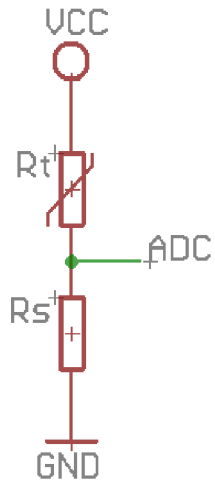
A class to use a NTC thermistor as a temperature sensor

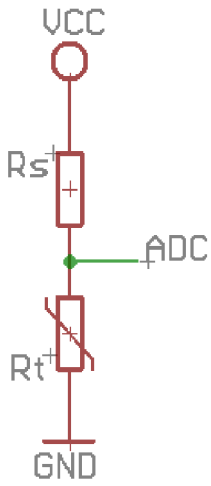
I've often looked at NTC resistors to measure exact temperature, but the amount of work to calculate the actual temperature did always put me off, until someone pointed me at the Steinhart/Hart equation. Let's call that person Tom. Tom is a mathematician and he understands how it all works. I don't, but I figured out how to use the equation. According to Tom, the two "hart's" use 'matrix mathematics' to record the characteristics of a NTC in 3 coefficients. Vishay does supply these coefficients for their NTC products. You can obtain those 3 values on their website when you supply a Vishay part number.
For other/unknown brand thermistors you can also calculate the 4 coefficients using the calculator on this website, the page shows a download link for an Excel spreadsheet. When supplied with 3 different temperature readings, the sheet will calculate 4 coefficients that will work accurately within the supplied temperature range, but not outside the supplied range. My Arduino class does a calculation for both methods (with 3 or 4 coefficients). I haven't tested the 4 coefficient method thoroughly, please leave me a message in the GitHub-issue section when you run into a problem, I like debugging better than uploading libraries. :-)
To use the NTC class, you'll need to pass the 3 (or 4) coefficients to the constructor, don't worry, an example sketch is available.
Starting on the next page, you'll find a description of all the available methods for the different settings.

This class provides two ways to connect the NTC. Both for a voltage divider between VCC and GND.
- The NTC on top and the series resistor below, Code name: NTC_TOP.
- The series resistor on top and the NTC below, Code name: NTC_BOTTOM.

NTC_TOP



NTC_BOTTOM



I chose VCC as the reference instead of the internal VREF, because then current and voltage do not have to be calculated.
The ratio between ADC:MaxADC is equal to the ratio of NTC:(NTC+series_resistance).
That makes the level of the actual supply voltage of the microcontroller and the supply voltage of the resistor-divider irrelevant.

For the supplied example I have used the coefficients for a Vishay NTHS1206N02N1002J thermistor. I could adjust it to show a precise temperature at 0 C and 100 C but I was unable to accurately check the temperature in between. If I only had a medical thermometer to compare it with, I guess those are precise. I can't remember when I last measured my body temperature, if you can, please let me know how accurate this calculation works for temperatures somewhere midrange of the 0..100 C trajectory.
Calibration of 0 C is quite simple, just mix water and ice and leave it for a minute to let the ice to warm up, and the water to cool down to the freezing point. After a while both ice and water are exactly 0 C.
Adjusting 100 C with boiling water is a bit more complex because the air pressure affects the boiling point. Only when the air pressure is exactly 1013.25 mbar/hPa, the boiling point is 100 C.
The class has a built in method to calculate the boiling point of water for a given air pressure in mbar (GetBoilingPointWater). This method is accurate between 950 and 1050 mbar. For a precise air pressure a barometer is needed and the measured temperature from the boiling water needs to be compared with the result from 'GetBoilingPointWater' instead of comparing it with 100 C.
Keep adjusting the value of the series resistor (in the calculation) to the point where the error is the same for both 0 C and the boiling point, either both positive or both negative. Then correct the result of every 'GetTemperature' reading with that error value, or use the method 'SetCorrection' to do it for you.
The calibration needs to be done with the Celsius scale because the correction value is in C, also GetBoilingPointWater returns a value in C. The values in Fahrenheit and Celsius are both converted and corrected from Kelvin during the calculation.

