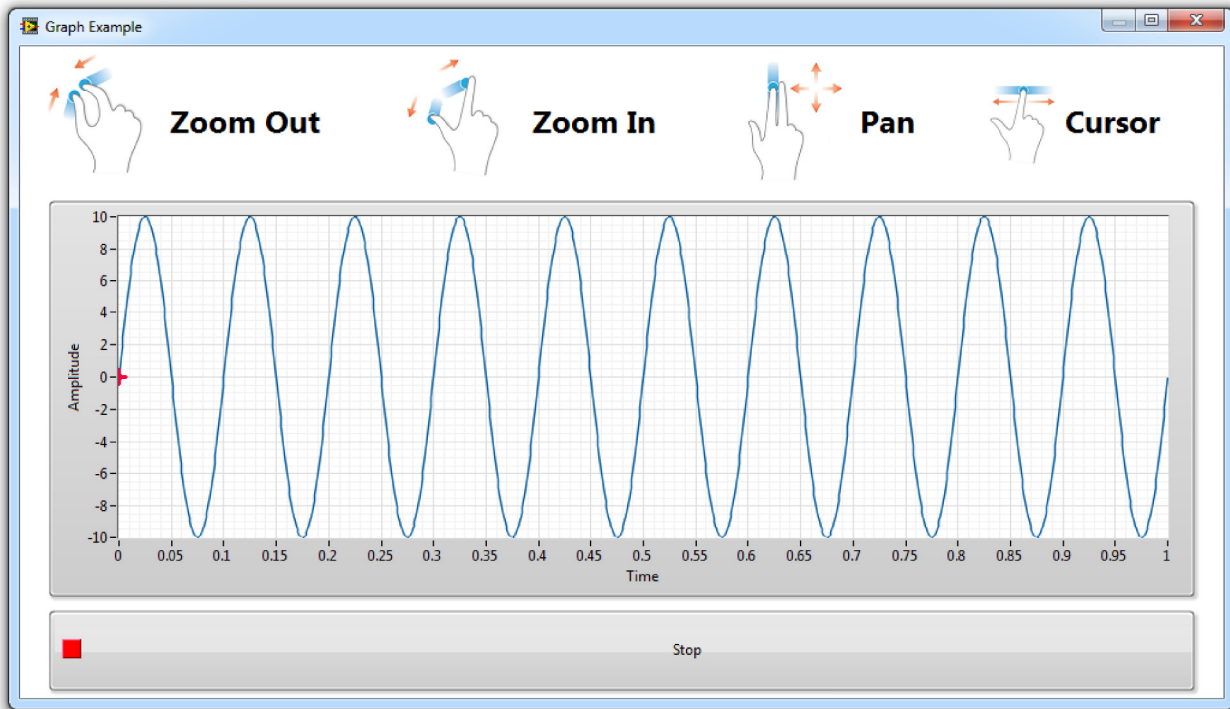


## WTG Graph Example VI

Installed With: LabVIEW

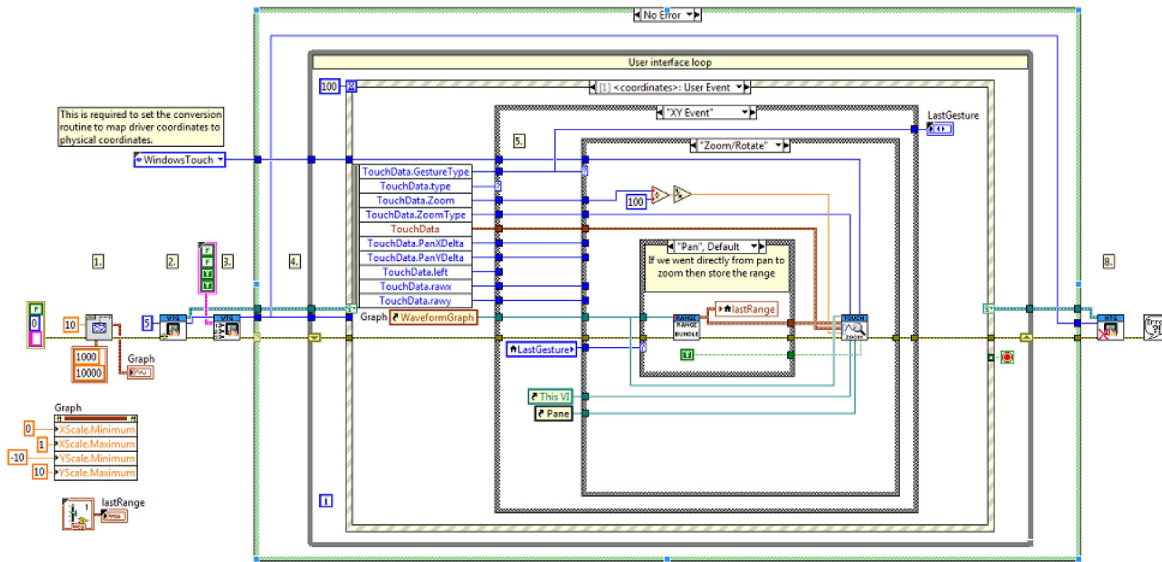
Demonstrates how to use the touchscreen toolkit to manipulate the x and y scales of a waveform graph. This example detects the zoom, rotate, and pan gestures from the toolkit and modifies the scales of the graph control. This can be useful for improving usability of LabVIEW touch applications display data to the user on the front panel and allowing the user to interact with the data as they would expect on a mobile device. These features are only possible with multi-touch capable monitors that can detect 2 or more touch points simultaneously. This example has been specifically developed for use with a MIMO Magic Touch 10.1" Capacitive Touch monitor and should be maximized on the monitor before running. The implementation uses the Windows 7 Touch API and the [WM\\_TOUCH](#) message.

Use zoom contract/expand gestures to zoom the graph in and out. This effectively changes the scale and centers the zoom around the center point between the two touch points. Place both fingers close together and move them around to pan the graph. Use a single touch along the graph to move the graph cursor.



The example uses one user interface loop that monitors for touch events from the touchscreen API through user events in the event structure. The x and y scales of the graph control are modified directly in the event handler. This can be done if the processing of the gesture does not take a significant amount of time. During initialization in this example, the touchscreen API has been configured to notify on every 5th touch event, which significantly reduces the number of events that must be handled by the user event loop, therefore, this allows enough time to process the touch event directly in the event handler. It is advised to handle and process touch and gesture messages in a separate loop or thread so that the user interface does not become unresponsive. The example here shows everything processed in the user event handler for simplicity, but more complex applications would likely require processing of touch and gesture messages in a separate thread.

1. Writes a sample waveform to the graph indicator for demonstration of zooming with gestures.
2. Initialize the touch driver and create an event registration callback for touch data. Report interval is set > 1 so that the user event loop is not flooded with touch events since it is directly manipulating the graph in the user event. Ideally, manipulation of front panel controls is done in a separate thread or loop.
3. Turn On Swipe, Zoom, and Pan gestures. If necessary, gesture thresholds can be also changed here using the SetGestureThreshold vis.
4. User interface/event loop used to monitor for button presses and receives touch data from the touchscreen driver.
5. On a touch move event, determine if the gesture is a pan, zoom, or single touch move. If the gesture transitions from a pan to zoom or zoom to pan, the last graph range must be saved in "lastRange" so that the range is saved and the graph does not jump when transitioning between gestures.
6. On a pan event, move the graph in proportion to the size of the graph so the data follows the users finger movement.
7. On single touch, control the graph cursor and map physical coordinates to the graph's data.
8. Close the event and connection to the driver.

