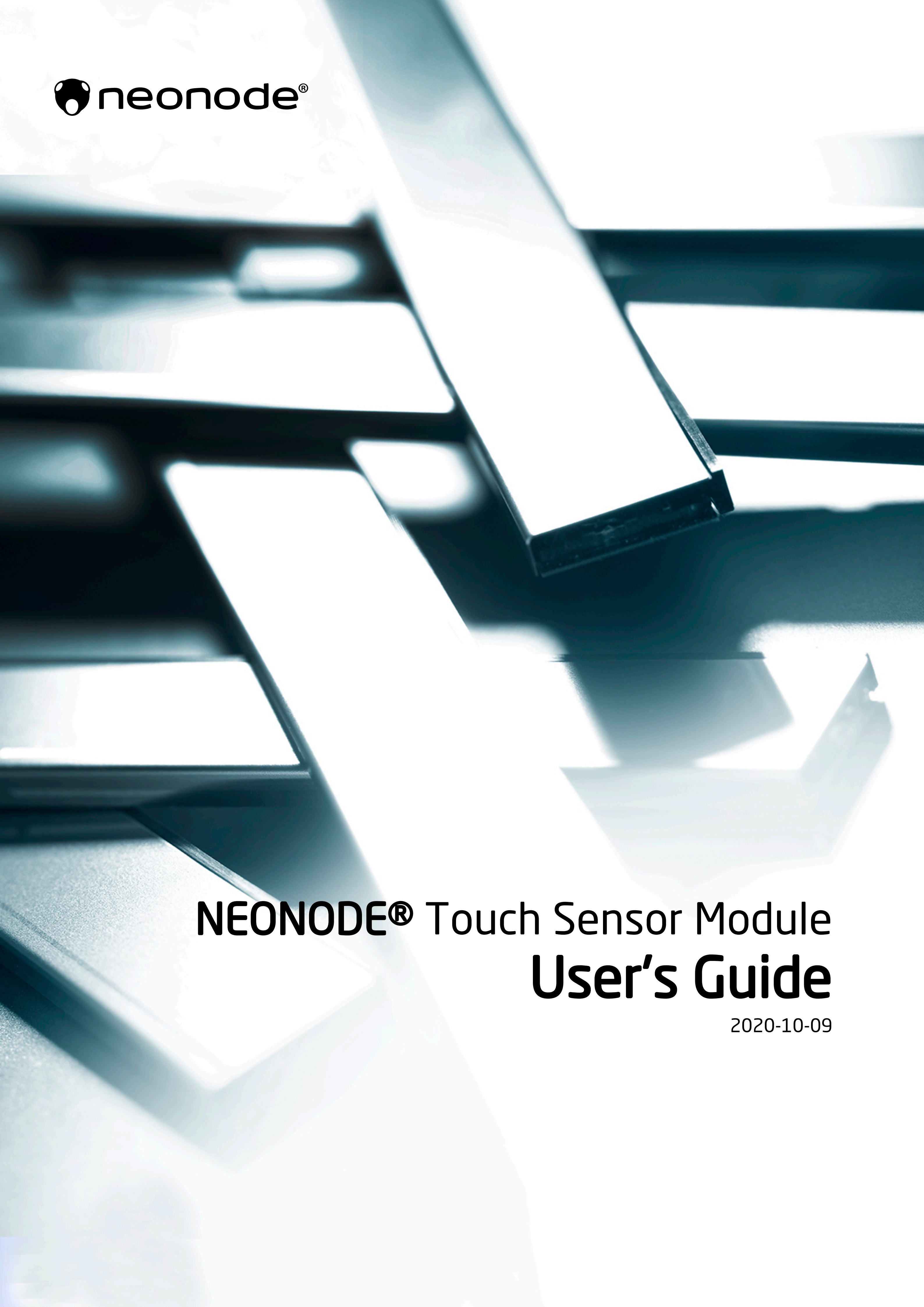


NEONODE® Touch Sensor Module
User's Guide

2020-10-09

Legal Notice

Neonode may make changes to specifications and product descriptions at any time, without notice. Do not finalize a design with this information. Neonode assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using Neonode components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

Neonode components are neither designed nor intended for use in FDA Class III applications, or similar life-critical medical equipment. Customers acknowledge and agree that they will not use any Neonode components in FDA Class III applications, or similar life-critical medical equipment, and that Neonode will not be responsible for any failure to meet the requirements of such applications or equipment.

No part of the materials contained in any Neonode document may be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine-readable form, in whole or in part, without specific written permission from Neonode Inc.

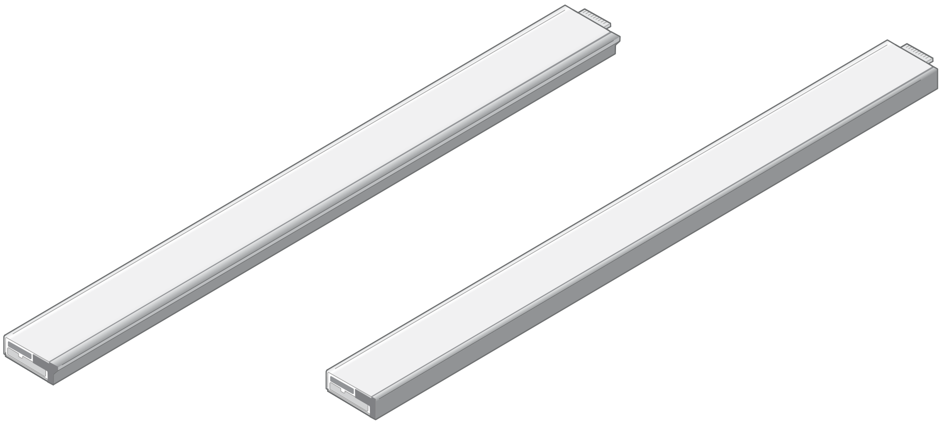
NEONODE, the NEONODE logo, and ZFORCE are trademarks of Neonode Inc. registered in the United States and other countries. All other trademarks are the property of their respective owners.

1 Table of Contents

2 Introduction

2.1 Product Overview

The Neonode Touch Sensor Module (previously referred to as zForce AIR) is a laser light based touch sensor module that can be integrated and used in various applications. The sensor module's characteristics are high scanning frequency, low latency, good touch accuracy and it can be used on any surface or even in-air. The Touch Sensor Module can be connected to the host system through a standard connector and communicate through a standard I2C or USB interface.



2.1.1 Main Features

- Enables touch on any surface or in-air
- Dual touch support
- High scanning frequency – up to 200Hz or more depending on sensor modules length
- Low touch latency
- High touch accuracy
- Idle mode for reduced current power consumption
- Configurable touch active area
- I2C and USB interface
- Standard 5V power supply

2.2 Product Variants

In order to fit in a wide range of applications, the Touch Sensor Module exists in two types, one for horizontal and one for vertical integration, and a number of different lengths.

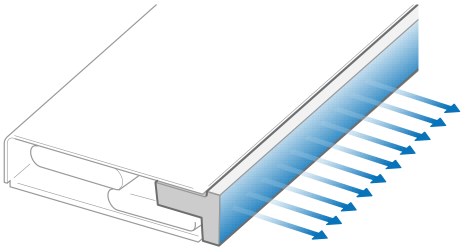
⚠ If the variant you are interested in is not available for purchase from your distributor, please contact the distributor or a Neonode sales representative (refer to www.neonode.com[1]) for more information.

[1] http://www.neonode.com/

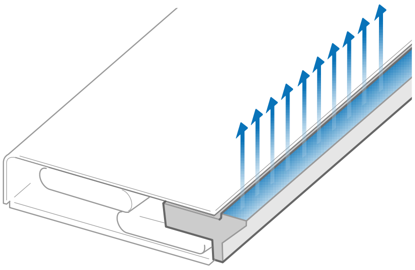
2.2.1 Sensor Module Orientation

The Touch Sensor Module is available in two types, one where the active area emerges straight out from the sensor module (0° type) and one where it emerges out from the sensor module at a 90° angle (90° type). This enables both vertical and horizontal integration.

0° Type



90° Type



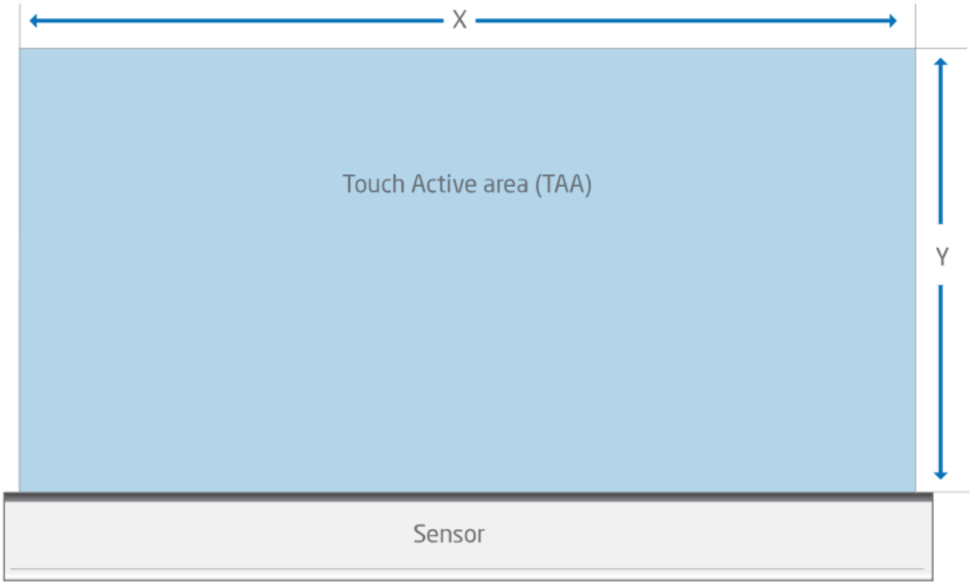
2.2.2 Sensor Module Length

The Touch Sensor Module is available in 43 different lengths. The length affects the Touch Active Area (TAA) in both X and Y directions.

2.2.3 Touch Active Area

The tables list all product variants, the product number, the TAA, and, if applicable, the TAA with Extended Range for each variant. See also Mechanical Data (see page 145).

Sensor modules with X ≥ 237.6 mm are long enough to use a scanning pattern that extends the active area in the Y-direction. The use of the Extended Range scanning pattern is supported from different firmware versions for different product variants, see the following tables.. Extended Range can affect the power consumption and the accuracy.



Product Number		TAA (mm)	
0° Type	**90° Type**	**X**	**Y**
NNAMC0430PC01	NNAMC0431PC01	43.2	14.9
NNAMC0500PC01	NNAMC0501PC01	50.4	29.8
NNAMC0580PC01	NNAMC0581PC01	57.6	29.8
NNAMC0640PC01	NNAMC0641PC01	64.8	44.7
NNAMC0720PC01	NNAMC0721PC01	72	44.7
NNAMC0790PC01	NNAMC0791PC01	79.2	59.6
NNAMC0860PC01	NNAMC0861PC01	86.4	59.6
NNAMC0940PC01	NNAMC0941PC01	93.6	74.5
NNAMC1010PC01	NNAMC1011PC01	100.8	74.5
NNAMC1080PC01	NNAMC1081PC01	108	89.4
NNAMC1150PC01	NNAMC1151PC01	115.2	89.4
NNAMC1220PC01	NNAMC1221PC01	122.4	104.3
NNAMC1300PC01	NNAMC1301PC01	129.6	104.3

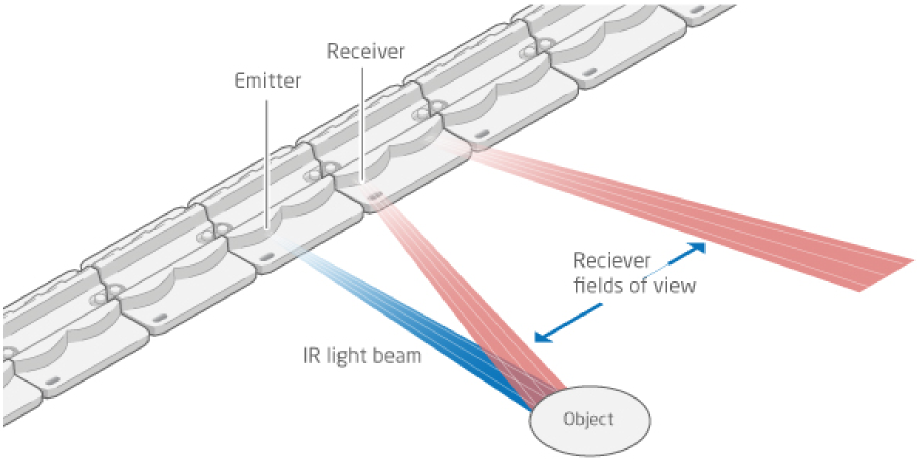
Product Number		TAA (mm)	
0° Type	90° Type	X	Y
NNAMC1370PC01	NNAMC1371PC01	136.8	119.2
NNAMC1440PC01	NNAMC1441PC01	144	119.2
NNAMC1510PC01	NNAMC1511PC01	151.2	134.0
NNAMC1580PC01	NNAMC1581PC01	158.4	134.0
NNAMC1660PC01	NNAMC1661PC01	165.6	148.9
NNAMC1730PC01	NNAMC1731PC01	172.8	148.9
NNAMC1800PC01	NNAMC1801PC01	180	163.8
NNAMC1870PC01	NNAMC1871PC01	187.2	163.8
NNAMC1940PC01	NNAMC1941PC01	194.4	178.7
NNAMC2020PC01	NNAMC2021PC01	201.6	178.7
NNAMC2090PC01	NNAMC2091PC01	208.8	193.6
NNAMC2160PC01	NNAMC2161PC01	216	193.6
NNAMC2230PC01	NNAMC2231PC01	223.2	208.5
NNAMC2300PC01	NNAMC2301PC01	230.4	208.5

Product Number		TAA (mm)		TAA, Extended Range (mm)		
0° Type	90° Type	X	Y	X	Y	From Firmware Version
NNAMC2380PC01	NNAMC2381PC01	237.6	208.5	237.6	223.4	Available on request
NNAMC2450PC01	NNAMC2451PC01	244.8	208.5	244.8	223.4	Available on request
NNAMC2520PC01	NNAMC2521PC01	252	208.5	252	238.3	Available on request
NNAMC2590PC01	NNAMC2591PC01	259.2	208.5	259.2	238.3	Available on request
NNAMC2660PC01	NNAMC2661PC01	266.4	208.5	266.4	253.2	Available on request
NNAMC2740PC01	NNAMC2741PC01	273.6	208.5	273.6	253.2	Available on request
NNAMC2810PC01	NNAMC2811PC01	280.8	208.5	280.8	268.1	Available on request

Product Number		TAA (mm)		TAA, Extended Range (mm)		
0° Type	90° Type	X	Y	X	Y	From Firmware Version
NNAMC2880PC01	NNAMC2881PC01	288	208.5	288	268.1	Available on request
NNAMC2950PC01	NNAMC2951PC01	295.2	208.5	295.2	283.0	Available on request
NNAMC3020PC01	NNAMC3021PC01	302.4	208.5	302.4	283.0	Available on request
NNAMC3100PC01	NNAMC3101PC01	309.6	208.5	309.6	297.9	Available on request
NNAMC3170PC01	NNAMC3171PC01	316.8	208.5	316.8	297.9	Available on request
NNAMC3240PC01	NNAMC3241PC01	324	208.5	324	312.8	Available on request
NNAMC3310PC01	NNAMC3311PC01	331.2	208.5	331.2	312.8	Available on request
NNAMC3380PC01	NNAMC3381PC01	338.4	208.5	338.4	327.7	Available on request
NNAMC3460PC01	NNAMC3461PC01	345.6	208.5	345.6	327.7	v1.49 Extended Range

Basic Principles

The Neonode Touch Sensor Module detect and trace objects by detecting diffusely reflected infrared light. The sensor module comprises an optical system arranged to combine emitted IR beams and receiver fields of view within the same apertures. IR light beams are emitted perpendicular to the output window, while receivers field of view is centered at a certain angle left and right.

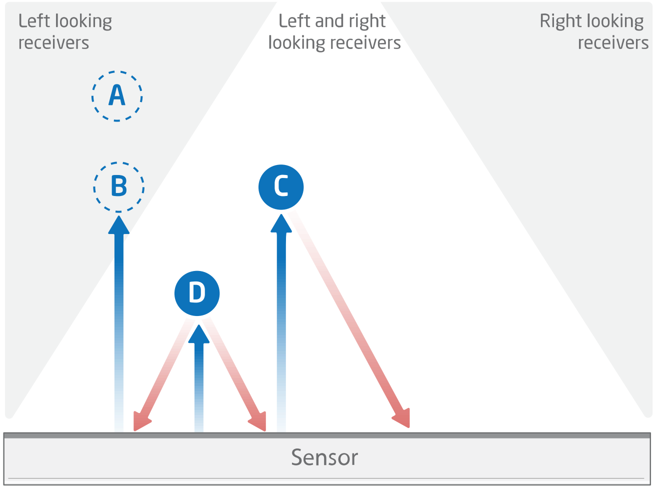


Each emitter-receiver combination covers a narrow region on the active area. An object present in the active area will affect several emitter-receiver channels, and the reported coordinates is the outcome of a center of gravity calculation on these signals.

2.3 Multi-Touch Functionality

The Touch Sensor Module determine an object's position by signals derived from emitter-receiver pairs and have the capacity to detect and track several objects at the same time. Both the hardware and the software have been optimized in order to support standard touch gestures like, pinch-to-zoom, rotate, swipe and tap. However, some combinations of two or more objects might require special consideration, which is described in more detail below.

2.3.1 Shadows



- An object directly behind another object cannot be illuminated. In the figure above, object A will not be detected since illumination is blocked by object B.
- The correct receiver must have a clear field of view. Object B is in a region covered only by left looking receivers. Object B will not be detected because its field of view is blocked by object D.
- Object C may be seen by both left and right looking receivers. Although the right looking field of view is blocked by object D, object C is detected by the left looking receiver.
- Object D is detected by both left and right looking receivers.

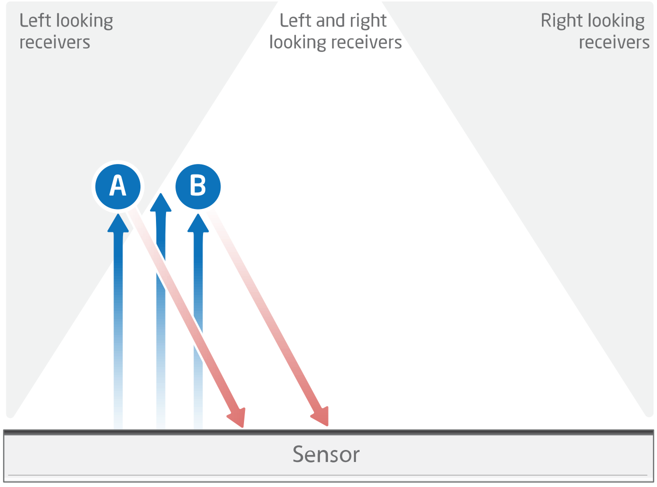
Shadow Trick

Note that in most cases, user experience is not affected by the shadow situations mentioned above. This is because of a functionality implemented in the sensor module's firmware called "shadow trick", which e.g. generates a smooth "rotate" feeling despite one touch object being shadowed during the rotate gesture. A previously detected object that can no longer be detected is still reported as present if:

- The object was last seen close to a location where it could be shadowed by another object.
- The potentially shadowing object is still detected and hasn't moved away from a shadowing location.

The shadow trick make multi-touch gestures such as "rotate" and "pinch-to-zoom" work better.

2.3.2 Adjacent Objects

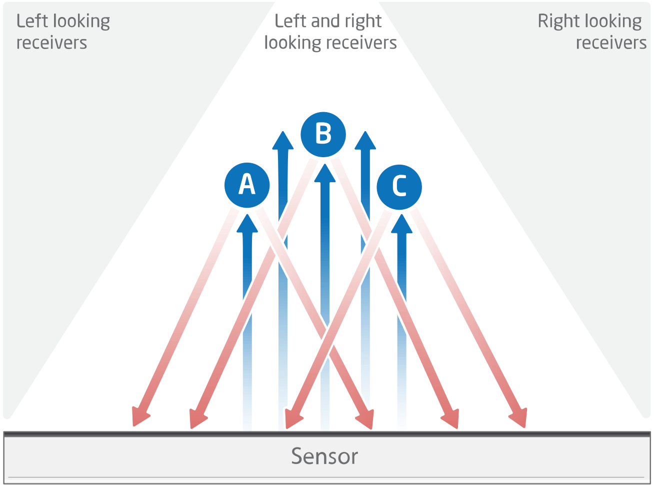


- In order to recognize two objects close to each other (A and B), a separation must allow at least one emitter-receiver channel (~10 mm) to pass freely between them. Otherwise, the two objects will be reported as one large object.

2.3.3 More Than Two Objects

When more than two objects are being tracked the likelihood that an object ends up being in the shadow of another object increases. Therefore, it is only recommended to enable more than two tracked objects if, for example:

- it is not vital to track all detected objects 100% in all possible combinations and locations at all time.
- When all objects are likely to be detected by the sensor module, for example when it is expected that all objects will be placed along a line that is parallel to the sensor module, as in the example below.



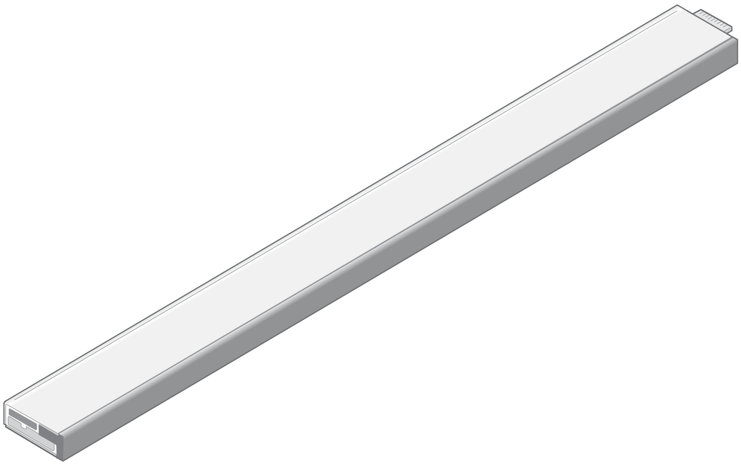
2.4 Applications

The Touch Sensor Module can be integrated for a wide range of applications, such as:

- PCs/Tablets
- TVs/Monitors
- Printers
- Mechanical key replacement
- White goods
- Smart furniture
- Interactive mirrors
- Elevator panels
- eReaders
- Instruments
- Vending Machines
- ATM/POS terminals
- Robotics
- Range finders
- Collision detectors
- ... and much more

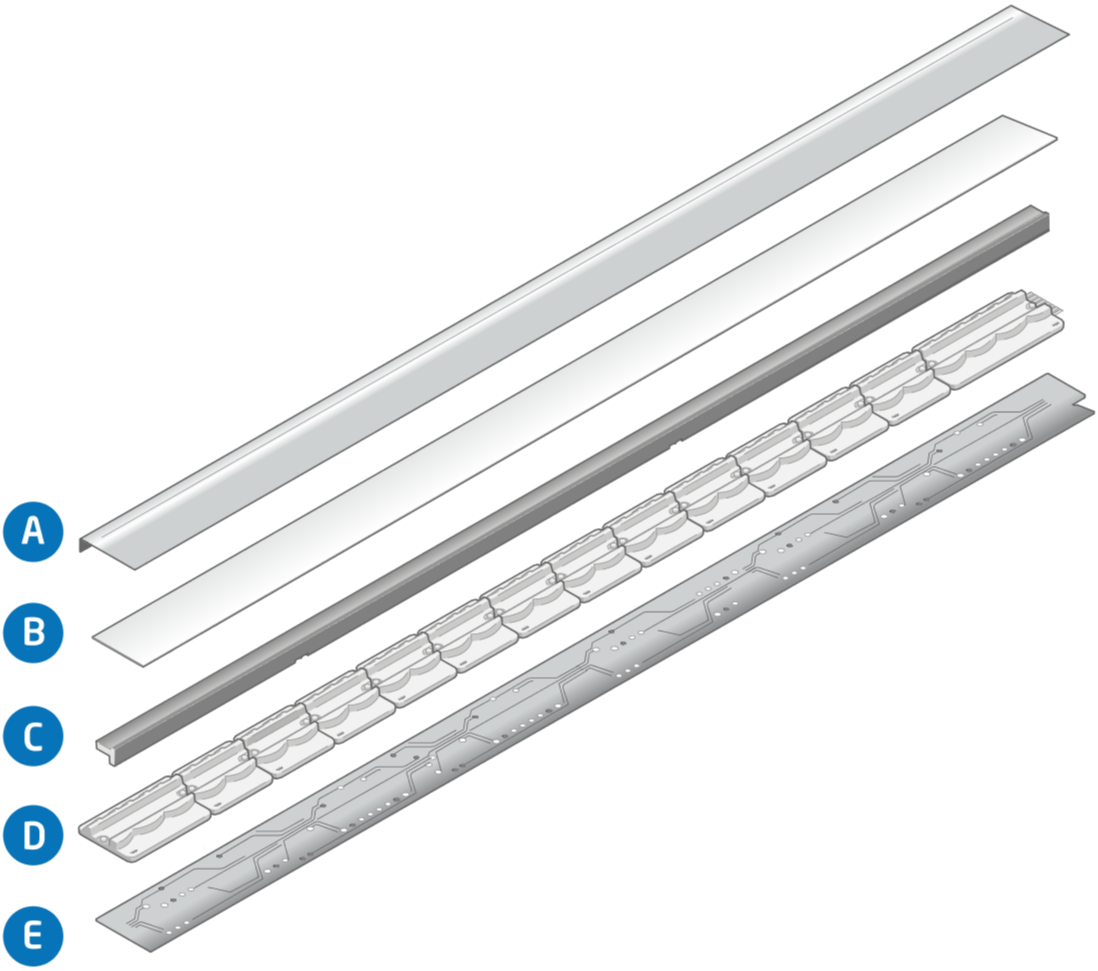
2.5 Sensor Module Design and Components

The Touch Sensor Module is a laser light based touch sensor module that can be used for various touch and in-air detection applications. The image below show the sensor module design (0° type). The connector is shown to the far right.



2.5.1 Exploded view

The image below shows the sensor module (0° type) in an exploded view.



Part	Description
A	Cover
B	Adhesive
C	Front light pipe – straight shooting or 90 degree shooting depending on sensor module's type
D	Lenses - the amount depends on sensor module's length
E	PCBA

2.5.2 Sensor Module Components

The PCBA is equipped with both active and passive components, for example:

- MCU

- Co-processor, a Neonode proprietary scanning IC
- Optical lenses, made out of polycarbonate
- VCSELs
- Photodiodes
- Other passive components

2.6 Product Integration

The Touch Sensor Module can be integrated into any host system through a physical connector with 8 contact pads. The connector supports both I2C and USB HID.

The sensor module communicates with messages that are defined in ASN.1-notation. ASN.1 is a standardized way (ISO/IEC 8824) to describe data regardless of language implementation, hardware system and operation system. The host system can communicate with the sensor module using the zForce communication protocol.[2]

To facilitate integration, Neonode has developed function libraries that are available for download.

[2] https://support.neonode.com/docs/display/AIRTSUsersGuide/zForce+Communication+Protocol

3 Getting started with Touch Sensor Module Evaluation

3.1 Getting Started with Sensor Evaluation - Plug and Play with USB

3.1.1 Required Equipment

The following equipment from the evaluation kit is required:

- 1 x Neonode Touch Sensor Module
- 1 x FPC cable with connector
- 1 x Interface board

Additional required equipment:

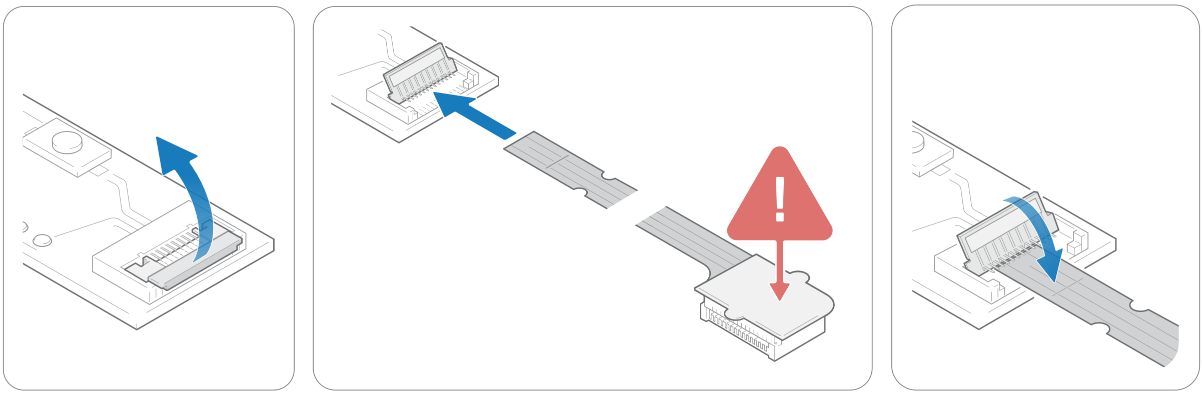
- Computer
  - Operating system: Windows 8.1 or Windows 10.
  - Software requirements: .NET Framework 4.5 or higher is required and can be downloaded from Microsoft's official website. Windows 8 and higher has this installed by default.
- USB cable with a Micro USB type B connector

  > Make sure that the USB cable transmits both power and data and not only power.

- (Optional) tape for mounting

3.1.2 Connecting Sensor Module

1. Connect the FPC cable to the interface board:



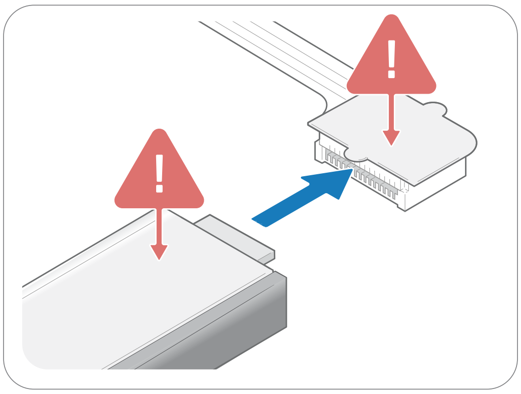
   a. Lift the flip lock on the interface board.
   b. Insert the FPC cable into the end of the connector, with the connector pads facing down, towards interface board. The yellow piece of PCB of the connector on the other side of the cable is facing

upwards. Make sure the direction is straight into the connector and the pads have reached the end of the connector.

c. Make sure the connector pads of the FPC cable are facing downwards, towards interface board. The sensor module risks damage if the FPC cable is connected in wrong direction.

d. Press down the flip lock.

2. Connect the FPC cable to the sensor module:

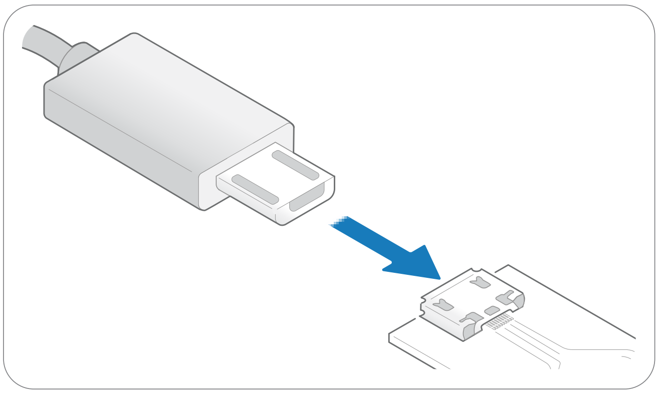


a. Place the sensor module so that the module's connector pads are facing downwards (steel surface upwards).

b. Insert the sensor module into the connector on FPC cable (yellow piece of PCB of the FPC connector still facing upwards).

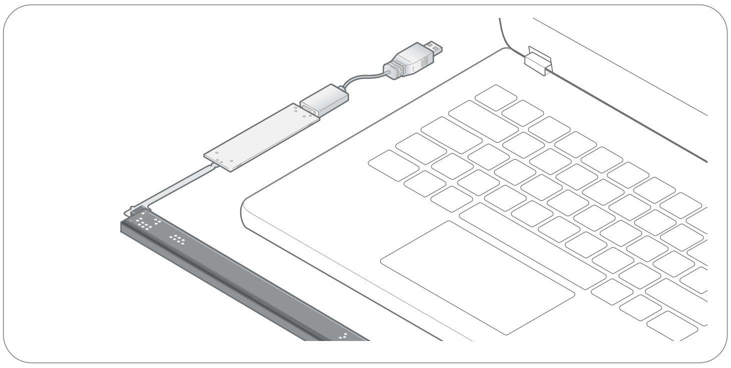
c. Make sure the direction of the pads is straight into the connector, and the pads have reached the end of the connector.

3. Connect a USB cable with a Micro USB type B connector to the interface board.

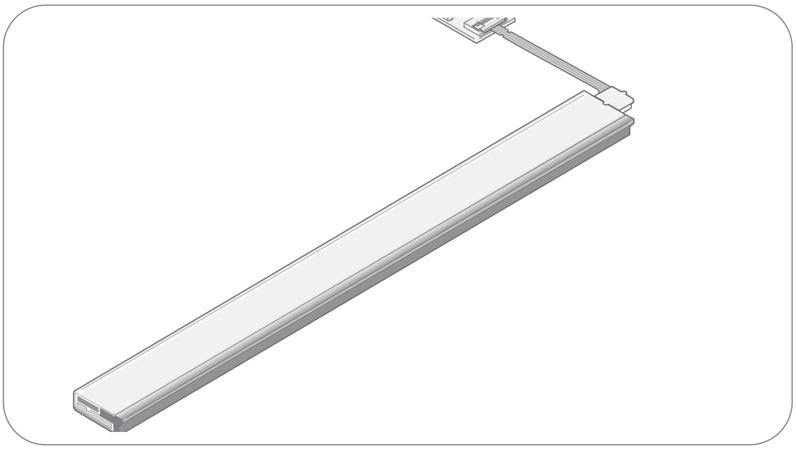


4. Make sure no object is within the touch active area of the sensor module before connecting power to the sensor through USB. The sensor calibrates itself when powered on and an object within the touch active area may interfere with the calibration.

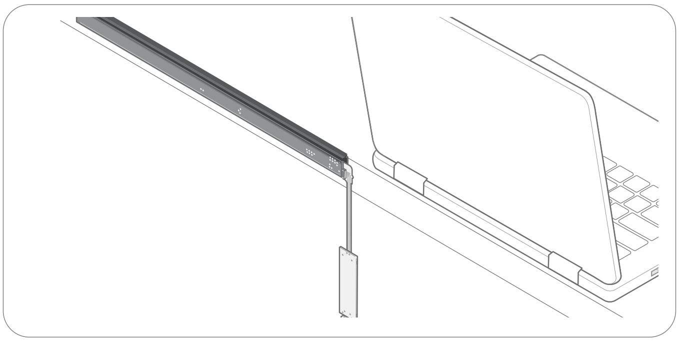
a. If the sensor module is of the 0° type: place the module on a table with the steel surface facing downwards and with the optical surface facing towards you.



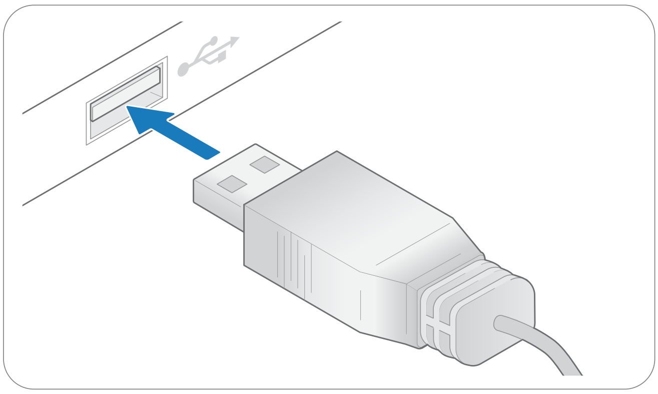
b. If the sensor module is of the 90° type: place the module on a table with the steel surface facing upwards, so the optical surface is facing upwards as well. Make sure no object is within the touch active area above the sensor module.



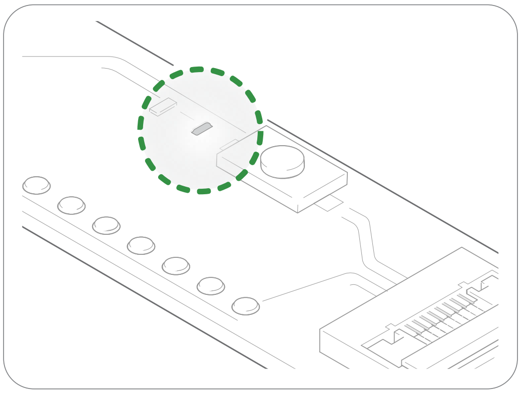
   i. Alternatively, you can mount the sensor module by using tape in order to fasten the steel surface to the edge of a table, with the optical surface facing towards you.



5. Insert the USB cable into a computer.



6. The green LED on the interface board lights up when connected.



7. When the sensor module has enumerated, it will act as a touch screen USB HID device.
8. Put an object in the touch active area, touch HID reports will be sent to your computer.
9. To visualize touches, you can for for example use Paint (default Windows application) and draw lines by moving you finger in the touch active area.