This is the declaration of the class

```
class NTC {
private:
    uint8_t Method, Scale, Equation;
    uint8_t decim; // for c_str() only
    double A, B, C, D, MaxADC, ADC, Rt, Rs, mBar;
    double Hyster, Factor, Rounder, Correct;
    bool bHyst;
    char fmt[16];
public:
    NTC( const double A, const double B, const double C );
    NTC( const double A, const double B, const double C, const double D );
    void SetDecimals( uint8_t decimals );
    double GetBoilingPointWater( double mBar );
    void SetScale( uint8_t scale );
    void SetMaxADC( uint16_t max );
    void SetValueADC( uint16_t adc );
    void SetMethod( uint8_t method );
    void SetHysteresis( bool Hysteresis );
    void SetCorrection( double Correction );
    void SetSeriesResistor( long Resistance );
    double GetTemperature( uint8_t scale );
    double GetTemperature( void );
    char *c_str( char *buf, uint8_t scale );
    char *c_str( char *buf );
};
```

getter: char *c_str(char *buf)

Returns the current temperature in in a formatted string.
To store the formatted value a pointer to a buffer is needed large enough to hold the result.
The number of decimals shown can be set with 'SetDecimals'.
To avoid jumping between two consecutive values, a hysteresis can be set with 'SetHysteresis'. The 'c_str' method is the only function that uses the 'decimals'- and 'hysteresis'-values.

getter: double GetTemperature(uint8_t scale) or double GetTemperature(void)

Returns the current temperature in double precision.

2 ways to call GetTemperature:

- Without parameter it will return the temperature for the scale that was last set with 'SetScale'. Available scales are: Celsius, Fahrenheit and Kelvin.
- With the scale parameter: SCALE_C, SCALE_F or SCALE_K.
  Of course the preferred settings have to be setup before a temperature can be calculated, at least the value of the ADC with analogRead. All other settings have a default value, see below.

setter: SetDecimals(uint8_t decimals)

Decimals are only used by the method c_str() to visualise the temperature reading with a fixed number of decimals. Although you can ask for 16 decimals, keep in mind that asking for more than 2 decimals is quite useless when the ADC resolution is only 10-bits or less.

The default value is: 1

setter: SetScale(int scale)

‘SetScale’ sets one these 3 scales: Celsius, Fahrenheit or Kelvin.
Available parameters are: SCALE_C, SCALE_F or SCALE_K.

The default value is: SCALE_C.

setter: SetMaxADC()

Set the maximum ADC value that can be read with the current setting. For most AVR microcontrollers that will be 1023 for a 10-bit analogue reading. Some newer AVR types and STM32 support 12-bit with 4095 as the highest reading.

The default value is: 1023

setter: SetValueADC()

Use ‘SetValueADC’ to pass the return value from analogRead to the ntc-class, after that the temperature is available immediately.

The default value is: 0, stored during startup but irrelevant

setter: SetMethod()

With ‘SetMethod’ we can indicate the position of the thermistor, either on top (NTC_TOP) or on the bottom (NTC_BOTTOM). See the image in the description. Because the microcontroller reads the voltage relative to the GND potential, it is important to know which wiring was used and which calculation method has to be performed.

The default value is: NTC_BOTTOM

setter: SetHysteresis()

To avoid jumping between two consecutive values pass the boolean value 'true' to this method. Depending on the number of decimals, a hysteresis will be calculated and used to suppress the jumping behaviour.

The default value is: false

setter: SetCorrection()

To adjust the result of the ‘GetTemperature’ method, a value can be supplied to correct the error in temperature reading. The correction value is added to the return value of ‘GetTemperature’.
- When the result of ‘GetTemperature’ is to high, the correction value should be negative.
- When the result of ‘GetTemperature’ is to low, the correction value should be positive.

The default value is: 0

setter: SetSeriesResistor()

‘SetSeriesResistor‘ passes the value of the series resistor to the class. Most multimeters do not have sufficient precision to measure a resistor accurately. Probably a 0.1% resistor is accurate enough to be used without calibration. For the 10k thermistor, I’ve used a combination of two resistors, 4k7 in series with 470R.
The ideal value for the Vishay NTHS1206N02N1002J is:
resistance at 0 C = 28160
resistance at 100 C = 945
The ideal series resistance (for a range from 0..100 C):
Rs is SQRT(28160 * 945) = 5159
4k7 will do fine, but 4700+470=5170 will give a slightly better ADC resolution.

You can fine tune the resistor value by measuring freezing and boiling point of water and alter the resistor value until both readings are correct.

The default value is: 4700