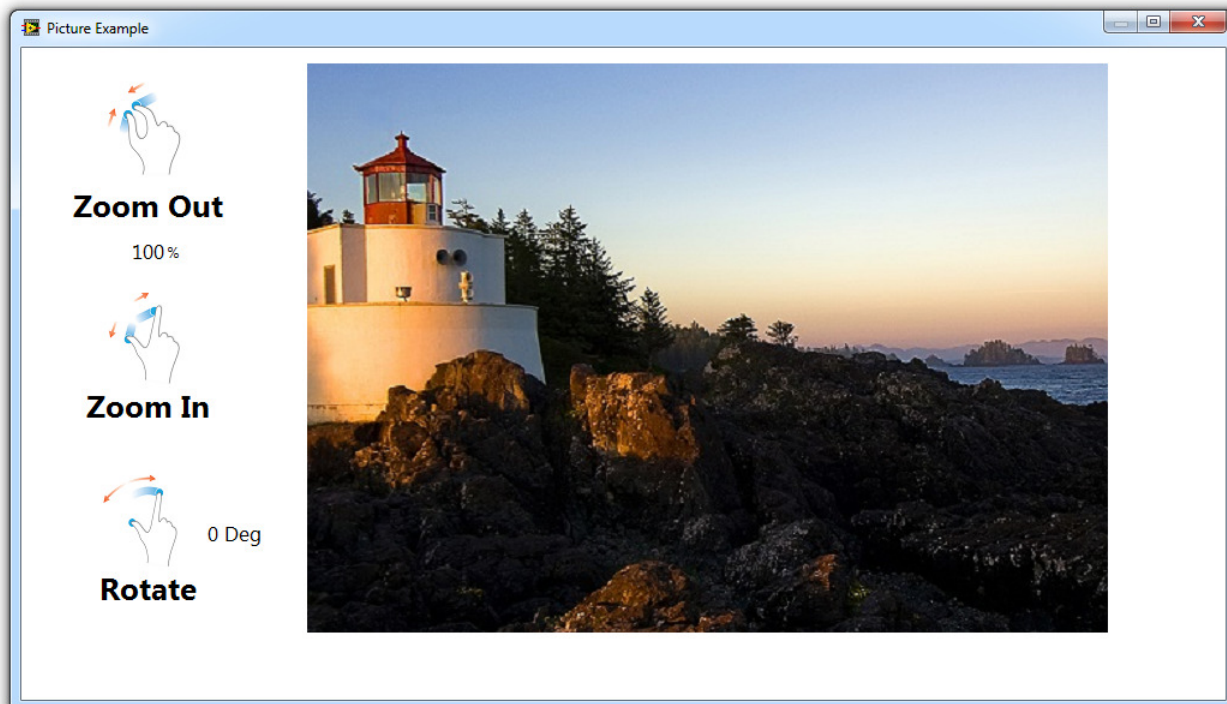


## WTG Picture Example VI

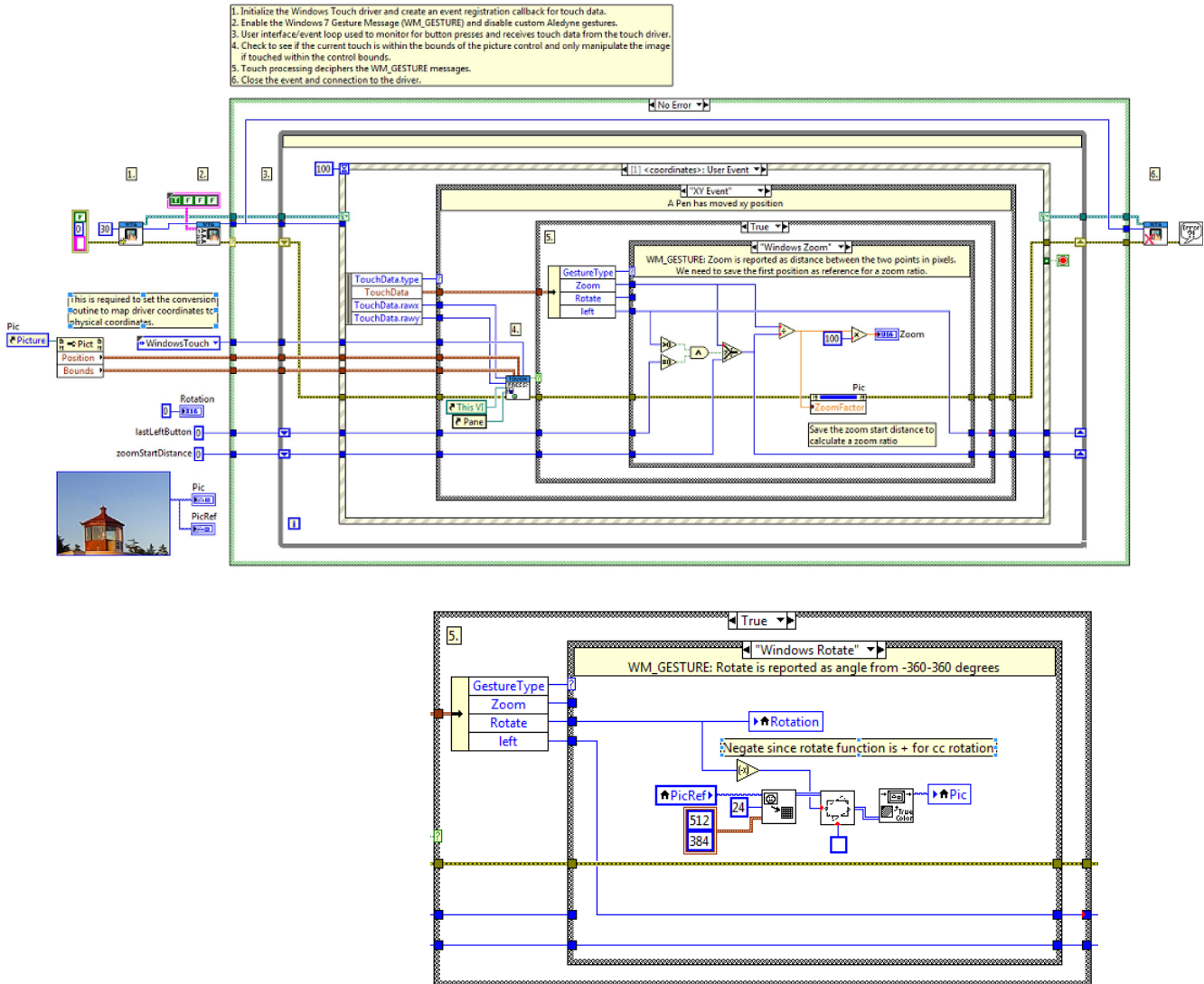
Installed With: LabVIEW

Demonstrates how to enable the Windows 7 Gesture Engine and messaging built into the [WM\\_GESTURE](#) message. This example connects to the touch driver, receives user events that are triggered off of [WM\\_GESTURE](#) and returns gesture data for Zoom and Rotate and uses the data to manipulate a picture control. This can be useful for extended touch applications. These features are only possible with multi-touch capable monitors that can detect 2 or more touch points simultaneously. This example has been specifically developed for use with a MIMO Magic Touch 10.1" Capacitive Touch monitor and should be maximized on the monitor before running.

Use zoom contract/expand gestures to zoom the picture in and out. Rotate one finger around the other as it remains stationary to rotate the image clockwise and counterclockwise.



The example uses one user interface loop that monitors for touch events from the touchscreen API through user events in the event structure. The zoom factor of the picture control is modified directly in the event handler. This can be done if the processing of the gesture does not take a significant amount of time. During initialization in this example, the touchscreen API has been configured to notify on every 30th touch event, which significantly reduces the number of events that must be handled by the user event loop, therefore, this allows enough time to process the touch event directly in the event handler. Modifying a picture control in this nature takes a significant amount of time so the number of touch events reports has to be reduced. It is advised to handle and process touch and gesture messages in a separate loop or thread so that the user interface does not become unresponsive. The example here shows everything processed in the user event handler for simplicity, but more complex applications would likely require processing of touch and gesture messages in a separate thread. Also note that for zoom control using WM\_GESTURE, the zoom quantity is reported as a distance between the two touch points in pixels, therefore, the position needs to be converted to a ratio based on the first touch position so that the image can be scaled in terms of a percentage of the original size, where 100% is the original size. When using the Aledyne zoom gesture, zoom is reported directly as a ratio of the original touch point distance.



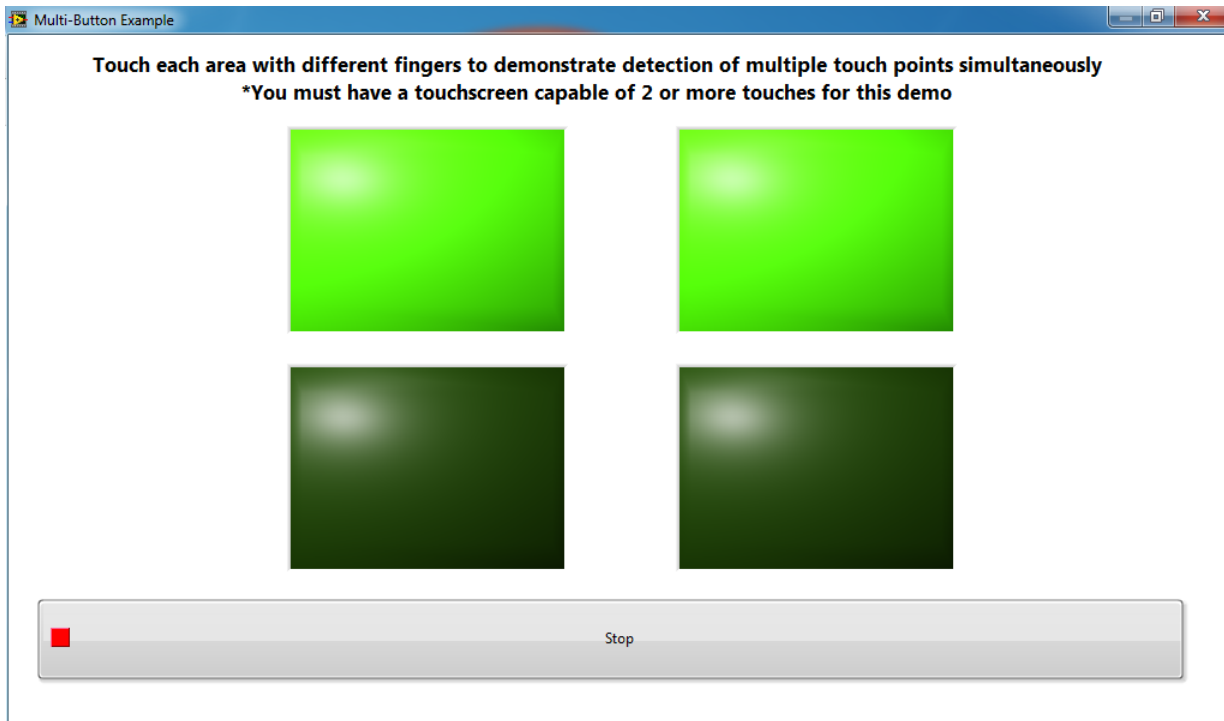
## WTG Multi-Button Example VI

Installed With: LabVIEW

Demonstrates how to use the touchscreen toolkit to detect touch on multiple areas of interest on the front panel simultaneously using the Windows 7 Touch API. A typical application would be to detect touch of two buttons simultaneously to ensure operators hands are away from machinery before starting a dangerous piece of machinery. This eliminates the need for external mechanical buttons. These features are only possible with multi-touch capable monitors that can detect 2 or more touch points simultaneously. The implementation uses the Windows 7 Touch API and the [WM\\_TOUCH](#) message.

Press any number of the 4 touch areas to see them detected simultaneously.





Note: Some touchscreens only support 2 touches. For all 4 touch areas to be recognized simultaneously, you must have a touchscreen that supports 4 or more simultaneous touch points. Refer to system properties to identify the number of supported touch points.