3.2 Getting Started with Sensor Evaluation - Workbench and USB

3.2.1 Required Equipment

The following equipment from the evaluation kit is required:

- 1 x Neonode Touch Sensor Module
- 1 x FPC cable with connector
- 1 x Interface board

Additional required equipment:

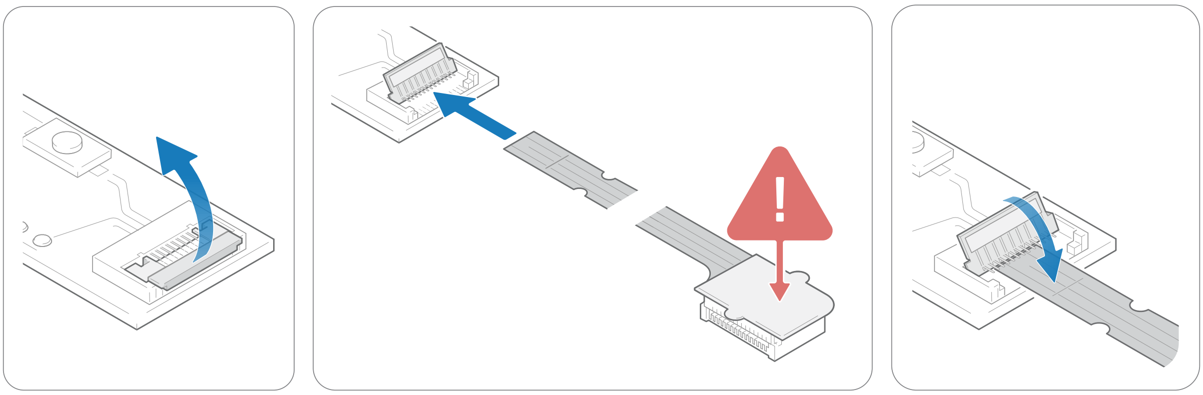
- Computer
  - Operating system: Windows 8.1 or Windows 10.
  - Software requirements: .NET Framework 4.5 or higher is required and can be downloaded from Microsoft's official website. Windows 8 and higher has this installed by default.
- USB cable with a Micro USB type B connector

> ⓘ Make sure that the USB cable transmits both power and data and not only power.

- (Optional) tape for mounting

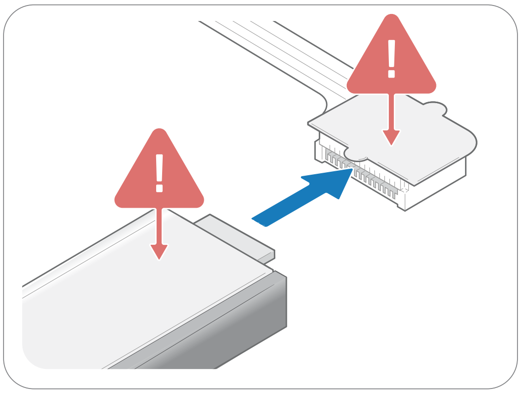
3.2.2 Connecting Sensor Module

1. Connect the FPC cable to the interface board:



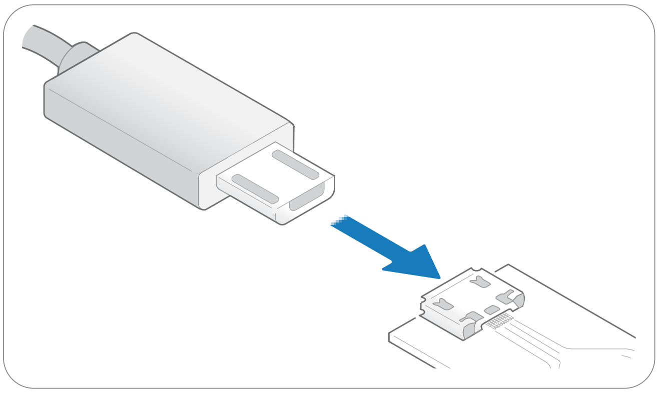
   a. Lift the flip lock on the interface board.
   b. Insert the FPC cable into the end of the connector, with the connector pads facing down, towards interface board. The yellow piece of PCB of the connector on the other side of the cable is facing upwards. Make sure the direction is straight into the connector and the pads have reached the end of the connector.
   c. Make sure the connector pads of the FPC cable are facing downwards, towards interface board. The sensor module risks damage if the FPC cable is connected in wrong direction.
   d. Press down the flip lock.

2. Connect the FPC cable to the sensor module:

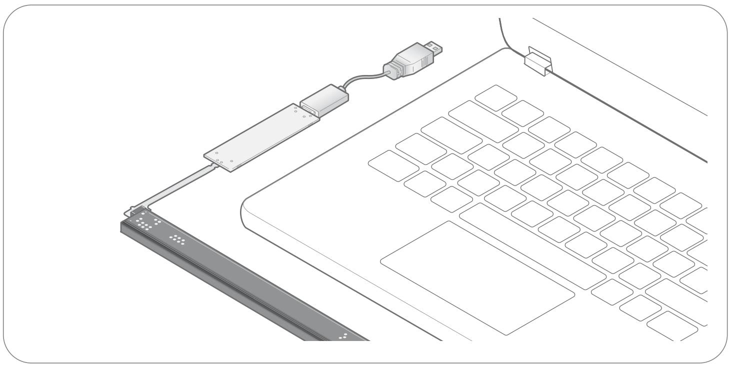


   a. Place the sensor module so that the module's connector pads are facing downwards (steel surface upwards).
   b. Insert the sensor module into the connector on FPC cable (yellow piece of PCB of the FPC connector still facing upwards).
   c. Make sure the direction of the pads is straight into the connector, and the pads have reached the end of the connector.

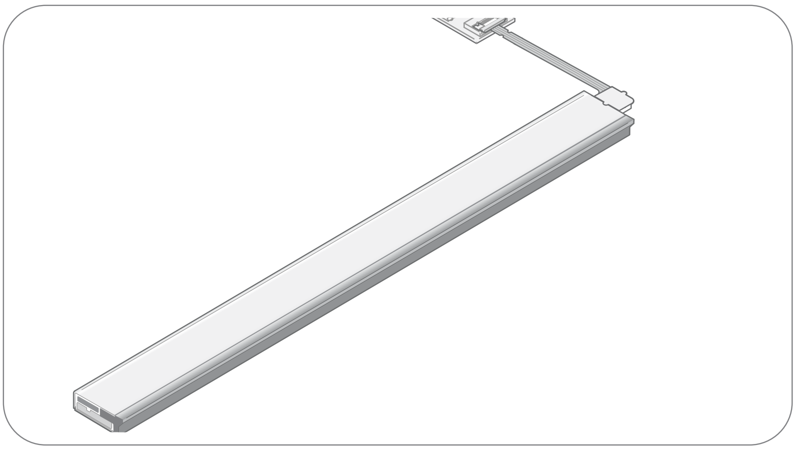
3. Connect a USB cable with a Micro USB type B connector to the interface board.



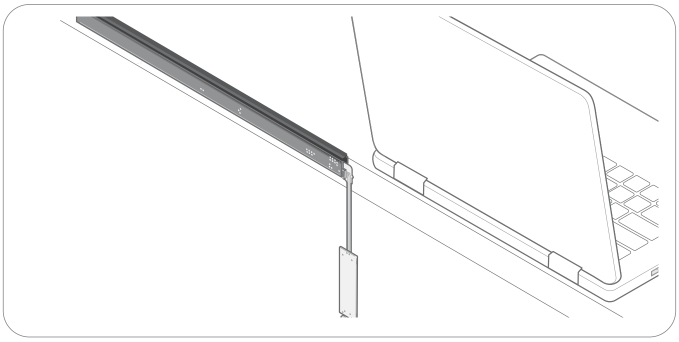
4. Make sure no object is within the touch active area of the sensor module before connecting power to the sensor through USB. The sensor calibrates itself when powered on and an object within the touch active area may interfere with the calibration.
   a. If the sensor module is of the 0° type: place the module on a table with the steel surface facing downwards and with the optical surface facing towards you.



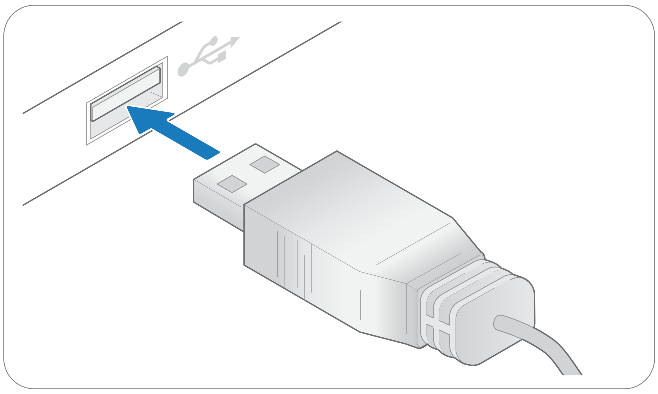
   b. If the sensor module is of the 90° type: place the module on a table with the steel surface facing upwards, so the optical surface is facing upwards as well. Make sure no object is within the touch active area above the sensor module.



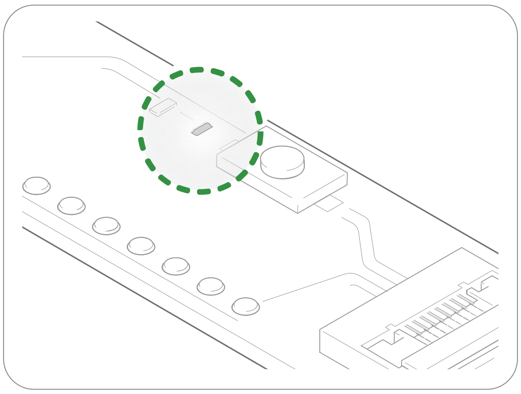
i. Alternatively, you can mount the sensor module by using tape in order to fasten the steel surface to the edge of a table, with the optical surface facing towards you.



5. Insert the USB cable into a computer.



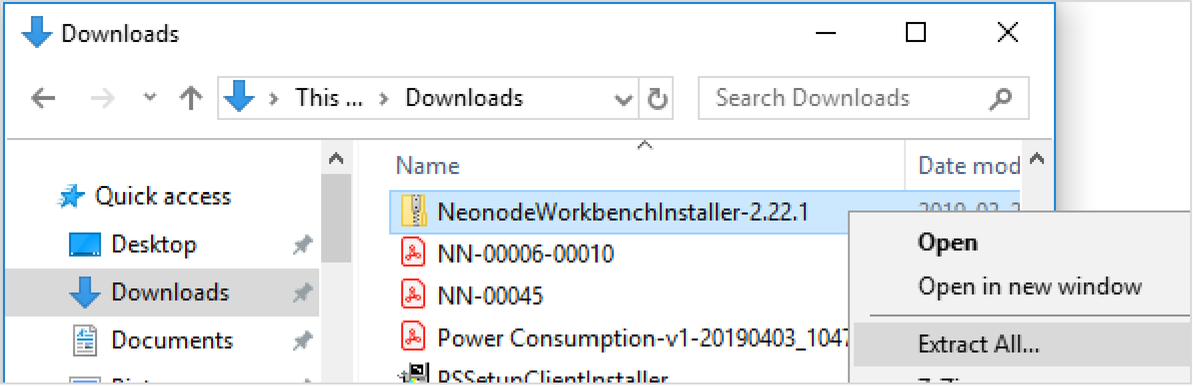
6. The green LED on the interface board lights up when connected.



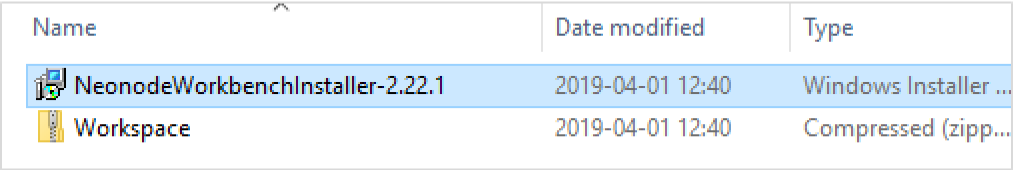
3.2.3 Install and Open Neonode Workbench

1. Download the latest release of the Workbench installation package from https://support.neonode.com/docs/pages/viewpage.action?pageId=2490816.

2. Unzip the installation package.



3. Open the installation package folder.
4. Run the Workbench installer (.msi file) and follow the instructions.

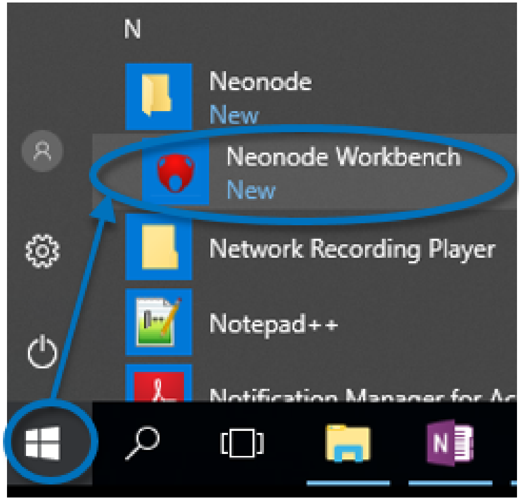


Name	Date modified	Type
NeonodeWorkbenchInstaller-2.22.1	2019-04-01 12:40	Windows Installer ...
Workspace	2019-04-01 12:40	Compressed (zipp...

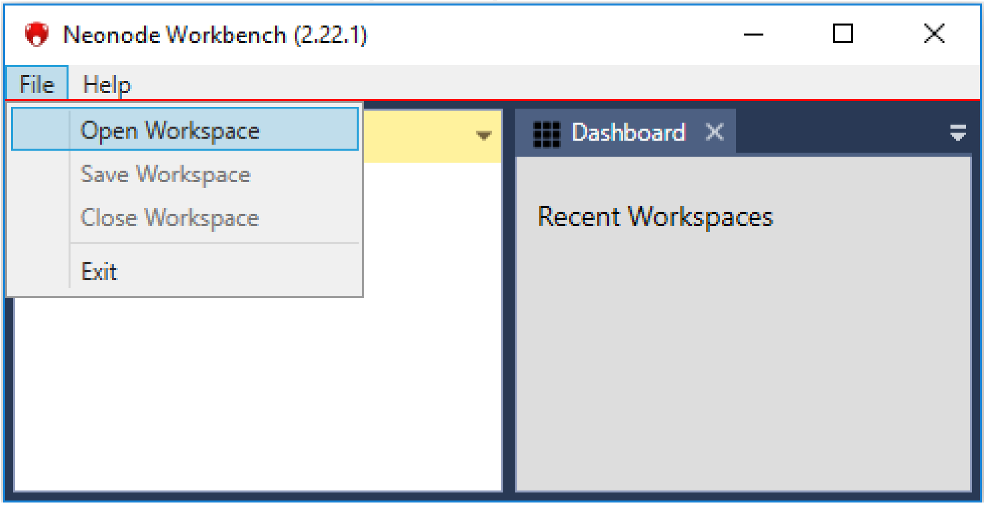
5. Open the installation package folder again.
6. Unzip the Workspace folder to a location where you have write permissions. Write permissions are required to save settings and user data.

> ⓘ In order for the Workbench application to operate, the files in the Workspace folder must be kept together. Move the entire folder if you want to relocate the workspace file.

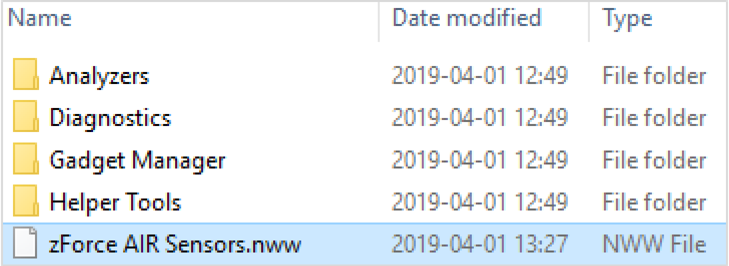
7. Open the Neonode Workbench application.



8. From the toolbar, select **File** >> **Open Workspace**.



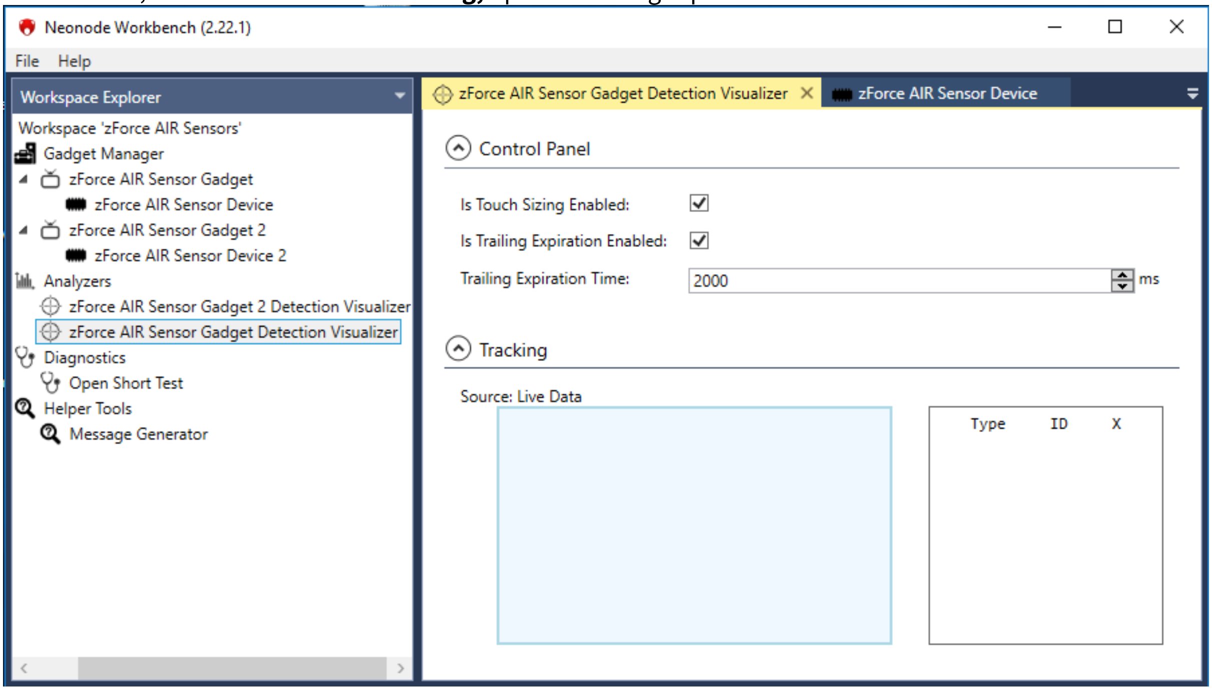
9. Navigate to the Workspace folder and double-click the .nww file inside the folder.



Name	Date modified	Type
Analyzers	2019-04-01 12:49	File folder
Diagnostics	2019-04-01 12:49	File folder
Gadget Manager	2019-04-01 12:49	File folder
Helper Tools	2019-04-01 12:49	File folder
zForce AIR Sensors.nww	2019-04-01 13:27	NWW File

3.2.4 Visualizing Touches with Workbench

1. In the left panel of the workspace, double-click **zForce AIR Sensor Gadget Detection Visualizer.** A tab with two sections, **Control Panel** and **Tracking,** opens in the right panel.


Neonode Workbench (2.22.1)
File
Help
Workspace Explorer
Workspace 'zForce AIR Sensors'
Gadget Manager
zForce AIR Sensor Gadget
zForce AIR Sensor Device
zForce AIR Sensor Gadget 2
zForce AIR Sensor Device 2
Analyzers
zForce AIR Sensor Gadget 2 Detection Visualizer
zForce AIR Sensor Gadget Detection Visualizer
Diagnostics
Open Short Test
Helper Tools
Message Generator
zForce AIR Sensor Gadget Detection Visualizer
zForce AIR Sensor Device
Control Panel
Tracking
Type
ID
X

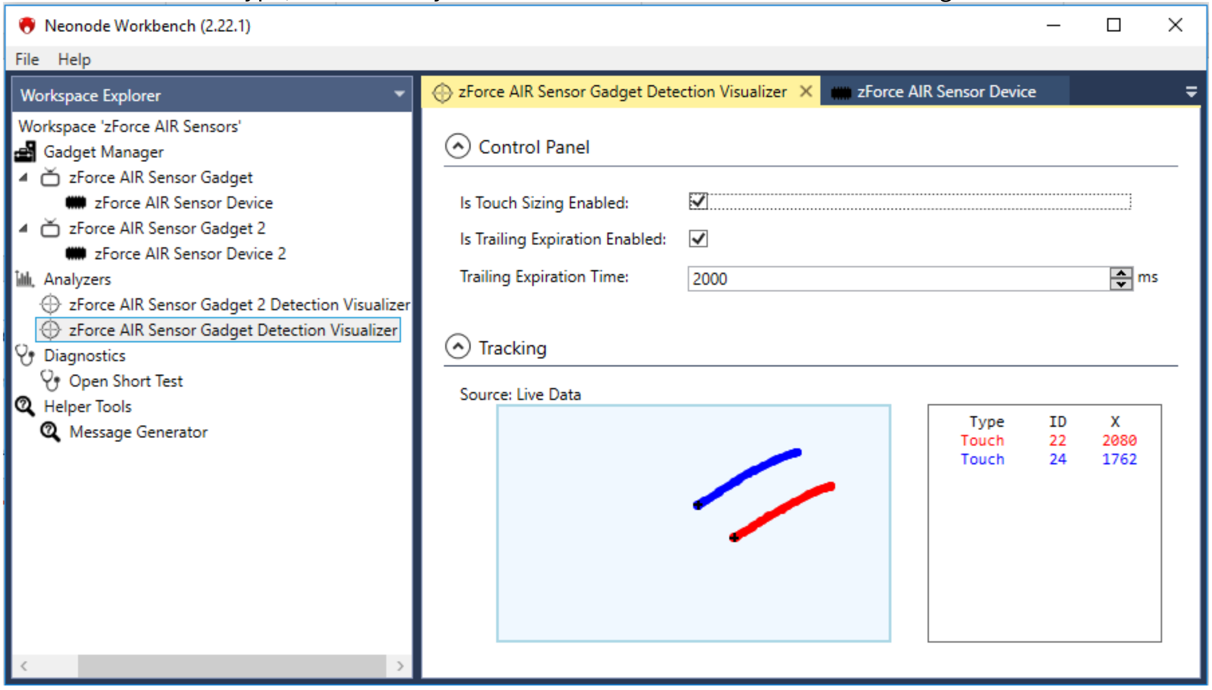
Is Touch Sizing Enabled: ☑

Is Trailing Expiration Enabled: ☑

Trailing Expiration Time: 2000 ms

Source: Live Data

2. If either section in the right panel is collapsed, click ⌄ to expand it.
3. In the right panel
    a. Enable or disable **Touch Sizing**. With Touch Sizing enabled, the size of a detected object is indicated by the size of the tracking cursor.
    b. Enable or disable **Trailing Expiration.** With Trailing Expiration enabled, the trail of a detected object is shown, indicating its movement.
4. Move one or more fingers or other objects in the active area of the sensor module. The registered touches show on the canvas. Type, ID an x- and y-coordinates of each touch is shown to the right of the canvas.


Neonode Workbench (2.22.1)
File
Help
Workspace Explorer
Workspace 'zForce AIR Sensors'
Gadget Manager
zForce AIR Sensor Gadget
zForce AIR Sensor Device
zForce AIR Sensor Gadget 2
zForce AIR Sensor Device 2
Analyzers
zForce AIR Sensor Gadget 2 Detection Visualizer
zForce AIR Sensor Gadget Detection Visualizer
Diagnostics
Open Short Test
Helper Tools
Message Generator
zForce AIR Sensor Gadget Detection Visualizer
zForce AIR Sensor Device
Control Panel
Tracking

Is Touch Sizing Enabled: ☑

Is Trailing Expiration Enabled: ☑

Trailing Expiration Time: 2000 ms

Source: Live Data

Type	ID	X
Touch	22	2080
Touch	24	1762

In Workbench, you can also

- Access sensor information such as firmware version.
- Configure the sensor module to explore different configurations.
- Perform a test to identify any damaged laser or photo diodes.
- Generate sensor messages in hexadecimal format without understanding the structure of the communication protocol message.

For further information, please refer to Workbench documentation [3]

3.3 Getting Started with Sensor Evaluation - I2C and Arduino

3.3.1 Table of Contents

[3] https://support.neonode.com/docs/display/Workbench/Getting+Started+with+Neonode+Workbench.

- Arduino Library (see page 31)

3.3.2 Required Equipment

3.3.3 Required Equipment using Interface Board

The following equipment from the evaluation kit is required:

- 1 x Neonode Touch Sensor Module
- 1 x FPC cable with connector
- 1 x Interface Board

Additional required equipment:

- An Arduino-compatible board. The I2C library described here supports most Arduino-compatible boards.
- An Arduino development environment, for example Arduino IDE.
- USB cable with a Micro USB type B connector

> ⓘ Make sure that the USB cable transmits both power and data and not only power.

- (Optional) tape for mounting

Required Equipment using Neonode Prototyping Board

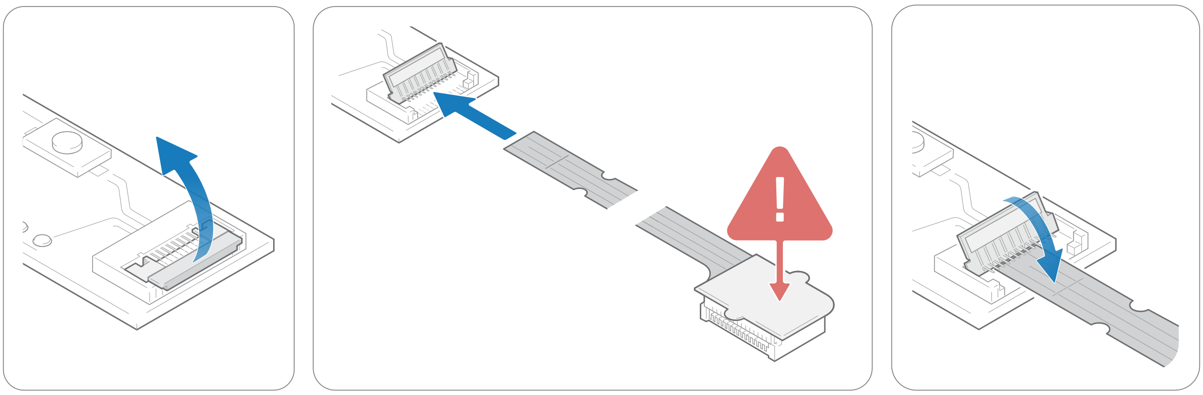
- 1 x Neonode Touch Sensor Module
- 1 x Neonode Prototyping Board
- An Arduino development environment, for example the Arduino IDE.
- USB cable with a Micro USB type B connector

> ⓘ Make sure that the USB cable transmits both power and data and not only power.

- (Optional) tape for mounting

3.3.4 Connecting Sensor Module using Interface Board

1. Connect the FPC cable to the interface board:



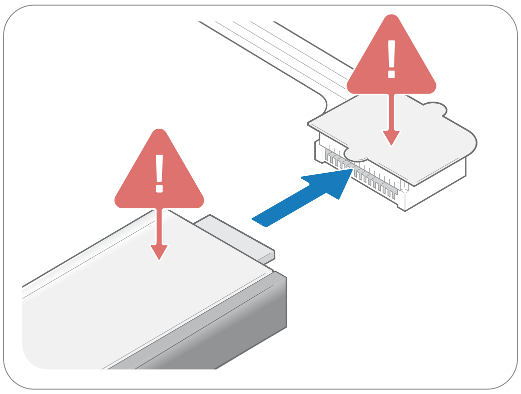
   a. Lift the flip lock on the interface board.
   b. Insert the FPC cable into the end of the connector, with the connector pads facing down, towards interface board. The yellow piece of PCB of the connector on the other side of the cable is facing

upwards. Make sure the direction is straight into the connector and the pads have reached the end of the connector.

c. Make sure the connector pads of the FPC cable are facing downwards, towards interface board. The sensor module risks damage if the FPC cable is connected in wrong direction.

d. Press down the flip lock.

2. Connect the FPC cable to the sensor module:

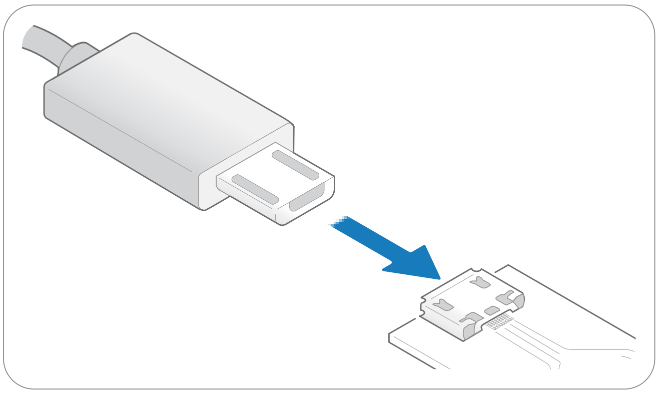


a. Place the sensor module so that the module's connector pads are facing downwards (steel surface upwards).

b. Insert the sensor module into the connector on FPC cable (yellow piece of PCB of the FPC connector still facing upwards).

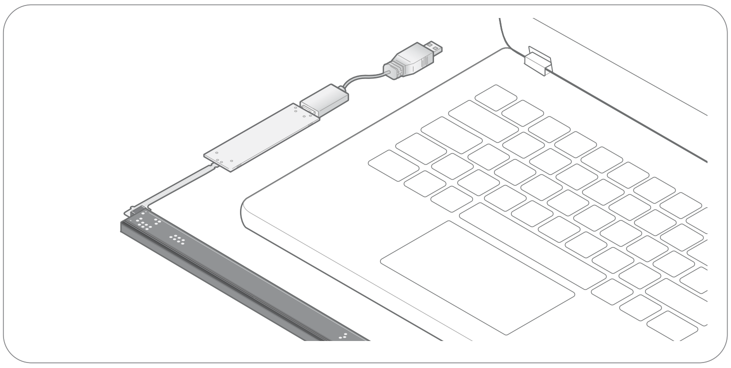
c. Make sure the direction of the pads is straight into the connector, and the pads have reached the end of the connector.

3. Connect a USB cable with a Micro USB type B connector to the interface board.

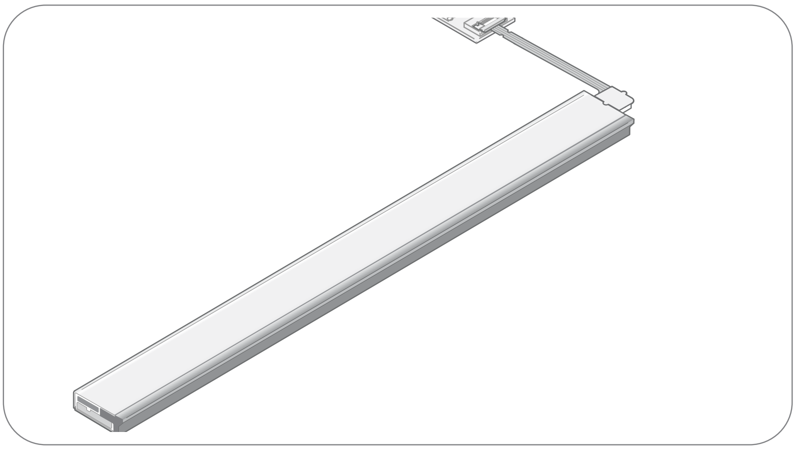


4. Make sure no object is within the touch active area of the sensor module before connecting power through USB. The sensor module calibrates itself when powered on and an object within the touch active area may interfere with the calibration.

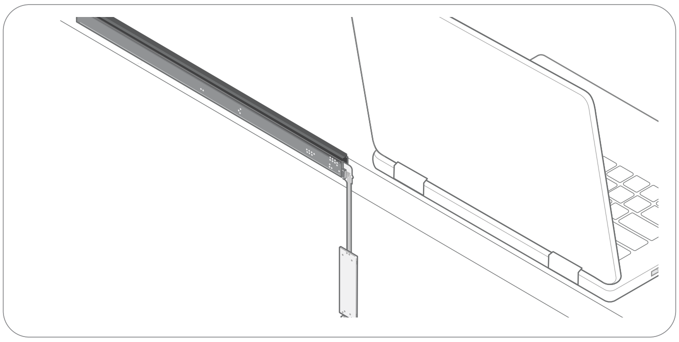
a. If the sensor module is of the 0° type: place the module on a table with the steel surface facing downwards and with the optical surface facing towards you.



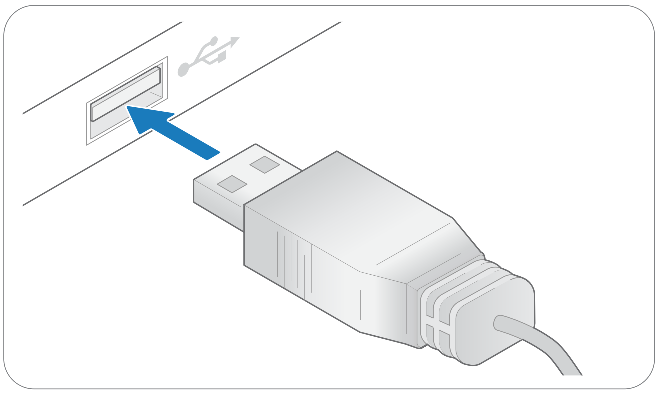
b. If the sensor module is of the 90° type: place the module on a table with the steel surface facing upwards, so the optical surface is facing upwards as well. Make sure no object is within the touch active area above the sensor module.



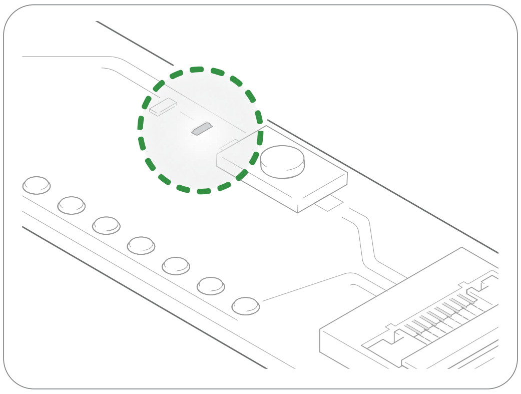
i. Alternatively, you can mount the sensor module by using tape in order to fasten the steel surface to the edge of a table, with the optical surface facing towards you.



5. Insert the USB cable into a computer.



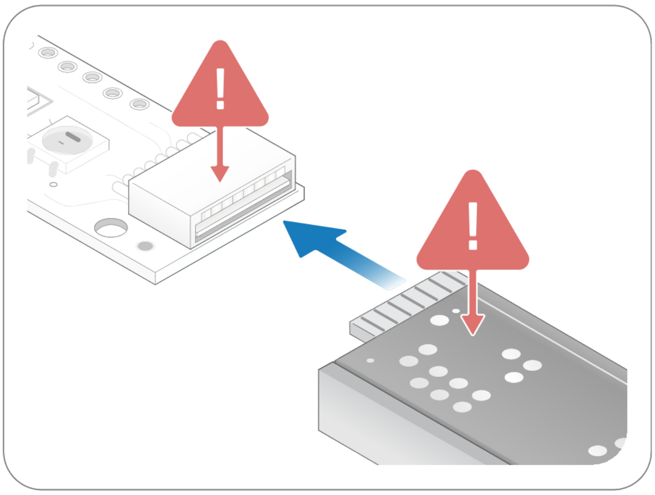
6. The green LED on the interface board lights up when connected.



3.3.5 Connecting Sensor Module using Neonode Prototyping Board

Evaluate Touch Sensor Module using Prototyping Board

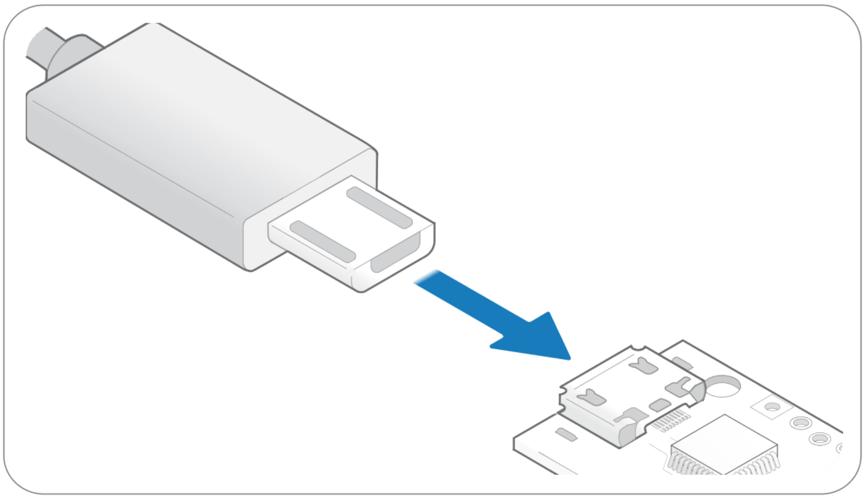
1. Connect the sensor Module to the Prototyping Board

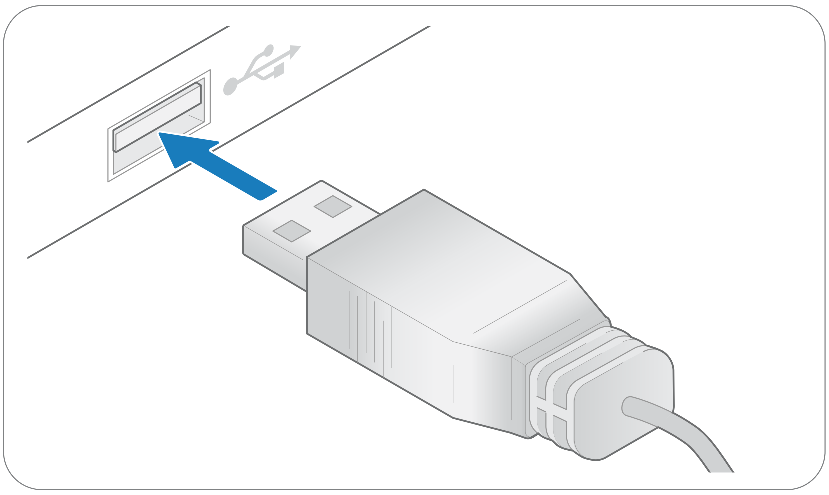


   a. Place the sensor module so that the module's connector pads are facing upwards (black surface upwards).

