List of public functions of the MPU9250_WE library

Function	Parameters	what it does
bool init( )	none	Init() first resets and then initiates the MPU9250 with some default register values. Returns true if the MPU9250 has responded.
void autoOffsets ( )	none	Measures acceleration and gyroscope values and calculates offset values. The MPU9250 should be positioned flat in its xy-plane.
void setAccOffsets (min/max values)	xMin, xMax, yMin, yMax, zMin, zMax (all float)	A more accurate method to set offsets. You need to determine the min/max raw acceleration values for the axes manually (2g range). Use this function *or* autoOffset().
void setGyrOffsets (x,y,z-Offsets)	xOffset, yOffset, zOffset	A method to set gyroscope offsets. You need to determine the raw gyroscope values, when only gravity acts on the sensor. Use this function *or* autoOffset()
void setGyrDLPF(DLPF-level)	MPU9250_DLPF_0 ..... MPU9250_DLPF_7	Sets the digital low pass filter to reduce noise. You can choose from 8 levels. You find more information in the example sketches.
void setSampleRateDivider(divider)	0...255	Divides the sample rate by (1+divider). It can only be applied if the corresponding DLPF is enabled and 0 < DLPF < 7.
void setGyrRange(range)	MPU9250_GYRO_RANGE_250, MPU9250_GYRO_RANGE_500, MPU9250_GYRO_RANGE_1000, MPU9250_GYRO_RANGE_2000	Sets the gyroscope range in degrees / second. The higher the range, the lower is the resolution. Default is 250.
void enableGyrDLPF( )	none	Enables the digital low pass filter for the gyroscope. The DLPF level needs to be set with setGyrDLPF().
void disableGyrDLPF(bandwidth)	MPU9250_BW_WO_DLPF_3600 or MPU9250_BW_WO_DLPF_8800	If you disable the DLPF, you need to choose the bandwidth which is 3600 or 8800 Hz. You find more information in the example sketches.
void setAccRange(range)	MPU9250_ACC_RANGE_2G, MPU9250_ACC_RANGE_4G, MPU9250_ACC_RANGE_8G, MPU9250_ACC_RANGE_16G	Sets the range for the accelerometer in g. You can set it to +/2, +/-4, +/-8 or +/- 16 g. The higher the range, the lower is the resolution. Default is 2g.
void enableAccDLPF(true/false)	true / false	Enables or disables the digital low pass filter.
void setAccDLPF(level)	MPU9250_DLPF_0 ..... MPU9250_DLPF_7	Sets the digital low pass filter to reduce noise. You can choose from 8 levels. You find more information in the sketches.
void setLowPowerAccDataRate(rate)	MPU9250_LP_ACC_ODR_0_24 ... MPU9250_LP_ACC_ODR_500	Sets the accelerometer output data rate in low power mode, which you enable with enableCycle(true). You can choose from 12 data rates, which are between 0.24 and 500 Hz.
void enableAccAxes(axes)	e.g.: MPU9250_ENABLE_X0Z	Enables/Disables axes for acceleration measurement. Example: MPU9250_ENABLE_X0Z means: x-axis and z-axis enabled, y-axis disabled. Default: all axes are enabled.
void enableGyrAxes(axes)	e.g.: MPU9250_ENABLE_X0Z	Enables/Disables axes for gyroscope measurements.
xyzFloat getAccRawValues( )	none	Returns a set (x,y,z) of raw acceleration values. xyzFloat is a struct which consists of three floats: x,y,z.
xyzFloat getCorrectedAccRawValues( )	none	Returns the "calibrated" raw values for acceleration.
xyzFloat getGValues( )	none	Returns g values which are based on the corrected raw acceleration values.
xyzFloat getAccRawValuesFromFifo( )	none	Returns acceleration raw values (one set of x,y,z values) from the Fifo.
xyzFloat getCorrectedAccRawValuesFromFifo( )	none	Returns corrected (calibrated) raw values from the Fifo.
xyzFloat getGValuesFromFifo( )	none	Return values from the Fifo as g values. These are calculated from the corrected raws.
float getResultantG(xyzFloat g-value)	g values as xyzFloat	Returns the resulting g value of the three axes (sum of the vectors which is not the sum of the x,y,z g values). If only gravity acts on the MPU9250, it should always return 1 g.
float getTemperature( )	none	Returns the temperature measured by the temperature sensor of the MPU9250.
xyzFloat getGyrRawValues( )	none	Returns the raw gyroscope values. xyzFloat is a struct which consists of three floats: x,y,z.
xyzFloat getCorrectedGyrRawValues( )	none	Returns the calibrated raw gyroscope values.
xyzFloat getGyrValues( )	none	Returns gyroscope values in degrees/second. Based on calibrated raws.
xyzFloat getGyrValuesFromFifo ( )	none	Returns gyroscope values (one set of x,y,z values) from the Fifo in degrees/second. Based on calibrated raws.
xyzFloat getMagValues( )	none	Returns the magnetic flux density for the x,y and z-axis in µTesla.
xyzFloat getAngles( )	none	Returns the angles of the x,y and z axis vs. the horizontal. It only works if only gravity acts on the MPU9250. The method works well below 60°, then the deviations increase. It's just the arcsin of the g values of the axes.
MPU9250_orientation getOrientation( )	none	Returns the axis with the highest positive acceleration. The return value is an enum called MPU9250_orientation. For "translation" have a look into MPU9250.h.
String getOrientationAsString( )	none	This function also returns the orientation, but - better to understand - as a string: "x up", "x down", "y up", "y down", "z up" or "z down".