System

Rating: **7.2** Windows Experience Index

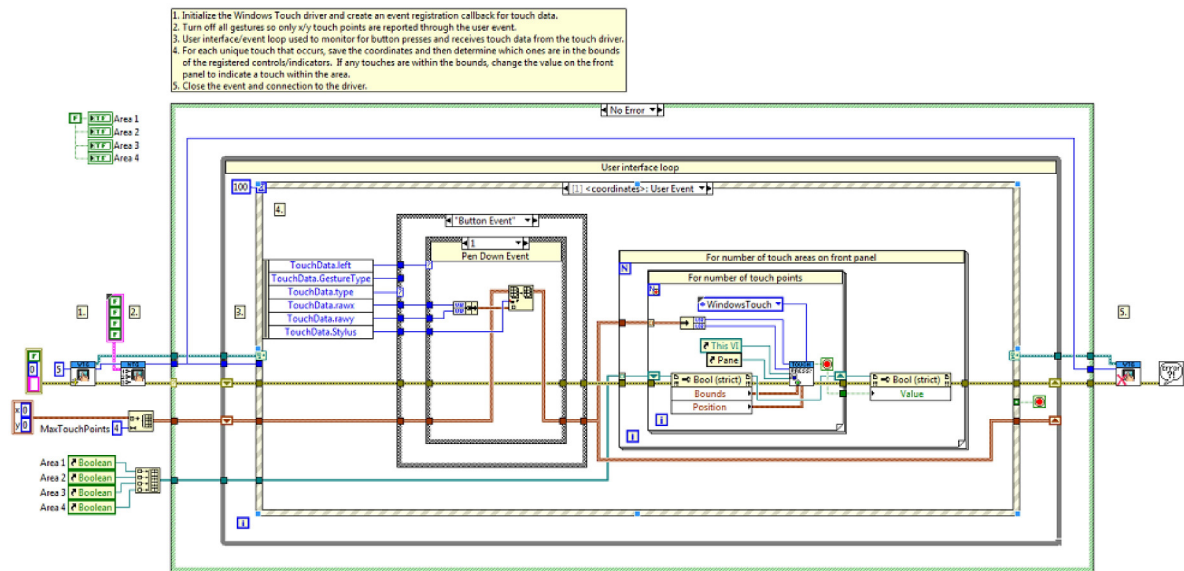
Processor: Intel(R) Core(TM) i7-3840QM CPU @ 2.80GHz 2.80 GHz

Installed memory (RAM): 16.0 GB

System type: 64-bit Operating System

Pen and Touch: Touch Input Available with 16 Touch Points

The example uses one user interface loop that monitors for touch events from the touchscreen API through user events in the event structure. The value property of the touch indicators are modified directly in the event handler. This can be done if the processing of the gesture does not take a significant amount of time. During initialization in this example, the touchscreen API has been configured to notify on every 5th touch event, which significantly reduces the number of events that must be handled by the user event loop, therefore, this allows enough time to process the touch event directly in the event handler. It is advised to handle and process touch and gesture messages in a separate loop or thread so that the user interface does not become unresponsive. The example here shows everything processed in the user event handler for simplicity, but more complex applications would likely require processing of touch and gesture messages in a separate thread.





## UPDD Touch Interface Help

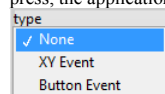
June 2014

The UPDD interface requires a separate driver (sold separately) which supports multiple Operating Systems and many different touchscreens. This is required for integration with NI standalone chassis, such as the cDAQ-9139. Refer to <http://www.touch-base.com/> or contact [sales@aledyne.com](mailto:sales@aledyne.com). Touch-Base is a leading developer and supplier of touchscreen and pointer device drivers. The Universal Pointer Device Driver is in use on many thousands of devices worldwide. This API provides a LabVIEW wrapper around the UPDD driver and a set of LabVIEW vi's in order to initialize the driver and receive events and touch data when any touchscreen connected is touched.

Please visit <http://touch-base.com/> to purchase and download the UPDD driver for your hardware if using the UPDD APIs in the toolkit. The downloaded UPDD driver will include a TBApi32/64.dll and an ACE\_UPDD\_5.6.2.dll which must both be copied into the directory of this toolkit (<LabVIEW>\vi.lib\Aledyne Engineering\Touchscreen Toolkit).

### Touch Data Elements Reported by the Toolkit:

**Type:** Defines if the event reported is an XY event, such as dragging a finger or stylus across the screen, or a Button event, which happens once on a depression or removal of a finger or stylus on the screen. When a button event occurs, the XY coordinates of the button press or release will also be reported for easy processing. Following a button press, the application will continually receive XY Events with position data until the button is released.



**Tick:** Relative time in ticks that the data was read from the UPDD driver.

**Rawx:** Raw x value of button press/release or drag received from the touch controller normalized to 0-4000 as reported by the UPDD driver. For the WTG API this is the screen coordinate in physical pixels.

**Rawy:** Raw y value of button press/release or drag received from the touch controller normalized to 0-4000 as reported by the UPDD driver. For the WTG API this is the screen coordinate in physical pixels.

**Calx:** The corresponding calibrated x value as reported by the UPDD driver. For the WTG API this is mostly unused except for in Zoom/Rotate mode it provides the center point of the two touches so that the application can zoom about the center point.

**Caly:** The corresponding calibrated y value as reported by the UPDD driver. For the WTG API this is mostly unused except for in Zoom/Rotate mode it provides the center point of the two touches so that the application can zoom about the center point.

**Left:** The state of the left button where 0 is released and 1 is pressed. The left button is used for touches. **This data is only valid for Button Events.**

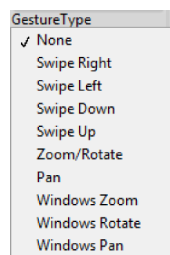
**Right:** The state of the right button where 0 is released and 1 is pressed. **This data is only valid for Button Events.**

**Timed:** Reserved for Future Use.

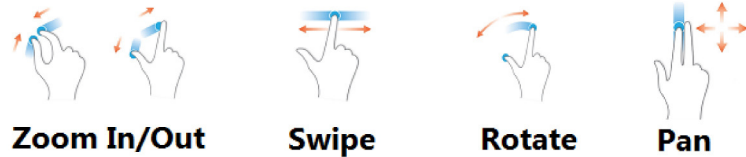
**DeviceHandle:** Used to identify the monitor/device that was pressed in a multi-monitor system. For the UPDD API, this should be used with UPDD\_GetDeviceIndex.vi to identify which monitor is reporting the data. For the WTG API, this is a unique identifier for the monitor that reported the touch event. In a single monitor environment, this is not needed.

**Stylus:** This is the pen/stylus number in a multi-touch application. If the monitor being used supports multi-touch, up to 16 independent pens will be reported based on the capabilities of the hardware. This can be used to create custom gestures using a capacitive touchscreen monitor. In a resistive touch application that only supports one coordinate, this will always be 0.

**GestureType:** The toolkit has built in gesture recognition for swiping, zooming, rotating, and panning. Swipe only requires single touch monitors but zoom, rotate, and pan will only work in multi-touch environments. Zoom and rotate are reported as one gesture but with independent values because zoom and rotate are interpreted as one gesture when using two fingers on a display. The application can monitor either Zoom or Rotate if it is only desired to perform one of the functions. The Windows Zoom, Windows Rotate, and Windows Pan are gesture events triggered by the Windows [WM\\_GESTURE](#) message when Windows specific gestures are enabled using the WTG API. If Pan and Zoom/Rotate are enabled, when the pan threshold is exceeded the Pan gesture will be cancelled and the gesture engine will transition to the Zoom/Rotate gesture if its zoom threshold is exceeded. If the distance between the two touch points is then reduced lower than the zoom threshold, the gesture will be canceled and if the distance is less than the pan threshold, the gesture engine will transition back to the Pan gesture.





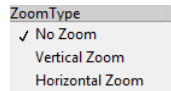


**Swipe:** Reports the difference in X movement for left to right and right to left swipe gestures and the difference in Y movement for down to up and up to down gestures. This is not reported until the gesture threshold is passed. The threshold can be customized using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi.

**Zoom:** Reports the zoom scale in a multi-touch environment when pinching and expanding motions are detected on the monitor. This is reported as a percentage of the initial spacing between the two touch points for the Aledyne gesture. So if two touch points are initially spaced 1 inch apart and are moved apart to 2 inches, a value of 200% will be reported. The threshold distance to which both fingers must be apart from one another before considering the gesture a zoom can be configured using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi. If a zoom gesture is being reported and the fingers move together again within the set threshold, the zoom gesture will be cancelled. If pan is enabled it will go back to pan mode. For WM\_GESTURE (Windows Zoom), the zoom quantity is reported as a distance in pixels between the two touch points. The quantity is not converted to a percentage of the initial spacing and, if this is required, it must be done in the user's application. Refer to WTG\_PictureExample.vi for an example.

**Rotate:** When one finger is rotated around the second finger which remains a pivot point, rotation is detected. The angle of rotation reported correlates to the rotation angle of one finger with respect to the other. The maximum reported angles are -90 to 90 degrees. If the rotation is not sensitive enough in the application, this number can simply be scaled or multiplied by some factor to make the rotation more sensitive to movement of the fingers on the screen.

**ZoomType:** Returns if the zoom orientation is horizontal or vertical based on touch positioning. If the relative angle is greater than 45 degrees from the horizontal axis, a vertical zoom type is returned.



**PanXDelta:** Reports the difference in X movement for left to right and right to left pan gestures. For right to left gestures this will be reported as a negative value. This is not reported until the gesture threshold is passed. The threshold can be customized using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi. If a pan is started and then the fingers are moved apart to commence a zoom gesture, once the pan threshold is exceeded, the gesture engine will automatically cancel the pan and will commence the zoom.

**PanYDelta:** Reports the difference in Y movement for down to up and up to pan gestures. For down to up gestures this will be reported as a negative value. This is not reported until the gesture threshold is passed. The threshold can be customized using UPDD\_SetGestureThreshold.vi or WTG\_SetGestureThreshold.vi. If a pan is started and then the fingers are moved apart to commence a zoom gesture, once the pan threshold is exceeded, the gesture engine will automatically cancel the pan and will commence the zoom.

Refer to "<LabVIEW>\examples\Aledyne Engineering\Touchscreen Toolkit" for examples of how to interface to the UPDD and WTG driver.

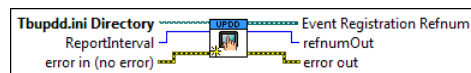
To provide feedback or request support, contact us [here](#).

© 2012–2014 Aledyne Engineering. All rights reserved.

## UPDD\_Initialize VI

**Installed With:** LabVIEW

Creates a user event and sends the reference to the UPDD\_DLL.dll wrapper Dll for registration of the callback in the TBApi.dll. The interval at which user events are reported can also be specified by **ReportInterval**. If this is set to 1, every received Windows event will be passed to LabVIEW. If this is 2, then every other event will be passed to LabVIEW. This can be increased to reduce the amount of interrupts generated by the driver. By default this is set to 1.



**Tbupdd.ini Directory** is the directory where the Tbupdd.ini driver file resides.

**ReportInterval** defines how often to send touch messages from the driver up to the LabVIEW user event. By default this is 1 so that every message is sent. Messages are sent about every 5-10ms. This can be increased so that LabVIEW is not interrupted too often. The maximum is 10.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

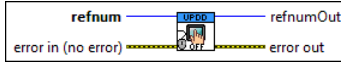
**Event Registration Refnum**

**refnumOut** is the reference to the UPDD instance.

## UPDD\_MousePointerDisable VI

Installed With: LabVIEW

Calls into the UPDD\_DLL wrapper DLL and calls TBApiMousePortInterfaceEnable(false) which disables the mouse port interface. This has no effect on Windows 8 and the mouse pointer cannot be disabled.



**refnum** is the reference to the UPDD instance.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

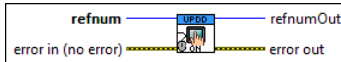
The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**refnumOut** is the reference to the UPDD instance.