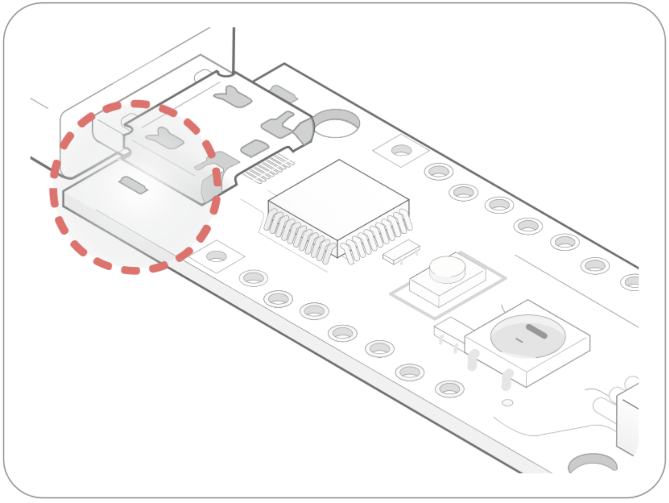
   b. Insert the sensor module to the Prototyping Board's sensor port.

2. The sensor module is now connected to the board, which expose all connections between the sensor module and the board. For details, refer to Electrical Integration (see page 48). Do not connect power until the following steps have been performed.
3. Make sure no object is within the touch active area of the sensor module before connecting power through USB. The sensor module calibrates itself when powered on and an object within the touch active area may interfere with the calibration.
   - a. If the sensor module is of the 0° type: place the module on a table with the steel surface facing downwards and with the optical surface facing towards you.
   - b. If the sensor module is of the 90° type: place the module on a table with the steel surface facing upwards, so the optical surface is facing upwards as well. Make sure no object is within the touch active area above the sensor module.
     - i. Alternatively, you can mount the sensor module by using tape in order to fasten the steel surface to the edge of a table, with the optical surface facing towards you.
4. Connect power to the sensor module through the USB.





a. A red light next to the micro USB port should turn on to indicate power transfer.



5. The Prototyping Board is now ready to be flashed.
6. For further information, please refer to Get Started With Neonode Prototyping Board[4].

3.3.6 Arduino Library

Please refer to our Arduino Library[5].

3.4 Getting Started with Sensor Evaluation - SDK and USB

3.4.1 Required Equipment

The following equipment from the evaluation kit is required:

- 1 x Neonode Touch Sensor Module
- 1 x FPC cable with connector
- 1 x Interface board

Additional required equipment:

- SDK System Requirements[6]
- USB cable with a Micro USB type B connector

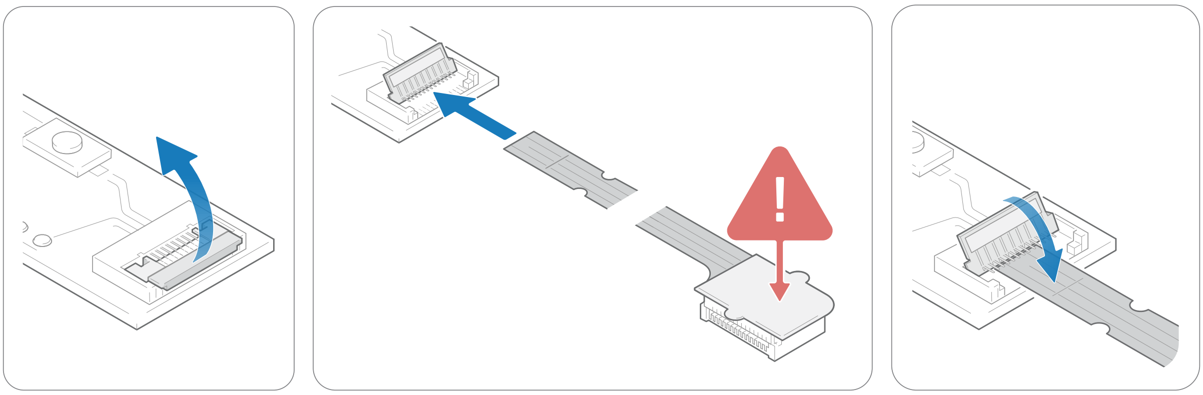
> ⓘ Make sure that the USB cable transmits both power and data and not only power.

- (Optional) tape for mounting.

[4] https://support.neonode.com/docs/display/NPB/Get+Started+with+Neonode+Prototyping+Board
[5] https://github.com/neonode-inc/zforce-arduino
[6] https://support.neonode.com/docs/display/SDKDOC/SDK+System+Requirements

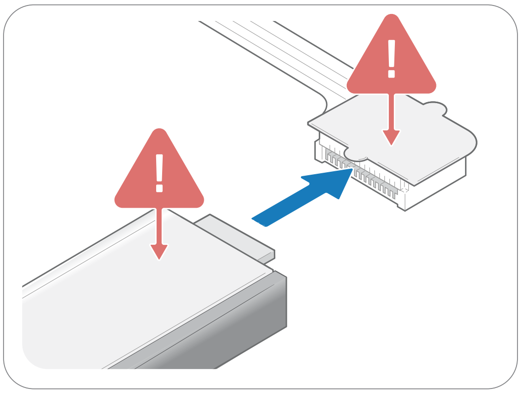
3.4.2 Connecting Sensor

1. Connect the FPC cable to the interface board:



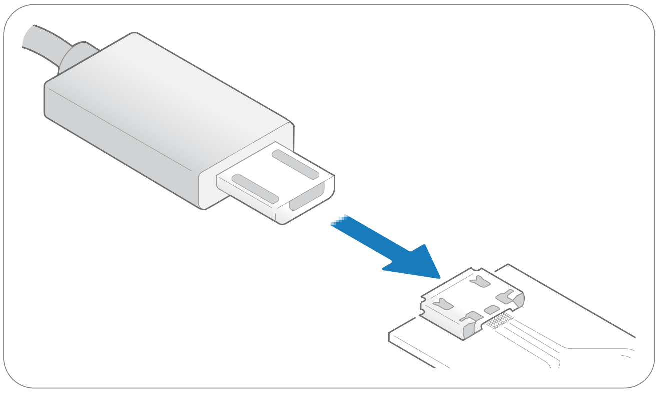
    a. Lift the flip lock on the interface board.
    b. Insert the FPC cable into the end of the connector, with the connector pads facing down, towards interface board. The yellow piece of PCB of the connector on the other side of the cable is facing upwards. Make sure the direction is straight into the connector and the pads have reached the end of the connector.
    c. Make sure the connector pads of the FPC cable are facing downwards, towards interface board. The sensor module risks damage if the FPC cable is connected in wrong direction.
    d. Press down the flip lock.

Connect the FPC cable to the sensor module:



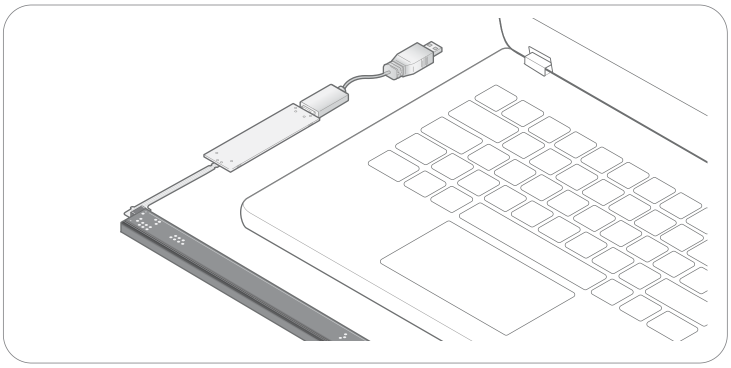
    a. Place the sensor module so that the module's connector pads are facing downwards (steel surface upwards).
    b. Insert the sensor module into the connector on FPC cable (yellow piece of PCB of the FPC connector still facing upwards).
    c. Make sure the direction of the pads is straight into the connector, and the pads have reached the end of the connector.

3. Connect a USB cable with a Micro USB type B connector to the interface board.

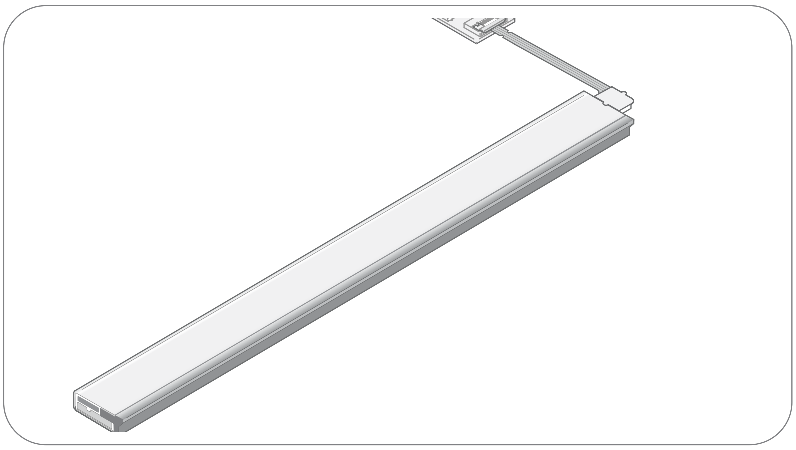


Make sure no object is within the touch active area of the sensor module before connecting power to the sensor through USB. The sensor calibrates itself when powered on and an object within the touch active area may interfere with the calibration.

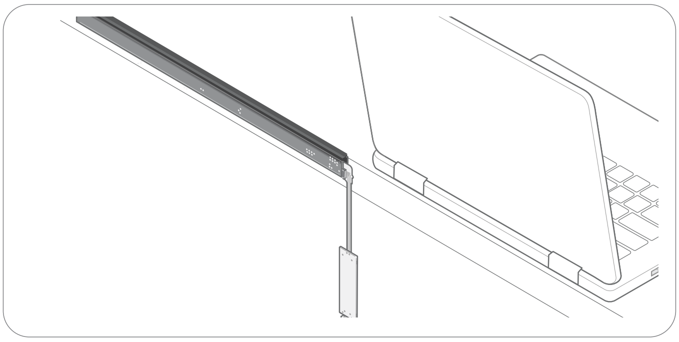
   a. If the sensor module is of the 0° type: place the module on a table with the steel surface facing downwards and with the optical surface facing towards you.



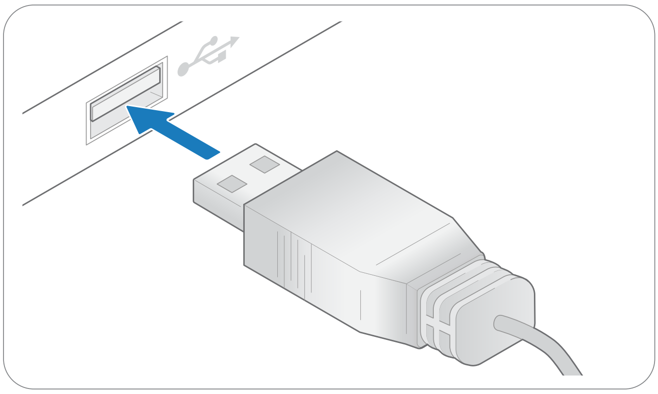
If the sensor module is of the 90° type: place the module on a table with the steel surface facing upwards, so the optical surface is facing upwards as well. Make sure no object is within the touch active area above the sensor module.



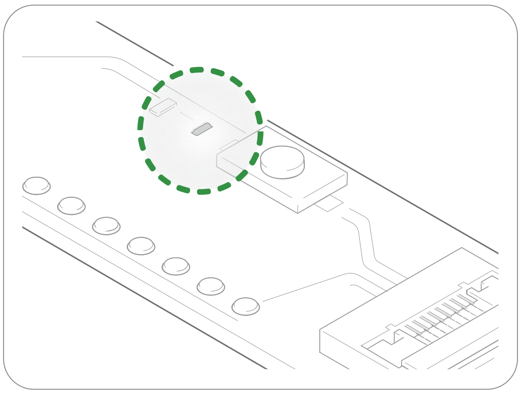
i. Alternatively, you can mount the sensor module by using tape in order to fasten the steel surface to the edge of a table, with the optical surface facing towards you.



5. Insert the USB cable into a computer.



6. The green LED on the interface board lights up when connected.



3.4.3 Download SDK

1. Follow the SDK guide for your OS.
   a. SDK Guide for Windows[7]
   b. SDK Guide for Linux[8]

For Further information, please refer to SDK Documentation[9].

[7] https://support.neonode.com/docs/display/SDKDOC/Getting+started+with+SDK+for+Windows
[8] https://support.neonode.com/docs/display/SDKDOC/Getting+started+with+SDK+for+Linux
[9] https://support.neonode.com/docs/display/SDKDOC

4 Getting Started with Software Integration

The Neonode Touch Sensor Module can be integrated into any host system that supports either the I2C or the USB HID transport protocol. The zForce communication protocol is based on I2C- or USB HID-transport of messages that are serialized according to the zForce ASN.1 Serialization Protocol. ASN.1 is a standardized way to describe data regardless of language implementation, hardware system and operating system (ISO/IEC 8824).

4.1 Use the zForce SDK

4.1.1 Communicating without Deserializing ASN.1-encoded Messages

The zForce SDK is compatible with USB and Windows or Linux and allows you to communicate with the sensor module without considering serialization or deserialization of ASN.1 encoded messages. It can also be used to create an application for communication with the sensor module. The SDK contains an example program to get you going. An explanation of the program in pseudocode is available here[10].

Download zForce SDK from Downloads[11] and please refer to the separate zForce SDK documentation[12].

4.2 Use the Touch Sensor Module Interface Library for Arduino

4.2.1 Communicating with I2C and Arduino

The Touch Sensor Module interface library is compatible with I2C and Arduino. It is a primitive function library and can be used to handle the communication with the sensor module. The library contains an example program to get you going.

For more information, refer to zForce Arduino Library (see page 100).

4.3 Communicating Using a System and a Programming Language of Your Choice

Learn more about the zForce Communication Protocol (see page 56) and write your own application to read and write data via one of the following transport modes:

- USB Raw HID Mode
- I2C Transport

Make sure to prepare the sensor module for communication, refer to Preparing the Sensor for Communication (see page 55).

Neonode provides the following help to get you started:

- A Message Generator, as part of the Neonode Workbench. This tool can be used to generate serialized messages according to the zForce ASN.1 Protocol. Refer to separate Neonode Workbench documentation[13] for further information.
- Examples of different implementations. Refer to Implementation Examples (see page 100).

[10] https://support.neonode.com/docs/display/SDKDOC/Example+Program+Pseudocode
[11] https://support.neonode.com/docs/display/Downloads/zForce+SDK+Downloads
[12] https://support.neonode.com/docs/pages/viewpage.action?pageId=21135404
[13] https://support.neonode.com/docs/display/Workbench/Getting+Started+with+Neonode+Workbench

- Support. For any questions, please refer to Neonode Help Center[14].

[14] https://helpcenter.neonode.com/hc/en-us/requests/new

5 Introduction to Contactless Touch Solutions

5.1 Introduction

Neonode Touch Sensor Modules are perfectly suited to create a contactless touch environment. Contactless integration allows interaction to an application or interface without having to touch any surface in order to provide a germ-free alternative. Made to prevent and reduce spread of bacteria and viruses.

When connecting the Touch Sensor Module through USB to a supported system, it will immediately be recognized as a (HID) touchscreen digitizer through Plug-and-Play[15]. The sensor module could then be positioned at a preferred distance parallel to the display. The gap would allow the end user to interact with an application by simply hovering their finger (or any other object) in front of the display in-air. Examples of this has been implemented in e.g. kiosks in order to provide a contactless alternative.

Another approach is to let the image display be projected through a special projection plate to create the illusion of a hologram. The image projection can then be controlled in-air by using the Touch Sensor Module. Examples of this have been implemented in e.g. elevators around the world.

We separate these two solutions as *Parallel Plane Solution* and *Holographic Display Solution*. The sensor module can also be connected and configured in our get-started tools, such as Workbench[16], SDK[17], or Arduino Library[18] (I2C) for easier implementation without having to deserialize the ASN.1 encoded message.

5.2 Parallel Plane Solution

The easiest and most cost-effective way to create a contactless touch solution is to position the Touch Sensor Module above the display, while leaving a gap in-between the sensor module and the chosen display or surface. By raising the Touch Sensor Module, it is possible to interact with an application of choice by simply hovering across the Touch Active Area (TAA).

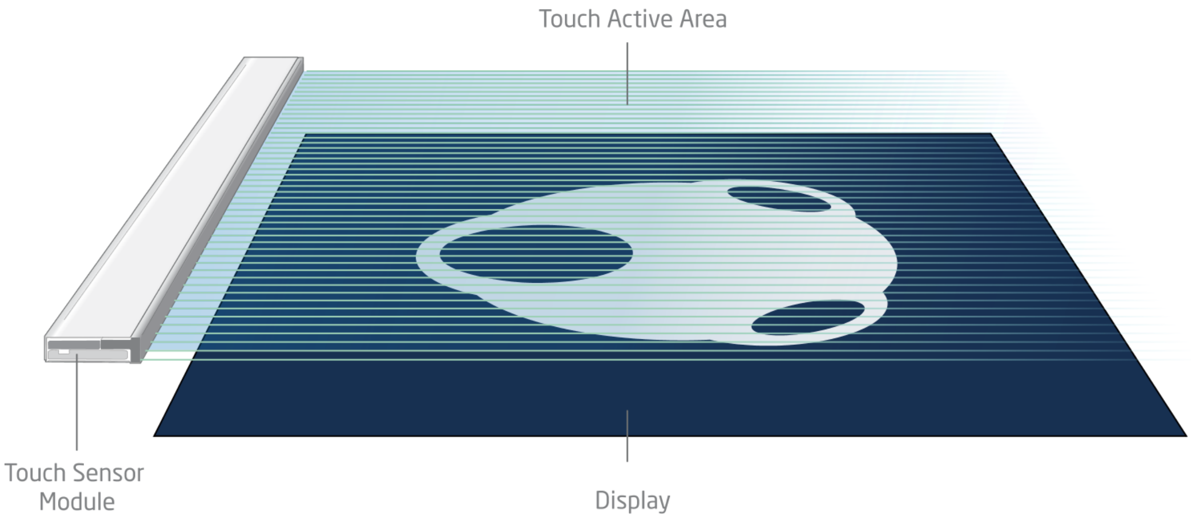
The Parallel Plane Solution is a perfectly suited contactless touch approach for elevators and a variety of kiosks appliances. That involves self-service kiosks, vending machines, Quick Service Restaurants, ATMs, or coffee Machines.

[15] https://support.neonode.com/docs/display/AIRTSUsersGuide/Getting+Started+with+Sensor+Evaluation+-+Plug+and+Play+with+USB

[16] https://support.neonode.com/docs/display/AIRTSUsersGuide/Getting+Started+with+Sensor+Evaluation+-+Workbench+and+USB

[17] https://support.neonode.com/docs/display/AIRTSUsersGuide/Getting+Started+with+Sensor+Evaluation+-+SDK+and+USB

[18] https://support.neonode.com/docs/display/AIRTSUsersGuide/Getting+Started+with+Sensor+Evaluation+-+I2C+and+Arduino



Since the Touch Sensor Module is recognized as a HID touchscreen digitizer by the host system, the Plug-and-Play[19] support allows an easy upgrade on existing systems. To upgrade an existing system, the sensor module will simply work upon plug-and-play if supported. The sensor module can also be implemented or configured using the evaluation tools (i.e. Workbench, example-SDK, Arduino Library), or a custom solution through USB HID or I2C.

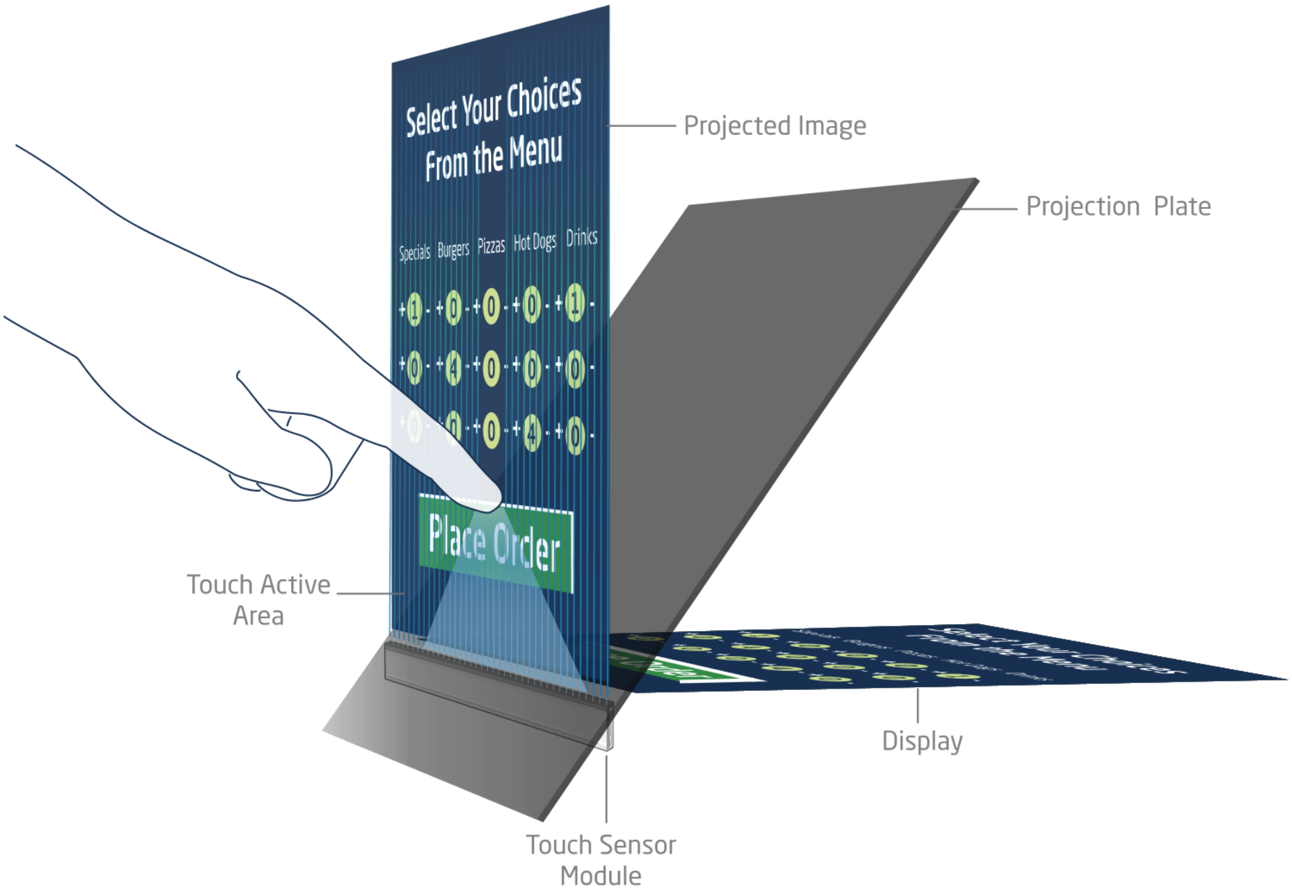
It is possible to apply a new set of configurations temporarily (RAM) with the use of our evaluation tools, or by simply sending a configuration message to the sensor module. These settings can change how the sensor module handles touch input data, which makes it possible to mount the sensor alongside any sides of the display regardless of its orientation.

For further information about software integration, please refer to section software integration.

5.3 Holographic Display Solution

In-air image projection enables the possibilities to interact with a floating interface. By projecting the image of the display through a projection plate, the image will appear to be floating in-air. The Touch Sensor Module can then be positioned to cover the projected image, to make it possible to interact with the projected interface by simply touching it.

[19] https://support.neonode.com/docs/display/AIRTSUsersGuide/Getting+Started+with+Sensor+Evaluation+-+Plug+and+Play+with+USB



The Holographic Display Solution enables the in-air image projection of an ordinary LCD to be projected 90 degrees away from the screen, in-air. Since there is a fixed angular relationship between the display, the projection sheet, and the final image, the installation geometry of the display and projection sheet determines the orientation of the final image.

The projection plate reduces the viewing angle of the projected image compared to the original display, which enables applications with privacy requirements.

5.3.1 Assembly

A projection plate consisting of at least two layers of mirror arrays positioned at 45° relative to the display redirects light emanating from the display an additional 45° and concentrates it to a position at equal distance from the plate, creating the floating image.

The Touch Sensor Module can then be positioned underneath the projection plate, where its interactive area (TAA) is covering the projected image. Please refer to Optical Requirements on External Window[20] for optical requirements.

5.4 Clean Environment

Contactless integration provides a germ- and smudge-free solution for challenging environments. The light reflective technology of the Touch Sensor Module allows the user to make interactions using any object, which is

[20] https://support.neonode.com/docs/display/AIRTSUsersGuide/Optical+Requirements+on+External+Window

beneficial for environments that require protective gear, such as gloves. Contactless integration is also an alternative for environments that cannot risk dirty equipment.

Unlike the standard touchscreen, a contactless holographic display requires less maintenance in regards of cleaning, as well as a lower risk of spreading bacteria, dirt, and viruses.

5.5 Software Implementation

The Touch Sensor Module can send touch notifications through USB HID pipe or I2C.

The parallel plane or holographic display solution can be implemented quickly with our Sensor Evaluation Tools[21]. To create a custom solution on a system of your choice, please refer to Get Started with Software Integration[22].

For further information, please refer to Neonode Support Team

5.5.1 Related Press Releases

- Contactless Touch Interaction with Images Floating In-Air[23]
- Neonode Enables Contactless Self Check-in Solution at Singapore Changi Airport[24]
- Neonode Touch Sensor Modules Selected for Touchless Holographic Elevator Rollout in China by Easpeed[25]
- N[26]eonode Touch Sensor Modules Selected for Touchless Holographic products by Yesar[27]

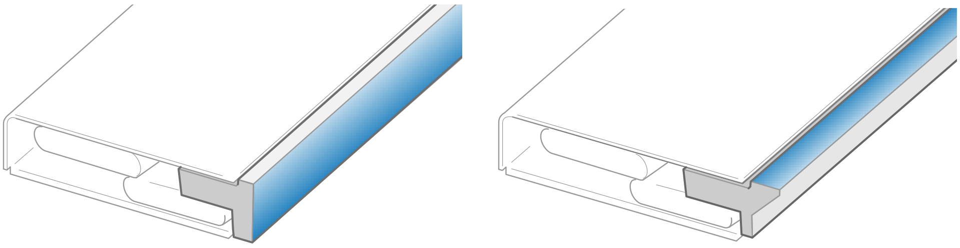
[21] https://support.neonode.com/docs/display/AIRTSUsersGuide/Getting+started+with+Touch+Sensor+Module+Evaluation
[22] https://support.neonode.com/docs/display/AIRTSUsersGuide/Getting+Started+with+Software+Integration
[23] https://neonode.com/contactless-touch-interaction-with-images-floating-in-air/
[24] https://neonode.com/neonode-enables-contactless-self-check-in-solution-at-singapore-changi-airport/
[25] https://neonode.com/neonode-touch-sensor-modules-selected-for-touchless-holographic-elevator-rollout-in-china-by-easpeed/
[26] https://neonode.com/neonode-touch-sensor-modules-selected-for-touchless-holographic-elevator-rollout-in-china-by-easpeed/
[27] https://neonode.com/neonode-touch-sensor-modules-selected-for-touchless-holographic-products-by-yesar/

6 Mechanical Integration

The Neonode Touch Sensor Module can be used for different purposes, such as detecting touches on a surface or objects in-air. Assembly requirements differ depending on what purpose the Touch Sensor Module fulfills. In addition, different industries have different standards and demands to fulfill. In-air detection applications generally require lower mounting tolerances.

6.1 Means of Integration

Neonode Touch Sensor Module comes in two types: one is designed to be integrated horizontally and the other vertically. This allows different types of assembly possibilities and better adaptation of the available space in the host system. The two sensor module designs are built on the same concept, but use two different front light pipes. One light path is unaffected; the other bent 90°.

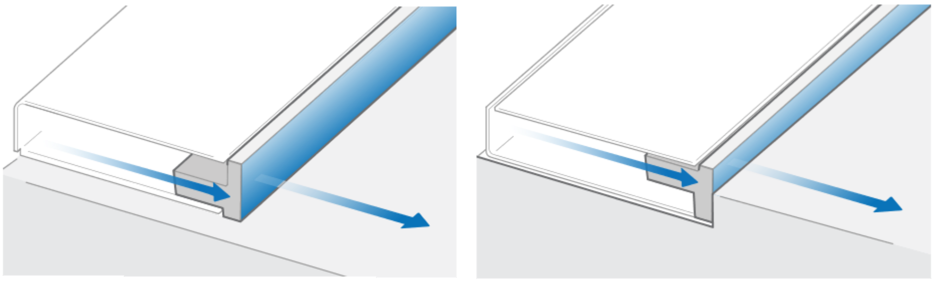


The front optical surface is not allowed to be blocked by the host system.

The short sides of the sensor module should be protected from light of high intensity. If there is a risk for exposure to strong light, covering the short sides with, for example, black tape might improve performance.

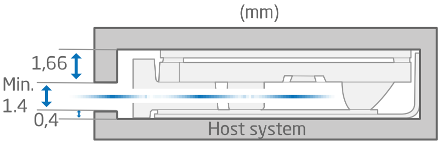
6.1.1 Horizontal Integration

Light is sent straight out and enables an active area in front of the module, in the same plane as the light path.



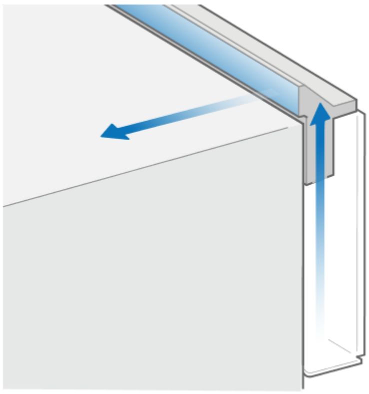
When integrating the Touch Sensor Module into a host system make sure not to interfere with the light path. For horizontal integration, the opening for the sensor modules light path must be minimum 1.4 mm.

If the host system have large tolerances, opening must be adjusted to always be minimum 1.4 mm.

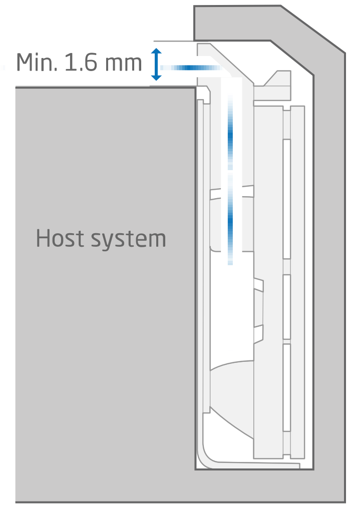


6.1.2 Vertical Integration

Light is bent 90 degrees within the Touch Sensor Module. This allows the sensor module to be assembled vertically but still have an active area in the horizontal plane.



To make sure not to interfere with the sensor modules light path, the opening must be minimum 1.6 mm. If the host system has large tolerances, the opening must always be adjusted to be minimum 1.6 mm. Also note that it is not allowed to mount, glue or in any other way affect the sensor module's optical surfaces since it will affect the performance. This applies to both the module's visible optical surface and the back of the mirror surface that bends the light path 90°.



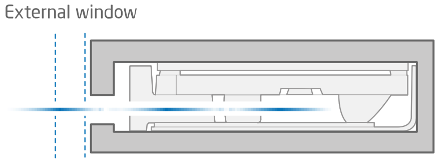
6.1.3 Options for Guiding and Fastening

- **Double adhesive tape** – for smaller sizes this can be used alone to hold the Touch Sensor Module. The host system geometry needs to provide a flat supporting surface.
- **Snaps** – Host system geometry provides some sort of snap features holding the Touch Sensor Module in place. These must be developed for each case to fit the host System cover and the surrounding.
- **Sandwiched** – the Touch Sensor Module is mounted by pressing the sensor modulebetween host system exterior cover and display. A structure (ribs, foam gasket or adhesive) is needed to make sure the Touch Sensor cannot move.

The Touch Sensor Module needs to be protected from outer pressure and forces that can bend the module and by that change the direction of the sensor light. The most common cause of bending is when a sensor module is mounted on a non-flat surface, so the host system supporting structure needs to be flat. High point load on the PCBA side of the sensor can also affect touch performance. If any kind of clamp or similar setup is used for fastening, make sure that the contact surface with the sensor module is as large as possible.

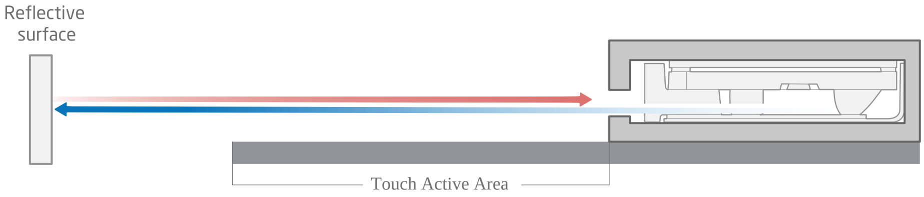
6.1.4 External Window

An external window is something placed between the sensor module and the desired touch active area, usually in form of a plastic or glass "window". It is of high importance for the function that these surfaces fulfill the optical demands stated in Optical Requirements on External Window (see page 143). It is important to know that each window the light passes through will reduce the sensor modules received signal levels, even though the requirements are fulfilled, which in some applications might reduce the maximum detection range.



6.1.5 External Reflective Surface

An external reflective surface is a surface located outside the active area, but close enough to be reached by the IR light emitted by the sensor module. Depending on the angle and the reflectance of the surface, reflected light can enter the sensor module and interfere with touch object detection. If the external reflective surface is close to the touch active area, it is recommended to make sure it has a low reflectance in the direction back towards the sensor module.



Lower reflectance can be achieved by

- Using a color on the reflective material which has low reflectance in the IR spectrum, e.g. black plastic or black paint.
- Using an angle of the reflective surface to divert the reflective light away from the sensor module, meaning that the surface can not be perpendicular to the module's optical axis. It need to differ with at least 3 degrees compared to the perpendicular angle.

6.2 Touch Applications

The sections below describe integration aspects specific for touch applications and do not concern in-air applications.

6.2.1 Touch Accuracy

Mechanical integration of Touch Sensor Module and assembly tolerances has a direct impact on touch accuracy. For this reason relaxed assembly tolerances might in some applications have an impact on the perceived touch performance. The best user experience is achieved when the projected Touch Active Area from the Touch Sensor Module perfectly overlaps the intended touch sensitive area on the host device, for example, the active area on a display.

The Touch Active Area of host system and the Touch Sensor Module needs to be well aligned. Translational tolerances in x and y directions and rotational tolerances will affect accuracy. See Translational Tolerances (see page 45) (x and y direction) and Rotational Tolerances (see page 46) (angle "b").

6.2.2 Hovering Touches

Hovering touch means that the Touch Sensor Module reports a touch event before the object reaches the surface. The basic principle of the Touch Sensor Module is that light is sent above the surface. To provide a good user experience the Touch Sensor Module software adjusts the signal and reports a touch first when the object reaches the surface.

Hovering touches is also direct linked to how the sensor module is integrated in the host system. It’s important that the mounting surface has the correct angle compared to the intended touch surface. Twisting and tilting of the Touch Sensor Module should always be avoided. Relaxed tolerances can lead to missed touches and increased hovering. See Translational Tolerances (see page 45) (z direction) and Rotational Tolerances (see page 46) (angle "a").

Furthermore, host system active area surface need to be flat or slightly concave. A convex surface can give false touches.

6.2.3 Assembly Tolerances

Translational Tolerances

Direction	Recommended Tolerances for Touch Applications
x-direction	±0.5 mm
y-direction	±0.5 mm
z-direction	0 mm to +0.5 mm

Translational tolerances affects the overlap between the display active area and the touch active area. For example, if the Touch Sensor Module is translated 0.5 mm in x-direction there will be a systematic touch offset of 0.5 mm for the complete sensor module in x-direction.

