List of public functions of the ADXL345_WE library

Function	Parameters	what it does
void init( )	none	Initiates the ADXL345 with some default register values.
void setCorrFactors (*min/max values*)	xMin, xMax, yMin, yMax, zMin, zMax (all float)	Recalculation of (delta g / delta raw) and the offset of each axis.
void setDataRate(*adxl345_dataRate*)	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.
adxl345_dataRate getDataRate( )	none	Returns the data rate as adxl345_dataRate. Look into ADXL345_WE.h for definition.
String getDataRateAsString( )	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.
byte getPowerCtlReg( )	none	This is a function for the more advanced users who might need the status of the Power Control Register.
void setRange(*range*)	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.
adxl345_range getRange( )	none	Returns the selected range as adxl345_range. Look into ADXL345_WE.h for definition.
String getRangeAsString( )	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.
void setFullRes(*bool*)	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.
xyzFloat getRawValues( )	none	Returns the raw values from the data registers. xyzFloat is a struct which contains three floats: x,y,z.
xyzFloat getCorrectedRawValues( )	none	Returns the corrected raw values (calibration applied). xyzFloat is a struct which contains three floats: x,y,z.
xyzFloat getGValues( )	none	Returns the g values. If calibration has been applied, it will provide data based on corrected raws. xyzFloat is a struct which contains three floats: x,y,z.
xyzFloat getAngles( )	none	Returns the angles base on a a very simple calculation (angle = arcsin g). Okay for g values up to ~0.6. It's the basis for determining the orientation. xyzFloat is a struct which contains three floats: x,y,z.
xyzFloat getCorrectedAngles( )	none	Use measureAngleOffsets first. Then getTilts will start with zero angles. It applies an additional angle offset.
float getPitch( )	none	Quite similar to the x values of getCorrectedAngles(). Uses no extra angles offset, but better for higher angles. It considers also the z-axis.
float getRoll( )	none	Quite similar to the y values of getCorrectedAngles(). Uses no extra angles offset, but better for higher angles. It considers also the z-axis.
void measureAngleOffsets( )	none	Place your ADXL345 flat (in XY plane). The function determins the angle offset which is internally used for tilts (not for angles).
adxl345_orientation getOrientation( )	none	Returns the orientation as adxl345_orientation. Look into ADXL345_WE.h for definition.
String getOrientationAsString( )	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
void setMeasureMode (bool)	true, false	If true, the ADXL345 measures at the selected data rate. If false, it goes into standby mode.
void setSleepMode(bool, frequency)	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).
void setSleepMode(bool)	true, false	Enables or disables the sleep mode without changing the wake up frequency.
void setAutoSleep(bool)	true, false	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.
bool isAsleep(bool)	none	Returns true if the ADXL345 is asleep, otherwise false.
void setLowPower(bool)	none	Enables Low Power Mode. I only provides advantages between 12.5 and 400 Hz data rates.
void setInterrupt(type, pin)	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).

void **setInterruptPolarity**( )	ADXL345_ACT_LOW ADXL345_ACT_HIGH	Sets the interrupt pins active-low or active-high. Default is active-high.
void **deleteInterrupt**(type)	ADXL345_OVERRUN ADXL345_WATERMARK ……..(see setInterrupt)	Disables the specified interrupt.
byte **readAndClearInterrupts**( )	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.
bool **checkInterrupt**(source, type)	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.
void **setLinkBit**(bool)	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
void **setFreeFallThresholds**(g value, time)	threshold in g threshold in milliseconds (max 1275 ms)	Sets the parameters for the free fall detection. Minimum g and minimum time.
void **setActivityParameters**(mode, axes, threshold)	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.
void **setInactivityParameters**(mode, axes, threshold, inact time)	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.
void **setGeneralTapParameters**(axes, threshold, duration, latent)	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.
void **setAdditionalDoubleTapParameters**(suppress, window)	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.
byte **getActTapStatus**( )	none	Returns the axes which were inolved in an activity or tap event. Look into ADXL345_WE.h for definition.
String **getActTapStatusAsString**( )	none	Returns the axes which were inolved in an activity or tap event as a better understandable string.
void **setFiFoParameters**(trigger int pin, samples)	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.
void **setFifoMode**(type)	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
byte **getFifoStatus**( )	none	Returns the FIFO status register. For advanced users. Look into the data sheet.
void **resetTrigger**( )	none	Enables the ADXL345 to detect a new trigger. Only relevant for FIFO trigger mode.