## UPDD\_MousePointerEnable VI

Installed With: LabVIEW

Calls into the UPDD\_DLL wrapper DLL and calls TBApiMousePortInterfaceEnable(true) which enables the mouse port interface. This has no effect on Windows 8.



**refnum** is the reference to the UPDD instance.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

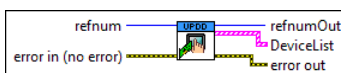
The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**refnumOut** is the reference to the UPDD instance.

## UPDD\_GetAllDevices VI


Installed With: LabVIEW


Iterates through the devices returned by TBApi.dll and populates an array of device names and handles.




The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more


information about the error displayed.


 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**DeviceList** is a list of detected touchscreen devices.

  **Handle**  
 **device\_name**

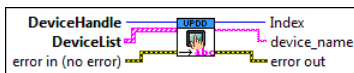
 **refnum** is the reference to the UPDD instance.

 **refnum** is the reference to the UPDD instance.

## UPDD\_GetDeviceIndex VI

Installed With: LabVIEW


Takes the input device handle and iterates through the device list to find a match. If a match is found, the device name and index in the device list array is returned. If no match is found, -1 is returned for the index and a blank string is returned for the device name.





**DeviceList** is the list of devices returned by GetAllDevices.


  **Handle**  
 **device\_name**

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 **DeviceHandle** to a specific device in the list.


The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

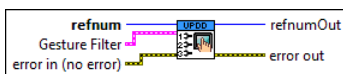
 **Index** of the device corresponding to **DeviceHandle**

 **device\_name** is the name of the device corresponding to **DeviceHandle**

## UPDD\_GestureFilter VI


Installed With: LabVIEW

Calls into the UPDD driver and enables/disables gesture features. Windows 7 Gestures do not apply for the UPDD driver and only the Aledyne custom gesture engine is enabled.



**Gesture Filter**

  **W7** defines if Windows 7 Gestures are enabled. If true, Windows 7 WM\_GESTURE messages are received and WM\_TOUCH and the Aledyne gesture engine is disabled. This setting does not do anything for the UPDD instance. Windows 7 gestures are automatically disabled and this setting is ignored for UPDD. This can only be enabled for the WTG instance.

 **Swipe** defines if the Aledyne swipe gesture is enabled. This can only be enabled if W7 is disabled.

**Zoom/Rotate** defines if the Aledyne zoom/rotate gesture is enabled. This can only be enabled if W7 is disabled.

**Pan** defines if the Aledyne pan gesture is enabled. This can only be enabled if W7 is disabled.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**refnum** is the reference to the UPDD instance.

**refnumOut** is the reference to the UPDD instance.

## UPDD\_SetGestureThreshold VI

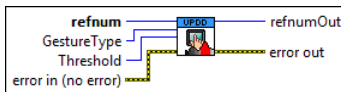
Installed With: LabVIEW

Sets the thresholds of gestures for finer tuning of gesture response. Currently, the only gestures that allow fine tuning are the Aledyne Swipe and Pan. These thresholds are defined in units of screen width and height as reported by the UPDD driver (on a scale of 0-4000).

For Swipe, when a single finger is swiped across the screen, once it has moved greater than the specified distance in UPDD units, the API will start registering the gesture. The default for the Swipe threshold is 200 units (5% of screen size).

For Pan, when two fingers are moved across the screen, and the two fingers start with a distance apart that is less than the specified distance in UPDD units, the API will start registering the gesture. The default for the Pan threshold is 700 units.

For Zoom, when a zoom is started by holding two fingers close together, the zoom percentage will stay at 100% until the distance the fingers are moved apart is greater than the specified distance in UPDD units. This prevents the start zoom to be really small when a zoom is started causing a large zoom percentage for a small movement apart. The default for the Zoom threshold is 700.



**refnum** is the reference to the UPDD instance.

**GestureType** is the type of gesture to be configured. Currently, the only gesture thresholds that can be configured are the Aledyne swipe and pan.

**Threshold** is the threshold to set for the gesture specified. The default for swipe is 200 units and the default for pan is 600 units.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

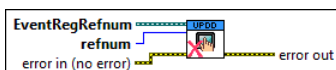
The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**refnumOut** is the reference to the UPDD instance.

## UPDD\_Close VI

Installed With: LabVIEW

Unregisters the event and closes the connection to UPDD\_DLL.dll.



**EventRegRefnum** for the event registered in UPDD\_Initialize.

**UPDD** **refnum** is the reference to the UPDD instance created by UPDD\_Initialize.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**STATUS** **TF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**U32** **code** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** **source** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**STATUS** **TF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

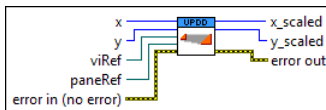
**U32** **code** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** **source** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## UPDD\_Scale VI

Installed With: LabVIEW

Scales the input x-y coordinates received from the UPDD driver to map to the coordinates of a front panel VI and pane. Note that the top-left coordinate of the touch panel will be mapped to the top-left coordinate of the VI pane referenced. Since the UPDD driver reports screen coordinates as a number from 0-4000, the value must be mapped to physical screen coordinates as a percentage of total resolution before mapping to pane coordinates.



**viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.

**x** is the physical x coordinate reported by the UPDD API. This is reported as a normalized coordinate of 0-4000.

**y** is the physical y coordinate reported by the UPDD API. This is reported as a normalized coordinate of 0-4000.

**paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**STATUS** **TF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**U32** **code** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** **source** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**STATUS** **TF** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**U32** **code** The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**abc** **source** The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

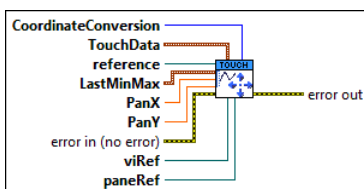
**U32** **x\_scaled** is the scaled x LabVIEW panel coordinate.

**U32** **y\_scaled** is the scaled y LabVIEW panel coordinate.

## PanGraph VI


Installed With: LabVIEW

Controls panning of a graph by scaling the min and max ranges of the x and y scales relative to the physical x/y coordinates provides to the input. If the x/y coordinates input map to a graph coordinate outside the current range, the graph will not be panned. Panning will only be executed if the touches are within the bounds of the graph control. **CoordinateConversion** identifies if the caller is using Windows Touch (WTG) or the UPDD driver as this determines how to map the physical coordinates to LabVIEW pane coordinates.



**TouchData** is the current touch data reported by the WTG or UPDD API. This is used to determine if the current touch is within the bounds of the graph control.

**abc**

 **reference** to a graph control used to change the scales programatically for panning.


**LastMinMax** is the last locked in graph range used as input to modify the scales based on **PanX** and **PanY**. This should remain the same from the beginning of a touch pan to the end of a touch pan and is used as a reference for modifying the scales.




The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



**PanX** is a scaling factor to apply to the x scale minimum and maximum. The difference between the max and min will be scaled by this amount.



**PanY** is a scaling factor to apply to the y scale minimum and maximum. The difference between the max and min will be scaled by this amount.



**viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.



**paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.





**CoordinateConversion** is used to identify if the Windows Touch (WTG) or UPDD API is being used. This is used to determine the proper mapping of physical coordinates to panel coordinates. The UPDD driver reports touch coordinates normalized to a scale of 0-4000 whereas the WTG driver reports touch coordinates relative to physical pixels.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

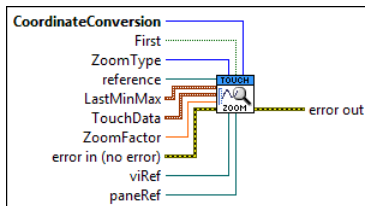
 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## GraphZoomAtPoint VI

Installed With: LabVIEW

Controls zooming of a graph by scaling the min and max ranges of the x and y scales relative to the physical x/y coordinates provided to the input. If the x/y coordinates input map to a graph coordinate outside the current range, the graph will not be zoomed. Zooming will only be executed if the touches are within the bounds of the graph control. Zoom is executed about the input x/y coordinate of **TouchData**. If **First** is true, the center point is used from calx/caly of **TouchData** and is stored for following calls to activate zooming about a center point. **CoordinateConversion** identifies if the caller is using Windows Touch (WTG) or the UPDD driver as this determines how to map the physical coordinates to LabVIEW pane coordinates.



**reference** to a graph control used to change the scales programatically.



**viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.




**paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.



**ZoomType** defines either an x axis or y axis zoom.

**LastMinMax** is the last locked in graph range used as input to modify the scales based on **PanX** and **PanY**. This should remain the same from the beginning of a touch pan to the end of a touch pan and is used as a reference for modifying the scales.



 **ZoomFactor** is the zoom amount as a percentage where 100 equates to 100% of the image size.



**First** should be set to TRUE on first touch. This forces the touch coordinate to be stored internally to this VI in order to control zooming about the point defined by calx and caly of **TouchData**.




**CoordinateConversion** is used to identify if the Windows Touch (WTG) or UPDD API is being used. This is used to determine the proper mapping of physical coordinates to panel coordinates. The UPDD driver reports touch coordinates normalized to a scale of 0-4000 whereas the WTG driver reports touch coordinates relative to physical pixels.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**TouchData** is the current touch data reported by the WTG or UPDD API. This is used to determine if the current touch is within the bounds of the graph control.






Also, calx and caly are used to determine the first point of touch to control the point where zoom is activated on a graph control.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event



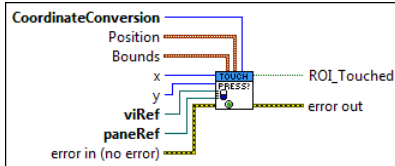
of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


-  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
-  The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
-  The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## ObjectTouched VI



**Installed With:** LabVIEW

Determines if the input **x** and **y** coordinates are within the object bounds defined in the **ROI\_Bounds** and **ROI\_Position** inputs. The Bounds and Position properties of a front panel's object that needs to be monitored for a touch can be wired to the **ROI\_Bounds** and **ROI\_Position** inputs. A reference to the front panel's vi and pane must also be wired to convert display coordinates to panel coordinates. **CoordinateConversion** identifies if the caller is using Windows Touch (WTG) or the UPDD driver as this determines how to map the physical coordinates to LabVIEW pane coordinates.






-  **y** is the physical y coordinate reported by either the WTG or UPDD driver. This VI scales the coordinate based on **CoordinateConversion**.



### Bounds

-  The **Width** of the front panel item to check for touch.
-  The **Height** of the front panel item to check for touch.