float getPitch( )	none	Returns the pitch tilt angle. The calculation is based on x, y and z and therefore better at higher angles than the getAngles method. The latter has a higher precision at small angles.
float getRoll( )	none	Returns the roll tilt angle. Otherwise same comments as for getPitch.
void sleep(true/false)	true / false	Enables / disables sleep mode. After disabling it takes some time before you will measure correct acceleration and gyroscope values (depends on mode / DLPF).
void enableCycle(true/false)	true / false	Enables / disables the low power mode. The MPU9250 toggles between active and sleep mode. The frequency depends on the low power data rate. Don't use the cycle mode to obtain gyrocope data.
void enableGyroStandby(true/false)	true / false	Enables/disables a lower power mode for the gyroscope with a shorter wake up time, i.e. when disabled you will need less time until you can measure accurate gyroscope data compared to the sleep mode.
void setIntPinPolarity(polarity)	MPU9250_ACT_HIGH, MPU9250_ACT_LOW	Sets the interrupt pin polarity active-high (default) or active-low.
void enableIntLatch(true/false)	true, false	If latch is enabled the interrupt pin level is held until the interrupt status is cleared. If latch is disabled the interrupt pulse is ~50µs (default).
void enableClearIntByAnyRead(true/false)	true, false	The interrupt can be cleared by any read (true) or it will only be cleared if the interrupt status register is read (false = default).
void enableInterrupt(type)	MPU9250_DATA_READY MPU9250_FIFO_OVF MPU9250_WOM_INT	Enables an interrupt type. The library has implemented three types: new data is ready to be read, fifo overflow or wake-on-motion interrupt. The latter is triggered by acceleration data which exceeds a defined threshold. If you want to enable more than one interrupt type, then call the function several times.
void disableInterrupt(type)	MPU9250_DATA_READY MPU9250_FIFO_OVF MPU9250_WOM_INT	Should be self-explaining (see also enableInterrupt)
bool checkInterrupt(source, type)	source (MPU9250_intType), interrupt type	If an interrupt occurred you might want to check if it was data ready, fifo overflow or wake on-motion. readAndClearInterrupts() returns the source, but as an enum: MPU9250_intType. Either you look up in MPU9250_WE.h how it is defined or you check with this function. The disadvantage is that you need to check one by one.
uint8_t readAndClearInterrupts( )	none	Returns which interrupt occurred asMPU9250_intType and clears the interrupt.
void setWakeOnMotionThreshold(thresh)	threshold (1…255)	Sets the threshold for the wake-on-motion interrupt. The LSB is 4 mg (= milli-g), i.e. 1 equals 4 mg, 255 equals 1020 mg.
void enableWakeOnMotion(wom_en, mode)	MPU9250_WOM_DISABLE/ MPU9250_WOM_ENABLE, MPU9250_WOM_COMP_DISABLE/ MPU9250_WOM_COMP_ENABLE	Enables/disables the wake-on-motion interrupt. Enables/disables the compare mode. In compare mode the current acceleration value is compared with the last measured value. If compare mode is disabled the baseline is the starting value, when the WOM interrupt was enabled.
void startFifo(type)	MPU9250_FIFO_ACC, MPU9250_FIFO_GYR, MPU9250_FIFO_ACC_GYR	If called, the MPU9250 starts writing data into the Fifo. Fifo must be enabled before. I have implemented three options: you can write acceleration data (max 85 x,y,z sample sets), gyroscope data (max 85 x,y,z sample sets) or acceleration and gyroscope data (max 42 sets) into the Fifo.
void stopFifo( )	none	Stops writing data into the Fifo.
void enableFifo(true/false)	true / false	Enables / disables the Fifo function.
void resetFifo( )	none	Sets the Fifo counter to zero.
int16_t getFifoCount( )	none	Returns the number of byte in the Fifo. Maximum is 512 Byte.
void setFifoMode(mode)	MPU9250_CONTINUOUS, MPU9250_STOP_WHEN_FULL	Sets continuous or stop-when-full mode. In continuous mode new data is continuously written into the Fifo. If full, the oldest data is replaced by new data. In the other mode, no new data is written to the Fifo is full.
int16_t getNumberOfFifoDataSets( )	none	Returns the number of complete data sets in the Fifo. E.g. a complete set of acceleration and gyroscope data consists of 2 (acc & gyro) x 3 (x,y,z) x 2 (2 bytes) = 12 bytes. If Fifo is full it contains 512 byte -> 512 / 12 = 42 complete sets, rest is 8.
void findFifoBegin( )	none	In the stop-when-full mode the Fifo will start at the beginning of a set. The last set will be incomplete. In the continuous it will end with a complete set. That means in continuous mode you (or the library) has to calculate at which byte (fifo count) the first complete set starts.
bool initMagnetometer( )	none	Initiates the magnetometer and reads adjustment factors from the ROM (which have been added there by the manufacturer).
void setMagOpMode(mode)	AK8963_PWR_DOWN, AK8963_TRIGGER_MODE, AK8963_CONT_MODE_8HZ, AK8963_CONT_MODE_100HZ	Sets the operational mode for the magnetometer (AK8963): power down, triggered mode (saves power vs. continuous mode), continuous mode at 8 Hz or 100 Hz.
bool isMagOverflow( )	none	Returns if magnetometer measures out of range.
uint8_t getStatus2Register( )	none	After you read measured data, you need to complete the read process by reading the status register 2. The library does this automatically.
void startMagMeasurement( )	none	Starts a magnetometer measurement in triggered mode and waits until the data is available.
bool isMagDataReady( )	none	Check if magnetometer data is ready to be read.