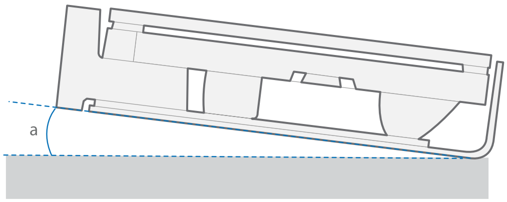
A 0 mm translation in z-direction means that the host system active area surface is positioned exactly at the edge of the light path. A positive translation means that the Touch Sensor Module, and therefore the light path, is translated up from the host system active area surface. This will not affect the touch accuracy in the sensor module, but it can affect the perceived touch performance, since it leads to increased hovering. A negative translation in z-direction should be avoided since parts of the light will be blocked which leads to no or reduced touch performance.

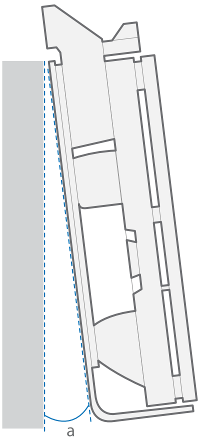
Rotational Tolerances

There are two types of rotations that can affect the performance; defined as the angles "a" and "b". Angle "a" affects the hovering and angle "b" affects the overlap between the intended active area and the sensor module's Touch Active Area. Both these issues will grow with larger display sizes. The angles are exaggerated in the pictures to better illustrate the problem. For any uncertainty regarding rotational tolerances for a specific application, contact Neonode for recommendations.

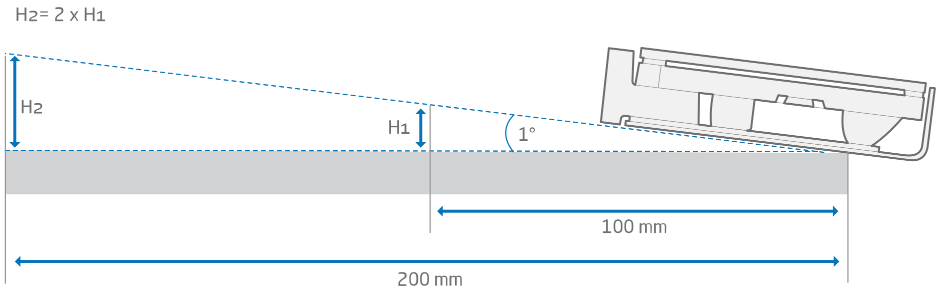
Angle "a"

The angle "a" is defined as shown in the images below.



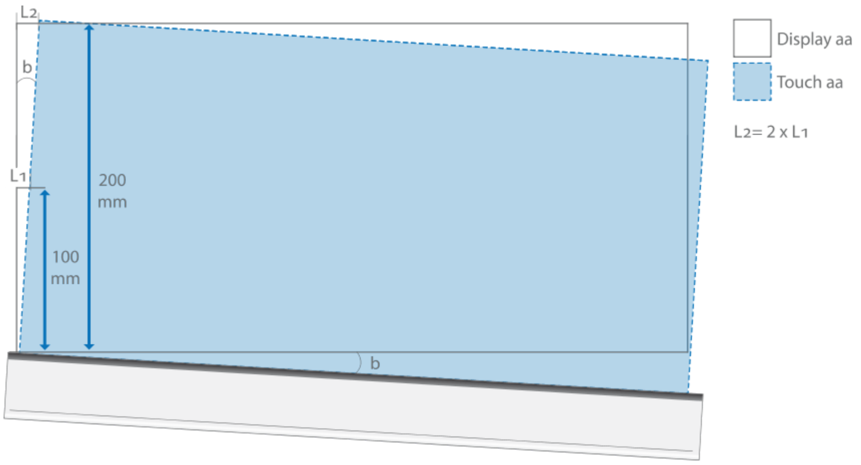


The example below illustrates how the problem increases with larger active areas.



Angle "b"

The angle "b" is defined as shown in the image below. How sensitive the Touch Sensor Module is for assembly rotations is directly linked to the size. At any given angle b, the Touch Active Area will be tilted twice as much at 200 mm compared to at 100 mm.



6.3 Models and Drawings

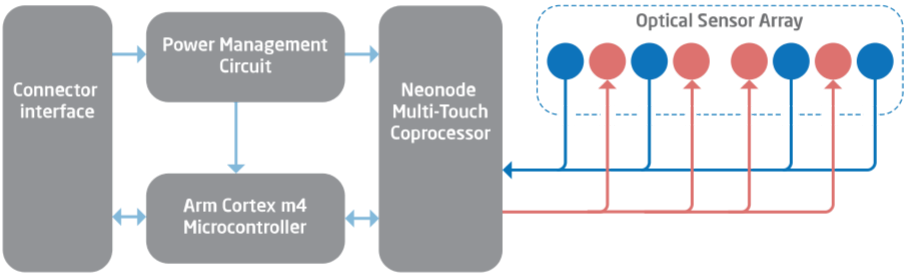
3D-models and 2D-drawings for Neonode Touch Sensor Module can be downloaded here[28].

[28] https://support.neonode.com/docs/display/Downloads

7 Electrical Integration

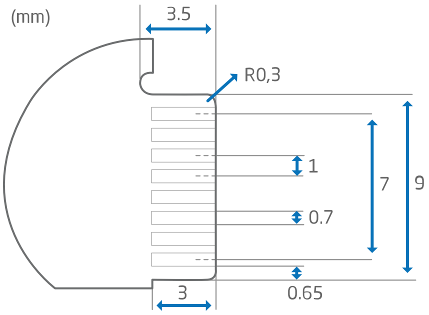
Electrostatic Sensitive Device!
To prevent equipment damage, use proper grounding techniques.

7.1 Electrical Block Diagram



7.2 Physical Connector

The Touch Sensor Module has 8 contact pads and a PCB outline that matches that of a standard 0.3-0.33 mm thick FFC/FPC with 1 mm pitch and top mounted connectors:



The contact pads are placed on the backside of the sensor module PCBA.

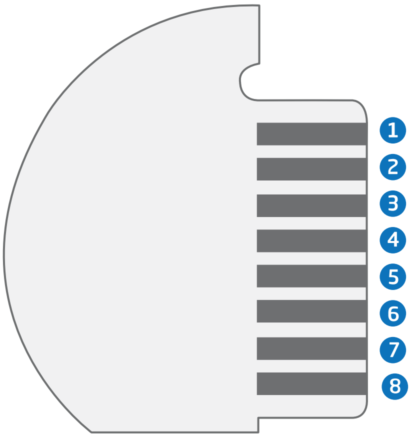
List of supported FFC connectors:

Supplier	Part number(s)
Molex[29]	527930870, 2005290080
Omron Electronic Components[30]	XF3M(1)-0815-1B
Wurth Electronics Inc[31].	686108148922

[29] http://www.molex.com/molex/home
[30] https://www.components.omron.com/
[31] http://www.digikey.com/en/supplier-centers/w/wurth-electronics

Almita Connectors[32]	BL124H-8R-TAND

7.3 Pin-Out



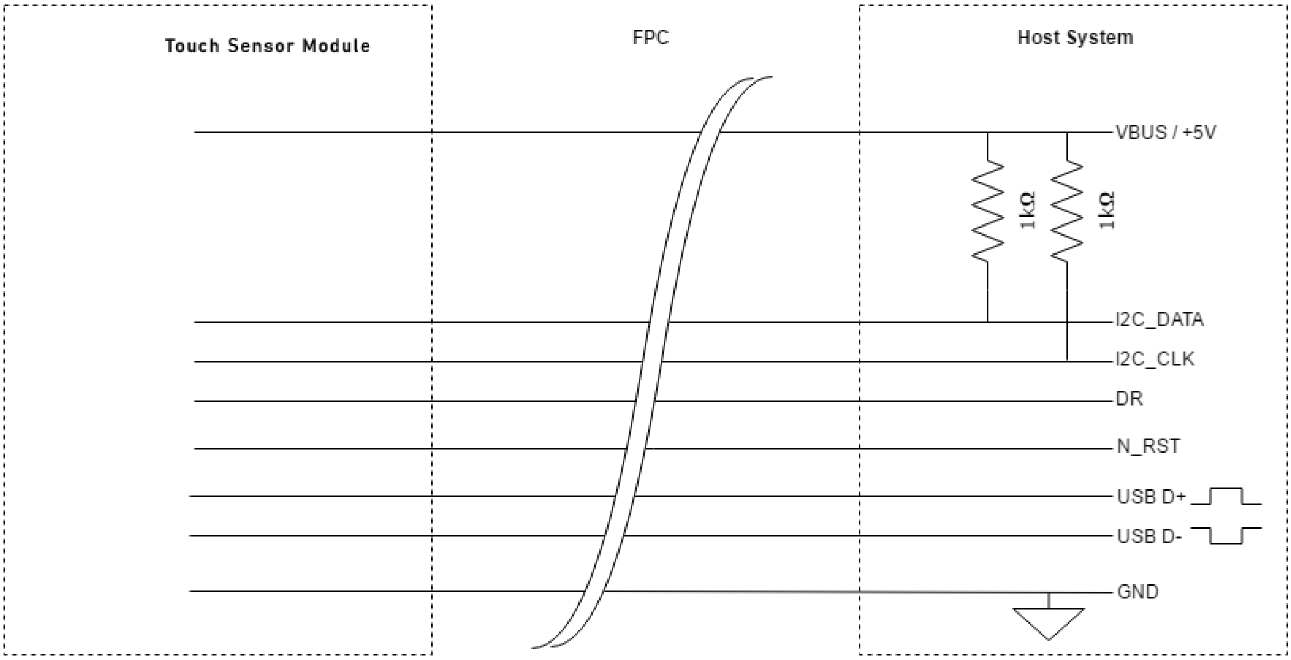
Function	Pin name	Pin No	Direction	Description	Comment
Power ground	GND	1	-	Ground	
Reset	N_RST	2	Input	Resets sensor module to initial state. Active low.	A minimum of a 300 ns low pulse is required
Data Ready	DR	3	Output	Indicates that sensor module has data to send	Push/pull output. Only used in I2C mode.
I2C Data	I2C_DATA	4	I/O	I2C data line	Requires external pull-up resistor
I2C Clock	I2C_CLK	5	Input	I2C clock line	Requires external pull-up resistor
USB DM	USB D-	6	I/O	USB DM line	USB 2.0 Compliant
USB DP	USB D+	7	I/O	USB DP line	USB 2.0 Compliant
Power supply	+5V	8	-	+5V power supply	USB 2.0 Compliant

Note: All pins use 3.3V voltage level and have 5V tolerance.

[32] http://www.almita-connectors.com/

7.4 Interface Configuration

The Touch Sensor Module provides two interfaces for communication with the host system, I2C and USB HID-device. The user can choose to connect one of them, or both. The typical Touch Sensor Module connection to a host system is shown in the following diagram:



7.4.1 USB Connection

The Touch Sensor Module provides a USB full-speed device interface through its 8-pin connector. In this connection, PIN 1, 6, 7, 8 (GND, USB D-, USB D+, VBUS) are used. After connecting the sensor module to the host system, it could be enumerated as a normal USB HID-device and act as a digitizer for a touch screen.

In this connection, only the default touch active area size could be used. Please refer to Product Variants (see page 0) for the actual values.

In this case, it is recommended to use two 1kΩ pull-up resistors to tie up I2C_DATA and I2C_CLK pins to VBUS or 3.3V power supply to avoid noise issue on I2C interface, and leave other pins as not connected. PIN 2 (N_RST) could be used to reset or enable/disable the sensor module.

USB Characteristics

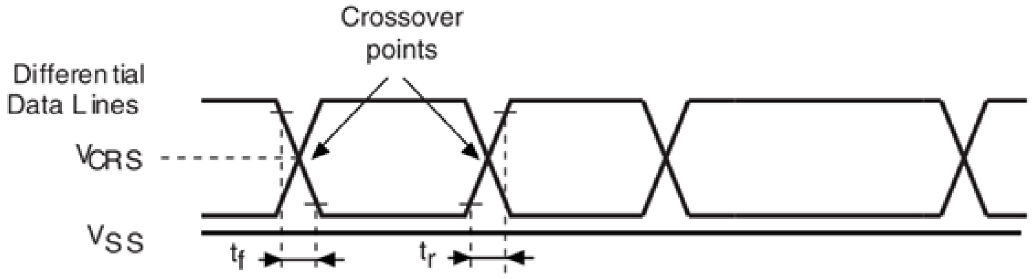
The USB interface meets the requirements of USB specifictions 2.0:

Symbol		Parameter	Conditions	Min	Typ	Max	Unit
Input levels	V_{DI}	Diff. Input sensitivity		0.2	-	-	V

Symbol		Parameter	Conditions	Min	Typ	Max	Unit
	V_{CM}	Diff. common mode range		0.8		2.5	
	V_{SE}	Single ended receiver threshold		1.3		2	
Output levels	V_{OL}	Output level low	1.5kΩ to V_{DD}	-	-	0.3	
	V_{OH}	Output level high	15kΩ to V_{SS}	2.8	-	3.6	
R_{PD}		D+/D-	V_{IN}=V_{DD}	17	21	24	kΩ
R_{PU}		D+/D-	V_{IN}=V_{SS}	1.5	1.8	2.1	

USB full speed timings: the definitions of data signal rise and fall time are presented in the following table and figure:

Driver characteristcs				
Symbol	**Parameter**	**Conditions**	**Min**	**Max**
t_r	Rise time	C_L = 50pF	4	20
t_f	Fall time	C_L = 50pF	4	20
t_{rfm}	Rise/fall time matching	t_r/t_f	90	110
V_{CRS}	Output signal crossover		1.3	2



7.4.2 I2C Connection

The Touch Sensor Module provides an I2C interface through its 8-pin connector. The interface runs in fast mode, which means the speed is 400kbps. With this interface, more advanced configuration could be performed. PIN 1, 3, 4, 5, 8 (GND, DR, I2C_DATA, I2C_CLK, VBUS) are used for this connection. It is recommended to use two 1kΩ pull-up resistors to tie up I2C_DATA and I2C_CLK pins to VBUS or 3.3V power supply to perform proper I2C communication. If USB connection is not used in parallel, PIN 6, 7 (USB D-, USB D+) could be left unconnected.

After the Touch Sensor Module is powered on, it will act as a slave device on a I2C bus. For further information about what commands could be send or receive through this interface, please refer to I2C Transport (see page 75).

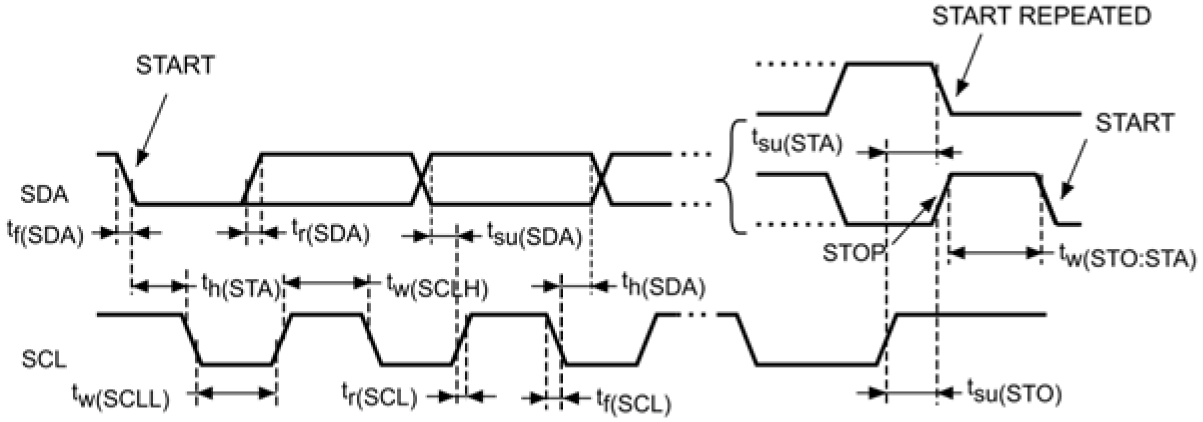
I2C Characteristics

The I2C interface meets the requirements of the standard I2C communication protocol with the following restrictions: SDA and SCL are mapped to I/O pins that are not “true” open-drain. When configured as open-drain, the PMOS connected between the I/O pin and VDD is disabled, but is still present.

The I2C characteristics are described in the following table:

Symbol	Parameter	Min	Max	Unit
$t_{w(SCLL)}$	SCL clock low time	4.7	-	us
$t_{w(SCLH)}$	SCL clock high time	4.0	-	
$t_{su(SDA)}$	SDA setup time	250	-	ns
$t_{h(SDA)}$	SDA data hold time	0	-	
$t_{r(SDA)}$	SDA and SCL rise time	-	1000	
$t_{r(SCL)}$				
$t_{f(SDA)}$	SDA and SCL fall time		300	
$t_{f(SCL)}$				
$t_{h(STA)}$	Start condition hold time	4.0	-	us
$t_{su(STA)}$	Repeated Start condition setup time	4.7	-	
$t_{su(STO)}$	Stop condition setup time	4.0	-	
$t_{w(STO:STA)}$	Stop to Start condition time (bus free	4.7	-	

Symbol	Parameter	Min	Max	Unit
t_{SP}	Pulse width of the spikes that are suppressed by the analog filter	0	50	ns
C_{load}	Capacitive load for each bus line	-	400	pF
V_{IL}	Input low level voltage		0.8	V
V_{IH}	Input high level voltage	2.3		V
V_{OL}	Output low level voltage		0.4	V
V_{OH}	Output high level voltage	2.9		V



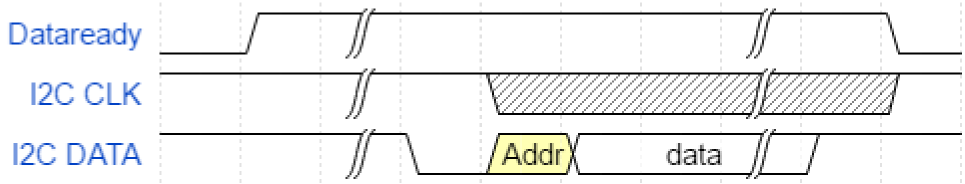
PIN 3 (DR) Characteristics

PIN 3 is used as signal output for DataReady (DR). The DataReady signal is only used in I2C communication.

The sensor module can only act as an I2C slave, therefore this pin is needed to notify the host to read the data in the output buffer of the sensor module:

- PIN 3 is set to high (1) when there is data in the buffer to be sent from the sensor module.
- PIN 3 is reset to low (0) when there is no data in the buffer to be sent from the sensor module.

PIN 3 can be used as an interrupt input or it can be read repeatedly by the host. When the pin is set to high, the FW will wait for the host to read data from the I2C bus. When the read transaction is finished, this Data Ready signal will be reset automatically by the sensor module. The following figure shows the timing behavior of a typical I2C transaction:



I2C Reading Sequence

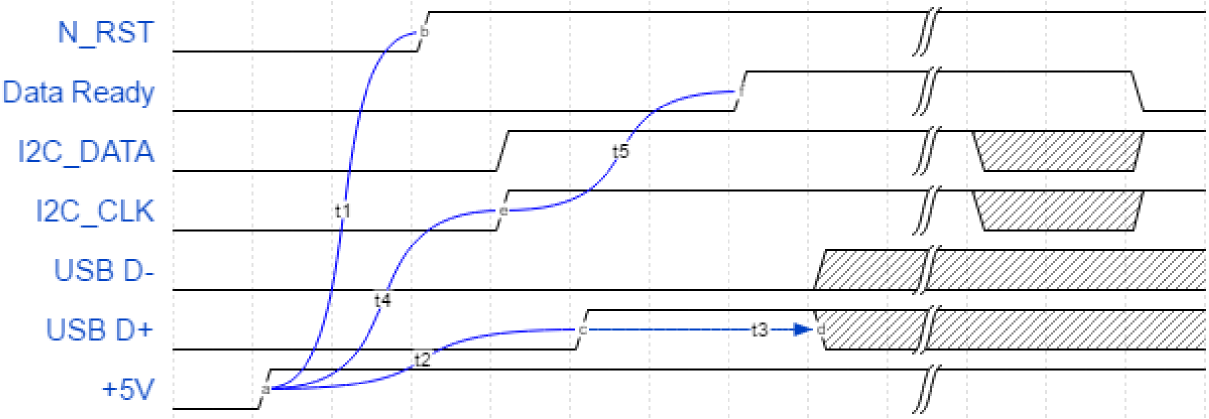
See I2C Transport (see page 75).

7.5 Power On and Boot Sequence

Power on timing latency are listed in the following table:

Name	Min	Typ.	Max	Unit	Comment
t1	-	15	20	ms	Delay time from power on to NRST to high voltage.
t2	-	60	70	ms	Delay time from power on to USB pins voltage ready.
t3	-	170	-	ms	Delay time from USB init ready to sending/receiving data.
t4	5	7	10	ms	Delay time from power on to I2C pins voltage ready.
t5	65	70	80	ms	Delay time from I2C pins voltage ready to triggering boot complete packet request.

The power on sequence is shown in the following figure.



8 Software Integration

8.1 Communication Protocol

Neonode Touch Sensor Module can communicate with a host system through USB HID transport or I2C transport. The structure of the communicated data is defined in ASN.1 notation and encoded using a defined set of encoding rules. For more information, refer to zForce Communication Protocol (see page 56).

8.1.1 Preparing the Sensor Module for Communication

When using the USB Raw HID interface or the I2C interface, the sensor module must be prepared for communication before it can receive messages. Refer to Preparing the Sensor for Communication (see page 55).

8.2 Available Function Libraries

Neonode has developed the following function libraries to facilitate integration of the sensor module:

- **The zForce Software Development Kit (SDK)** is a complete function library for communication via the USB HID interface. The SDK allows the user to communicate with the sensor module via USB without deciphering the serialized ASN.1-messages. Read more under the separate SDK documentation.[33]
- **The zForce interface library for Arduino** is a function library for communication via the I2C interface. This library is primitive, but the I2C read and write functions are public, so the user can receive sensor messages and send any information, that is correctly encoded, to the sensor module. Refer to zForce library interface for Arduino (see page 100).

8.3 Preparing the Sensor for Communication

The following procedures are required to prepare a sensor module for communication with I2C or USB Raw HID mode.

8.3.1 USB Raw HID Mode

The Touch Sensor Module has a setting called *operation mode*, that determines which data will be sent to the host. When communicating over USB there are two different operation modes that deliver touch data, called *detection mode* and *detection mode HID.* By default, the sensor module is set in the operation mode detection mode HID. This mode delivers touch data to the operating system according to the HID standard for touch screen devices. In order for the host to receive touch data that is serialized according to Neonode's ASN.1 serialization protocol, the operation mode has to be set to detection mode. In other words, setting the operation mode to detection mode allows the host to receive touch data over USB Raw HID mode.

Do the following procedure to prepare the sensor module over USB Raw HID mode.

1. Power on the Touch Sensor Module.
2. Wait for the OS to enumerate the device and then enumerate the sensor module in your application. Depending on which operating system that is used, the application might need permission from the operating system to access the device.
3. Set the sensor module to the correct operation mode (detection mode) by sending a Feature Report with report id 1.

[33] https://support.neonode.com/docs/pages/viewpage.action?pageId=21135404

4. Wait for the sensor module to signal that there is data to read. This comes as an Input Report 2 or you can poll Feature Report with id 2 for new data.
5. Read the response data from Feature Report with id 2. The data that is read reflects the current setting in the sensor module.

The Touch Sensor Module is now ready to communicate. After the procedure, the sensor module is enabled by default and will start sending ASN.1 serialized touch notifications. To disable the touch notifications, a Disable request must be sent. Refer to zForce Message Specification (see page 77) for examples of requests, responses and notifications.

For further details on how to communicate with the Touch Sensor Module over Feature Report 1 and 2 refer to USB HID Transport (see page 60).

8.3.2 I2C

Use the following procedure to prepare the Touch Sensor MOdule over I2C.

1. Power on the Touch Sensor Module.
2. Wait for sensor module to assert Data Ready pin (DR).
3. Initiate 2 byte I2C read operation. Payload of this read should be EE XX where XX is the amount of bytes to read in a second I2C read operation.
4. Read XX amount of bytes (number of bytes to read is indicated by second byte of first I2C Read Operation). Now read a message called BootComplete. The message should be

   ```
   F0 11 40 02 00 00 63 0B 80 01 YY 81 02 03 YY 82 02 00 YY
   ```

   where YY is usually "00" but can have another value. This signals that the sensor module is now booted.
5. To enable the Touch Sensor Module to start sending touch notifications, do the following:
   a. Send an Enable command:

      ```
      EE 09 40 02 02 00 65 03 81 01 00
      ```

   b. Read the response. The response should be:

      ```
      EF 09 40 02 02 00 65 03 81 01 00
      ```

The Touch Sensor Module is now ready to communicate. When DR is asserted the sensor module will send a touch notification or a new BootComplete. A BootComplete indicates that the sensor module has restarted for some reason; Enable must then be set again. For more details, refer to I2C Transport (see page 75).

8.4 zForce Communication Protocol

To communicate with Neonode Touch Sensor Module, you need two things: to send/receive messages and to encode/decode the messages.

The sending and receiving can be done in one of the following ways:

- **USB HID Touch Digitizer**: the sensor module can be used as a standard HID Touch Digitizer to report touch data to the OS.
- **USB RAW HID**: The sensor module uses HID Get and Set Feature Reports as a pipe to read and write. For more information, refer to USB HID Transport (see page 60)

- **I2C transport**: The sensor module has support for I2C communication with an extra pin for signaling when data is ready to be read, which allows the host system to be interrupt driven. The sensor module takes the role of an I2C slave and has the I2C address 0x50. For more information, refer to I2C Transport (see page 75).

For the encoding/decoding, you need to understand the structure of the messages and the protocol that is used to serialize them:

The structure of the sensor messages is defined in ASN.1 notation. ASN.1 is a standardized way (ISO/IEC 8824) to describe data regardless of language implementation, hardware system and operation system. For more information, refer to zForce Message Specification (see page 77).

The zForce communication protocol uses the Distinguished Encoding Rules (DER) to serialize messages. For more information, refer to Understanding the zForce ASN1 Protocol (see page 89).

8.4.1 Serialization Protocol Quick Start

The Touch Sensor Module communicates with messages that are serialized according to the zForce serialization protocol. Here is a quick introduction to the messages you need to send to get started and also how you interpret received messages.

Encoding Integers

ASN.1 encoded integers, for example values representing scanning frequency or touch active area size, are represented by one or more bytes:

- If the integer is between 0 and 127, it is represented by one byte (`00` to `7F`).
- If the integer is between 128 and 32767, it is represented by two bytes (`00 80` to `7F FF`).

The length of the message therefore varies depending on parameter values.

Enabling Touch Sensor Modules

Do the following to enable a Touch Sensor Module, that is, tell the module to start sending touch notifications.

1. Enable the sensor module by sending a request with an Enable command:

   ```
   EE 09 40 02 02 00 65 03 81 01 00
   ```

   This enables the sensor module to send touch notifications.
2. Read the response. The response should be:

   ```
   EF 09 40 02 02 00 65 03 81 01 00
   ```

3. The Touch Sensor Module is now Enabled. After this, wait for the sensor module to indicate it has something to send, which means that the device will send a Touch Notification or a BootComplete. A BootComplete indicates that the device has restarted for some reason, so rerun the initialization and enable the sensor module to start receiving touch notifications again.

Disabling Touch Sensor Modules

Do the following to disable a Touch Sensor Module, that is, tell the module to stop sending touch notifications.

1. Disable the sensor module by sending a request with the following command:

   ```
   EE 08 40 02 02 00 65 02 80 00
   ```

   This disables the sensor module to send touch notifications.
2. Read the response. The response should be:

   ```
   EF 08 40 02 02 00 65 02 80 00
   ```
3. The Touch Sensor Module is now Disabled.

Device Configuration

Device configuration is a command that includes different settings for the sensor module, for example the Touch Active Area. When sending a device configuration message, all of the settings specified are not required to be sent, however the response from the sensor module will include the full device configuration message.

This is an example message that changes the touch active area using the Device Configuration command:

```
EE 1A 40 02 02 00 73 14 A2 12 80 02 00 B5 81 01 43 82 02 06 98 83 02 04 34 85 01 FF
```

The message is explained in the table below:

Part	Description
`EE 1A`	ID for request followed by length of total payload (0x1A = 26 bytes)
`40 02 02 00`	Device address (always the same for Neonode Touch Sensor Modules)
`73 14`	ID for Device Configuration followed by the length of the total Device Configuration payload (20 bytes)
`A2 12`	ID for Sub Touch Active Area followed by the length of Sub Touch Active Area payload
`80 02 00 B5`	ID for xMin followed by payload length and an integer value (0x00B5 = 181)
`81 01 43`	ID for yMin followed by payload length and an integer value (0x43 = 67)
`82 02 06 98`	ID for xMax followed by payload length and an integer value (0x0698 = 1688)
`83 02 04 34`	ID for yMax followed by payload length and an integer value (0x0434 = 1076)
`85 01 FF`	ID for "Invert y axis" followed by length of payload and a Boolean (0xFF= True)

The response from the sensor module to the above message will contain the full device configuration message, also the parts not set in the request.

Setting the Touch Active Area should be done before enabling the sensor module with the ENABLE request.

Reflective Edge Filter

In the Device Configuration command there is something called Reflective Edge filter (from firmware version 1.45 and later). This setting is useful if there is a highly reflective material, right outside of the Touch Active Area. If such conditions are present, enabling this feature could increase the touch performance.

Command to Enable Reflective Edge Filter

```
EE 09 40 02 02 00 73 03 85 01 80
```

Command to Disable Reflective Edge Filter

```
EE 09 40 02 02 00 73 03 85 01 00
```

Setting Frequency

To set the finger frequency to 200 Hz and idle frequency to 63 Hz use a request with the following command:

```
EE 0D 40 02 00 00 68 07 80 02 00 C8 82 01 3F
```

ⓘ Neonode Touch Sensor Module does not support Stylus mode, and setting the stylus frequency does not do anything.

Decoding Touch Notifications

A packet can contain from one up to 10 touches, and optionally a timestamp. On packets where the timestamp is not included, the 58 02 TT TT bytes are missing from the end and the length bytes are adjusted accordingly, For One touch (below), F0 15 in the beginning will be F0 11 and A0 0F in the middle will be A0 0B. The same bytes are decreased by 4 for Two and Three touches.

One Touch

A packet that contains one touch will look like:

```
F0 15 40 02 02 00 A0 0F 42 09 II VV XX XX YY YY GG GG JJ 58 02 TT TT
```

where the data is defined as follows:

Syntax	Meaning
II	ID of this specific touch object. More than one can be tracked simultaneously.
VV	Event: 00 = DOWN (new object). 01 = MOVE (existing object has moved). 02 = UP (existing object is no longer being tracked).
XX XX	X Coordinate of the object. This is in Network Byte Order / Big Endian / Motorola Endian.
YY YY	Y Coordinate of the object. See above.
GG GG	Size of object on the X Axis.
JJ	This value should be ignored.
TT TT	Timestamp of the touch.

Two Touches

A packet that contains two Touches will look like:

```
F0 20 40 02 02 00 A0 1A 42 09 II VV XX XX YY YY GG GG JJ 42 09 II VV XX XX YY YY GG
GG JJ 58 02 TT TT
```

where the first "II" and the following bytes up to and including "JJ" are from the first touch, and the second "II" and the following bytes up to and including "JJ" are from the second touch.

Three Touches

A packet that contains three Touches will look like:

```
F0 2B 40 02 02 00 A0 25 42 09 II VV XX XX YY YY GG GG JJ 42 09 II VV XX XX YY YY GG
GG JJ 42 09 II VV XX XX YY YY GG GG JJ 58 02 TT TT
```

For a more thorough explanation of the serialization protocol, refer to Understanding the zForce ASN.1 Serialization Protocol (see page 89).

8.4.2 USB HID Transport

When connected via USB, the Touch Sensor Module communicates in Full Speed (12 Mbit/s) in two modes: Raw HID mode (also called HID Pipe) and HID Touch Digitizer mode. HID Touch Digitizer mode is initiated automatically as soon as the sensor module is plugged in. In order to use Raw HID mode, the module's operation mode must be changed. For more information, refer to Initializing Sensor Modules (see page 55).

HID Touch Digitizer Mode

The Touch Sensor Module acts as a HID Input device and communicates directly with the OS, and is completely plug and play.

> **Ubuntu 17.10 - 18.04**
> The sensor module is not recognized as a touch screen in the Ubuntu versions 17.10 - 18.04. This has been fixed and works fine in Ubuntu 18.10.

Raw HID Mode / HID Pipe

This mode uses two HID Feature Reports to communicate with the host.

- Send data to the sensor module by writing to Feature Report 1.
- Read data from the sensor module by reading from Feature Report 2.

Refer to zForce Message Specification (see page 77) for examples of requests, responses and notifications.

USB Communication in Different Operating Systems

Depending on the system and programming language you are using to write and read from a feature report, the implementation differs. For example, in Windows, this is abstracted and the hid.dll offers a function to get and set feature reports, while in for example a UNIX based OS, you might have to implement your own function to get and set a feature report.

The **communication** heading below describes how to implement your own get and set feature report functions, using a control transfer.

> **USB Permission**
> Depending on operating system you might need explicit permission for your program to have access to the HID device.

How to Implement Custom Functions for Raw HID Communication

In order to communicate with the sensor module the data flow type called *control transfer* should be used. The control transfer usually takes the following parameters:

- Request Type (*int)*
- Request (*int)*
- Value (*int)*
- Index (*int)*
- Data (*byte array)*
- Length (*int)*
- Timeout (*int)*

Writing to Feature Report 1

When writing to the sensor module, a full 257 bytes need to be sent no matter how long the actual message is. Take a look at the code snippet below, to see how this could be done.

```
uint8_t operationMode[] = { 0x01, 0x17, 0xEE, 0x15, 0x40, 0x02, 0x02, 0x00,
                                                        0x67, 0x0F, 0x80, 0x01, 0xFF,
0x81, 0x01, 0x00,
                                                        0x82, 0x01, 0x00, 0x83, 0x01,
0x00, 0x84, 0x01, 0x00 }; // The first two bytes are the header. First byte is
feature report, second byte is length of the following data.
uint8_t data[257];
memcpy(data, operationMode, sizeof(operationMode));

int requestType = 0x00 | (0x01 << 5) | 0x01; // USB_HOST_TO_DEVICE | USB_TYPE_CLASS |
USB_RECIPIENT_INTERFACE
int request = 0x09; // SET_CONFIGURATION = 0x09
int value = 0x0301; // 0x03 for feature report, 0x01 for feature report 1
int index = 0x0000;
int length = sizeof(data);
int timeout = 0;

 connection.controlTransfer(
            requestType,
            request,
            value,
            index, data, length, timeout);
```

Reading from Feature Report 2

When reading from feature report 2, the message is always 258 bytes long.

```
uint8_t data[258];

int requestType = 0x80 | (0x01 << 5) | 0x01; // USB_DEVICE_TO_HOST | USB_TYPE_CLASS |
USB_RECIPIENT_INTERFACE
int request = 0x01; // CLEAR_FEATURE = 0x01
int value = 0x0302; // 0x03 for feature report, 0x02 for feature report 2
int index = 0x0000;
int length = sizeof(data);
int timeout = 0;

connection.controlTransfer(
            requestType,
            request,
            value,
            index, data, length, timeout);
```

HID Report Descriptor (click to expand)

HID Report Descriptor

> ⓘ The HID Report Descriptor is subject to change. The descriptor below is from firmware version 1.47.