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.




-  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
-  The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
-  The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


### Position


-  The **Left** coordinate in LabVIEW front panel coordinates of the front panel item to check for touch.
-  The **Top** coordinate in LabVIEW front panel coordinates of the front panel item to check for touch.

**CoordinateConversion** is used to identify if the Windows Touch (WTG) or UPDD API is being used. This is used to determine the proper mapping of physical coordinates to panel coordinates. The UPDD driver reports touch coordinates normalized to a scale of 0-4000 whereas the WTG driver reports touch coordinates relative to physical pixels.


The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

-  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
-  The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
-  The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 **paneRef** is a reference to the top level vi pane to map physical coordinates to panel coordinates.

 **viRef** is a reference to the top level vi to map physical coordinates to panel coordinates.

 **x** is the physical x coordinate reported by either the WTG or UPDD driver. This VI scales the coordinate based on **CoordinateConversion**.

 **ROI\_Touched** is a TRUE if the specified region is touched.

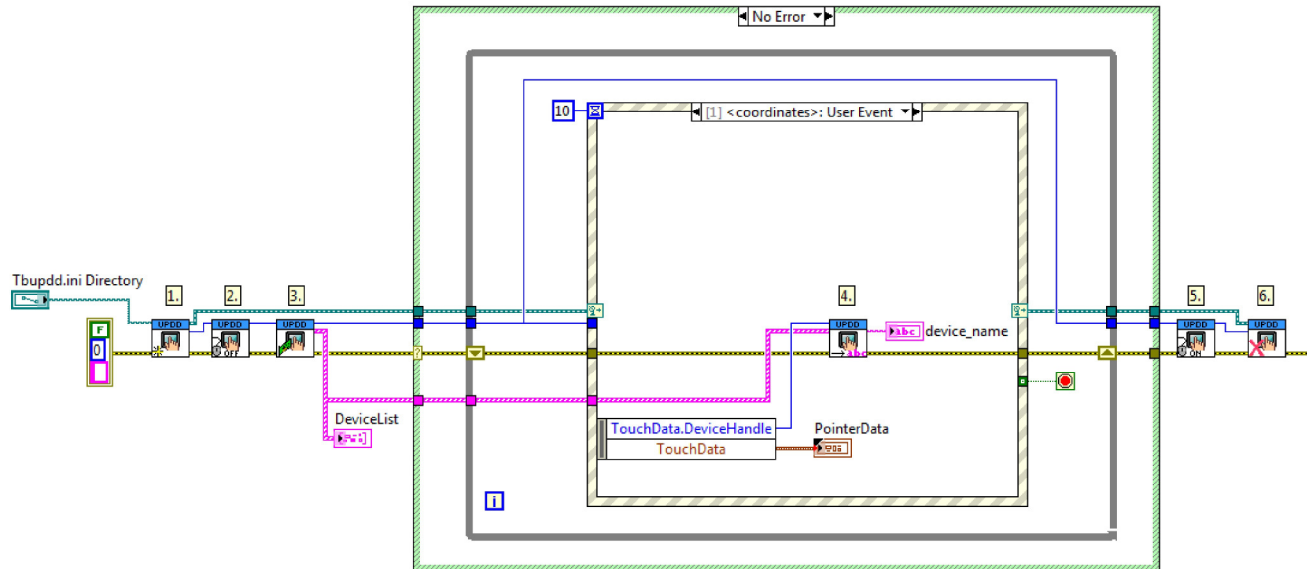
## UPDD Example VI

**Installed With:** LabVIEW

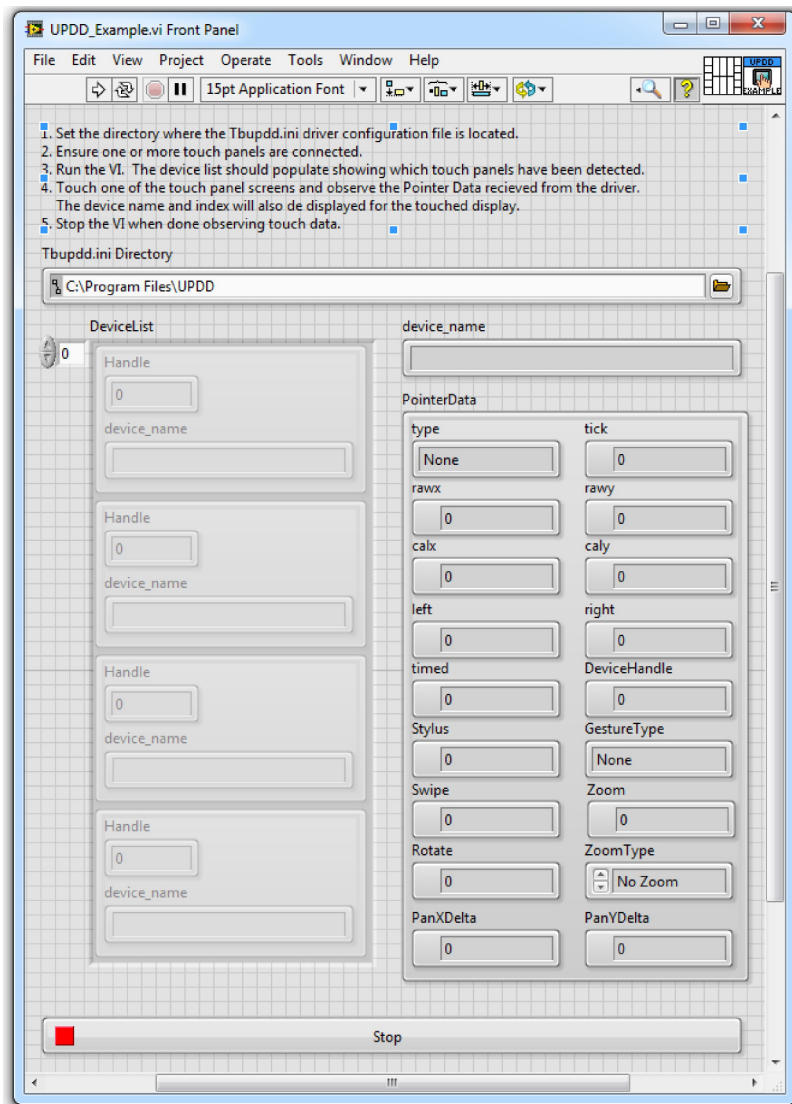
A fully featured example that shows the integration between LabVIEW and the UPDD driver. This example demonstrates the initialization of the UPDD driver by creating a user event which is then monitored for touch events. If a touch event occurs, the user event will be triggered and the touch coordinates along with the device handle will be returned. The device handle can be used to provide a device identifier for the specific touchscreen that was touched in the event multiple touchscreens are connected. The UPDD toolkit will report multi-touch coordinates (from simultaneous presses) if the hardware connected supports multi-touch, such as a capacitive touch monitor.

Also, the mouse pointer is disabled at the beginning such that a touch event on any screen will not override the mouse control. This is useful if it is not desirable that the touchscreen overrides the mouse and the programmer wishes to passively monitor the touch coordinates. In this way the machine can be used normally while still receiving touch events.

1. Initialize the UPDD driver and create an event registration callback for touch data.
2. Turn off the mouse port interface so the touches don't override the mouse control.
3. Get a list of touch panels that we will receive events from.
4. When a touch event is received, decode which panel it came from and display the device name.
5. Turn the mouse port interface back on to return mouse control.
6. Close the event and connection to the driver.



On the front panel, the directory where the tbupdd.ini driver file must be provided. Typically this is with the UPDD device driver install. Contact Touch-Base, Ltd. for more information. The DeviceList will populate when the VI is started showing all detected touch devices. This is used for identifying the touched device in a multi-device configuration when a touch event occurs. After initialization and event registration, whenever a touch occurs the coordinates and other driver information will be displayed in PointerData.

