

## List of public functions of the ADXL345\_WE library

Function	Parameters	what it does
<code>bool init( )</code>	none	Initiates the ADXL345 with some default register values. Returns connection status (I2C and SPI).
<code>void setSPIClockSpeed( <i>clockSpeed</i> )</code>	SPI clock speed as unsigned long int.	Sets the SPI clock speed.
<code>void setCorrFactors ( <i>min/max values</i> )</code>	xMin, xMax, yMin, yMax, zMin, zMax (all float)	Recalculation of (delta g / delta raw) and the offset of each axis.
<code>bool setDataRate( <i>adxl345_dataRate</i> )</code>	ADXL345_DATA_RATE_3200 ..... ADXL345_DATA_RATE_0_10	Sets the data rate. You can choose from 16 values between 3200 Hz and 0.1 Hz. Returns connection status (only I2C).
<code>adxl345_dataRate getDataRate( )</code>	none	Returns the data rate as <code>adxl345_dataRate</code> . Look into ADXL345_WE.h for definition.
<code>String getDataRateAsString( )</code>	none	Since you might not want to go into the details of the library this is an option to get the data rate range as a better understandable string.
<code>byte getPowerCtlReg( )</code>	none	This is a function for the more advanced users who might need the status of the Power Control Register.
<code>bool setRange( <i>range</i> )</code>	ADXL345_RANGE_16G ADXL345_RANGE_8G ADXL345_RANGE_4G ADXL345_RANGE_2G	Sets the g range. I have implemented the full resolution as default, so you have no disadvantage using 16 g range. Returns connection status (only I2C).
<code>adxl345_range getRange( )</code>	none	Returns the selected range as <code>adxl345_range</code> . Look into ADXL345_WE.h for definition.
<code>String getRangeAsString( )</code>	none	Since you might not want to go into the details of the library this is an option to get the range as a better understandable string.
<code>uint8_t getDeviceID( )</code>	none	Returns the device ID, which should be 0xE5
<code>bool isConnected( )</code>	none	Returns whether the ADXL345 is still connected. For this purpose, it tries to read the device ID. Returns connection status (I2C and SPI).
<code>bool setFullRes( <i>bool</i> )</code>	true, false	Enables / disables full resolution. If disabled, resolution is 10 bit for all ranges. If enabled, resolution increases from 10 to 13 bit going from 2 g to 16 g range. Returns connection status (only I2C).
<code>bool getRawValues( <i>xyzFloat &amp;</i> )</code>	Raw values as <code>xyzFloat</code> . <code>xyzFloat</code> is a struct which contains three floats: x,y,z.	Assigns the current raw values of the ADXL345 to the <code>xyzFloat</code> struct. Returns connection status (only I2C).
<code>bool getCorrectedRawValues( <i>xyzFloat &amp;</i> )</code>	Corrected raw values as <code>xyzFloat</code> .	Assigns the corrected raw values (calibration applied) to the <code>xyzFloat</code> . Returns connection status (only I2C).
<code>bool getGValues( <i>xyzFloat &amp;</i> )</code>	Acceleration values as <code>xyzFloat</code> .	Assigns the acceleration ("g") values to the <code>xyzFloat</code> . If calibration has been applied, it will provide data based on corrected raws. Returns connection status (only I2C).
<code>bool getAngles( <i>xyzFloat &amp;</i> )</code>	Angles as <code>xyzFloat</code> .	Assigns the angles based on a simple calculation (angle = arcsin g) to the <code>xyzFloat</code> . Okay for g values up to ~0.6. It's the basis for determining the orientation. Returns connection status (only I2C).
<code>bool getCorrectedAngles( <i>xyzFloat &amp;</i> )</code>	Corrected angles as <code>xyzFloat</code> .	Use <code>measureAngleOffsets</code> first. It applies an additional angle offset to start with angles = 0. Returns connection status (only I2C).
<code>float getPitch( )</code>	none	Quite similar to the x values of <code>getCorrectedAngles()</code> . Uses no extra angles offset, but better for higher angles. It considers also the z-axis.
