

## List of commands (public functions) of the INA226\_WE library

Function	Parameters	what it does
<code>void Init( )</code>	none	initiates the INA226 with some default register values
<code>void reset_INA226( )</code>	none	reset of the device
<code>void getI2cErrorCode( )</code>	none	returns the current error code from endTransmission(); 0 = success.
<code>void setCorrectionFactor( <i>factor</i> )</code>	factor (float)	if INA226 current values differ from currents measured with calibrated equipment, you can apply a factor
<code>void setAverage( <i>mode</i> )</code>	AVERAGE_X  X = 1, 4, 16, 64, 128, 256, 512, 1024	sets the number of samples that are averaged for one measurement
<code>void setConversionTime( <i>time</i> )</code>	CONV_TIME_X  X = 140, 204, 332, 588, 1100, 2116, 4156, 8244	sets time for conversion for shunt and bus voltage in microseconds
<code>void setMeasureMode( <i>mode</i> )</code>	CONTINUOUS, TRIGGERED, POWER_DOWN	sets the mode; for POWER_DOWN please use the powerDown function since it remembers the mode before power-down
<code>void setResistorRange( <i>resistorValue</i>, <i>range</i> )</code>	resistorValue in Ohm (float), range in Ampere (float)	Sets resistor value in case you don't use the INA226 module with 0.1 Ohm. <b>Don't use setCurrentRange if you are using this function.</b>
<code>float getShuntVoltage_mV( )</code>	none	delivers shunt voltage in mV
<code>float getBusVoltage( )</code>	none	delivers bus voltage in V
<code>float getCurrent_mA( )</code>	none	delivers current in mA
<code>float getBusPower_mW( )</code>	none	delivers the power in mW
<code>void startSingleMeasurement( )</code>	none	starts single shot measurement and waits until data is available
<code>void powerDown( )</code>	none	switches the module off and saves the configuration before
<code>void powerUp( )</code>	none	switches the module on after Power Down and writes back the configuration (modes, gains, etc)
<code>void waitUntilConversionCompleted( )</code>	none	waits until the current conversions and calculations are completed.
<code>void setAlertPinActiveHigh( )</code>	none	by default the the alert pin is active-low; this function changes this
<code>void enableAlertLatch( )</code>	none	the alert flag is set and the alert pin is active, when the limit in the alert register is exceeded; by default it will be deleted with the next measurement in limit; with enableAlertLatch the flag will have to be cleared manually, which gives better control
<code>void setAlertType( <i>type</i>, <i>limit</i> )</code>	types: SHUNT_UNDER, SHUNT_OVER, BUS_UNDER, BUS_OVER, CURRENT_UNDER, CURRENT_OVER, POWER_OVER limit: float	sets the alert type and the limit:  SHUNT_OVER/_UNDER: limit in mV BUS_OVER / _UNDER: limit in V CURRENT_OVER / _UNDER: limit in mA POWER_OVER: limit in mW
<code>void readAndClearFlags( )</code>	none	reads the Mask/Enable register; this clears the overflow, conversion ready and limit alert flags; the status of the flags are saved in the following bool variables: - overflow - convAlert - limitAlert