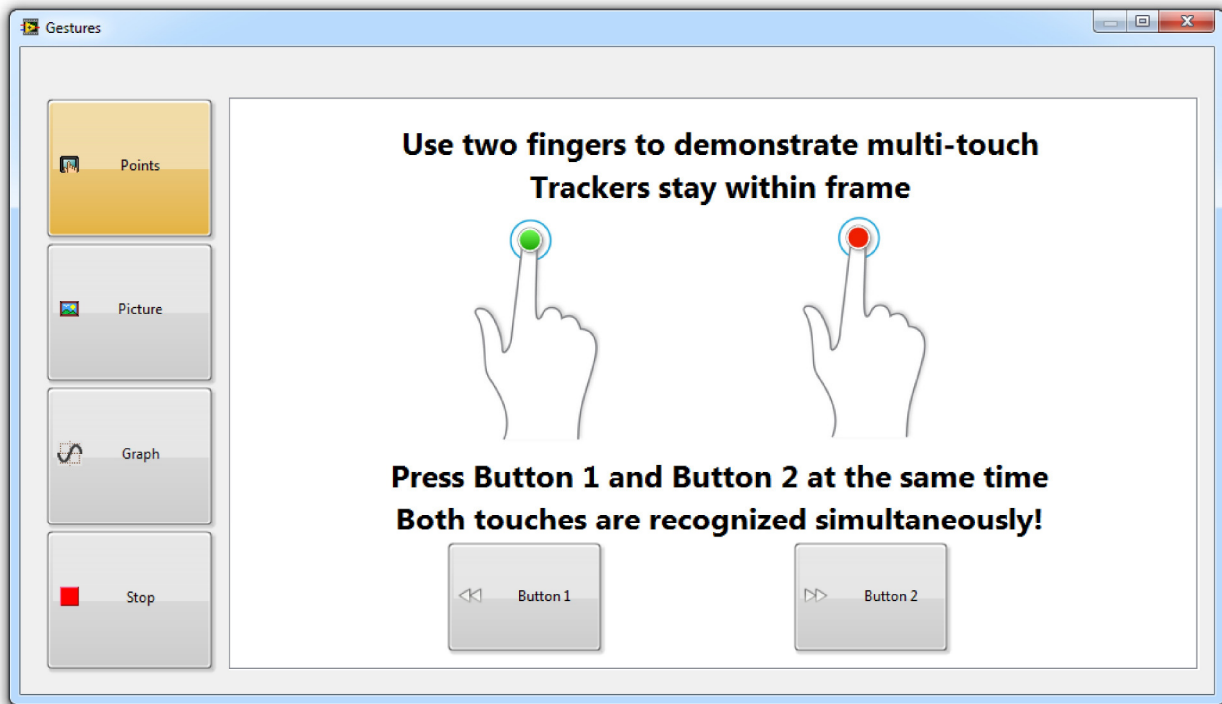


## UPDD Gestures Example VI

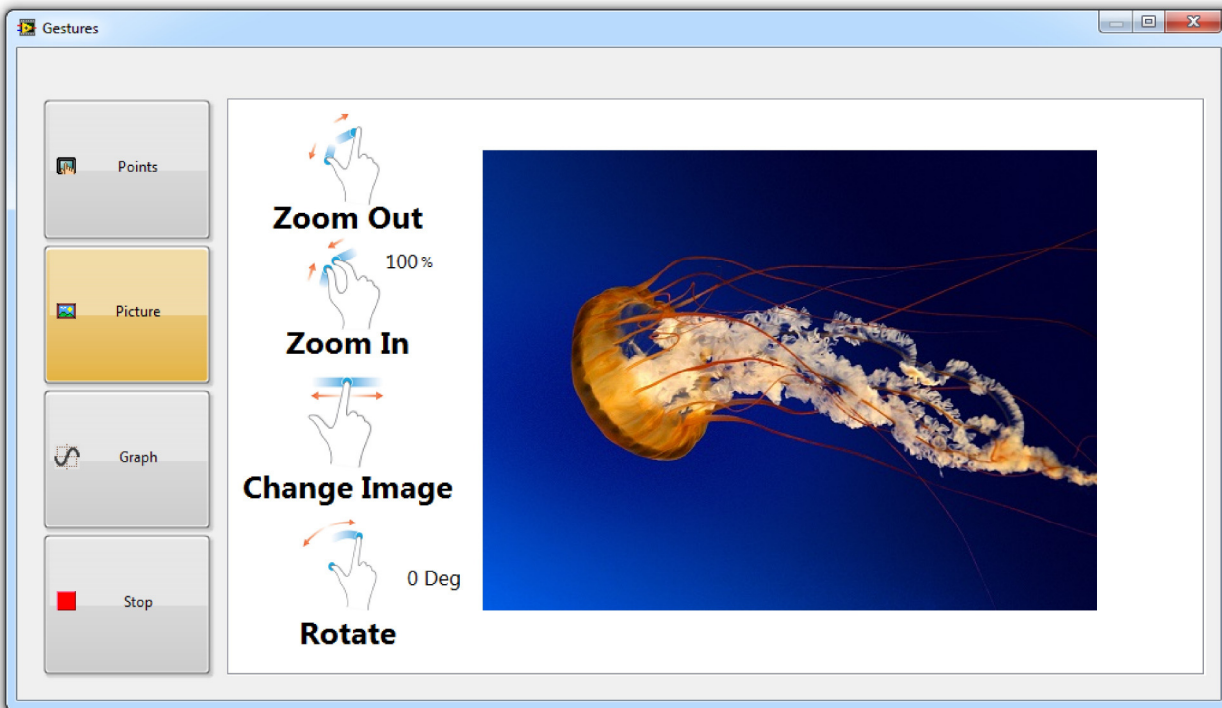
**Installed With:** LabVIEW

A fully featured example that shows how the UPDD toolkit can be used in multi-touch environments to process gestures and detect touches using the UPDD driver. This example demonstrates detection and dragging of two independent touch points, detecting presses of multiple buttons simultaneously, flipping by swipe, zooming and rotating images in a picture control, and zooming/panning of a waveform graph. This example has been specifically developed for use with a MIMO Magic Touch 10.1" Capacitive Touch monitor and should be maximized on the monitor before running.

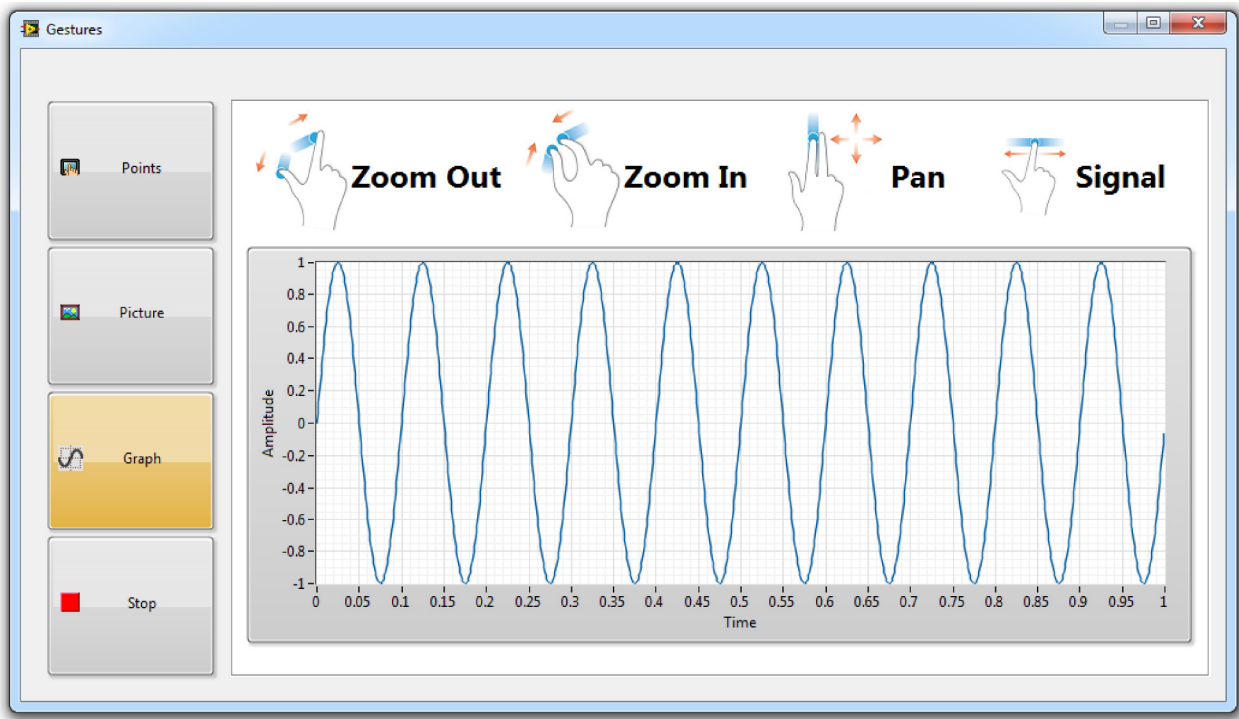
Use multiple touches to demonstrate tracking two movements within the touch area. Also, pressing both Button 1 and Button 2 simultaneously show how to detect multiple button presses at the same time.



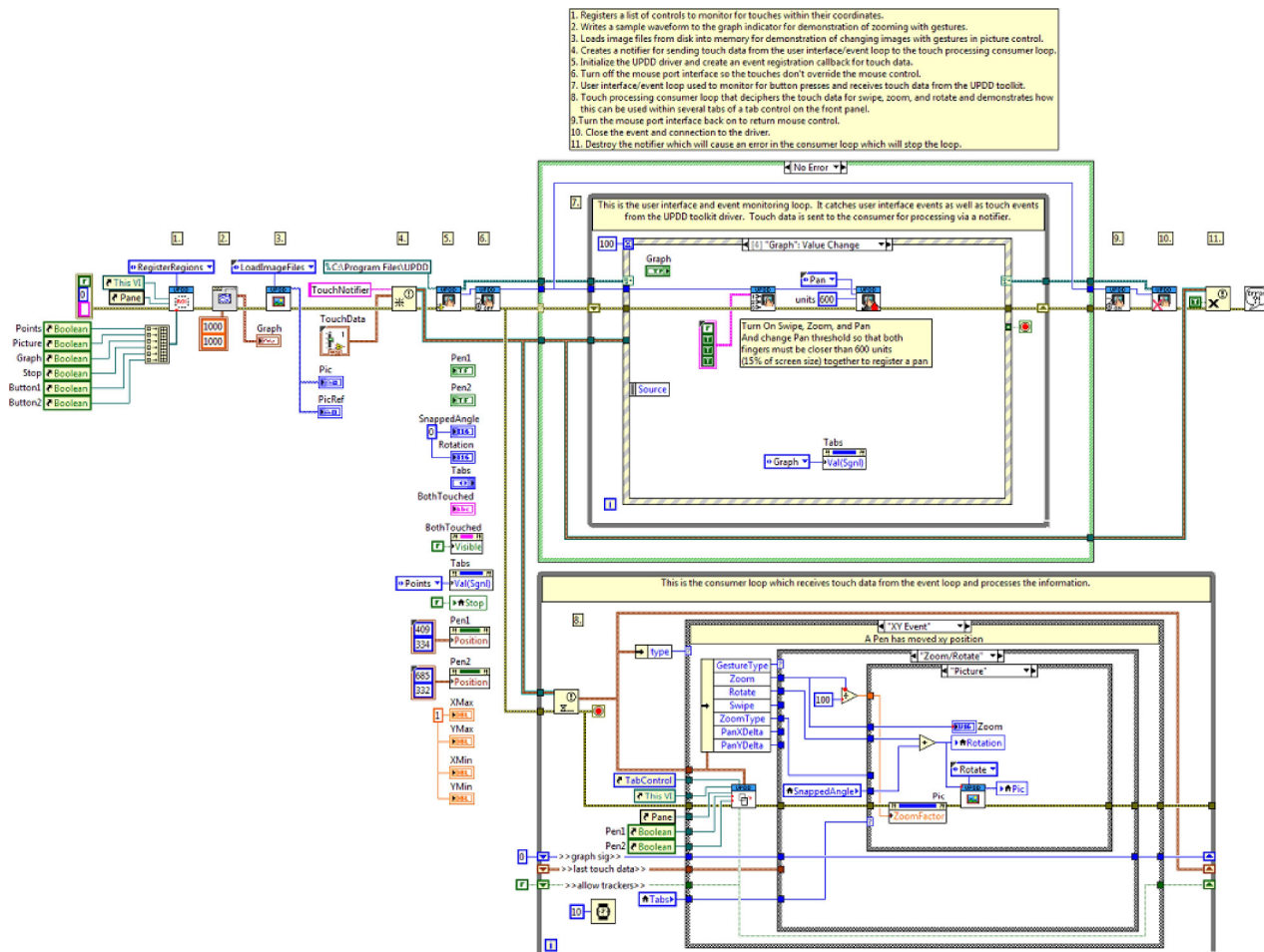
Use a left to right and right to left swiping motion to change the image. Use zoom contract/expand gestures to zoom the picture in and out. Rotate one finger around the other as it remains stationary to rotate the image clockwise and counterclockwise. The image will snap to increments of 90 degrees when the touch points are removed.