<code>float getRoll( )</code>	none	Quite similar to the y values of <code>getCorrectedAngles()</code> . Uses no extra angles offset, but better for higher angles. It considers also the z-axis.
<code>void measureAngleOffsets( )</code>	none	Place your ADXL345 flat (in XY plane). The function determines the angle offset which is internally used for tilts (not for angles).
<code>xyzFloat getAngleOffsets( )</code>	none	Returns the angle offsets. You could save the offsets e.g. in the EEPROM for re-use after a reset.
<code>void setAngleOffsets( <i>offsets</i> )</code>	Angle offsets as const <code>xyzFloat</code>	Sets the angle offsets.
<code>adxl345_orientation getOrientation( )</code>	none	Returns the orientation as <code>adxl345_orientation</code> . Look into ADXL345_WE.h for definition.
<code>String getOrientationAsString( )</code>	none	Since you might not want to go into the details of the library this is an option to get the orientation as a better understandable string: z up (flat), z down, x up, x down, y up, y down
<code>bool setMeasureMode ( <i>bool</i> )</code>	true, false	If true, the ADXL345 measures at the selected data rate. If false, it goes into standby mode. Returns connection status (only I2C).
<code>bool setSleepMode( <i>bool</i>, <i>frequency</i> )</code>	true, false  ADXL345_WUP_FQ_8 ADXL345_WUP_FQ_4 ADXL345_WUP_FQ_2 ADXL345_WUP_FQ_1	Enables or disables the sleep mode and sets the wake up frequency (1 Hz, 2 Hz, 4 Hz or 8 Hz). Returns connection status (only I2C).
<code>bool setSleepMode( <i>bool</i> )</code>	true, false	Enables or disables the sleep mode without changing the wake up frequency. Returns connection status (only I2C).

<code>bool setAutoSleep( bool, frequency )</code>	true, false ADXL345_WUP_FQ_8 ADXL345_WUP_FQ_4 ADXL345_WUP_FQ_2 ADXL345_WUP_FQ_1	Enables / disables auto sleep mode. You also need to enable inactivity. The link bit must also be set, this does the library in the background. If you disable auto sleep, you'll have to delete the link bit manually if you don't want to be set. Returns connection status (only I2C).
<code>bool setAutoSleep( bool )</code>	true, false	Enables or disables the auto sleep mode without changing the wake up frequency. Returns connection status (only I2C).
<code>bool isAsleep( )</code>	none	Returns true if the ADXL345 is asleep, otherwise false.
<code>bool setLowPower( bool )</code>	true / false	En-/Disables Low Power Mode. It only provides advantages between 12.5 and 400 Hz data rates. Returns connection status (only I2C).
<code>bool isLowPower( )</code>	none	Checks whether Low Power Mode is enabled or not.
<code>bool setInterrupt( type, pin )</code>	ADXL345_OVERRUN ADXL345_WATERMARK ADXL345_FREEFALL ADXL345_INACTIVITY ADXL345_ACTIVITY ADXL345_DOUBLE_TAP ADXL345_SINGLE_TAP ADXL345_DATA_READY  INT_PIN_1, INT_PIN_2	Sets an interrupt and specifies the interrupt output pin for that interrupt. Overrun, watermark and data ready are always enabled, so you can only change the pin (default is INT1).
<code>bool setInterruptPolarity( )</code>	ADXL345_ACT_LOW ADXL345_ACT_HIGH	Sets the interrupt pins active-low or active-high. Default is active-high. Returns connection status (only I2C).
<code>bool deleteInterrupt( type )</code>	ADXL345_OVERRUN ADXL345_WATERMARK .....(see setInterrupt)	Disables the specified interrupt. Returns connection status (only I2C).
<code>uint8_t readAndClearInterrupts( )</code>	none	Returns the interrupt source register which is cleared by doing this. The return value is a byte. For the details look into the library.
<code>bool checkInterrupt( source, type )</code>	Return value of readAndClearInterrupts  ADXL345_OVERRUN ADXL345_WATERMARK .....(see setInterrupt)	Checks if the return value matches a certain interrupt type. Might be easier to use than to evaluate the return value of readAndClearInterrupt by yourself. Returns connection status (only I2C).