Item	Data
Usage Page (Digitizer)	05 0D
Usage (Touch Screen)	09 04
Collection (Application)	A1 01
Report ID (4)	85 04
Usage (Contact Count Maximum)	09 55
Logical Minimum (0)	15 00
Logical Maximum (-1)	25 FF
Report Size (8)	75 08
Report Count (1)	95 01
Feature (Data,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	B1 02
Report ID (3)	85 03
Usage (Contact Count)	09 54
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Scan Time)	09 56
Logical Maximum (65535)	27 FF FF 00 00
Report Size (16)	75 10
Unit Exponent (-4)	55 0C
Unit (SI Lin: Time (s))	66 01 10
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Finger)	09 22
Collection (Logical)	A1 02
Usage (Tip Switch)	09 42
Logical Maximum (1)	25 01
Report Size (1)	75 01

Item	Data
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Contact Identifier)	09 51
Logical Maximum (127)	25 7F
Report Size (7)	75 07
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Generic Desktop)	05 01
Usage (X)	09 30
Logical Maximum (65535)	27 FF FF 00 00
Report Size (16)	75 10
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Y)	09 31
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Digitizer)	05 0D
Unit Exponent (-2)	55 0E
Unit (SI Lin: Length (cm))	65 11
Usage (Width)	09 48
Usage (Height)	09 49
Report Count (2)	95 02
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
End Collection	C0
Usage (Finger)	09 22

Item	Data
Collection (Logical)	A1 02
Usage (Tip Switch)	09 42
Logical Maximum (1)	25 01
Report Size (1)	75 01
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Contact Identifier)	09 51
Logical Maximum (127)	25 7F
Report Size (7)	75 07
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Generic Desktop)	05 01
Usage (X)	09 30
Logical Maximum (65535)	27 FF FF 00 00
Report Size (16)	75 10
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Y)	09 31
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Digitizer)	05 0D
Unit Exponent (-2)	55 0E
Unit (SI Lin: Length (cm))	65 11
Usage (Width)	09 48
Usage (Height)	09 49

Item	Data
Report Count (2)	95 02
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
End Collection	C0
Usage (Finger)	09 22
Collection (Logical)	A1 02
Usage (Tip Switch)	09 42
Logical Maximum (1)	25 01
Report Size (1)	75 01
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Contact Identifier)	09 51
Logical Maximum (127)	25 7F
Report Size (7)	75 07
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Generic Desktop)	05 01
Usage (X)	09 30
Logical Maximum (65535)	27 FF FF 00 00
Report Size (16)	75 10
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Y)	09 31
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Digitizer)	05 0D

Item	Data
Unit Exponent (-2)	55 0E
Unit (SI Lin: Length (cm))	65 11
Usage (Width)	09 48
Usage (Height)	09 49
Report Count (2)	95 02
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
End Collection	C0
Usage (Finger)	09 22
Collection (Logical)	A1 02
Usage (Tip Switch)	09 42
Logical Maximum (1)	25 01
Report Size (1)	75 01
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Contact Identifier)	09 51
Logical Maximum (127)	25 7F
Report Size (7)	75 07
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Generic Desktop)	05 01
Usage (X)	09 30
Logical Maximum (65535)	27 FF FF 00 00
Report Size (16)	75 10
Report Count (1)	95 01

Item	Data
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Y)	09 31
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Digitizer)	05 0D
Unit Exponent (-2)	55 0E
Unit (SI Lin: Length (cm))	65 11
Usage (Width)	09 48
Usage (Height)	09 49
Report Count (2)	95 02
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
End Collection	C0
Usage (Finger)	09 22
Collection (Logical)	A1 02
Usage (Tip Switch)	09 42
Logical Maximum (1)	25 01
Report Size (1)	75 01
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Contact Identifier)	09 51
Logical Maximum (127)	25 7F
Report Size (7)	75 07
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Generic Desktop)	05 01

Item	Data
Usage (X)	09 30
Logical Maximum (65535)	27 FF FF 00 00
Report Size (16)	75 10
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Y)	09 31
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Digitizer)	05 0D
Unit Exponent (-2)	55 0E
Unit (SI Lin: Length (cm))	65 11
Usage (Width)	09 48
Usage (Height)	09 49
Report Count (2)	95 02
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
End Collection	C0
Usage (Finger)	09 22
Collection (Logical)	A1 02
Usage (Tip Switch)	09 42
Logical Maximum (1)	25 01
Report Size (1)	75 01
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Contact Identifier)	09 51
Logical Maximum (127)	25 7F

Item	Data
Report Size (7)	75 07
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Generic Desktop)	05 01
Usage (X)	09 30
Logical Maximum (65535)	27 FF FF 00 00
Report Size (16)	75 10
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage (Y)	09 31
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Usage Page (Digitizer)	05 0D
Unit Exponent (-2)	55 0E
Unit (SI Lin: Length (cm))	65 11
Usage (Width)	09 48
Usage (Height)	09 49
Report Count (2)	95 02
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
End Collection	C0
End Collection	C0
Usage Page (Vendor-Defined 1)	06 00 FF
Usage (Vendor-Defined 1)	09 01
Collection (Application)	A1 01
Report ID (1)	85 01

Item	Data
Usage (Vendor-Defined 1)	09 01
Report Size (8)	75 08
Report Count (1)	95 01
Logical Minimum (0)	15 00
Logical Maximum (-1)	25 FF
Feature (Data,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	B1 02
Usage (Vendor-Defined 2)	09 02
Report Count (255)	95 FF
Feature (Data,Var,Abs,NWrp,Lin,Pref,NNul,Vol,Buf)	B2 82 01
Report ID (2)	85 02
Usage (Vendor-Defined 1)	09 01
Report Count (1)	95 01
Feature (Data,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	B1 02
Usage (Vendor-Defined 2)	09 02
Report Count (255)	95 FF
Feature (Data,Var,Abs,NWrp,Lin,Pref,NNul,Vol,Buf)	B2 82 01
Usage (Vendor-Defined 3)	09 03
Report Size (1)	75 01
Logical Maximum (1)	25 01
Report Count (1)	95 01
Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)	81 02
Report Size (7)	75 07
Input (Cnst,Ary,Abs)	81 01
Report Size (8)	75 08

Item	Data
Report ID (128)	85 80
Usage (Vendor-Defined 1)	09 01
Feature (Cnst,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	B1 03
Report ID (130)	85 82
Usage (Vendor-Defined 1)	09 01
Feature (Cnst,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)	B1 03
End Collection	C0

Parsed reports by Report ID

Input Report 2		
Bit offset	**Bit count**	**Description**
0	1	Internal use
1	7	(Not used)
Input Report 3		
Bit offset	**Bit count**	**Description**
0	8	Contact count
8	16	Scan Time
24	1	Tip Switch
25	7	Contact identifier
32	16	X
48	16	Y
64	16	Width
80	16	Height
96	1	Tip Switch
97	7	Contact identifier

104	16	X
120	16	Y
136	16	Width
152	16	Height
168	1	Tip Switch
169	7	Contact identifier
176	16	X
192	16	Y
208	16	Width
224	16	Height
240	1	Tip Switch
241	7	Contact identifier
248	16	X
264	16	Y
280	16	Width
296	16	Height
312	1	Tip Switch
313	7	Contact identifier
320	16	X
336	16	Y
352	16	Width
368	16	Height
384	1	Tip Switch
385	7	Contact identifier
392	16	X

408	16	Y
424	16	Width
440	16	Height
Feature Report 1 - Write		
Bit offset	**Bit count**	**Description**
0	8	Payload size (bytes)
8	2040	Payload
Feature Report 2 - Read		
Bit offset	**Bit count**	**Description**
0	8	Payload size (bytes)
8	2040	Payload
Feature Report 4		
Bit offset	**Bit count**	**Description**
0	8	Contact count maximum
Feature Report 128		
Bit offset	**Bit count**	**Description**
0	8	Internal use
Feature Report 130		
Bit offset	**Bit count**	**Description**
0	8	Internal use

8.4.3 I2C Transport

Introduction

Each I2C bus consists of two lines (signals): a clock line (SCL) and a data line (SDA). The devices on the I2C bus are either masters or slaves. The master is always the device that drives the clock line. The slaves are the devices that respond to the master. Only the master can initiate a transfer over the I2C bus. Each slave has a unique address.

> All bytes in this section are written in hexadecimal form, if not indicated otherwise.

> The Touch Sensor Module does not communicate using registers or memory accesses, so the documentation for the STM32 microcontrollers is not applicable.

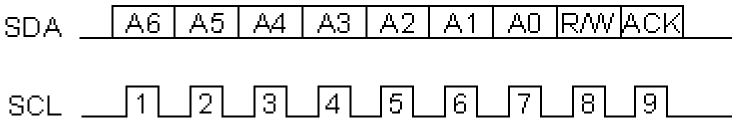
The DataReady signal

The Touch Sensor Module uses a signal called DataReady (DR) to inform the host that there is data to read and that a read operation can be initialized. Refer to Electrical Integration (see page 53) for more information.

Module Address

The slave I2C address of the sensor module is 0x50 (7 bit). The address itself consists of 7 bits but 8 bits are always sent. The extra bit informs the slave if the master is reading from or writing to it. If the bit is 0 the master is writing to the slave. If the bit is 1 the master is reading from the slave.

The resulting address bytes are 0xA1 (read) and 0xA0 (write).



Syntax

The I2C Transport Protocol is simple and the syntax is identical in both directions (read or write):

Byte number	**1**	**2**	**3 to n**
Description	FrameStart, a constant. The value is always EE.	DataSize, the number of bytes in the payload. Datasize >= 1.	Payload (serialized ASN.1 message).

Examples:

1. The payload is 8 bytes long. The transmission reads:

```
EE 08 [payload]
```

2. The payload is 25 bytes long. The transmission reads:

```
EE 19 [payload]
```

Sending Data

To send data to the Touch Sensor Module, the host initiates a write operation for the sensor module's address and writes the full payload.

(Do not confuse the I2C FrameStart constant with the type byte that indicates that a serialized message is a request. The value is EE for both bytes, but the meaning is completely different.)

Example: Sending a request with an ENABLE command to the sensor module:

```
EE 0B EE 09 40 02 02 00 65 03 81 01 00
```

> ⚠ **DataReady**
> If the sensor module signals DataReady, it is NOT permitted to initiate an I2C write operation as the sensor module is waiting for the host to initiate a read operation.
> Always wait for the corresponding response from one command before sending another command. Note that not all commands have a response command. If there is no corresponding response command, the host is free to issue another command.

Receiving Data

The sensor module triggers the DataReady signal when there is data for the host to receive. To maximize the performance and minimize the load on the I2C bus, the host is expected to read data in a certain sequence:

1. The sensor module asserts DataReady
2. The host initiates a two bytes I2C read operation for the I2C 7-bit address 0x50.
3. The sensor module fills in the first two bytes, FrameStart and DataSize.
4. The host initiates an I2C read operation for address 0x50 and reads XX bytes (as indicated by DataSize).
5. The sensor module deasserts DataReady.

BootComplete

When the sensor module has finished booting, and hardware such as the I2C module have been configured, the command BootComplete is put into the send queue. The DataReady signal is triggered and the host needs to read the data to acknowledge that the module is ready to operate. If the sensor module powers on before the host system does, the host system needs to check if the DataReady signal is active, in which case it needs to read the data.

> ⚠ Do not send any commands to the sensor module before BootComplete has been read.

8.4.4 zForce Message Specification

Definition and Encoding

The message structure that is used in the zForce communication protocol is defined in the zForce PDU definition file. PDU stands for Protocol Data Unit, the specific block of information that is going to be transferred. The definition is written according to Abstract Syntax Notation One (ASN.1), a standardized way to describe data regardless of language implementation, hardware system and operation system (ISO/IEC 8824).

To get the message data format used for information transfer, a set of encoding rules are applied to the ASN.1 definitions. The zForce communication protocol uses the Distinguished Encoding Rules (DER) to serialize the information that is going to be transferred. For more information, refer to Understanding the zForce ASN1 Protocol (see page 89).

ASN.1 defines a number of universal types, from which all other types inherit traits. The result is that a set of encoding rules that covers the universal types can serialize any PDU, as long as the identifier numbers and the definition categories are available. The identifier numbers and definition categories for the zForce PDU are included in the PDU definition file.

Downloading the definition file

Download the zForce® PDU definition file here[34].

The zForce message

The communication protocol uses three types of messages:

- Requests
- Responses
- Notifications

The host sends a request to the sensor module, and the device responds with a response. The device may send notifications to the host at any time.

All messages contain a virtual device address. Virtual devices are functionally isolated from each other, and communicate separately with the host. There are two types of virtual devices:

- Platform – represents the system.
- Air – represents one Neonode Touch Sensor Module.

A sensor module will always contain one platform virtual device and can contain any number of instances of other virtual device types.

In the definition file, all sensor messages are described as instances of the top level PDU `ProtocolMessage` which have three child PDUs:

PDU	Content	Explanation
Request	deviceAddress	Specifies the virtual device within the sensor module that receives the request.

[34] https://support.neonode.com/docs/display/Downloads/Communication+Protocol+Downloads

PDU	Content	Explanation
	command	The command specifies what is being requested.
Response	deviceAddress	Specifies the virtual device within the sensor module that sends the response.
	command	The command contains the result or requested information of the request.
Notification	deviceAddress	Specifies the virtual device within the sensor module that sends the notification.
	notificationMessage	The message from the sensor module to the host.
	notificationTimestamp (optional)	Timestamp.

PDU Description in GSER Notation

The PDU specifies the following message specification templates, notifications and pairs of requests and responses:

-
  - Application Interface (see page 79)
  - Device Information (see page 80)
  - Device Count (see page 81)
  - Frequency (see page 81)
  - Touch Sensor (see page 82)
  - Operation Mode (see page 82)
  - Touch Format (see page 83)
  - Enable Execution (see page 85)
  - Touch Notifications (see page 86)
  - Information (see page 86)
  - Configuration (see page 87)

The messages described here are encoded using the man-readable Generic String Encoding Rules (GSER), as the messages encoded according to DER can be difficult to read for a human.

A short introduction to the GSER notation:

- Sequences and/or sets (items containing sub items) are shown as curly brackets: **{ <sub elements> }**
- Values encoded as octet strings are written as hexadecimal octets enclosed within single quotes and suffixed with H: **'FF9900'H**
- Bit strings are also shown as octet strings when the number of bits is a multiple of 8, otherwise each bit is shown as a single 1 or 0, and suffixed with B: **'11010101001'B**
- In a choice element, the selected type is denoted by its name followed by a colon: **request:**

Refer to Generic String Encoding Rules (GSER) for ASN.1 Types[35] for a full reference.

The tool FFASN1Dump[36] can transcode from GSER to DER:

[35] https://tools.ietf.org/html/rfc3641#page-6

[36] http://www.bellard.org/ffasn1/

```
ffasn1dump -I gser -O der zforce_pdu_def.asn ProtocolMessage <input file> <output
file>
```

> Currently ffasn1dump does not handle identifiers for Integer values. For this reason, they need to be replaced with numerical values.

Application Interface

The application interface specifies what requests can be made and what responses and notifications they activate. Messages are specified according to the following templates:

request

```
request: {
  deviceAddress <address>,
  <request command>
}
```

response

```
response: {
  deviceAddress <address>,
  <command response>
}
```

notification

```
notification: {
  deviceAddress <address>,
  <notification>
  notificationTimestamp <timestamp>
}
```

Where

- Address is an octet string with 2 octets, device type and index: '<device type><index>'H. The available device types are:

Device type	Device
0	Platform
1	Core

Device type	Device
2	Air
3	Plus
4	Lightning

Example: ‘0000’H for platform (there can be only one platform device per sensor system).

- Timestamp is an integer representing int16 counting at 32768 Hz.

Device Information

The deviceInformation command fetches for example the product id and the FW version. What information that is available differs depending on the type of device. The following example shows a response from a platform device.

request command

```
deviceInformation {
}
```

command response

```
deviceInformation {
  platformInformation {
    platformVersionMajor 7,
    platformVersionMinor 0,
    protocolVersionMajor 1,
    protocolVersionMinor 5,
    firmwareVersionMajor 1,
    firmwareVersionMinor 0,
    hardwareIdentifier "Demo board",
    hardwareVersion "R2",
    asicType nn1002,
    numberOfAsics 1,
    mcuUniqueIdentifier '340038000E51333439333633'H,
    projectReference "DEMO_1.0_REL",
    platformReference "734cebd",
    buildTime "16:01:14",
    buildDate "2016-07-01"
      }
}
```

The fields have the following meaning:

- platformVersion: FW platform version, version 7.0 in the example.
- protocolVersion: communication protocol version, version 1.5 in the example.

- firmwareVersion: product FW version, version 1.0 in the example.
- hardware: product hardware, configuration and revision.
- asicType: which type of the Neonode optical scanner ASIC is used, and count.
- mcuUniqueIdentifier: identifier created at mcu manufacturing.
- projectReference: FW GIT tags or hashes. Product specific. Uniquely identifies the FW revision.
- platformReference: FW GIT tags or hashes. Uniquely identifies generic firmware base commit for the platform.
- buildTime: time of build in Central European Time, a string.
- buildDate: date of build, a string.

Device Count

The deviceCount command enumerates the available virtual devices.

request command

```
deviceCount {
}
```

command response

```
deviceCount {
  totalNumberOfDevices 1,
  airDevices 1
}
```

Device type instances are indexed from zero. The response shown here means that the only virtual device available is Air[0].

Frequency

The frequency command changes the update frequency of all sensor modules globally, that is for all devices on all platforms.

The following update frequencies can be set, if enabled in the product:

- finger: activated when objects with characteristics matching regular fingers are detected.
- stylus: activated for narrow stylus-like objects. (Not enabled for Neonode Touch Sensor Module.)
- idle: activated when no objects are detected, in order to minimize power usage.

The unit is Hz.

request command

```
  frequency {
    finger 30,
    idle 10
  }
```

The response contains the current frequency settings of the product:

command response

```
  frequency {
    finger 30,
    idle 10
  }
```

In this example, the sensor module update frequency will be 30 Hz as long as finger-like objects were recently detected. When no objects are detected, the frequency will drop to 10 Hz.

Touch Sensor

There are a number of different sensor module products that can co-exist on the same physical device. There are some product-specific commands, but the ones listed here are general.

The Touch Sensor Module will be used as example, which means that the device address will be the first Air virtual device

address

```
'0200'H
```

Operation Mode

The operationMode command sets what processing to perform on the sensor modules signals, and what diagnostics that are exposed.

The following example sets the operation mode to normal object detection:

request command

```
  operationMode {
    detection TRUE,
    signals FALSE,
    ledLevels FALSE,
    detectionHid FALSE,
    gestures FALSE
  }
```

command response

```
  operationMode {
    detection TRUE,
    signals FALSE,
    ledLevels FALSE,
    detectionHid FALSE
  }
```

> As can be seen gestures are missing in the response. This is a valid response, since the device is built with a subset of the protocol, or an older forward-compatible version.

Touch Format

The touchFormat command retrieves the binary format of the detected objects.

request command

```
  touchFormat {
  }
```

command response

```
  touchFormat {
    touchDescriptor { id, event, loc-x-byte1, loc-x-byte2, loc-y-byte1, loc-y-byte2,
size-x-byte1, size-y-byte1 }
  }
```

The touchDescriptor is a bit string, where each bit signifies one byte of payload being included in the touchNotification octet strings. A touchNotification is the concatenation of those bytes. The following table lists all bits. Bits used in the example are marked green.

Name	Description	Comment
id	Touch Identifier	
event	Up/Down/Move	0=Down; 1=Move; 2=Up; 3=Invalid; 4=Ghost
loc-x-byte1	X coordinate	
loc-x-byte2	X expanded	for higher precision
loc-x-byte3	X expanded	for higher precision
loc-y-byte1	Y coordinate	
loc-y-byte2	Y expanded	for higher precision
loc-y-byte3	Y expanded	for higher precision
loc-z-byte1	Z coordinate	
loc-z-byte2	Z expanded	for higher precision
loc-z-byte3	Z expanded	for higher precision
size-x-byte1	X size	
size-x-byte2	X size	for higher precision
size-x-byte3	X size	for higher precision
size-y-byte1	Y size	
size-y-byte2	Y size	for higher precision
size-y-byte3	Y size	for higher precision
size-z-byte1	Z size	
size-z-byte2	Z size	for higher precision
size-z-byte3	Z size	for higher precision
orientation	Orientation	Hand orientation
confidence	Confidence	Ignore. This value is not reliable for the Touch Sensor Module.
pressure	Pressure	

Location and size coordinates can be specified with up to 3 bytes. The byte order in decreasing significance - big-endian. For example:

- 1 byte: location x = loc-x-byte1

- 2 bytes: location x = (loc-x-byte1 << 8) + loc-x-byte2
- 3 bytes: location x = (loc-x-byte1 << 16) + (loc-x-byte2 << 8) + loc-x-byte3

Location is signed, and size is not.

The location coordinate scale is one of two systems, depending on which detector is used:

- Physical: Robair Air and Core detectors: The unit is 0.1 mm. A coordinate value of 463 thus means 46.3 mm from origin.
- Relative: Triangles and Shape Air detectors: Fraction of the largest screen dimension as fixed point with 14 bits after the radix point (q14). On a widescreen display, the horizontal axis ranges [0, 2^{14}[, and vertical [0, 2^{14} * 9/16[([0, 16383], [0, 9215]).

> ⓘ Touch Sensor Module uses Robair, thus the unit is 0.1 mm.

Size is in mm.

Confidence and pressure are fractions of the full values, in percent.

Enable Execution

The enable command activates the Touch Sensor Module, and notifications of detections start to stream.

request command

```
enable {
  enable 0
}
```

command response

```
enable {
  enable
}
```

To deactivate the Touch Sensor Module, send the disable command:

request command

```
enable {
  disable NULL
}
```

command response

```
  enable {
    disable NULL
  }
```

Touch Notifications

A detected object is reported with a touchNotification. The touchNotification payload is a touchDescriptor bit string. Every concurrently tracked object is represented by its own touchNotification payload.

notification

```
notificationMessage touchNotifications: {
    '0001013600730A0A64'H
  },
```

The following table shows the value of the example payload interpreted with the touch descriptor.

Name	Description	Comment	Value
id	Touch Identifier		0
event	Up/Down/Move	0=Down; 1=Move; 2=Up; 3=Invalid; 4=Ghost	1
loc-x-byte1	X coordinate		1
loc-x-byte2	X expanded	for higher precision	54
loc-y-byte1	Y coordinate		0
loc-y-byte2	Y expanded	for higher precision	115
size-x-byte1	X size		10
size-y-byte1	Y size		10

The touchNotification is from a Core device and translates to "Object 0 moved. Location is (31.0, 11.5) mm. Size is 10x10 mm."

Information

The command deviceInformation retrieves some information about the virtual device instance.

request command

```
deviceInformation {
}
```

command response

```
deviceInformation {
  deviceInstanceInformation {
    productVersionMajor 1,
    productVersionMinor 38,
    physicalWidth 1584,
    physicalHeight 1341,
    numberOfSignalAxes 0
  }
}
```

The response contains the deviceInstanceInformation structure, with the following parts:

Part	Description
productVersion	The specific type version of the virtual device.
physical	Size in unit 0.1 mm. See section Touch Format for the relationship to location coordinates.
numberOfSignalAxes	Only applicable for Core devices. The number of sensor arrays, each monitoring one dimension/axis of a touch sensor. Generally 2.

Configuration

Some configurations of the Touch Sensor Module can be changed at run-time. The deviceConfiguration request command and command response are identical, except some configuration items in the request may be omitted in order to leave them in their current state.

For instance, to set object size restrictions only, omit all other items:

request command

```
deviceConfiguration {
  sizeRestriction {
    maxSizeEnabled TRUE,
    maxSize 100,
    minSizeEnabled FALSE
  }
}
```

The command response contains the state of all configuration items:

command response

```
deviceConfiguration {
  subTouchActiveArea {
    lowBoundX 0,
    lowBoundY 0,
    highBoundX 1584,
    highBoundY 1341,
    reverseX FALSE,
    reverseY FALSE,
    flipXY FALSE,
    offsetX 0,
    offsetY 0
  },
  sizeRestriction {
    maxSizeEnabled FALSE,
    maxSize 0,
    minSizeEnabled FALSE,
    minSize 0
  },
  detectionMode default,
  numberOfReportedTouches 2,
  hidDisplaySize {
    x 1584,
    y 1341
  }
}
```

The items are:

- subTouchActiveArea: Crop the sensor module to a rectangle between the specified low and high coordinates in each dimension. Offset can be applied and flip the X and Y axis. Origin of reported locations is set to low coordinates, or if reversed, the high coordinate with increasing coordinates toward low.
- sizeRestriction: Limit detection to objects within this size range. Unit is 0.1 mm.
- detectionMode, one of the following:
  - default: finger and stylus
  - finger: Finger only
  - mergeTouches: Merges all touch objects into one

    - insensitiveFTIR: Unsupported
- numberOfReportedTouches: Maximum number of reported tracked objects.
- hidDisplaySize: Scaling the coordinate system when using the sensor module in HID Touch Digitizer mode.

8.4.5 Understanding the zForce ASN.1 Serialization Protocol

All communication with the Touch Sensor Module is serialized with Neonodes ASN.1 serialization protocol, and when implementing your own solution for the sensor module it is vital to know how to encode and decode it. This article explains the basics of ASN.1 DER/BER encoding and then walks you through the encoding of a DeviceConfiguration Message from the zForce ASN.1 Protocol.

Downloading the definition file

Download the zForce® PDU definition file here[37].

Type, Length, Value

All ASN.1 messages are structured by type, length and value:

- Type describes the type of the whole message or a subpart of a message. Type includes information on class and tag number.
- Length defines how many bytes there are in the message or in other words, the size of the value.
- Value is the actual value you are sending to the device, it is either a primitive value or a list. The primitive value types that are used in this protocol are:
    - Integer
    - Boolean
    - Octet String
    - Bit String

The Type Byte(s)

Type bytes can be constructed or primitive. A constructed type is a list, and in this protocol, all lists are defined by a sequence, and will hereafter be referred to as sequences.
Class tags and tag numbers are used to encode a type byte. The following ASN.1 class tags are used:

Class	Bit 8	Bit 7	Description
Universal	0	0	Not used in our protocol
Application	0	1	Shared
Context-specific	1	0	Local. For example specific to an Application
Private	1	1	Only used to describe the type of the whole message, Request/ Response/Notification

[37] https://support.neonode.com/docs/display/Downloads/Communication+Protocol+Downloads

The encoding for tag numbers up to and including 30 (higher code numbers are encoded differently, but those are not used in our protocol):

Bit position		8th	7th	6th	5th	4th	3rd	2nd	1st
Specific bit value	**Binary representation**	1000 0000	0100 0000	0010 0000	0001 0000	0000 1000	0000 0100	0000 0010	0000 0001
	Decimal representation	128	64	32	16	8	4	2	1
	Hexadecimal representation	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01
Description		Reserved for class tag	Reserved for class tag	Reserved for primitive/ sequence	Reserved for tag number	Reserved for tag number	Reserved for tag number	Reserved for tag number	Reserved for tag number

In short this means that the 8th and 7th bits are reserved to specify the class, and the 6th bit is reserved to show if it is a sequence. Bits 5 to 1 are used to specify the tag number.

Example: The type byte for the command deviceConfiguration

The command as seen in the protocol:

```
deviceConfiguration [APPLICATION 19] Sequence
```

This tells us that the deviceConfiguration is a sequence with the class tag Application and 19 as tag number. The deviceConfiguration type byte looks like this when it is represented as an octet:

Bit position	8th	7th	6th	5th	4th	3rd	2nd	1st
Bit	0	1	1	1	0	0	1	1
Hexadecimal Value		0x40	0x20	0x10			0x02	0x01

Description	Class tag	Class tag	Sequence tag	Tag number value	Tag number value	Tag number value	Tag number value	Tag number value

All of the hexadecimal numbers are then added together to get the type byte → 0x40 + 0x20 + 0x10 + 0x02 + 0x01 = 0x73.

The Length Byte(s)

The length byte defines the number of bytes in the value byte(s) that follows it, and if the number is 127 or below, the length byte is only one byte. If the number is 128 or higher, the length byte splits into two pieces: The first piece, is the first byte that describes the amount of length bytes that follows it, and the second piece is the unsigned integer that holds the whole length value.

Example: The length byte for a value that is 50 bytes long

The decimal value 50 translates to 0x32 in hexadecimal representation and 0011 0010 in binary representation

Bit position	**8th**	**7th**	**6th**	**5th**	**4th**	**3rd**	**2nd**	**1st**
Bit	0	0	1	1	0	0	1	0
Hexadecimal Value			0x20	0x10			0x02	
Description	If this bit is set the length of the value is 128 bytes or longer	Value	Value	Value	Value	Value	Value	Value

Example: The length bytes for a value that is 300 bytes long

The length bytes for a value that is 300 bytes long consists of three bytes. The first byte indicates that the following two bytes ((0x01) 0000 0001 (0x2C) 0010 1100) should be added together. The first byte is described in the following table:

Bit position	**8th**	**7th**	**6th**	**5th**	**4th**	**3rd**	**2nd**	**1st**
Bit	1	0	0	0	0	0	1	0
Hexadecimal Value	0x80						0x02	
Description	If this bit is set, the length of the value is 128 bytes or longer	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes

The Value Byte(s)

The value byte(s) can be as few as one byte or they can be a whole sequence following the Type Length Value format:

- Integers are represented by one or more bytes. An integer between 0 and 127 is represented by one byte (00 to 7F). An integer between 128 and 32767 is represented by two bytes (00 80 to 7F FF).
- A Boolean only requires one byte, since it is either true or false. The Boolean is either 0x00 or 0xFF.
- An Octet String takes up just as many bytes as the number of octets.
- Bit string: the number of bytes needed to represent a bit string depends on the number of bits and is easily explained in code:

```
int valueLength = 0;

 void CalculateValueLength()
 {
         valueLength = bitString.Length / 8;

         if((bitString.Length % 8) != 0)
         {
                valueLength++;
         }
 }
```

Request, Response, and Notification

When the host is communicating with the sensor module, all of the messages are defined as either request, response, or notification.

- Request is a message that is sent to the sensor module from the host.
- Response is a message that responds to the request.
- Notification is a message that is read by the host and is generated without any input from the host, for example a touch message or a boot complete message.

The messages look like this in the protocol:

```
*Message Definition*
    ProtocolMessage ::= CHOICE {
        request [PRIVATE 14] Message,
        response [PRIVATE 15] Message,
        notification [PRIVATE 16] Notification
    }
```

All three have the class tag private, and they are all sequences which means that the 8th, 7th, and 6th bits are all set. In binary this evaluates to 1110 0000 which in hexadecimal translates to 0xE0. Now all that needs to be done is to define which type of message it is, which in this case is either tag number 14, 15, or 16. In order to define a request, tag number 14 (0x0E) needs to be added to 0xE0, which sums up to 0xEE.

Device Address

All messages include a Device Address. Most messages go to the Air Device, but some go to the Platform Device. In the protocol, the DeviceAddress looks like this:

```
*Device Address Definition*
    DeviceAddress ::= [APPLICATION 0] OCTET STRING (SIZE (2))
    -- Addressing information used when multiple touch devices are present
    -- in the system.
    -- Byte0 - deviceType, Byte1 - deviceIndex
    -- DeviceTypes:   0x00    -   Platform
    --                0x01    -    zForce Core
    --                0x02    -    Air (Touch Sensor Module)
    --                0x03    -    zForce Plus
    --                0x04    -    Lighting devices
```

Example: Evaluate the device address

How to evaluate the address:

1. Type byte: The deviceAddress has the class tag Application, the tag number 0 and contains two octet strings. An octet string is primitive, and therefore the 6th bit is set to 0. In binary this evaluates to 0100 0000 and in hexadecimal that is 0x40.
2. Length byte: 2 (two octets).
3. Value bytes: The first byte is deviceType, the second is deviceIndex.
    a. Device type: In this case this is the Touch Sensor Module (Air), represented by 0x02.
    b. Device index: On a Touch Sensor Module (Air) the index is always 0.

The complete device address then evaluates to 0x40 0x02 0x02 0x00:

Type	Length	Value
DeviceAddress	Value length	deviceType followed by deviceIndex
0x40	0x02	0x02 0x00

Encoding a Device Configuration Message

Device Configuration will be used as an example as it is a message that contains a large number of values with both context-specific primitive values and context-specific sequences.

```
*Device Configuration ASN.1 Protocol*
-- Instance specific settings for a device
   deviceConfiguration [APPLICATION 19] Sequence {
            -- Set / get the number of touches to be tracked:
            numberOfTrackedTouches    [0] INTEGER (0..255) OPTIONAL,
            -- Set / get the minimal distance for updating a tracked touch in move
state
            trackingMinDistanceMove    [1] INTEGER (0..16383) OPTIONAL,
            -- Set / get the sub touch active area low bound in X coordinate
            subTouchActiveArea        [2] Sequence {
                -- Write Request and Read Response only:
                -- Set / get the sub touch active area low bound in X coordinate
                lowBoundX        [0] INTEGER (0..16383) OPTIONAL,
                -- Set / get the sub touch active area low bound in Y coordinate
                lowBoundY       [1] INTEGER (0..16383) OPTIONAL,
                -- Set / get the sub touch active area high bound in X coordinate
                highBoundX    [2] INTEGER (0..16383) OPTIONAL,
                -- Set / get the sub touch active area high bound in Y coordinate
                highBoundY    [3] INTEGER (0..16383) OPTIONAL,
```

All settings are optional and therefore does not require all settings to be defined in the message that is sent to the sensor module.

We want to set the following settings in the sensor module

SubTouchActiveArea:

- LowBoundX: 500
- LowBoundY: 500
- HighBoundX: 2000
- HighBoundY: 2000

This is how to do it (the length bytes are represented by XX, and are added in the last step):

Step	What to do	Details	Result	The message
1	Add the code for a **request**, since the message will be sent from the host to the sensor module: 0xEE 0xXX	See Request, response and notification (see page 92).	EE XX	EE XX
2	Add the **device address** 0x40 0x02 0x02 0x00	See Device Address (see page 93).	40 02 02 00	EE XX **40 02 02 00**

3	Add the bytes for **deviceConfiguration**	The command deviceConfiguration has the class tag Application, tag number 19 and is a sequence.	Binary: 0111 0011 Hexadecimal: 0x73.	EE XX 40 02 02 00 **73**
4	Add the bytes for **SubTouchActiveArea**	SubTouchActiveArea is a context-specific sequence with tag number 2.	Binary: 1010 0010 Hexadecimal: 0xA2	EE XX 40 02 02 00 73 **XX A2 XX**
5	Add the variables inside of SubTouchActiveArea..	All the variables are context-specific and primitive (binary 1000 0000, hexadecimal 0x80). Depending on which variable that is currently being added, add the specific tag number.	Per variable: Binary: 1000 0000 plus tag number. Hexadecimal: 0x80 plus tag number.	EE XX 40 02 02 00 73 XX A2 XX 80 XX 01 F4 81 XX 01 F4 82 XX 07 D0 83 XX 07 D0
6	Add the length bytes of the message.	Add the number of bytes for each specific part of the message.	-	EE **18** 40 02 02 00 73 **12** A2 **10** 80 **02** 01 F4 81 **02** 01 F4 82 **02** 07 D0 83 **02** 07 D0

8.5 Updating Firmware

The firmware on a Neonode Touch Sensor Module can be updated easily using the **Firmware Update** application for Windows. Download it here[38].

8.5.1 Prerequisites

Hardware requirements

- CPU: 1 GHz
- RAM: 512 MB
- Disk space: 20 MB
- Internet connection (for the automatic download of necessary Windows drivers)
- The interface board provided with the evaluation kit.

[38] https://support.neonode.com/docs/display/Downloads/Firmware+Update+Software

Operating System requirements

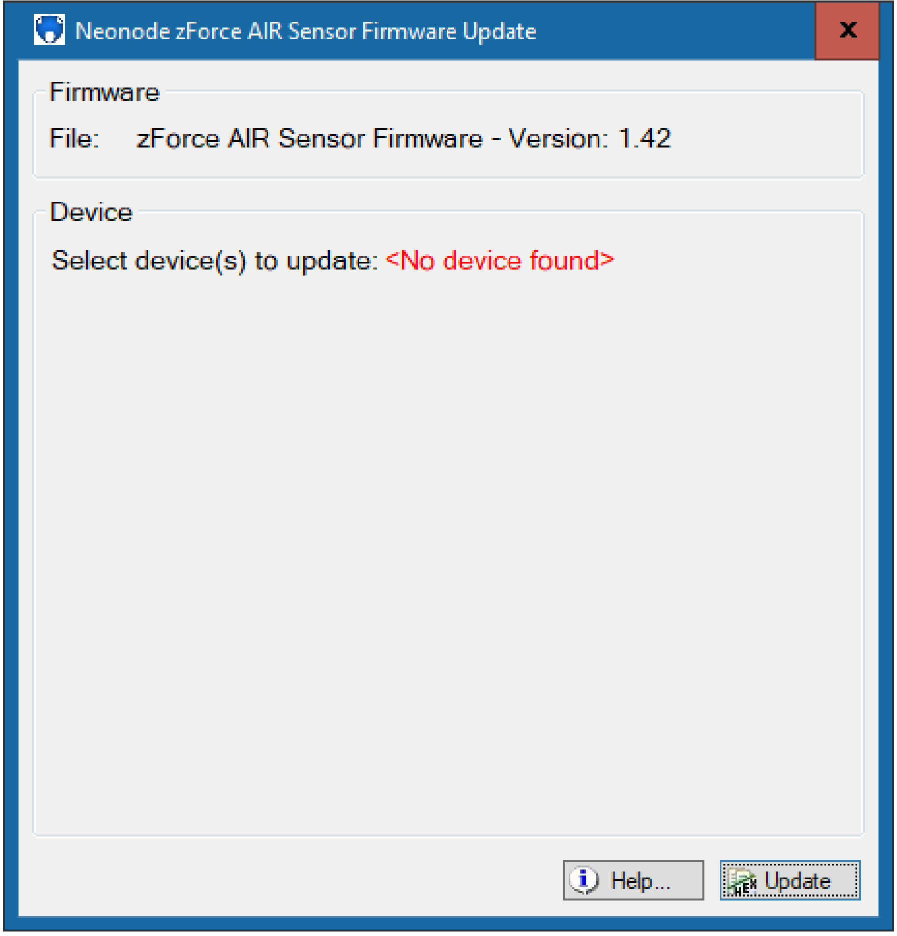
- Windows 10
- Windows 8.1

Software requirements

- .NET Framework 4.5 or higher is required and can be downloaded from Microsoft's official website. Windows 8 and higher has this installed by default.