Use horizontal zoom gestures to zoom in and out in the horizontal (time) axis. Use vertical zoom gestures to zoom in and out in the vertical (Amplitude) axis. Place two fingers close together on the graph then move left/right/down/up to pan the graph. Use horizontal swiping movements to change the signal type.



The example uses two functional loops. The top loop is a producer loop that monitors for touch events from the toolkit driver as well as for user events. Touch events are passed to the bottom (consumer loop) using a lossy notifier. Since it is streaming XY data it can be lossy. However, it is important that the button press/release events are always detected. To make sure this happens, the producer loop waits for a listener before sending the event. The consumer loop reads the touch data and determines how to process the gestures to control either the multi-touch trackers, the picture, or graph control.

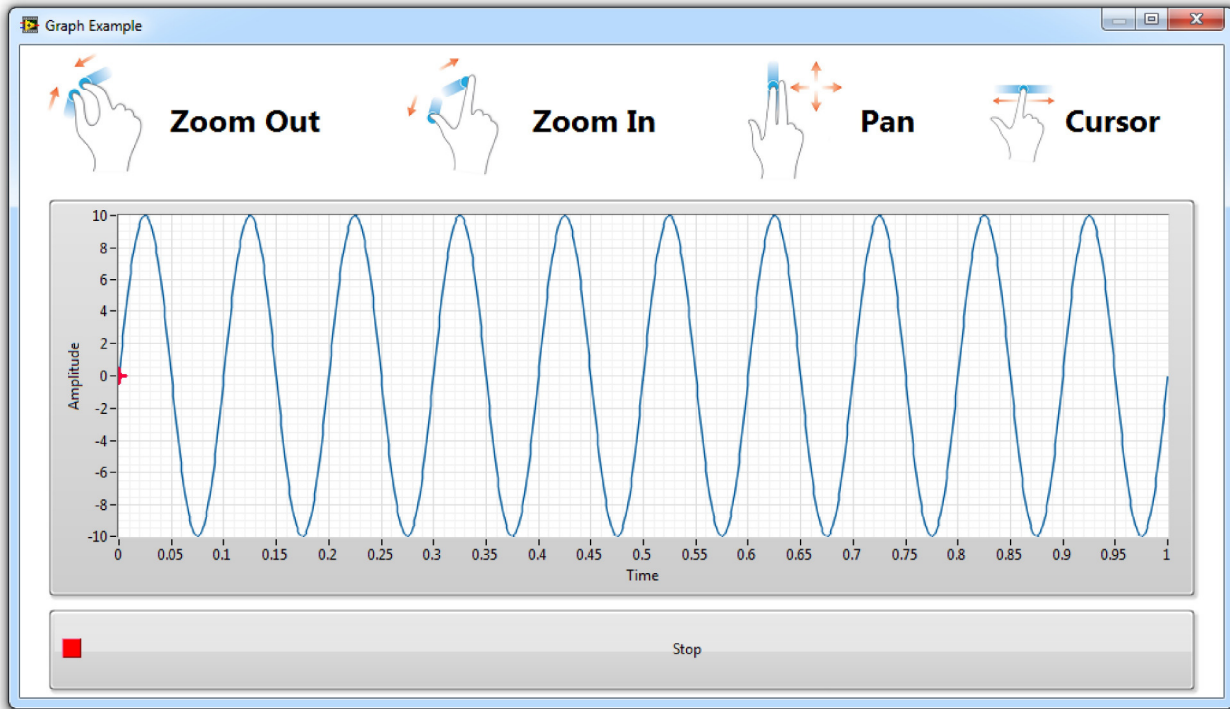


## UPDD Graph Example VI

Installed With: LabVIEW

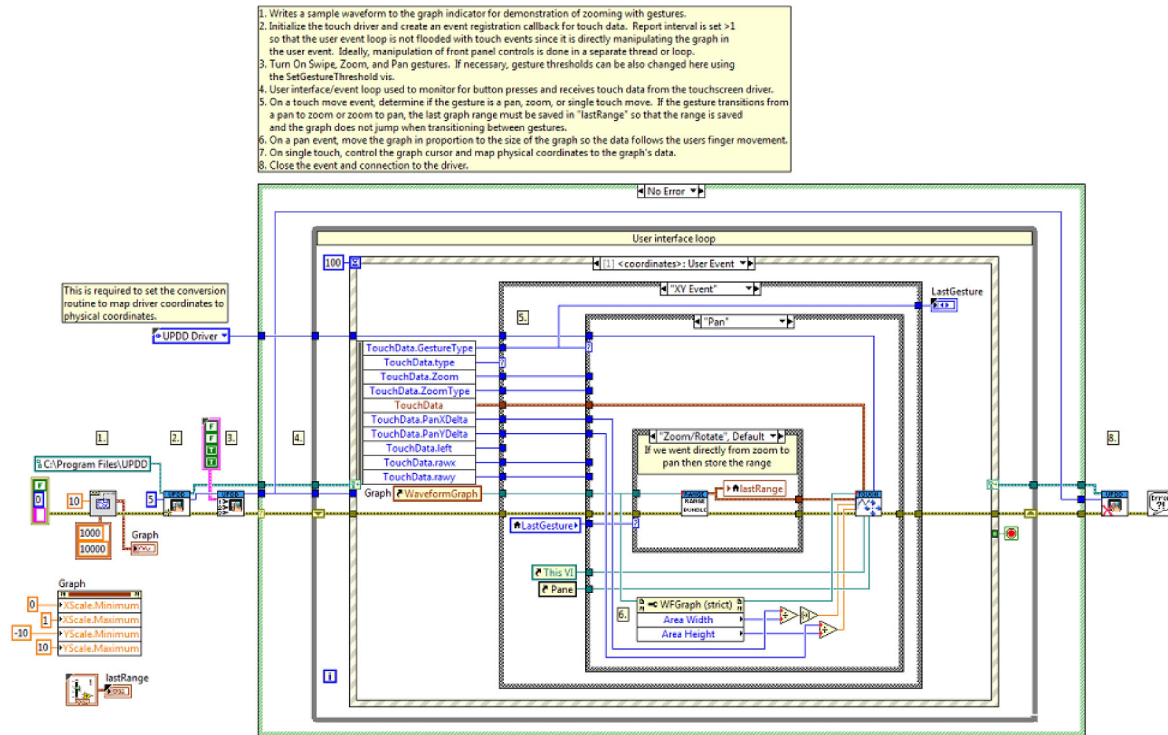
Demonstrates how to use the UPDD API of the touchscreen toolkit to manipulate the x and y scales of a waveform graph. This example detects the zoom, rotate, and pan gestures from the toolkit and modifies the scales of the graph control. This can be useful for improving usability of LabVIEW touch applications display data to the user on the front panel and allowing the user to interact with the data as they would expect on a mobile device. These features are only possible with multi-touch capable monitors that can detect 2 or more touch points simultaneously. This example has been specifically developed for use with a MIMO Magic Touch 10.1" Capacitive Touch monitor and should be maximized on the monitor before running.

Use zoom contract/expand gestures to zoom the graph in and out. This effectively changes the scale and centers the zoom around the center point between the two touch points. Place both fingers close together and move them around to pan the graph. Use a single touch along the graph to move the graph cursor.



The example uses one user interface loop that monitors for touch events from the touchscreen API through user events in the event structure. The x and y scales of the graph control are modified directly in the event handler. This can be done if the processing of the gesture does not take a significant amount of time. During initialization in this example, the touchscreen API has been configured to notify on every 5th touch event, which significantly reduces the number of events that must be handled by the user event loop, therefore, this allows enough time to process the touch event directly in the event handler. It is advised to handle and process touch and gesture messages in a separate loop or thread so that the user interface does not become unresponsive. The example here shows everything processed in the user event handler for simplicity, but more complex applications would likely require processing of touch and gesture messages in a separate thread.



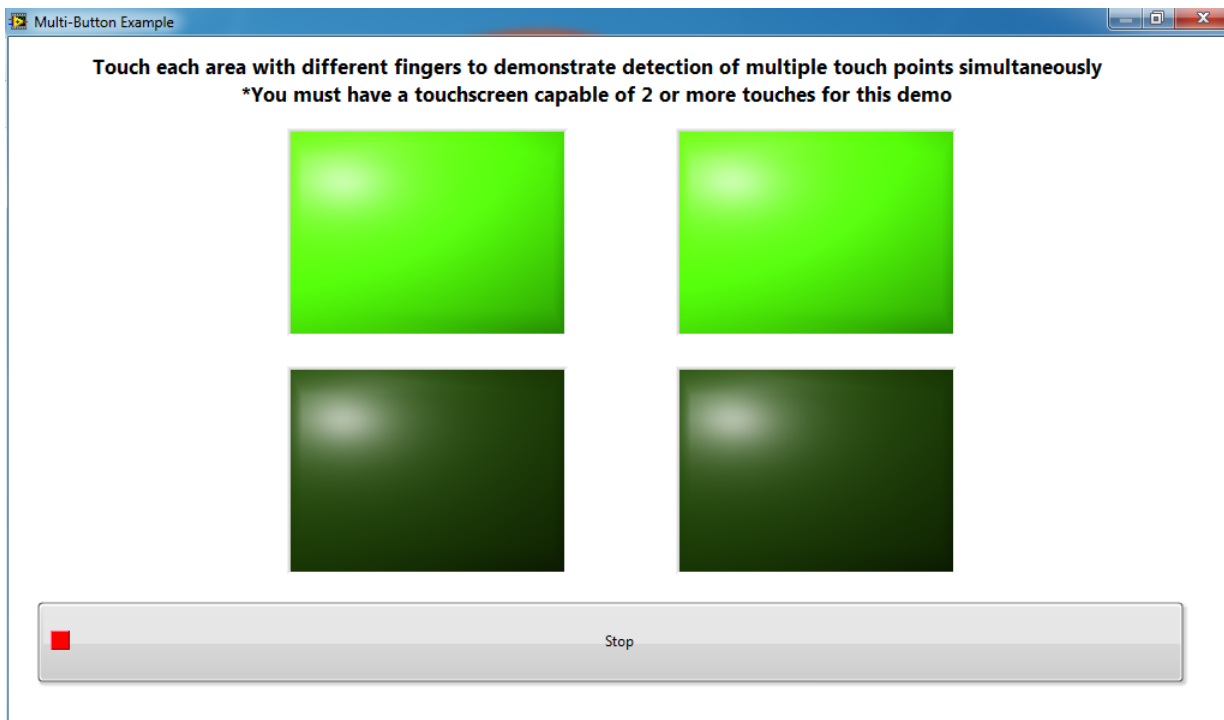


## UPDD Multi-Button Example VI

Installed With: LabVIEW

Demonstrates how to use the touchscreen toolkit to detect touch on multiple areas of interest on the front panel simultaneously using the UPDD API. A typical application would be to detect touch of two buttons simultaneously to ensure operators hands are away from machinery before starting a dangerous piece of machinery. This eliminates the need for external mechanical buttons. These features are only possible with multi-touch capable monitors that can detect 2 or more touch points simultaneously. This example receives touch data from the UPDD driver API instead of the Windows Touch API. This is required for systems that do not have the Windows Touch API, like Windows XP and Windows Embedded Standard 7 (WES7) without the Windows Pen and Touch Add-on.