<code>bool setLinkBit( bool )</code>	true, false	Sets or deletes the link bit, which influences the behavior of inactivity and activity. For details look into the data sheet or try ADXL345_activity_inactivity_interrupt.ino
<code>bool setFreeFallThresholds( g value, time )</code>	threshold in g threshold in milliseconds (max 1275 ms)	Sets the parameters for the free fall detection. Minimum g and minimum time. Returns connection status (only I2C).
<code>bool setActivityParameters( mode, axes, threshold )</code>	ADXL345_DC_MODE ADXL345_AC_MODE  ADXL345_X00 ADXL345_XY0 ADXL345_0Y0 ..... ADXL345_XYZ  Threshold in g	Sets the activity parameters. In DC mode the actual accelerations is compared with the specified threshold. In AC mode the threshold is the acceleration when enabling activity. The second parameter specifies the axes to be involved. The third parameter is the minimum threshold which leads to an activity interrupt. Returns connection status (only I2C).
<code>bool setInactivityParameters( mode, axes, threshold, inact time )</code>	same as for setActivityParameters plus inact time in seconds	Sets the inactivity parameters. Same as for the activity parameters, but threshold is a maximum value for inactivity. In addition a time must be specified after which the interrupt will be triggered. Maximum is 255 seconds. Returns connection status (only I2C).
<code>bool setGeneralTapParameters( axes, threshold, duration, latent )</code>	ADXL345_X00 ADXL345_XY0 ADXL345_0Y0 ..... ADXL345_XYZ  threshold in g  duration in milliseconds (max. 159 ms)  latent in milliseconds (max 318 ms)	Sets parameters needed for both tap and double tap detection. Axes are the involved axes. Threshold is minimum threshold in g. Duration is the maximum period that acceleration must be above the threshold. Latent is the time after which a new tap can be detected. Returns connection status (only I2C).
<code>bool setAdditionalDoubleTapParameters( suppress, window )</code>	true, false  window in milliseconds (max 318 ms)	Sets additional parameters needed for double tap detection. If the suppress bit is set a second tap during latency time invalidates the first tap to become a double tap, even if another tap is detected in the window time. The window specifies the time period after the end of the latency time in which the second tap must be detected to get a double tap. Returns connection status (only I2C).
<code>uint8_t getActTapStatus( )</code>	none	Returns the axes which were involved in an activity or tap event. Look into ADXL345_WE.h for definition.

<code>String getActTapStatusAsString( )</code>	none	Returns the axes which were involved in an activity or tap event as a better understandable string.
<code>bool setFifoParameters( trigger int pin, samples )</code>	ADXL345_TRIGGER_INT_1 ADXL345_TRIGGER_INT_2  number of samples in the Fifo buffer (max 32) / in trigger mode: the number of samples which are kept in FIFO before the trigger event	The trigger int pin specifies the int pin which an interrupt will be the trigger event. It is only relevant for the trigger mode.  The number of samples specifies the size of the FIFO for fifo and stream mode. In trigger mode the specified number is kept in the FIFO and then filled up to 32.  Returns connection status (only I2C).
<code>bool setFifoMode( type )</code>	ADXL345_BYPASS ADXL345_FIFO ADXL345_STREAM ADXL345_TRIGGER	FIFO - you choose the start, ends when FIFO is full (at defined limit)  STREAM - FIFO always filled with new data, old data replaced if FIFO is full; you choose the stop  TRIGGER - FIFO always filled up to 32 samples; when the trigger event occurs only defined number of samples is kept in the FIFO and further samples are taken after the event until FIFO is full again.  ADXL345_BYPASS - no FIFO  Returns connection status (only I2C).
<code>uint8_t getFifoStatus( )</code>	none	Returns the FIFO status register. For advanced users. Look into the data sheet.
<code>bool resetTrigger( )</code>	none	Enables the ADXL345 to detect a new trigger. Only relevant for FIFO trigger mode. Returns connection status (only I2C).