8.5.2 Procedure

1. Download the wanted firmware here[39], and run the application.
   The new firmware version is displayed, under **Firmware**



Neonode zForce AIR Sensor Firmware Update

Firmware

File: zForce AIR Sensor Firmware - Version: 1.42

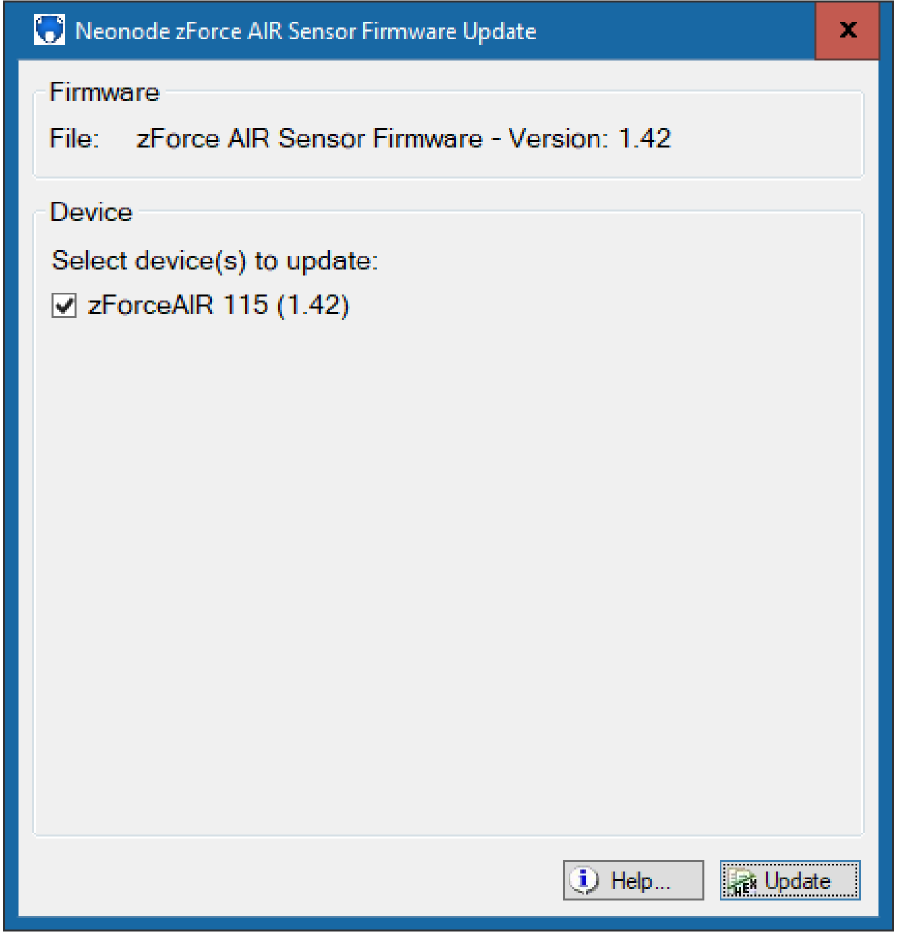
Device

Select device(s) to update: <No device found>

2. Connect the Touch Sensor Module to the computer via the provided interface board. Refer to Getting started with Touch Sensor Module Evaluation (see page 16) for details on how to connect the sensor.

[39] https://support.neonode.com/docs/display/Downloads/Firmware+Update+Software

The connected sensor's current firmware version is displayed, under **Device**.



Neonode zForce AIR Sensor Firmware Update

Firmware

File: zForce AIR Sensor Firmware - Version: 1.42

Device

Select device(s) to update:

☑ zForceAIR 115 (1.42)

3. Make sure the checkbox for the sensor is checked and click **Update**.
   The new firmware will be loaded into the sensor module. If it is the first time the application is used to update the firmware, an internet connection will be required to install necessary drivers.
   A progress bar will be displayed with status during the firmware update. This process may take a few seconds to a minute depending on the computer.
   When the firmware update is completed, a pop up message is displayed. The sensor module can be used right away after the update is finished.

8.5.3 Troubleshooting

If the updating fails or the device does not show up in the application, check the following possible scenarios. If the problem persists, please contact support.

The sensor module does not appear under Device in the application after it is plugged in

1. Click **Help...** and follow the instructions.

The progress bar gets stuck at 0% to 5% during the firmware update

1. Click **Help...** and follow the instructions to set the sensor module in DFU mode.
2. Check "*STM device in DFU mode*" or "*STM32 BOOTLOADER*" device under "*Universal Serial Bus controller*" or "*Other devices*" in Windows Device Manager to ensure drivers are working properly. Uninstall or update the driver if necessary.

The firmware update fails with the message "*Failed to download new firmware*" or the progress bar gets stuck at about 50 to 60%

1. Click **Help...** and follow the instructions.

The progress bar gets stuck at 100% during firmware update with the message "*Waiting for USB configuration...*",

This is because the sensor module failed to leave "boot mode" but the firmware is already updated.

1. Click on **Cancel** button and re-plug the sensor. The new firmware is updated and the sensor is ready to be used.

8.5.4 Firmware Release Notes 1.49

Contents of Release

- Bug fix.
- Modified the relationship between Hid Display Size and Offset.
- Extended range.

Fixed bugs

- Fixed an off by one error in the reported Physical Height.

Features and Modifications

- The settings Hid Display Size and Offset (in Device Configuration) are now independent of each other and changing one of these settings will not affect the other.
- Added an additional firmware package for NNAMC3460PC01 and NNAMC3461PC01 with support for extended range.

8.5.5 Firmware Release Notes 1.47

Contents of Release

- Bug fixes

Fixed bugs

- Fixed a bug where Flip XY did not work as expected, when ReversY and ReverseX was also set to true.
- FIxed a bug for "Sub Area Offset" when using "Flip XY".

8.5.6 Firmware Release Notes 1.46

Contents of Release

- Bug fixes

Fixed bugs

A bug was fixed in the gesture feature, that caused poor performance.

8.5.7 Firmware Release Notes 1.45

Contents of Release

- Added a reflective edge filter
- Bug fixes
- Changed name of the hardware id

New features

A new reflective edge filter has been added and is disabled by default. Instructions on how to enable it is available here[40].

Changed the name of the hardware id to "zForceAIR sensor". Previously called "AirModule".

Enhancements

Fixed bugs

A bug was fixed in the X size of the touching object and the Y size is no longer reported

[40] https://support.neonode.com/docs/display/AIRTSUsersGuide/Communication+Protocol+Quick+Start

9 Implementation Examples

The following examples are here to give you an idea of different ways to use the sensor module, and different ways of integrating the sensor into your system.

9.1 Interface Library for Arduino

9.1.1 Use Case

This library is useful if you would like to easily get started with a prototype, understand how to communicate with the sensor module using I2C or if you would like to see how the library has implemented the communication.

9.1.2 Introduction

The library offers an easy way to communicate with the Touch Sensor Module as well as some primitive parsing of the ASN.1 serialized messages. This makes it easy for the end user to get x and y coordinates from touch notifications or set different settings such as the correct touch active area. The library does not have support for all messages available in the ASN.1 protocol, however the I2C read and write functions are public and can be used if any setting or request not supported by the library needs to be sent/read from the sensor module.

9.1.3 Open Source

This library is distributed under the GNU LGPL v2.1 open source license and is available on GitHub[41] along with additional documentation as well as a full example program. For questions regarding how to use the library, please refer to our help center[42].

9.1.4 How to use the library

Main Loop

The library is built around using zforce.GetMessage() as the main loop for reading messages from the sensor module. GetMessage checks if the data ready pin is high and if it is, the function zforce.Read() will be called. The read function takes a buffer parameter which is used to store the data from the sensor module.

[41] https://github.com/neonode-inc/zforce-arduino
[42] https://helpcenter.neonode.com/hc/en-us/requests/new

GetMessage

```
Message* Zforce::GetMessage()
{
  Message* msg = nullptr;
  if(GetDataReady() == HIGH)
  {
    if(!Read(buffer))
    {
      msg = VirtualParse(buffer);
      ClearBuffer(buffer);
    }
  }

  return msg;
}
```

When GetMessage has been called it is up to the end user to destroy the message by calling zforce.DestroyMessage() and passing the message pointer as a parameter.

Send and Read Messages

The library has support for some basic settings in the Touch Sensor Module, for example zforce.SetTouchActiveArea(). When writing a message to the sensor module the end user has to make sure that data ready is not high before writing. This is done by calling GetMessage and reading whatever might be in the I2C buffer.

When a message has been sent, the sensor module always creates a response that has to be read by the host. It could take some time for the sensor module to create the response and put it on the I2C buffer, which is why it is recommended to call the GetMessage function in a do while loop.

Send and Read

```
  // Make sure that there is nothing in the I2C buffer before writing to the sensor
module
  Message* msg = zforce.GetMessage();
  if(msg != NULL)
  {
    // Here you can read whatever is in the message or just destroy it.
    zforce.DestroyMessage(msg);
  }

  // Send Touch Active Area request
  zforce.TouchActiveArea(50,50,2000,4000);

  // Wait for the response to arrive
  do
  {
    msg = zforce.GetMessage();
  } while (msg == NULL);

  // See what the response contains
  if(msg->type == MessageType::TOUCHACTIVEAREATYPE)
  {
    Serial.print("minX is: ");
    Serial.println(((TouchActiveAreaMessage*)msg)->minX);
    Serial.print("minY is: ");
    Serial.println(((TouchActiveAreaMessage*)msg)->minY);
    Serial.print("maxX is: ");
    Serial.println(((TouchActiveAreaMessage*)msg)->maxX);
    Serial.print("maxY is: ");
    Serial.println(((TouchActiveAreaMessage*)msg)->maxY);
  }
  // Destroy the message
  zforce.DestroyMessage(msg);
```

9.2 USB HID Communication Example

9.2.1 Use Case

This sort of implementation should be used when you want to communicate using USB, and you are not able to use either Neonode Workbench or the zForce SDK to configure the sensor, due to system requirements.

9.2.2 Introduction

The zForce AIR sensor supports communication over I2C and USB. Both the Neonode Workbench, and the zForce SDK uses USB hidpipe to communicate with the sensor, but unfortunately not all systems can use the SDK or the

Workbench application. This might be the case if you are planning to connect the sensor to (for example) an Android device, and need to change some settings, which is also why this example will be in Java using the Android SDK.

What is HID?

HID stands for Human Interface Device, and is generally a device that takes input from humans and delivers it to a host system. The zForce AIR sensor has two different modes for the USB connection, which are the HID Digitizer mode, and the Raw HID mode.

Raw HID mode

When the sensor is connected to a system it is automatically put into the HID Digitizer mode, but in order to communicate with it, and send/read messages from it in a program, it has to be put into (also known as hidpipe) Raw HID mode. This is done by enumerating the sensor in your program, and sending a command to it.

9.2.3 Connecting to the sensor

Enumerating

The first thing you have to do in your program, is to make sure that the sensor is talking to your program instead of communicating with the OS. This is done by enumerating the sensor, and the code can look a bit different depending on which OS or SDK you are using, but the general idea is still the same, and usually follows the USB standard.

In Java, using the Android SDK, the code looks like this:

```
manager = (UsbManager) getSystemService(Context.USB_SERVICE); // Create a UsbManager
Map<String, UsbDevice> connectedDevices = manager.getDeviceList(); // Get all the
connected USB devices and save them in a map


for (UsbDevice device : connectedDevices.values()) // Loop through all the connected
devices
{
    if(device.getVendorId() == 0x1536 && device.getProductId() == 0x0101) // If a
zForce AIR Sensor is found,
    {
        mPermissionIntent = PendingIntent.getBroadcast(this, 0, new
 Intent(ACTION_USB_PERMISSION), 0);
        IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
        registerReceiver(NmUsbReceiver, filter);
        manager.requestPermission(device, mPermissionIntent);

        Nintf = device.getInterface(0);
        // NEONODE 0=IN, no out
        Nendpoint = Nintf.getEndpoint(0);
        Nconnection = manager.openDevice(device);
        Nconnection.claimInterface(Nintf, forceClaim);
        configure_neonode();
        read_neonode();
    }
}
```

```
 public void SetOperationMode ()
 {
    byte[] sendArray = new byte[257];

    byte[] detectionMode = {
            (byte) 0x01,(byte) 0x17,(byte) 0xEE,(byte) 0x15,(byte) 0x40,(byte) 0x02,
(byte) 0x02,
            (byte) 0x00,(byte) 0x67,(byte) 0x0F,(byte) 0x80,(byte) 0x01,
            (byte) 0xFF,(byte) 0x81,(byte) 0x01,(byte) 0x00,(byte) 0x82,
            (byte) 0x01,(byte) 0x00,(byte) 0x83,(byte) 0x01,(byte) 0x00,
            (byte) 0x84,(byte) 0x01,(byte) 0x00
    };

    System.arraycopy(detectionMode, 0, sendArray, 0, detectionMode.length);

    int requestType = UsbConstants.USB_DIR_OUT | UsbConstants.USB_TYPE_CLASS |
UsbConstants.USB_INTERFACE_SUBCLASS_BOOT;
    int request = 0x09;
    int value = 0x0301;
    int index = 0x0000;

    // write to device to set it in detectionMode, 0x01 for feature report 1
    int read_response = sendFeatureReport(0x01, sendArray, sendArray.length);

    String responseString = "Request";
    Log.i(responseString, Integer.toString(read_response));
    String byteString = byteArrayToHex(detectionMode);
    Log.i(responseString, byteString);


    //Read the response of the previous request
    byte readBuffer[] = new byte[258];

    // read from the device, 0x02 for feature report 2
    read_response = getFeatureReport(0x02, readBuffer, readBuffer.length);

    responseString = "Response";
    Log.i(responseString, Integer.toString(read_response));
    byteString = byteArrayToHex(readBuffer);
    Log.i(responseString, byteString);

 }

    public int SendFeatureReport(int reportId, byte[] data, int length) {
        if ((reportId & 0xFF) != reportId)
            throw new IllegalArgumentException("reportId may only set the lowest 8
bits");
        return zForceAirConnection.controlTransfer(
                UsbConstants.USB_DIR_OUT | UsbConstants.USB_TYPE_CLASS |
UsbConstants.USB_INTERFACE_SUBCLASS_BOOT,
                0x09,
                reportId | 0x0300,
                0x0000, data, length, 0);
    }

    public int GetFeatureReport(int reportId, byte[] data, int length) {
        if ((reportId & 0xFF) != reportId)
            throw new IllegalArgumentException("reportId may only set the lowest 8
bits");
        return zForceAirConnection.controlTransfer(
                UsbConstants.USB_DIR_IN | UsbConstants.USB_TYPE_CLASS |
UsbConstants.USB_INTERFACE_SUBCLASS_BOOT,
                0x01,
                reportId | 0x0300,
                0x0000, data, length, 0);
    }
```

9.3 Sub Touch Active Area

9.3.1 Introduction

The Touch Sensor Module can be adjusted by changing the module's device configuration. By configuring these settings, the size or orientation of the Touch Active Area (TAA) can for instance be modified. Device Configuration is implemented according to our ASN.1 protocol (see page 56), and most of the parameters can be configured in Workbench, zForce SDK, or Arduino Library (I2C). The parameters that have been configured using our evaluation tools or your own application, are stored in the ram memory. Meaning that the new configuration will have to be applied after each reboot. Please refer to section **Device Configuration List** to find all avaliable parameters.

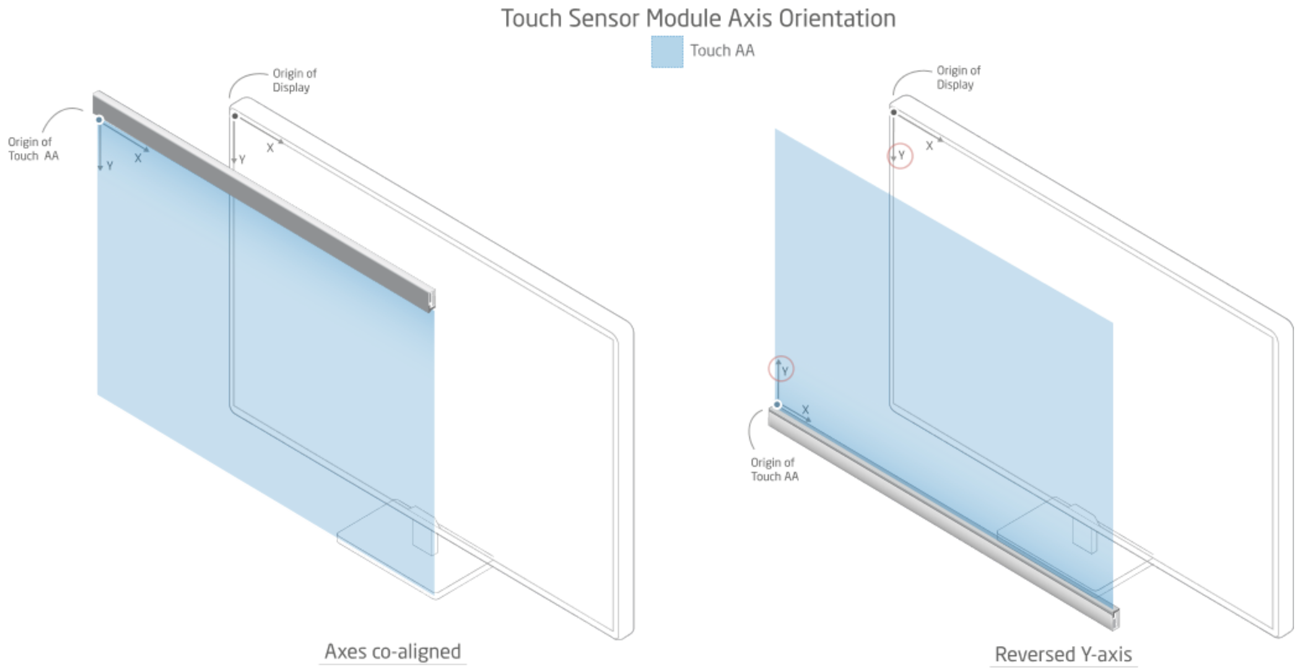
Index

9.3.2 Axis Orientation

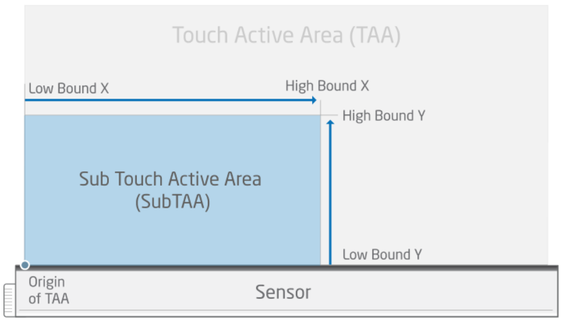
When mapping a reported touch from the Touch Active Area to a display, it is important that the orientation of the TAA, as well as the display are taken into account. When positioning a Touch Sensor Module over a display to achieve touch functionality, both systems needs to be co-aligned in order for a reported touch to be projected in the same corresponding position of the screen. If both systems are not oriented in the same way, the reported touches are going to be reversed depending of the setup. Luckily, the orientation of the reported touches can be configured to counter this by using *ReverseX/ReverseY* or *FlipXY* in Device Configuration. These settings rotates or flips the given coordinates of the reported touches, which allows the sensor to be mounted at any of the 4 sides of the screen

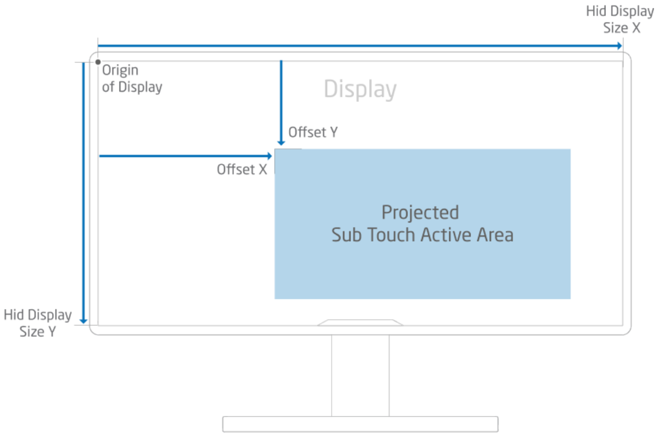
Almost all displays has a reference point (or origin) positioned at the upper left corner of the screen, where the x-axis points in the direction to the right, with the y-axis pointing downwards. The origin of the sensor module's Touch Active Area is positioned on the left hand side, when having the TAA facing downwards, with the black side of the sensor module facing outwards.

This means that if a sensor module is positioned above a display, with the black side facing outwards, and its TAA covering the screen - Both coordinates systems would be co-aligned and a reported touch would project on the display seamlessly. But if we were to flip the sensor module to cover the screen from underneath the display, a reported touch would appear to be reversed in y-direction since their coordinate systems would no longer be co-aligned, as illustrated below.



9.3.3 Device Configuration List





Configuration	Range	Measurement	Description
Idle Frequency	Frequency (see page 139)	Hz	Scanning frequency when the Touch Sensor Module have not registered any touch object.
Finger Frequency	Frequency (see page 139)	Hz	Scanning frequency when the Touch Sensor Module have registered and tracking a touch object.
Number Of Reported Touches	1-10 touches	Quantity	The number of touches that can be reported simultaneously.
Max Size Enabled	True/False	Boolean	Allows a maximum touch object size.
Max Size	Mechanical Data[43]	10^{-1}·mm	Max limitation of a touch object (iif *Max Size Enabled =True*).
Min Size Enabled	True/False	Boolean	Allows a minimum touch object size.
Min Size	Mechanical Data[44]	10^{-1}·mm	Min limitation of a touch object (iif *Min Size Enabled =True*).
Low Bound X	Mechanical Data[45]	10^{-1}·mm	Start position of the (Sub)TAA in x-direction.
Low Bound Y	Mechanical Data[46]	10^{-1}·mm	Start position of the (Sub)TAA in y-direction.
High Bound X	Mechanical Data[47]	10^{-1}·mm	End position of the (Sub)TAA in x-direction.
High Bound Y	Mechanical Data[48]	10^{-1}·mm	End position of the (Sub)TAA in y-direction.
Reverse X	True/False	Boolean	Reverses the x-coordinates of reported touches.

[43] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[44] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[45] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[46] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[47] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[48] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data

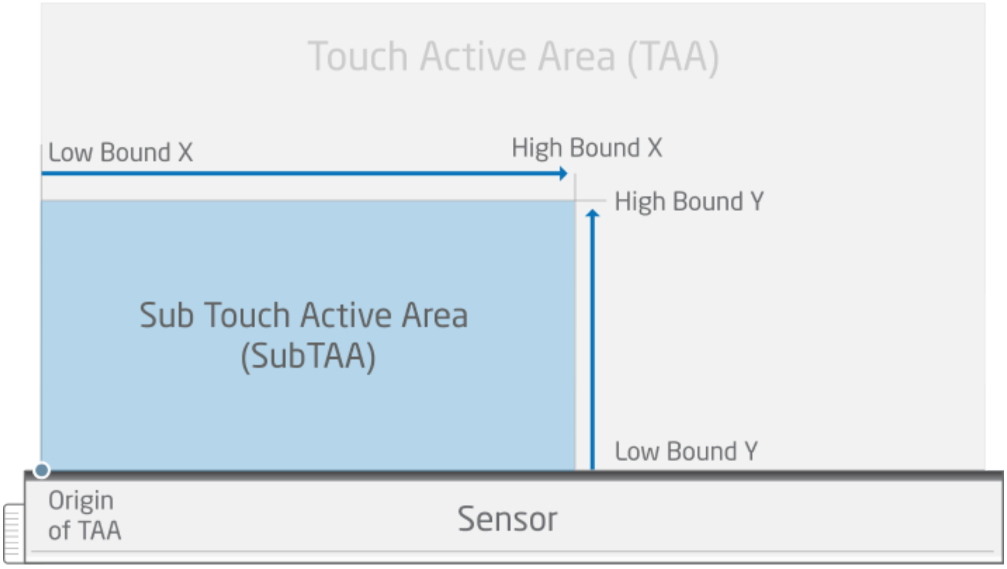
Reverse Y	True/False	Boolean	Reverses the y-coordinates of reported touches.
Flip XY	True/False	Boolean	Swaps the y- with the x-coordinates of the reported touches.
Offset X	Mechanical Data[49]	10^{-1}·mm	Offsets the projected (Sub)TAA in x-direction, on the display.
Offset Y	Mechanical Data[50]	10^{-1}·mm	Offsets the projected (Sub)TAA in y-direction, on the display.
Hid Display Size X	Mechanical[51] Data[52]	10^{-1}·mm	Width of the display.
Hid Display Size Y	Mechanical[53] Data[54]	10^{-1}·mm	Height of the display.

Low- and High Bound

The *Low* or *High Bound* of the x- or y-axis can be configured in order to make a Touch Active Area (TAA) smaller, creating a so-called *Sub Touch Active Area* (SubTAA). This limits the interactive area, and touches would only be reported within the SubTAA.

The SubTAA is measured from its origin, in the range of *Low Bound* and *High Bound,* in x- or y-direction*.*

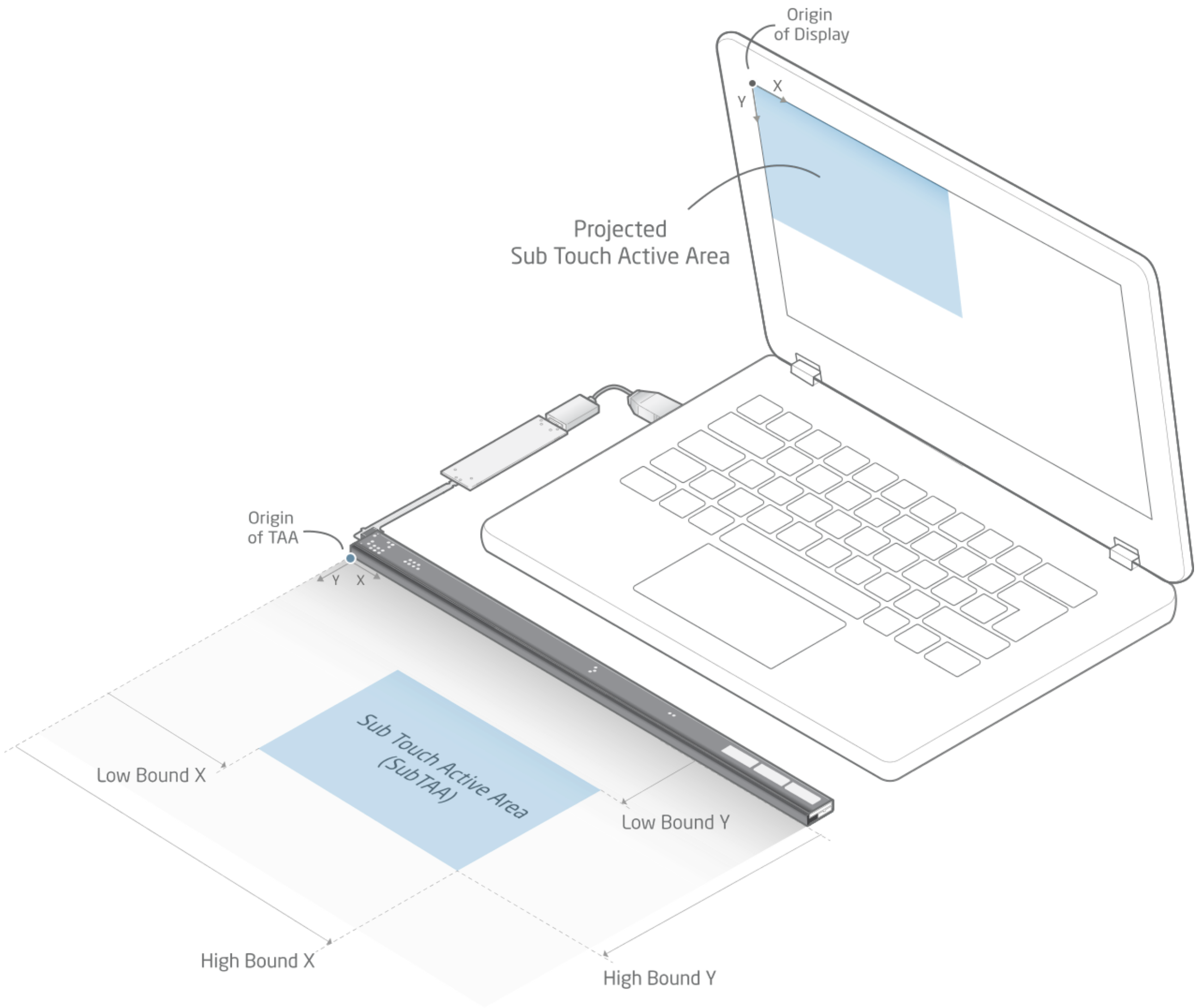
[49] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[50] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[51] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[52] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[53] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data
[54] https://support.neonode.com/docs/display/AIRTSUsersGuide/Mechanical+Data



Where,

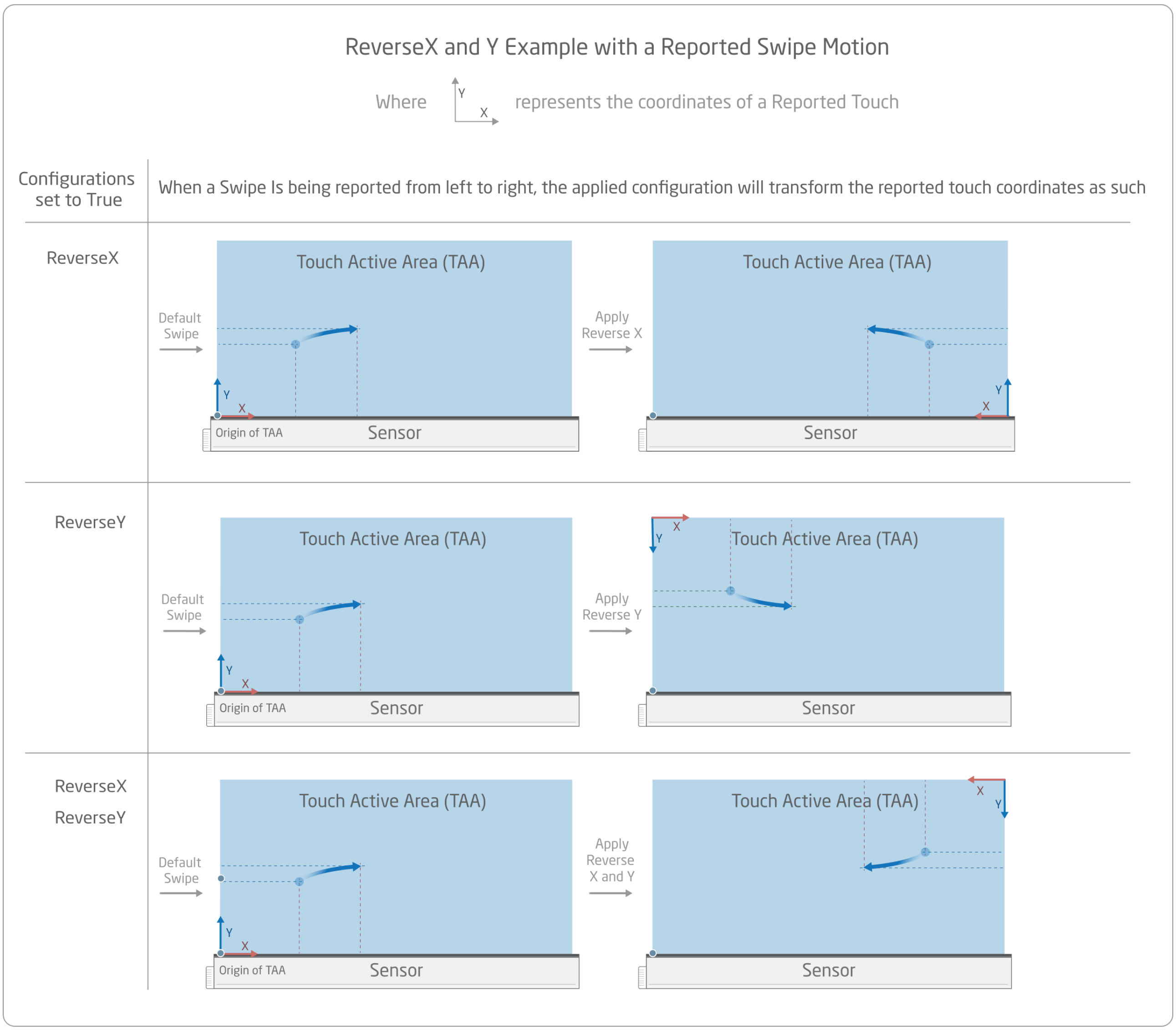
- *Low Bound X* - Start position of the (Sub)TAA in x-direction, measured from the origin of the TAA.
- *High Bound X* - End position of the (Sub)TAA in x-direction, measured from the origin of the TAA.
- *Low Bound Y* - Start position of the (Sub)TAA in y-direction, measured from the origin of the TAA.
- *High Bound Y* - End position of the (Sub)TAA in y-direction, measured from the origin of the TAA.

When projecting a SubTAA to a display within the host system, the projected SubTAA would then be mapped to the display's origin, regardless of its position (as shown in the illustration below).



9.3.4 Reverse X and Y

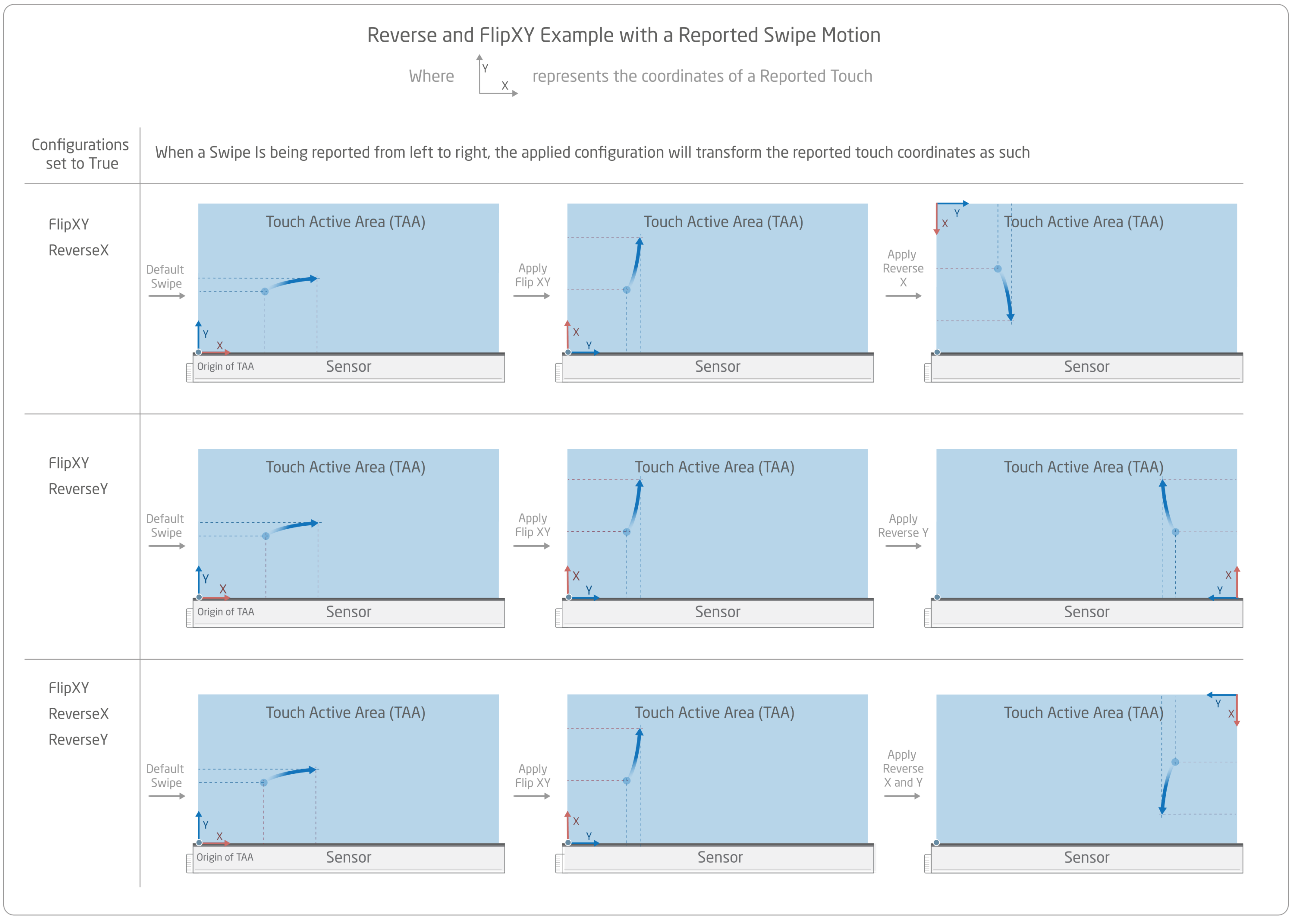
Reverse X or *Y* reverses the coordinates of a reported touch in x- or y-direction, without effecting the orientation of the (Sub) Touch Active Area. When positioning a sensor module, the axes of the Touch Active Area should either be positioned in the same orientation as the display, or have the reported touch coordinates adjusted. Please consider the following example.