Press any number of the 4 touch areas to see them detected simultaneously.





Note: Some touchscreens only support 2 touches. For all 4 touch areas to be recognized simultaneously, you must have a touchscreen that supports 4 or more simultaneous touch points. Refer to system properties to identify the number of supported touch points.

## System

Rating: **7.2** Windows Experience Index

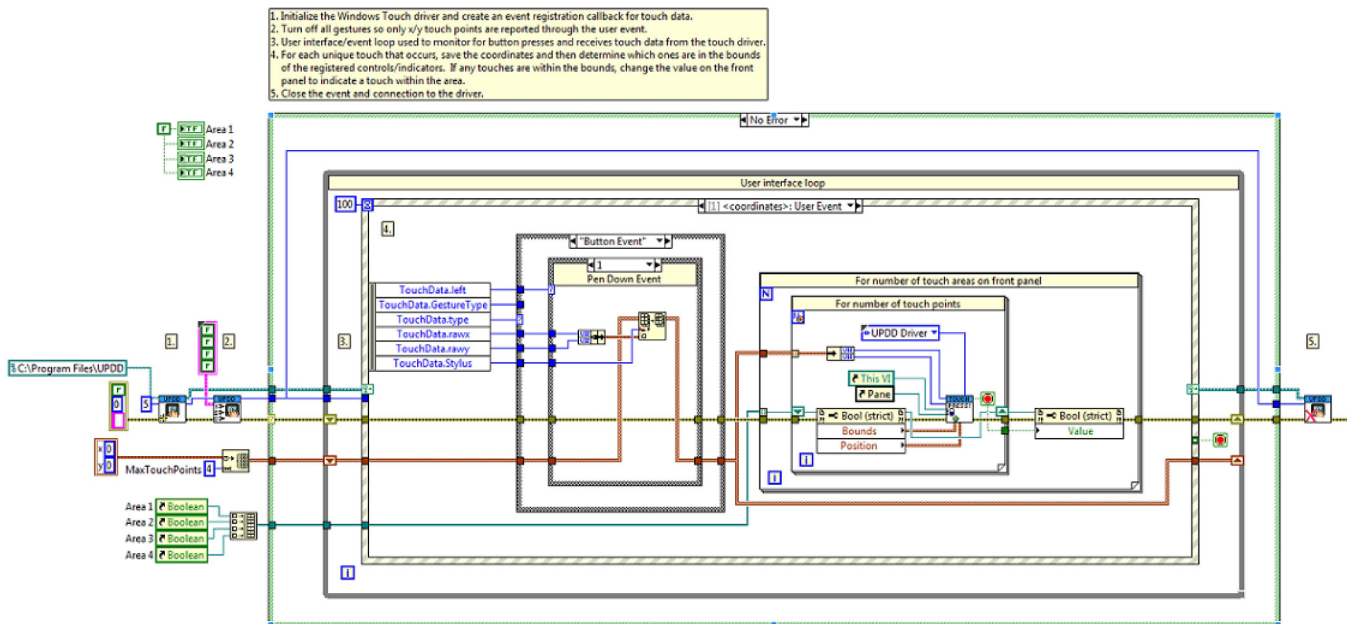
Processor: Intel(R) Core(TM) i7-3840QM CPU @ 2.80GHz 2.80 GHz

Installed memory (RAM): 16.0 GB

System type: 64-bit Operating System

Pen and Touch: Touch Input Available with 16 Touch Points

The example uses one user interface loop that monitors for touch events from the touchscreen API through user events in the event structure. The value property of the touch indicators are modified directly in the event handler. This can be done if the processing of the gesture does not take a significant amount of time. During initialization in this example, the touchscreen API has been configured to notify on every 5th touch event, which significantly reduces the number of events that must be handled by the user event loop, therefore, this allows enough time to process the touch event directly in the event handler. It is advised to handle and process touch and gesture messages in a separate loop or thread so that the user interface does not become unresponsive. The example here shows everything processed in the user event handler for simplicity, but more complex applications would likely require processing of touch and gesture messages in a separate thread.





## Sensor Interface Help

May 2015

Part of the Touchscreen Toolkit provides a LabVIEW interface to the Windows 7 and 8 Sensor API. Refer to [here](#) for more information. Windows 7 and 8 include native support for sensors, which are devices that can measure physical phenomena such as temperature or location. Today, software developers can write programs that use sensors, but a lack of standardization makes programming for sensors an arduous task. After a sensor-based program is completed, it is usually forever dependent on a particular type of hardware. Using one or more vertical solutions to enable deployment of a software-based system has limited the integration of sensors with computer hardware and, until now, Windows-based computers have been no exception. Windows 7 and 8 include native support for sensors, expanded by a new development platform for working with sensors, including location sensors, such as GPS devices. The Windows Sensor and Location platform provides a standard way for device manufacturers to expose sensor devices to software developers and consumers, while providing developers with a standardized application programming interface (API) for working with sensors and sensor data.



This API is mainly targeted for Windows based tablets. Small tablets are becoming very commonplace and they are a great platform for touch-based HMI interfaces powered by LabVIEW. With this toolkit, a developer can make use of many onboard sensors that already exist within tablet based hardware. These include accelerometers, gyroscopes, inclinometers and light sensors. Reading the data from these types of sensors is possible through the Sensor API.

Refer to "<LabVIEW>\examples\Aledyne Engineering\Touchscreen Toolkit\" for examples of how to use the Sensor API.

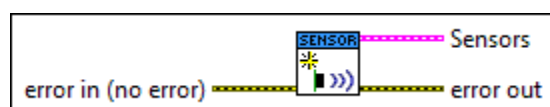
To provide feedback or request support, contact us [here](#).

© 2012–2014 Aledyne Engineering. All rights reserved.

## Initialize Sensors VI

**Installed With:** LabVIEW

Initializes the interface to the Windows sensor interface API. Returns which sensors are detected from the ones that are supported.



The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**Sensors** defines which sensors are present on the current system.



**Accelerometer**



**Inclinometer**



**Gyroscope**



**Orientation**



**Light**


The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



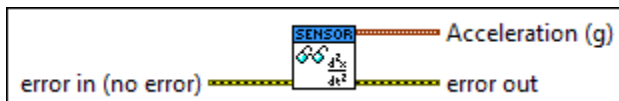
The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



## Read Acceleration VI


**Installed With:** LabVIEW


Reads the x, y, and z axes acceleration values in g's. If the sensor is not present in the system, the call will complete successfully without error, however the returned data will be invalid. Use the returned structure from Initialize Sensors.vi to determine which sensors are available.





The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



**Acceleration** is the accelerometer measurement in g's.


  **X-axis** acceleration, in g's.


 **Y-axis** acceleration, in g's.

 **Z-axis** acceleration, in g's.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

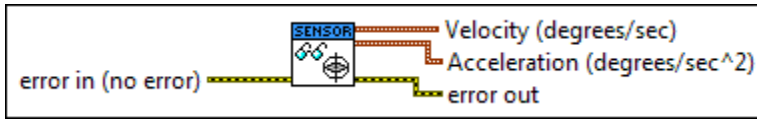
 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


## Read Gyroscope VI


## Installed With: LabVIEW


Reads the x, y, and z axes acceleration values in g's. If the sensor is not present in the system, the call will complete successfully without error, however the returned data will be invalid. Use the returned structure from Initialize Sensors.vi to determine which sensors are available.




The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


**Velocity** is the velocity measurement of the gyroscope in degrees per second.


 **X-axis** velocity in degrees per second.


 **Y-axis** velocity in degrees per second.

 **Z-axis** velocity in degrees per second.


**Acceleration** is the acceleration measurement of the gyroscope in degrees per second squared.


 **X-axis** acceleration in degrees per second squared.


 **Y-axis** acceleration in degrees per second squared.

 **Z-axis** acceleration in degrees per second squared.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

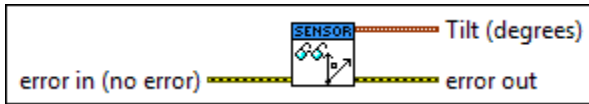
 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


## Read Inclinometer VI


**Installed With:** LabVIEW


Reads the x, y, and z axes tilt angle in degrees. If the sensor is not present in the system, the call will complete successfully without error, however the returned data will be invalid. Use the returned structure from Initialize Sensors.vi to determine which sensors are available.



The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


**Tilt** is the system's tilt measurement in degrees.


 **X-axis** tilt, in degrees.


 **Y-axis** tilt, in degrees.

 **Z-axis** tilt, in degrees.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

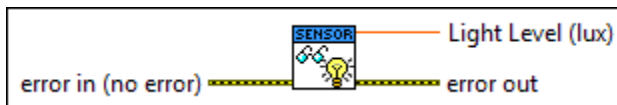
 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



## Read Light VI


**Installed With:** LabVIEW


Reads the x, y, and z axes acceleration values in g's. If the sensor is not present in the system, the call will complete successfully without error, however the returned data will be invalid. Use the returned structure from Initialize Sensors.vi to determine which sensors are available.




The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.



  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 **Light Level** is the Illuminance level, in lux.

The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## Read Orientation VI

**Installed With:** LabVIEW

Reads the x, y, and z axes acceleration values in g's. If the sensor is not present in the system, the call will complete successfully without error, however the returned data will be invalid. Use the returned structure from Initialize Sensors.vi to determine which sensors are available.






The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or



 a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

  The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

**Orientation** is the physical orientation of the device.

0: 0 degrees (Vertical) Counterclockwise

 1: 90 degrees (Vertical) Counterclockwise



2: 180 degrees (Vertical) Counterclockwise


3: 270 degrees (Vertical) Counterclockwise


4: Flat (Horizontal) Orientation

The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

  The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.