9.3.5 Flip XY

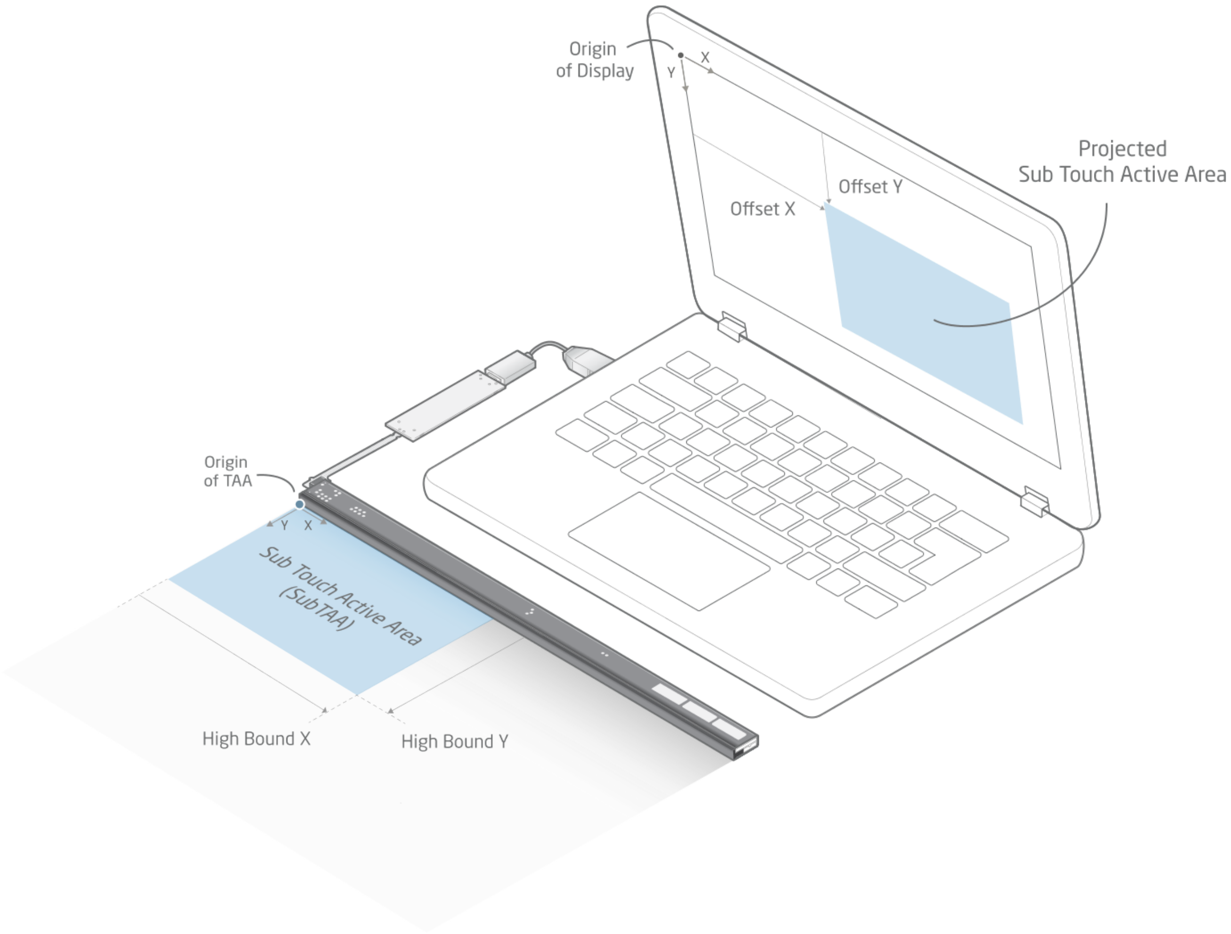
Flip XY swaps the x- with the y-axis of the reported touches*,* without effecting the orientation of the (Sub) Touch Active Area. When swapping axes, the reported touch data would then be sent to the host system containing the new x- and y coordinates.

FlipXY together with *ReverseX/Y* can be used together which allows the Touch Sensor Module to be positioned around all edges of a display. When positioning a sensor module, the axes of the Touch Active Area should either be positioned in the same orientation as the display, or have the reported touch coordinates adjusted. Please consider the following example.


Reverse and FlipXY Example with a Reported Swipe Motion
Where Y X represents the coordinates of a Reported Touch
Configurations set to True
When a Swipe Is being reported from left to right, the applied configuration will transform the reported touch coordinates as such
FlipXY
ReverseX
FlipXY
ReverseY
FlipXY
ReverseX
ReverseY
Default Swipe
Apply Flip XY
Apply Reverse X
Apply Reverse Y
Apply Reverse X and Y
Touch Active Area (TAA)
Origin of TAA
Sensor
X
Y

9.3.6 Offset X and Y

Offset X or *Y* configures the position of the mapped (Sub) Touch Active Area onto the display for the given axes. When adding the offset to a SubTAA, the position of the projected SubTAA will then be moved according to the following illustration.



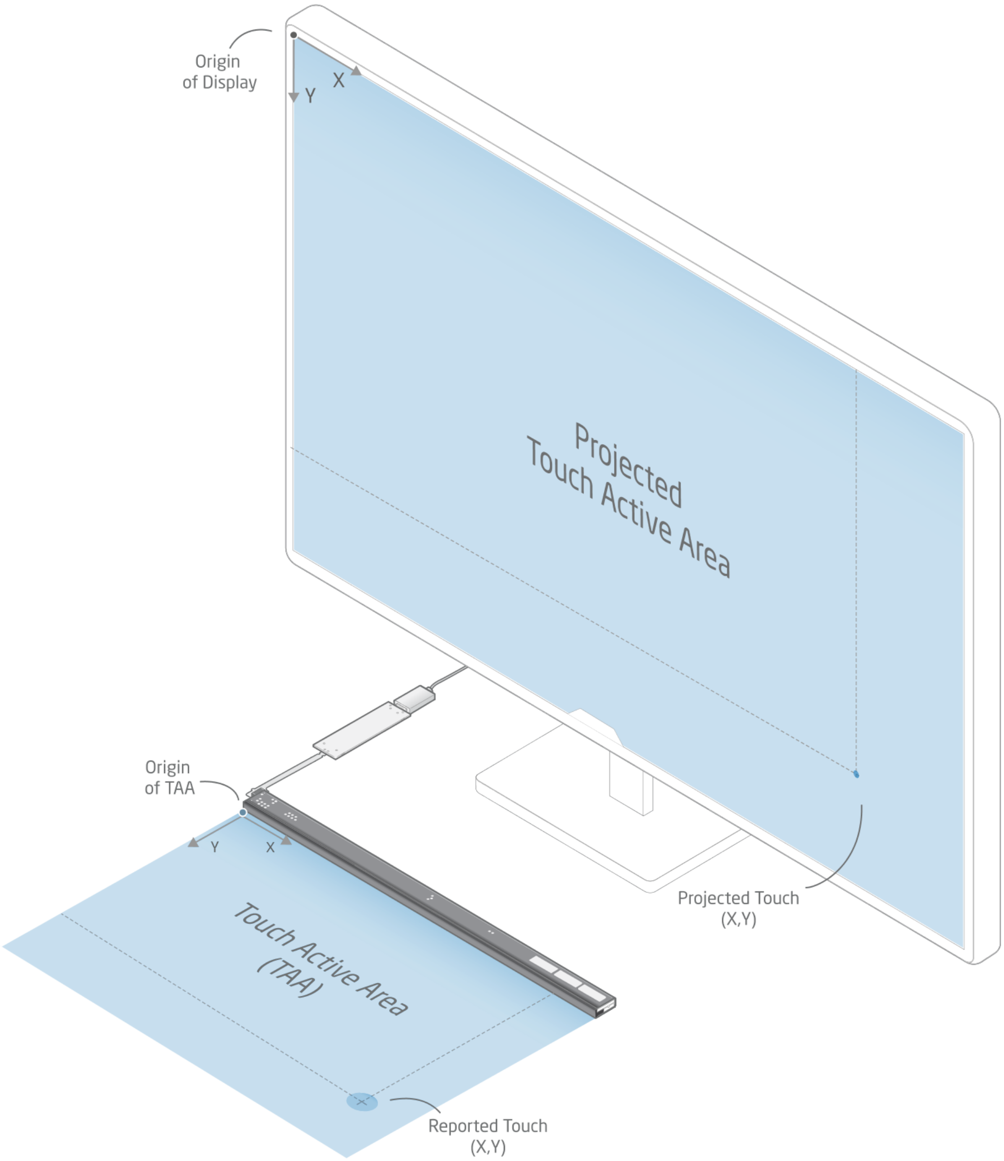
9.3.7 Hid Display Size X and Y

Hid Display Size X or *Y* represent the width and height of a display in the host system. The default value is set to match the size of the sensor module's Touch Active Area, creating a proportional mapping regardless of the display size. In practice, this means that if a display is larger than the TAA, a projected touch would by default on the display proportionally, as if the projected TAA were "stretched" to fit the display size and ratio. An alternative to a proportional mapping would be to create a selective touch area, where only a part of the display have touch functionality.

Proportional Mapping

When *Hid Display Size* is set to the size of the Touch Active Area, the mapping of a reported touch would get projected proportionally, to fit the size of the display.

For instance, if a display within the host system is larger than the TAA, a reported touch will get projected as the illustration show below.

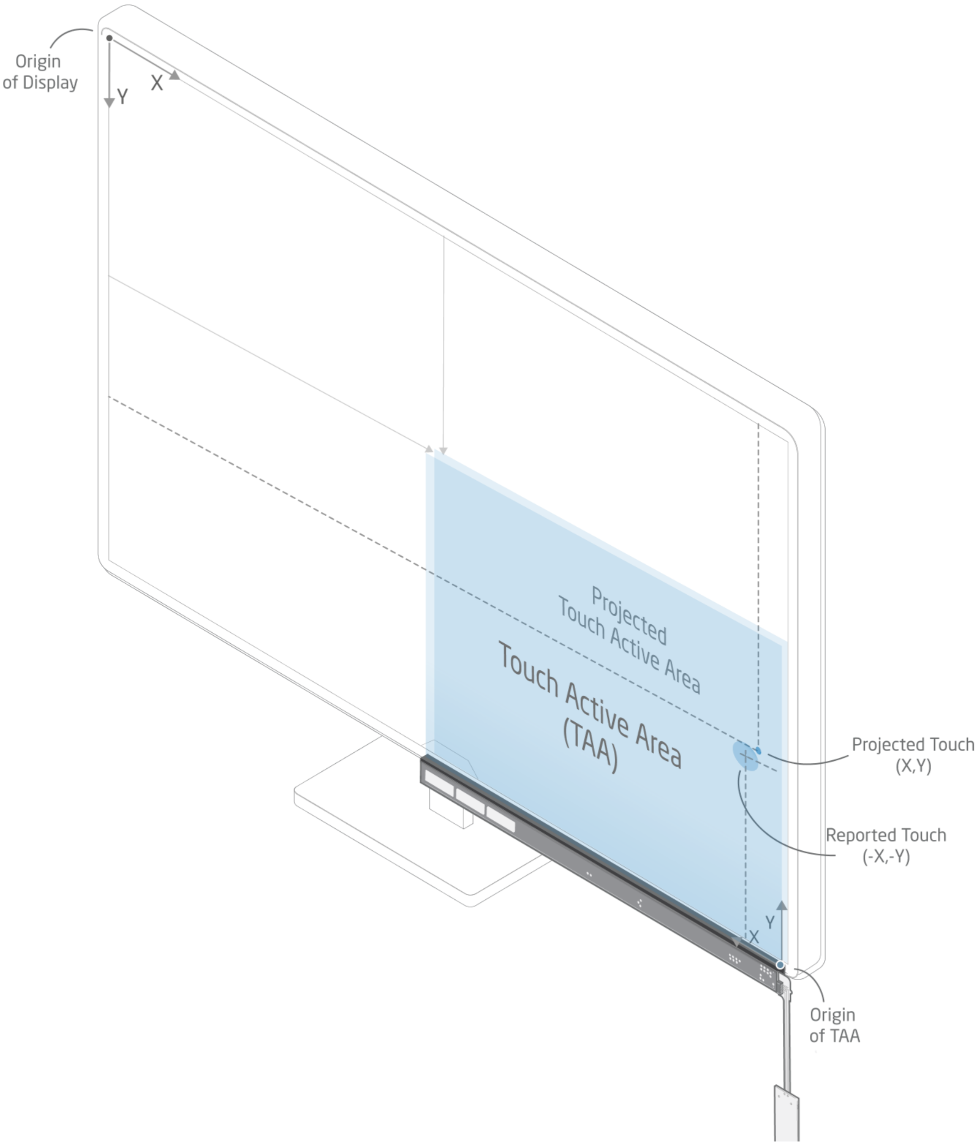


Selective Touch Area

It is possible to create a so-called Selective Touch Area, where only a part of the display have touch functionality.

For instance, if a Touch Sensor Module is positioned over a larger display, we can create a touch surface over that specific part of the screen. Please refer to Selective Touch Area[55] for further information and examples.

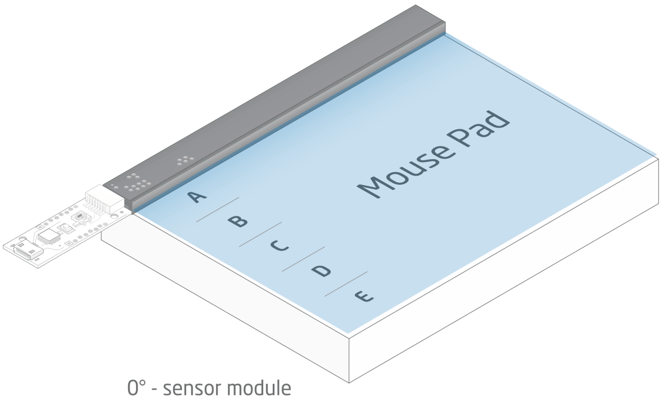
[55] https://support.neonode.com/docs/display/AIRTSUsersGuide/Selective+Touch+Area


Origin
of Display
Y
X
Projected
Touch Active Area
Touch Active Area
(TAA)
Projected Touch
(X,Y)
Reported Touch
(-X,-Y)
Y
X
Origin
of TAA

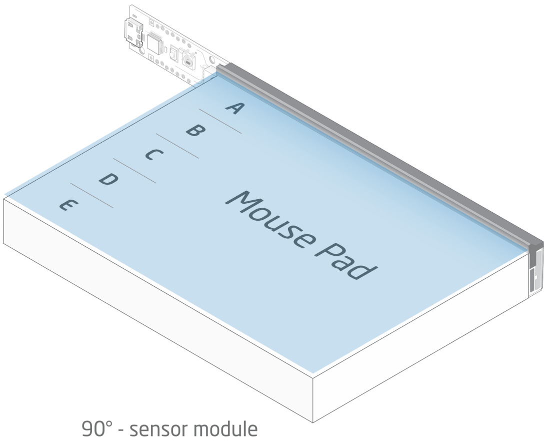
9.4 I2C Mouse pad and Keyboard

The Touch Sensor Module can be used to send keyboard or mouse input to a computer through I2C. An example of how this can be achieved is included in the Arduino Library[56] example zForceKeyboardMouse[57]. The example uses the official Arduino libraries *<Mouse.h>* and *<Keyboard.h.>* to communicate with the host system.

The Touch Sensor Module can be placed as an in-air solution or on any surface, providing plenty of alternatives for a wide range of users. The sensor module can for instance be configured to work as a personalized control panel, or mixer board. Due to the customizable layout and controls, It could be used in development, as well as assisting people who needs a customized workspace.



0° - sensor module



90° - sensor module

9.4.1 Description

The zForceKeyboardMouse example showcase how touch input can be converted as mouse or keyboard input. The example enables SAMD micro based boards (e.g. Neonode Prototyping Board[58]) to send keystrokes or mouse navigation to the host system by using the official Arduino libraries *<Mouse.h>* and *<Keyboard.h>,* through HID.

The example divides the Touch Active Area into one mouse pad and a keyboard section, containing 5 buttons (A-E). The mouse section works like a relative mouse pad, where the cursor moves relatively to its previous position. The Keyboard buttons are programmed to print the letters A-E when a touch is being preformed in each corresponding section.

We can access the touch data from the sensor module through the zForce Arduino library

[56] https://support.neonode.com/docs/display/AIRTSUsersGuide/Touch+Sensor+Module+Interface+Library+for+Arduino
[57] https://github.com/neonode-inc/zforce-arduino
[58] https://support.neonode.com/docs/display/NPB/User%27s+Guide+-+Neonode+Prototyping+Board

```
int16_t x = ((TouchMessage*)touch)->touchData[0].x;                        //Touch
input position x coordinate
int16_t y = ((TouchMessage*)touch)->touchData[0].y;                        //Touch
input position y coordinate
int8_t event = ((TouchMessage*)touch)->touchData[0].event;        //Touch input event
state, i.e TouchUp, TouchDown...
```

Touch handling Mouse Pad

Since the Touch Sensor Module is recognized as a touch screen digitizer, the touch input data needs to be extracted in order to translate it as mouse input. To translate the absolute position input coordinates to its relative position (movement), we subtract the current touch data position with the previous tracked object position.

```
Mouse.move((x - previousTouch.x), (y - previousTouch.y));         //"x" and "y" are
the location of the reported touch coordinates

/"previousTouch.x" and "previousTouch.y" are the location of the

/ previous reported coordinate.
```

A left-click mouse action is preformed once "tapping" or "clicking" withing the mouse section. The left-click action is fired when a reported touch has the event data "UP". In practice, this means that the end user would needs to both "press" and "release" their finger in order to preform a touch. The touch sensitivity can be adjusted by the global variable *holdTime,* which acts like a timer for how long a "touch" can take.

In order to use the mouse pad, we use the collected touch information from the sensor module:

```
switch (event)
{
  case 0:  // DOWN event
    previousTouch.x =  x;
    previousTouch.y =  y;
    globalMillis = millis();
    Serial.println("Mouse Input - DOWN");

    break;

  case 1: // MOVE event
    if ((millis() - globalMillis) >= holdTime)
    {
      Mouse.move((x - previousTouch.x), (y - previousTouch.y));
      Serial.println("Mouse Input - Moving cursor");
    }
    previousTouch.x = x;
    previousTouch.y = y;
    break;

  case 2: // UP event
    Serial.print("Mouse Input - UP");
    if (millis() - globalMillis < holdTime)  // mouse "left click", sensitivity
    { // can be tuned by changing "holdTime"
      Mouse.click(MOUSE_LEFT);
      Serial.print("(Left-Click)");
    }
    Serial.println("");
    break;
  default: break;
}
```

In-Air Contactless Touch Adjustment

For in-air solutions, we recommend to preform all "left-clicks" or "key presses" on the DOWN-event instead of the UP-event. By making this adjustment, the left-click will be fired when the end user press down their finger (instead of when releasing). This gives the Touch Active Area a more tactile response, which is optimal for in-air solutions

To make a left-click trigger on the DOWN-event instead, please consider the following adjustments:

```
switch (event)
  {
    case 0:  // DOWN event
      previousTouch.x =  x;
      previousTouch.y = y;
      globalMillis = millis();
      Mouse.click(MOUSE_LEFT);         //Left-click on DOWN-event
      break;

    case 1: // MOVE event
      if ((millis() - globalMillis) >= holdTime)
      {
        Mouse.move((x - previousTouch.x), (y - previousTouch.y));
      }
      previousTouch.x = x;
      previousTouch.y = y;
      break;

    case 2: // UP event
      break;
     default: break;
  }
```

Touch Handling Keyboard

The keyboard buttons will send a key press depending on the position of the touch object. Each buttons are sectioned by the variable *keyboardBoundary,* and are thereafter sectioned further to distinguish each key.

If a reported touch with a DOWN-event is positioned within the *keyboardBoundary*, a second look-up will be preformed in order to evaluate which key to print.

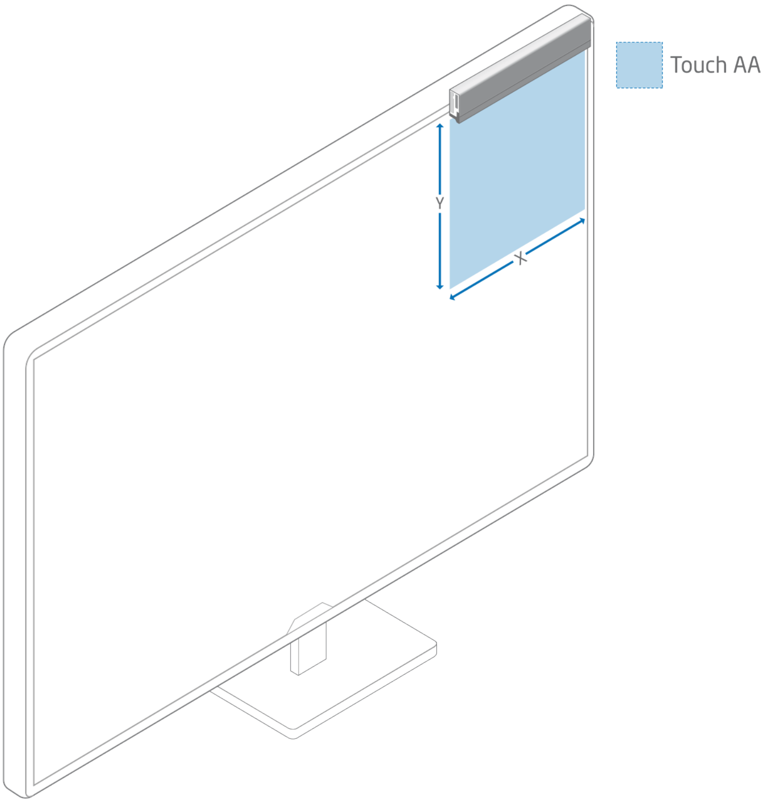
```
 if (event == 0) { // DOWN event
    //assign Key to the given interval

    if (y < 250)
    Keyboard.print('A'); //Print Key "A"
    else if (y < 500)
    Keyboard.print('B'); //Print Key "B"
    else if (y < 750)
    Keyboard.print('C'); //Print Key "C"
    else if (y < 1000)
    Keyboard.print('D'); //Print Key "D"
    else
    Keyboard.print('E'); //Print Key "E"
  }
```

9.5 Selective Touch Area

9.5.1 Use Case

This method of configuring sensor module(s) can be used to get one or multiple touch areas on a larger screen or on a large projected area, further referred to as Screen.

9.5.2 Illustrations



9.5.3 Introduction

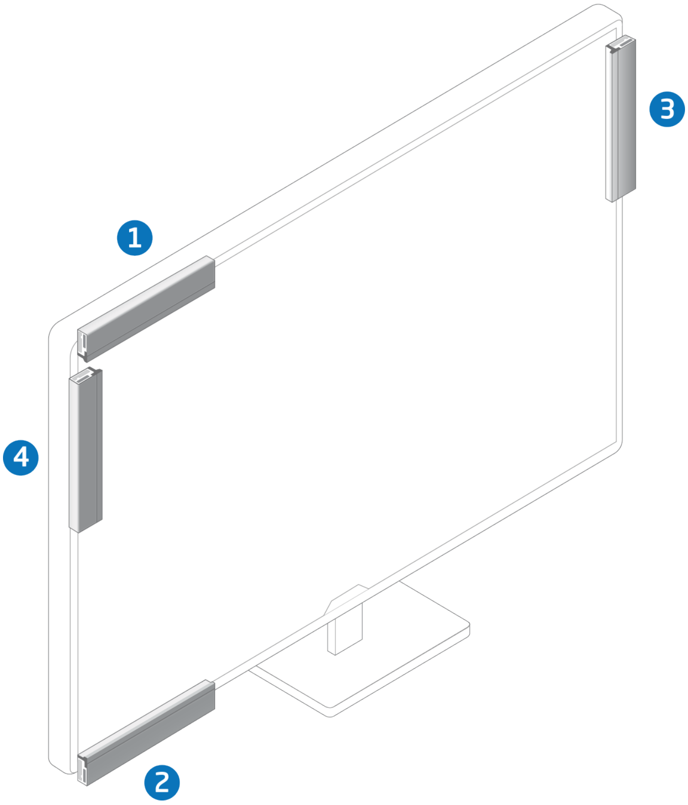
A Touch Sensor Module can be mounted on four sides of a screen and with the connector to either the right or the left (PCB or silver side down) but we recommend mounting it with the silver side towards the screen, as the touch area is then closer to the screen.

This will give a total of eight different configurations.

It is possible to mount a sensor module "on" or "in" the screen. However this will cover/block part of the screen and is therefore not a part of this example.

1. Top: Sensor module on top of the screen facing down.
2. Bottom: Sensor module on bottom of the screen facing up.
3. Right: Sensor module on right side of the screen facing left.

4. Left: Sensor module on the left side of the screen facing right.



9.5.4 HID Display Size

HID Display Size is the physical size of the screen in tenths of millimeters (1/10 ; 1 mm = 10)

9.5.5 Configurations With One Sensor Module

NOTE: Values in **bold** has to be changed from default to make the configuration work.

Top - Connector to the right

Configuration	**1 (Top - Connector to the right)**
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>

Configuration	1 (Top - Connector to the right)
Sub Area Reverse X	**True**
Sub Area Reverse Y	False
Sub Area Flip XY	False
Sub Area Offset X	**The offset distance left side of the screen and the (left side of the) sensor module.** If the sensor module sticks out to the left of the screen change "Sub Area High Bound X" . Subtract the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).
Sub Area Offset Y	Zero (0) If you have the sensor above the screen change the "Sub Area Low Bound Y" to the distance between screen and sensor module.
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

Top - Connector to the left

Configuration	1 (Top - Connector to the left)
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>
Sub Area Reverse X	False
Sub Area Reverse Y	False
Sub Area Flip XY	False

Configuration	1 (Top - Connector to the left)
Sub Area Offset X	**The offset distance left side of the screen and the (left side of the) sensor module**. If the sensor module sticks out to the left of the screen change "Sub Area Low Bound X" . Add the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).
Sub Area Offset Y	Zero (0) If you have the sensor above the screen change the "Sub Area Low Bound Y" to the distance between screen and sensor.
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

Bottom - Connector to the right

Configuration	2 (Bottom - Connector to the right)
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>
Sub Area Reverse X	**True**
Sub Area Reverse Y	**True**
Sub Area Flip XY	False
Sub Area Offset X	**The offset distance left side of the screen and the (left side of the) sensor module .** If the sensor module sticks out to the left of the screen change "Sub Area High Bound X" . Subtract the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).

Configuration	2 (Bottom - Connector to the right)
Sub Area Offset Y	**This is the value of "Hid Display Size Y" minus "Sub Area High Bound Y" plus "Sub Area Low Bound Y"** If the sensor module is below the screen change the "Sub Area Low Bound Y " to the distance between screen and sensor module. Also update the value of "Sub Area Offset Y" to correspond to the new offset.
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

Bottom - Connector to the left

Configuration	2 (Bottom - Connector to the left)
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>
Sub Area Reverse X	False
Sub Area Reverse Y	**True**
Sub Area Flip XY	False
Sub Area Offset X	**The offset distance left side of the screen and the (left side of the) sensor module.** If the sensor module sticks out to the left of the screen change "Sub Area Low Bound X" . Add the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).

Configuration	2 (Bottom - Connector to the left)
Sub Area Offset Y	**This is the value of "Hid Display Size Y" minus "Sub Area High Bound Y" plus "Sub Area Low Bound Y"** If the sensor module is below the screen change the "Sub Area Low Bound Y " to the distance between screen and sensor module. Also update the value of "Sub Area Offset Y" to correspond to the new offset.
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

Left - Connector to the top

Configuration	3 (Left - Connector to the top)
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>
Sub Area Reverse X	False
Sub Area Reverse Y	False
Sub Area Flip XY	**True**
Sub Area Offset X	Zero (0) If you have the sensor module to the left of the screen change the "Sub Area Low Bound Y" to the distance between screen and sensor.

Configuration	3 (Left - Connector to the top)
Sub Area Offset Y	**The offset is the distance from the sensor module to the top of the screen.** **Max offset is "Hid Display Size Y" minus "Sub Area High Bound X" plus "Sub Area Low Bound X".** If the sensor module sticks out to the top of the screen change "Sub Area Low Bound X" . Add the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

Left - Connector to the bottom

Configuration	3 (Left - Connector to the bottom)
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>
Sub Area Reverse X	False
Sub Area Reverse Y	**True**
Sub Area Flip XY	**True**
Sub Area Offset X	Zero (0) If you have the sensor module to the left of the screen change the "Sub Area Low Bound Y" to the distance between screen and sensor.
Sub Area Offset Y	**The offset is the distance from the sensor module to the top of the screen.** If the sensor module sticks out to the top of the screen change "Sub Area High Bound X" . Subtract the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).

Configuration	3 (Left - Connector to the bottom)
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

Right - Connector to the top

Configuration	4 (Right - Connector to the top)
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>
Sub Area Reverse X	**True**
Sub Area Reverse Y	False
Sub Area Flip XY	**True**
Sub Area Offset X	**This is the value of "Hid Display Size X" minus "Sub Area High Bound Y" plus "Sub Area Low Bound Y".** If you have the sensor module to the right of the screen change the "Sub Area Low Bound Y" to the distance between screen and sensor module.
Sub Area Offset Y	**The offset is the distance from the sensor to the top of the screen.** If the sensor module sticks out to the top of the screen change "Sub Area Low Bound X" . Add the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

Right - Connector to the bottom

Configuration	**4 (Right - Connector to the bottom)**
Sub Area Low Bound X	<Default values from sensor modules original configuration>
Sub Area Low Bound Y	<Default values from sensor modules original configuration>
Sub Area High Bound X	<Default values from sensor modules original configuration>
Sub Area High Bound Y	<Default values from sensor modules original configuration>
Sub Area Reverse X	**True**
Sub Area Reverse Y	**True**
Sub Area Flip XY	**True**
Sub Area Offset X	**This is the value of "Hid Display Size X" minus "Sub Area High Bound Y" plus "Sub Area Low Bound Y".** If you have the sensor module to the right of the screen change the "Sub Area Low Bound Y" to the distance between screen and sensor module.
Sub Area Offset Y	**The offset is the distance from the sensor module to the top of the screen.** If the sensor module sticks out to the top of the screen change "Sub Area High Bound X" . Subtract the distance the sensor module sticks out from the left edge of the screen and the SW (Software origin).
Hid Display Size X	**Width of screen in tenths of millimeters.**
Hid Display Size Y	**Height of screen in tenths of millimeters.**

9.5.6 Configurations With Multiple Sensor Modules

If you would like to configure multiple sensors modules for touch on different parts of the screen, please use the settings above for each sensor. If you would like to make this setting with Neonode Workbench, please use the Workspace designated for multiple sensor modules. If you would like to create your own solution, the zForce SDK can be used to implement this.

9.6 Touch Device Handling

9.6.1 Introduction

When the Touch Sensor Module sends touch notification, it contains event data that describes the state of the current tracked touch (*DOWN, MOVE, UP*). When connecting the Touch Sensor Module over USB HID, It is recognized as a Touch Screen Digitizer by the Operating System, where a "touch" is recognized on the *UP* event. In practice, this means that the end user would have to both "press" and "release" their finger (or touch object) before a touch can be preformed. Similar to a tapping motion.

For in-air contactless touch solutions, the application gets a stronger tactile response when the application to trigger on the *DOWN* event instead of the *UP* event. This way, a touch will be recognized as soon as a the object is entering the Touch Active Area. In this article, we will show how this can be achieved by binding buttons to the *DOWN event* of a tracked touch, using a WPF example application. For I2C communications, a demonstration of this can be found in I2C Mouse pad and Keyboard (see page 117).

9.6.2 USB Touch Screen Digitizer and Mouse

For the application to handle input from the Touch Sensor Module, it needs to both be able to read input from a touchscreen digitizer, as well as a mouse. The sensor module first sends touch input to the host system, which would then get translated to mouse input. Therefore, it is important to make sure that the application can handle both input devices, as well as separating the input devices from one another. Please refer to section *WPF Touch Device Handling Application Example* for an MVVM application example using WPF.

⚠ How to bind the touch event to trigger a button command might differ depending on your framework. It is usually with the same principle as this WPF application shows, but using different formatting.

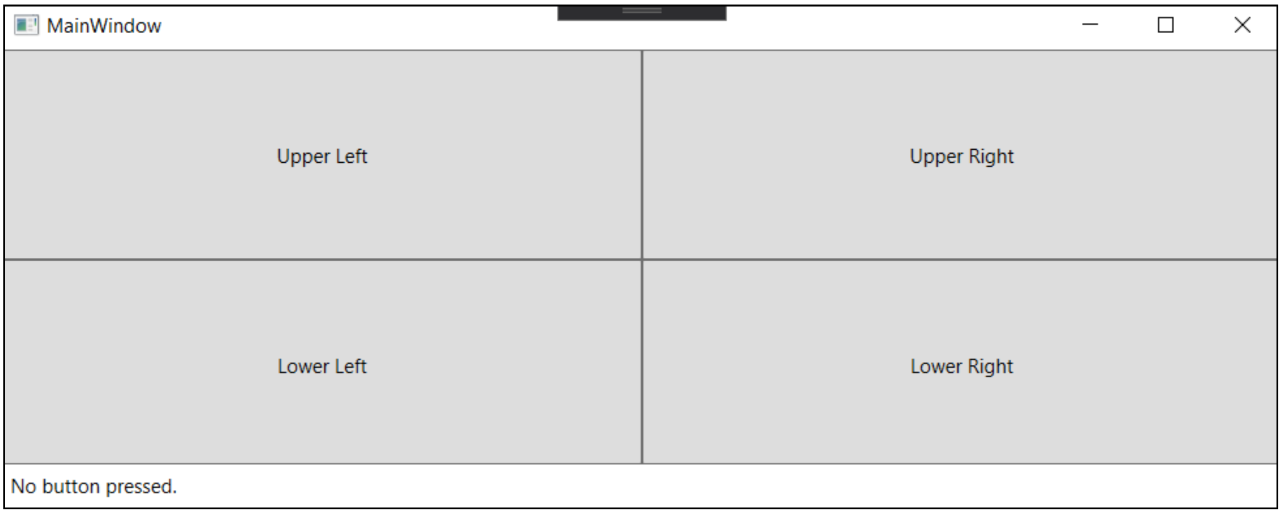
A "touch" has 4 significant events called, *TouchEnter*, *TouchDown*, *TouchLeave*, and *TouchUp* that is used to describe a *TOUCH* event. The wanted touch events (i.e. *TouchDown*) could then be handled in the application to trigger a button, for example.

Touch Event	Action
Touch Enter	Fired when an object enters the area. Can be either from the side, or a new object registering.
Touch Down	Fired when a new object enters the area. Note: Only fired on new events. If a new object enters straight into this area (or bound button), both TouchEnter and TouchDown are fired.
Touch Leave	Fired when an object leaves the area. Both when it is no longer tracked at all, and when it leaves the boundaries of the button.
Touch Up	Fired when a object leaves the area and is no longer tracked at all

To get a better understanding of the different touch events, please consider the following example.

Example

Imagine 4 buttons that are positioned in a 2x2 pattern called *UpperLeft*, *UpperRight*, *LowerLeft*, *LowerRight.* Each button can be configured to be fired on one on multiple touch events.



No button pressed.

Step-by-Step walkthrough of the touch events in the application

1. **Start:** No touches exist anywhere.
2. An object (finger) is detected within UpperLeft. (i.e. you put your finger inside the UpperLeft button).
   a. The *TouchDown* event is fired inside UpperLeft (i.e. the handler is called, if it is registered to handle the *TouchDown* case).
   b. The *TouchEnter* event is fired inside UpperLeft.
3. The object is moved from UpperLeft to UpperRight.
   a. The *TouchLeave* event is fired inside UpperLeft. Note that *TouchUp* is NOT fired as we are still tracking the object!).
   b. The *TouchEnter* event is fired inside UpperRight. Note that *TouchDown* is NOT fired as this is NOT a new object!).
4. The object is moved from UpperRight to LowerRight.
   a. The *TouchLeave* event is fired inside UpperRight.
   b. The *TouchEnter* event is fired inside LowerRight.
5. The object is removed from the LowerRight area. I.e. you removed your finger.
   a. The *TouchLeave* event is fired in the LowerRight area.
   b. The *TouchUp* event is fired in the LowerRight area.
6. No objects are tracked anymore.

WPF Application Example for Touch Handling Demonstration

System Requirements

- Windows 8, or higher
- Visual Studio 2019
- .Net Core

Download the Example WPF Application, here.

Introduction

The WPF Touch Device Handling Application both reads and separates *TOUCH*- from *MOUSE events*. The application contains of 6 buttons that triggers from different touch- or mouse events. A text is printed in the lower left corner to indicate when a button has been fired, and by what event. The application can be used to test out the effects of the different touch events. The application code is available for reference, which is created in the framework WPF.

To use a Command to handle touches, one must add the NuGet package "Microsoft.Xaml.Behaviors.Wpf", which has been done in this project already.

Event Button Description

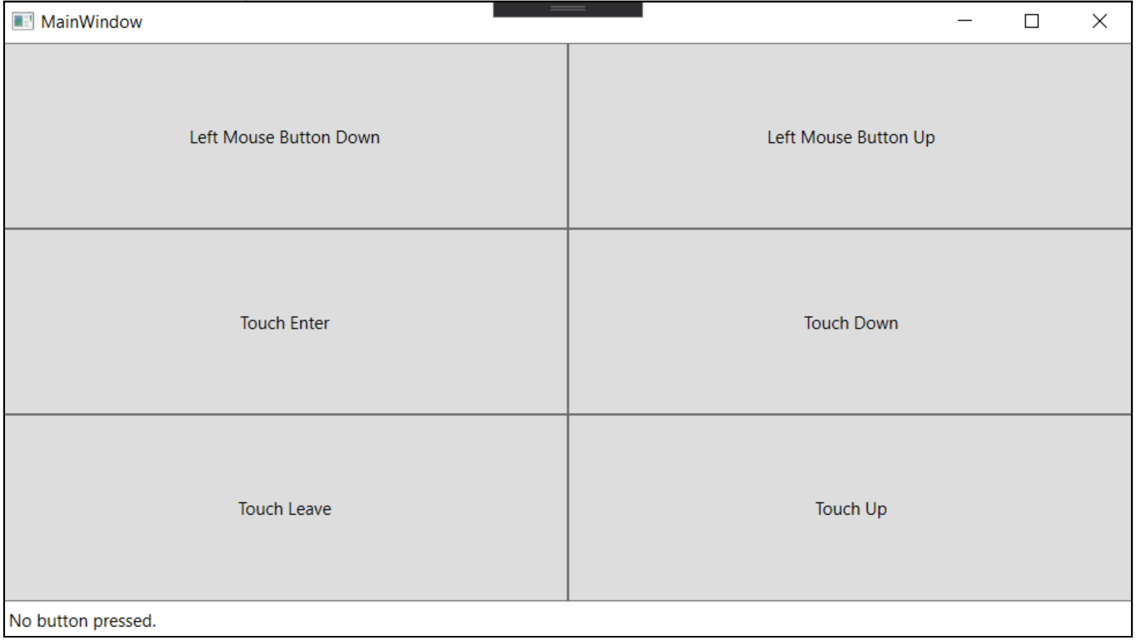
In the application, the 6 buttons will each represent an area that different touch and mouse events reacts to. The buttons will trigger to the following actions.

Button Name	Action
Left Mouse Button Down	Triggers from *MOUSE* event, from pressing down left mouse button.
Left Mouse Button Up	Triggers from *MOUSE* event, from releasing left mouse button.
Touch Enter	Triggers from *TOUCH* event, from entering the event area. Can be either from the side, or a new object registering.
Touch Down	Triggers from *TOUCH* event, when a new touch object enters the event area. Note: Only fired on new events. If a new object enters straight into this area (button in this case), both *TouchEnter* and *TouchDown* are triggered.
Touch Leave	Triggers from *TOUCH* event, when a touch object leaves the event area. Both when it is no longer tracked at all, and when it leaves the boundaries of the button.
Touch Up	Triggers from *TOUCH* event, when a touch object leaves the area and is no longer tracked at all.

How to run the Application

1. Download the example application.
2. Open the solution in Visual Studio 2019.
    a. After opening the solution the first time, build the solution once, or it will show an error: "The name "*MainViewModel*" does not exist in the namespace "*clr-namespace:WpfTouchDemo.ViewModels;assembly=WpfTouchDemo.ViewModels*".
    After building the first time it is able to resolve the name.
3. Connect the Touch sensor Module to the computer.
4. Run the program (F6).

a. When you run the program, the application will start, and 6 buttons will appear.



No button pressed.

b. Maximize the application window

5. Test pressing the buttons using both a mouse device, and the Touch Sensor Module.
    a. read the latest triggered event response in the lower left corner, and

Binding Description

The input events are bound in the following two files, and those are.

- MainWindow.xaml - Make sure to bind the button commands correctly here.
- MainViewModel.cs - All code that manage button presses is here.

The trigger event *TouchEnterButtonCommand* for example, is bound using the following method.

MainViewModel.xaml

```
 <behaviors:InvokeCommandAction Command="{Binding TouchEnterButtonCommand}" /> //
Binding TouchEnterButtonCommand
```

MainViewModel.cs. TouchEnterButtonCommand : Command

```
//Bind TouchEnterButtonCommand
this.TouchEnterButtonCommand = new Command ((parameter) => { this.SelectedButtonLabel
= "Touch Enter."; }, () => true);
```

Where *TouchEnterButtonCommand* is bound to the button "Touch Enter", as such:

Section from MainViewModel.xaml

```
    // Touch Enter handling using Command.
        <Button Name="TouchEnterButton" Grid.Column="0" Grid.Row="1" Content="Touch
Enter">
            <behaviors:Interaction.Triggers>
                <behaviors:EventTrigger EventName="TouchEnter">
                    <behaviors:InvokeCommandAction Command="{Binding
TouchEnterButtonCommand}" /> //Binding TouchEnterButtonCommand
                </behaviors:EventTrigger>
            </behaviors:Interaction.Triggers>
        </Button>
```

Touch Evaluation Process

Hence, the Touch Sensor Module both sends touch events, as well as mouse events (translated from touch events). The application should however only process one. If it is not taken to consideration, the application would presumably react on both event input.

> ⚠ Mouse events can both be from a mouse device, or promoted from a touch device. If the application reacts open the "fake" mouse events, there might be a small delay, which is why only the first input event should be read. The first event in this case would always be a touch event.

The application distinguish touch events from the translated touch events (mouse events) by checking if the events were originally sent from a touch screen digitizer.

The following code is an example from *WPF example application,* which *c*oncludes if the application should read event- or touch data.

Section from MainViewModel.cs

```
switch ( parameter )
                {
                    case TouchEventArgs eventArgs:
                        // This is a Touch Event.
                        this.SelectedButtonLabel = "Touch Down in Touch Down Area.";
                    break;
                    case MouseEventArgs eventArgs:
                        // this is a Mouse Event. What we don't know is if it is
promoted from a Touch Event or from a "real mouse".
                        if ( eventArgs.StylusDevice == null )
                        {
                            // This is from a mouse, i.e. not promoted from a Touch
Event. We ignore it if it was promoted, or we get
                            // two clicks.
                            this.SelectedButtonLabel = "Left Mouse Button Down in
Touch Down Area.";
                        }
                    break;
                }
```

10 Specifications

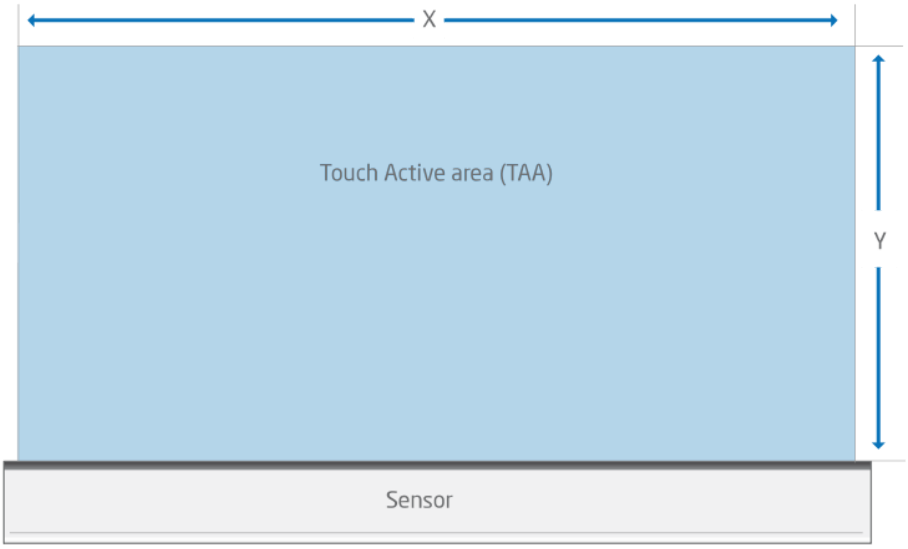
10.1 Specifications Summary

10.1.1 Touch Performance Specification

Item	Specification
Input methods	Finger, hand or glove.
Minimum object size (diameter)	5 mm
Number of touch objects	1, 2, or more, depending on application
Touch resolution	0.1 mm
Touch activation force	0 N (no activation force required)
Touch Active Area	Up to 345.6 x 327.7 mm. For details, refer to information on product variants in Introduction (see page 6).
Response time	~50 ms (initial touch, at 33 Hz in idle mode) 10 ms (continuous tracking, at 100 Hz in active mode)
Scanning frequency	Configurable up to 900 Hz, depending on product variant. For details, refer to Touch Performance (see page 139).

Touch accuracy

The specified values are valid for the used test setup. For more information, refer to Performance Test Methods (see page 149). The touch accuracy is measured inside the TAA, using a silicone based cylindrical test rod with a diameter of 16 mm.



Touch Accuracy for Normal Range Sensor Modules of the 90° **and 0° Types**

Product Number	**Typical Value (mm)**	**μ ± 2σ (mm)**
NNAMC346XPC01	1.5	3.5
NNAMC310XPC01	1.5	3.5
NNAMC295XPC01	1.5	3.5
NNAMC230XPC01	1.5	3.5
NNAMC209XPC01	1.5	3.5
NNAMC180XPC01	1.5	3.5
NNAMC158XPC01	1.5	3.5
NNAMC122XPC01	1.5	3.5
NNAMC115XPC01	1.5	4

Typical Value: The accuracy on average, within the TAA.
μ ± 2σ: 95% of reported touch positions deviate less than this value. (2σ standard deviation).
Product number: "X" indicates if the sensor module is of type 0º ("0") or 90º ("1").

The accuracy specification (normal range, 0° and 90°) is valid for units produced from 15th January 2020. Please contact our support team for specification regarding earlier produced sensor modules, or general questions about the accuracy specification.

Touch Accuracy for Extended Range Sensor Modules of the 90**° and 0° Types**

Product Number	Typical Value (mm)	μ ± 2σ (mm)
NNAMC346XPC01	2.5 [1]	5 [1]

Typical Value: The accuracy on average, within the TAA.
μ ± 2σ: 95% of reported touch positions deviate less than this value. (2σ standard deviation).
Product number: "X" indicates if the sensor module is of type 0º ("0") or 90º ("1").
[1] Preliminary value.

The accuracy specification (extended range, 0° and 90°) is valid for units produced from 15th January 2020. Please contact our support team for specification regarding earlier produced sensor modules, or general questions about the accuracy specification.

Technical Specification

Item	Sensor module Variant	Specification
Module size (LxHxW)	0° Type	L x 3.46 x 14.5 mm L depending on product variant.
	90° Type	L x 3.46 x 16.05 mm L depending on product variant.
Power consumption I2C interface Active mode (100 Hz)	NNAMC0720PC01, NNAMC0721PC01	57 mW
	NNAMC2090PC01, NNAMC2091PC01	80 mW
	NNAMC3460PC01, NNAMC3461PC01	104 mW
	NNAMC3460PC01, NNAMC3461PC01, Extended Range	135 mW
Power consumption I2C interface Idle mode (33 Hz)	NNAMC0720PC01, NNAMC0721PC01	44 mW
	NNAMC2090PC01, NNAMC2091PC01	45 mW
	NNAMC3460PC01, NNAMC3461PC01	47 mW

Item	Sensor module Variant	Specification
	NNAMC3460PC01, NNAMC3461PC01, Extended Range	61 mW

10.2 Touch Performance

10.2.1 Touch Object Requirement

The Neonode Touch Sensor Module detect and trace objects by detecting diffusely reflected infrared light.

Requirements on the object to detect include:

- A minimum reflectance of 30% in the near IR-spectrum is needed for proper signal levels, that is, the object can not be too dark.
- Object surface must be diffuse. A glossy or mirror-like object may not scatter enough light towards correct receivers in order to generate a reliable detection.
- An object must be ≥ 5 mm to ensure sufficient signal levels. This is closely related to reflectance. A white, diffuse object may be smaller than a dark, glossy one.

10.2.2 Touch Accuracy

Specification

Measured touch coordinate error in X and Y axis is less or equal than the specified value for about 95% of the cases.

Touch coordinate error data is calculated by subtracting the actual position and measured position in X and Y axis.

Definition

The touch accuracy of the Touch Sensor Module can be described statistically with the normal distribution and a standard deviation of 2 sigma. This means that the touch position reported by the sensor module will deviate less than the specified value in 95% of the cases.

10.2.3 Response Time

The specification of response time reflects the reaction speed of a Touch Sensor Module.

Specification

- **Initial touch**: ~50 ms, at 33 Hz scanning frequency (default frequency in idle mode).

- **Continuous tracking**: 10 ms, at 100 Hz scanning frequency (default frequency in active mode).

Increasing the scanning frequency decreases the response time.

Definition

Initial Touch

The specified response time for the **initial touch** starts from the instant an object is presented in the sensor module's touch active area and stops when the module starts to send the first touch notification for this object. The specified response time consists of two numbers reflecting the best case and the worst case, with the average response time right in the middle.

The response time (t) can be calculated for different idle mode frequencies (f) can be calculated by the formulas below:

Best case: *t = 16 ms*

Worst case: *t = 1/f + 16 ms*

Average: *t = (1/f + 32 ms) / 2*

In touch applications, an object will be detected slightly before it reaches the touch surface, making the perceived response time shorter.

Continuous Tracking

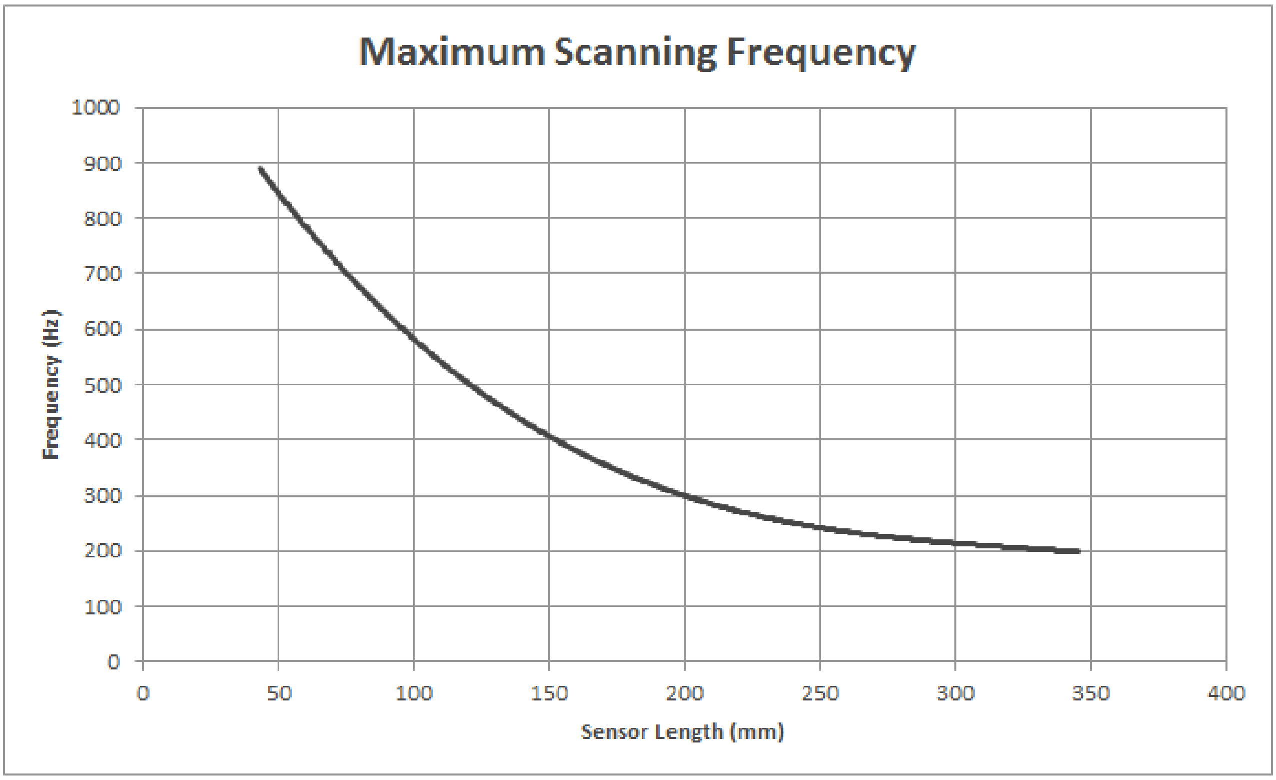
After the first touch notification, the sensor module will **continuously track** and send touch notifications to update the object position. The response time is therefore defined as the time between subsequent touch notifications.

The response time (t) can be calculated for different active mode frequencies (f) can be calculated by the formula below:

t = 1/f

10.2.4 Scanning Frequency

The scanning frequency can be set using the Neonode API. The default value is 100 Hz in active mode, that is, when an object is detected or tracked. The default value in idle mode, that is, when no object is detected or tracked, is 33 Hz. The maximum scanning frequency depends on the product variant (sensor module's length). See the following chart.

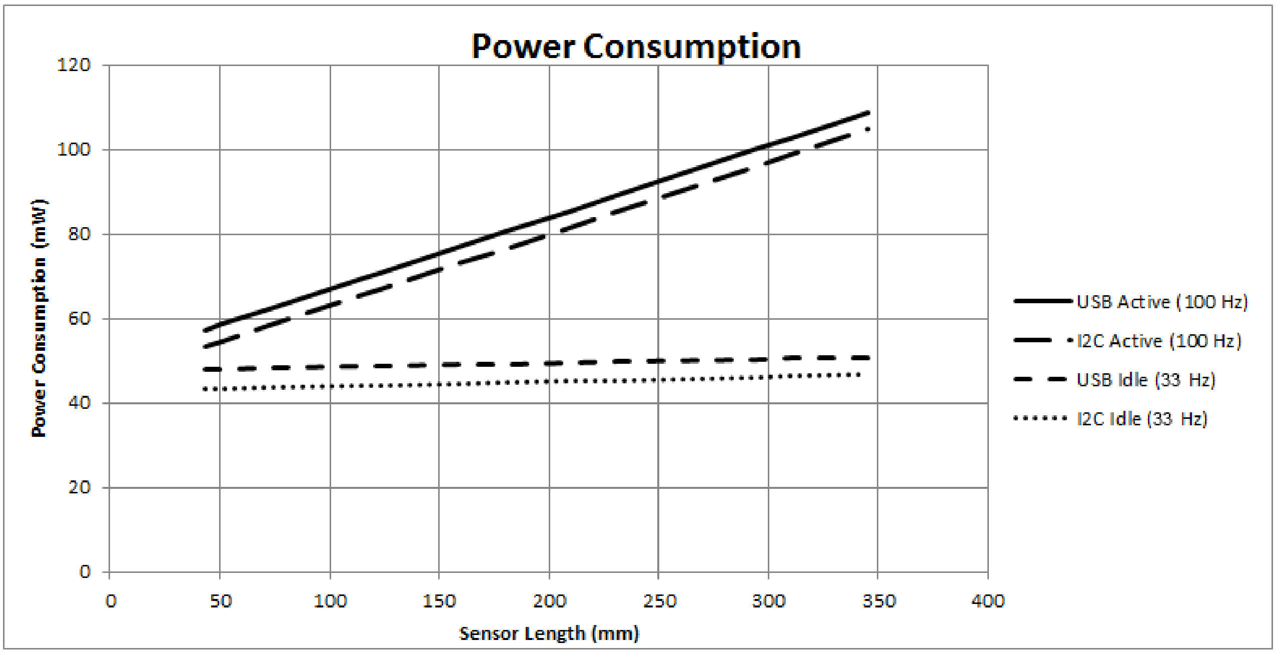


The maximum scanning frequency for product variants NNAMC3460PC01 and NNAMC3461PC01 with Extended Range is 175 Hz.

10.3 Power Consumption

10.3.1 Specification

The graph below shows the power consumption for various sensor module's lengths, in active and idle mode. In active mode, the scanning frequency is set to 100 Hz, and one object is presented in active area. In idle mode the scanning frequency is set to 33 Hz, with a clean active area. With higher scanning frequency or more detected objects, the power consumption might slightly higher than the values in the graph. The sensor module will only be in active mode when a touch object is being detected or tracked.



.

From firmware version 1.49 and higher, the sensor module types NNAMC3460PC01 and NNAMC3461PC01 are provided with Extended Range, and their power consumption increases 30% in both USB active mode and USB idle mode. The power consumption for module types shorter than 237 mm is not affected by Extended Range.

10.3.2 Definition

The power consumption is calculated from the current consumption when supplying the sensor module with 5 V.

The current consumption is, in turn, defined as the average current that goes through a sensor module. This is measured from the +5V power pin and reflects how much electric energy that is consumed by the whole sensor module. In real time, the current is not a stable value. Since the Touch Sensor Module has a low power consumption design, the processor and some peripheral circuits will switch to sleep mode during the time between two scan periods, to save power. Therefore, the current is frequently changing during run time.

According to the different working modes of the Touch Sensor Module, the current consumption value also changes between Active mode and Idle mode.

10.4 Environmental Requirements

10.4.1 Operating and Storage Conditions

Condition	Operation	Storage
Temperature	−20°C to +65°C	−40°C to +85°C
Humidity	5% to 95%	0% to 95%

Condition	Operation	Storage
Altitude	≤5000 m	≤15 km

10.4.2 ESD rating

EN55024
(61000-4-2)
Direct contact discharge: 4 kV
Indirect contact discharge: 4 kV
Air discharge: 8 kV

10.4.3 Agency Approvals

RoHS, IEC60825-1 Class 1

10.5 Electrical Requirements

10.5.1 Absolute Maximum Ratings

Parameter	Max Rating	Unit
Supply voltage	-0.3 to 6.0	V
Input voltage on I/O pins	-0.3 to 5.5	V

10.5.2 Recommended Operating Conditions

Parameter	Min	Typ	Max	Unit
Supply voltage	4.50	5.00	5.50	V

10.6 Optical Requirements on External Window

Most applications will require an outer cover window, for design cosmetics and protection against dust and humidity.

The optical properties on cover windows placed in front of the Touch Sensor Module are essential in order to maintain a high touch performance. If light is lost, scattered or diverted it will lead to shorter detection range and lower touch accuracy.

10.6.1 Optical Requirements

Window material must be optically clear, without absorption and have optical quality surfaces.

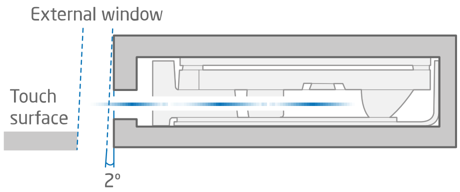
- Transmission: > **88 % at 945nm**
- Haze: **< 3%**
- Surface finish: **SP1-A2 (max Ra 0.05μm).**

Proven plastic materials include optical grade acrylic (PMMA) and polycarbonate. For glass windows, transmission at 945 nm must be verified. Many borosilicate glasses (such as Borofloat) work well, but some common window glasses show substantial absorption due to high iron content.

10.6.2 Geometrical Constraints

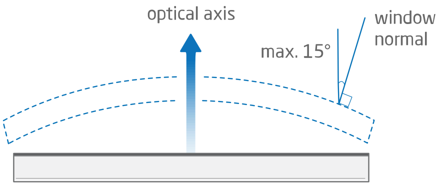
The Touch Sensor Module is an optical system that both emits and receives IR-light at different incident angles. When the light hits a transparent material, most of the light is transmitted through the material and exit on the other side. But in reality the amount of light being transmitted is angle dependent, why some shape constraints exist on windows placed in front of the sensor module:

- **Window surfaces must be parallel.**
  A wedge, or lens shaped window will shift light beams out of the active area.
- It is a good practice to install the window at a slight angle (~2°) to reduce reflected stray light. See the image below. The angle can be up to approximately 30° without affecting performance.

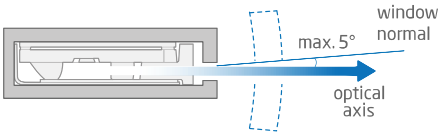


- A slight curvature on the window can be allowed.

- In x-direction, a maximum angle of 15° between window normal and sensor module's optical axis is recommended, for all parts of the window within the sensor module's TAA.



- In z-direction, the angle should be maximum 5°.



, which corresponds to a minimum radius of 12 mm for the surface closest to the sensor module.

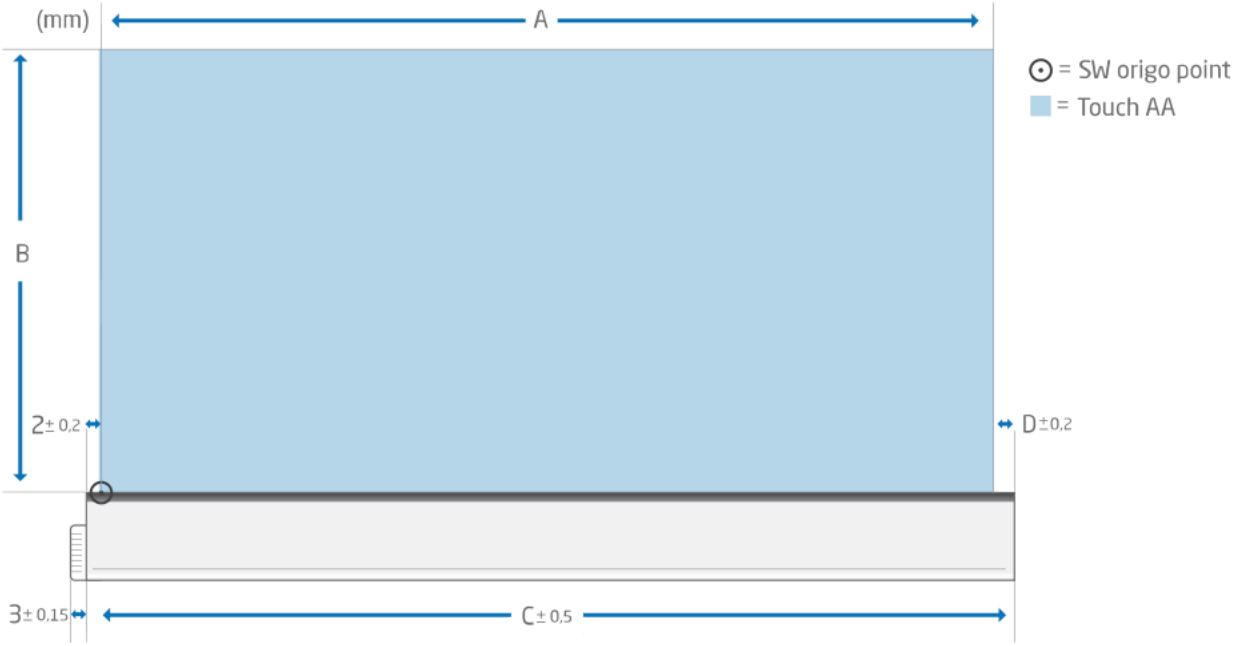
- Keep window thickness as small as mechanically feasible, to reduce absorption losses.

10.7 Mechanical Data

10.7.1 Physical Dimensions and Position of Origin

Top View

Dimensions **C** and **D** vary between the Touch Sensor Module types (0° and 90°) and therefore also the Touch Active Area (TAA) sizes (**A** and **B**). For Touch Sensor Module types with A ≥ 237.6 mm, dimension B also depends on the installed firmware version.



Product number		Measurements (mm)			
0°	**90°**	**A**	**B**	**C**	**D**
NNAMC0430PC01	NNAMC0431PC01	43.2	14.9	47.2	2
NNAMC0500PC01	NNAMC0501PC01	50.4	29.8	55.9	3.5
NNAMC0580PC01	NNAMC0581PC01	57.6	29.8	61.6	2
NNAMC0640PC01	NNAMC0641PC01	64.8	44.7	70.3	3.5
NNAMC0720PC01	NNAMC0721PC01	72	44.7	76	2

Product number		**Measurements (mm)**			
0°	**90°**	**A**	**B**	**C**	**D**
NNAMC0790PC01	NNAMC0791PC01	79.2	59.6	84.7	3.5
NNAMC0860PC01	NNAMC0861PC01	86.4	59.6	90.4	2
NNAMC0940PC01	NNAMC0941PC01	93.6	74.5	99.1	3.5
NNAMC1010PC01	NNAMC1011PC01	100.8	74.5	104.8	2
NNAMC1080PC01	NNAMC1081PC01	108	89.4	113.5	3.5
NNAMC1150PC01	NNAMC1151PC01	115.2	89.4	119.2	2
NNAMC1220PC01	NNAMC1221PC01	122.4	104.3	127.9	3.5
NNAMC1300PC01	NNAMC1301PC01	129.6	104.3	133.6	2
NNAMC1370PC01	NNAMC1371PC01	136.8	119.2	142.3	3.5
NNAMC1440PC01	NNAMC1441PC01	144	119.2	148	2
NNAMC1510PC01	NNAMC1511PC01	151.2	134.0	156.7	3.5
NNAMC1580PC01	NNAMC1581PC01	158.4	134.0	162.4	2
NNAMC1660PC01	NNAMC1661PC01	165.6	148.9	171.1	3.5
NNAMC1730PC01	NNAMC1731PC01	172.8	148.9	176.8	2
NNAMC1800PC01	NNAMC1801PC01	180	163.8	185.5	3.5
NNAMC1870PC01	NNAMC1871PC01	187.2	163.8	191.2	2
NNAMC1940PC01	NNAMC1941PC01	194.4	178.7	199.9	3.5
NNAMC2020PC01	NNAMC2021PC01	201.6	178.7	205.6	2
NNAMC2090PC01	NNAMC2091PC01	208.8	193.6	214.3	3.5
NNAMC2160PC01	NNAMC2161PC01	216	193.6	220	2
NNAMC2230PC01	NNAMC2231PC01	223.2	208.5	228.7	3.5
NNAMC2300PC01	NNAMC2301PC01	230.4	208.5	234.4	2

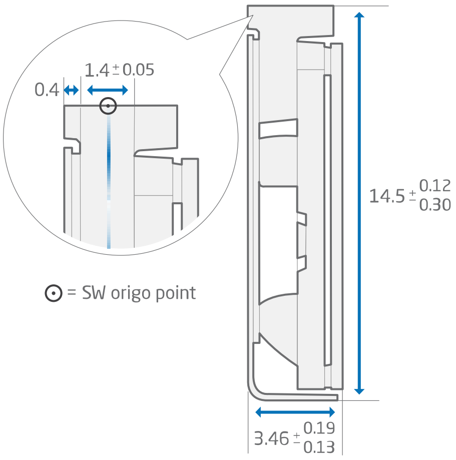
Product number		Measurements, Non-Extended Range (mm)					Measurements, Extended Range (mm)				
0°	90°	A	B	C	D	From Firmware Version	A	B	C	D	From Firmware Version
NNAMC2380PC01	NNAMC2381PC01	237.6	208.5	243.1	3.5	v1.38	237.6	223.4	243.1	3.5	Available on request
NNAMC2450PC01	NNAMC2451PC01	244.8	208.5	248.8	2	v1.38	244.8	223.4	248.8	2	Available on request
NNAMC2520PC01	NNAMC2521PC01	252	208.5	257.5	3.5	v1.38	252	238.3	257.5	3.5	Available on request
NNAMC2590PC01	NNAMC2591PC01	259.2	208.5	263.2	2	v1.38	259.2	238.3	263.2	2	Available on request
NNAMC2660PC01	NNAMC2661PC01	266.4	208.5	271.9	3.5	v1.38	266.4	253.2	271.9	3.5	Available on request
NNAMC2740PC01	NNAMC2741PC01	273.6	208.5	277.6	2	v1.38	273.6	253.2	277.6	2	Available on request
NNAMC2810PC01	NNAMC2811PC01	280.8	208.5	286.3	3.5	v1.38	280.8	268.1	286.3	3.5	Available on request
NNAMC2880PC01	NNAMC2881PC01	288	208.5	292	2	v1.38	288	268.1	292	2	Available on request
NNAMC2950PC01	NNAMC2951PC01	295.2	208.5	300.7	3.5	v1.38	295.2	283.0	300.7	3.5	Available on request
NNAMC3020PC01	NNAMC3021PC01	302.4	208.5	306.4	2	v1.38	302.4	283.0	306.4	2	Available on request
NNAMC3100PC01	NNAMC3101PC01	309.6	208.5	315.1	3.5	v1.38	309.6	297.9	315.1	3.5	Available on request
NNAMC3170PC01	NNAMC3171PC01	316.8	208.5	320.8	2	v1.38	316.8	297.9	320.8	2	Available on request
NNAMC3240PC01	NNAMC3241PC01	324	208.5	329.5	3.5	v1.38	324	312.8	329.5	3.5	Available on request
NNAMC3310PC01	NNAMC3311PC01	331.2	208.5	335.2	2	v1.38	331.2	312.8	335.2	2	Available on request

Product number		Measurements, Non-Extended Range (mm)					Measurements, Extended Range (mm)				
0°	90°	A	B	C	D	From Firmware Version	A	B	C	D	From Firmware Version
NNAMC3380PC01	NNAMC3381PC01	338.4	208.5	343.9	3.5	v1.38	338.4	327.7	343.9	3.5	Available on request
NNAMC3460PC01	NNAMC3461PC01	345.6	208.5	349.6	2	v1.38	345.6	327.7	349.6	2	v1.49 Extended Range

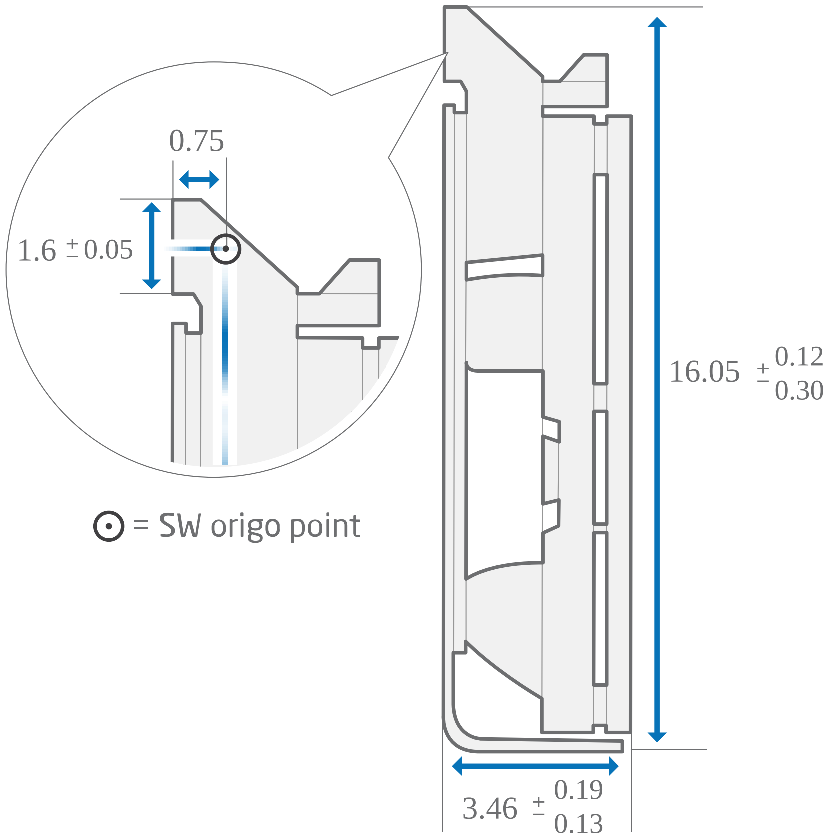
Side View

These measurements are identical for all sensor module's lenghts but vary some between the 0° and 90 ° types. The position of origin is marked with "zero software".

0° Type



90 ° Type



10.7.2 Packaging

Please refer to Packaging Blueprints in Downloads[59]for further information.

10.8 Test Specifications and Definitions

10.8.1 Performance Test Methods

For specifications for the performance test methods, contact Neonode Support[60].

[59] https://support.neonode.com/docs/display/Downloads/Packaging+Blueprints
[60] https://helpcenter.neonode.com/hc/en-us/requests/new

10.8.2 Reliability Test

Reliability Test Report

The test report can be found here[61].

Reliabilty Test Specification Overview

TEST ITEM	TEST CONDITION	QUANTITY
Neonode Touch Sensor Module:		
Low temperature operation test	Ta=-20°C, 96 hrs, Powered On	3
Low temperature storage test	Ta= -40°C, 96 hrs, Power Off	3
Temperature cycling operation test	Ta: 0°C / 65°C, RH: 60% Cycle time: 60 min, Duration: 20 minutes at each temperature extreme with a 10 minute transition time. 240 cycles in total ; Power On, Functionality test every 24 hours.	3
Thermal shock	Ta: +85°C to -40°C Humidity: Off Duration: 25 minutes at each temperature extreme with a 5 minute transition time. 120 cycles in total Operation mode: The DUT is powered off during testing.	3
Condensation Test	Ta: +25ºC to +65ºC RH: 95% 50 cycles, cycle time:30 + 30 min ; Power Off	3
Vibration test	Wave form : random Vibration level : 1.0 GRMS Bandwidth : 10-300 Hz Cycle: X,Y,Z, 60 min each Number of cycles: 4 DUT on vibration table: Clamped on 3 positions along sensor module.	3

[61] https://support.neonode.com/docs/display/Downloads/PDF+Documents

Shock test	Shock level : 50 G Waveform : half sine wave, 2 msec Direction :±X, ±Y, ±Z One time each direction	3
Pressure Intensity	Pressure on surface with a flat ø 10 mm rod with a force of 10 N. The force should be applied at the center of the PCB. Test for 168 hrs. Power Off	3
Contact wear test	Wave form : random Vibration level : 1.0 GRMS Bandwidth : 10-300 Hz Cycle: X,Y,Z, 60 min Number of cycles: 1 One time each direction DUT on vibration table: Clamped on 3 positions along sensor module. The FPC is clamped to the table at one position (center). The FFC-connector will not be clamped to the table.	3
Cleansing test	Rubbed 5 times using a piece of cotton soaked with 2-5 ml (depending on size of object). Apply the solutions below in following order on the same place of the object: o Denatured alcohol (ethanol 99,5%). o All purpose cleaning solution containing ammonia (e.g. AJAX Tornado). o Water.	3
ESD	EN55024 (61000-4-2) Direct contact discharge: 2,4,8 kV Indirect contact discharge: 2,4,8 kV Air discharge: 4,8,16 kV	3