

aWXIron

1.2

Generated by Doxygen 1.8.6

Sun Mar 29 2015 21:41:59



# Contents

<b>1</b>	<b>SMD Soldering Station for Weller RT Series Tips</b>	<b>1</b>
<b>2</b>	<b>User Interface overview</b>	<b>3</b>
2.1	Encoder usage . . . . .	3
2.2	Joined mode . . . . .	4
2.3	Standby . . . . .	4
<b>3</b>	<b>Calibration Process</b>	<b>5</b>
<b>4</b>	<b>Todo List</b>	<b>7</b>
<b>5</b>	<b>Class Index</b>	<b>9</b>
5.1	Class List . . . . .	9
<b>6</b>	<b>File Index</b>	<b>11</b>
6.1	File List . . . . .	11
<b>7</b>	<b>Class Documentation</b>	<b>13</b>
7.1	aDSChannels::_eepromCalibrationValue_t Union Reference . . . . .	13
7.1.1	Detailed Description . . . . .	13
7.1.2	Member Data Documentation . . . . .	13
7.1.2.1	c . . . . .	13
7.1.2.2	v . . . . .	13
7.2	aDSChannel Class Reference . . . . .	13
7.2.1	Detailed Description . . . . .	16
7.2.2	Member Enumeration Documentation . . . . .	16
7.2.2.1	HeatingState_t . . . . .	16
7.2.3	Constructor & Destructor Documentation . . . . .	16
7.2.3.1	aDSChannel . . . . .	16
7.2.3.2	~aDSChannel . . . . .	16
7.2.4	Member Function Documentation . . . . .	16
7.2.4.1	_analogRead . . . . .	16
7.2.4.2	_analogWrite . . . . .	16
7.2.4.3	_digitalRead . . . . .	17

7.2.4.4	<a href="#">_digitalWrite</a>	17
7.2.4.5	<a href="#">_turnOffPWM</a>	17
7.2.4.6	<a href="#">getADCValue</a>	17
7.2.4.7	<a href="#">getCalibration</a>	18
7.2.4.8	<a href="#">getHeatState</a>	18
7.2.4.9	<a href="#">getLEDState</a>	18
7.2.4.10	<a href="#">getStandbyMode</a>	18
7.2.4.11	<a href="#">getTemperature</a>	18
7.2.4.12	<a href="#">hasFocus</a>	18
7.2.4.13	<a href="#">isTempHasChanged</a>	19
7.2.4.14	<a href="#">service</a>	19
7.2.4.15	<a href="#">setCalibration</a>	19
7.2.4.16	<a href="#">setFocus</a>	19
7.2.4.17	<a href="#">setStandbyMode</a>	19
7.2.4.18	<a href="#">setTemperature</a>	19
7.2.4.19	<a href="#">setup</a>	20
7.2.4.20	<a href="#">syncTempChange</a>	20
7.2.4.21	<a href="#">updateLEDState</a>	20
7.2.5	<a href="#">Member Data Documentation</a>	20
7.2.5.1	<a href="#">ADC_TO_TEMP_GAIN</a>	20
7.2.5.2	<a href="#">ADC_TO_TEMP_OFFSET</a>	20
7.2.5.3	<a href="#">BLINK_UPDATE_RATE</a>	20
7.2.5.4	<a href="#">CNTRL_GAIN</a>	21
7.2.5.5	<a href="#">DEFAULT_TEMPERATURE_OFFSET</a>	21
7.2.5.6	<a href="#">DEFAULT_TEMPERATURE_SLOPE</a>	21
7.2.5.7	<a href="#">m_adcValue</a>	21
7.2.5.8	<a href="#">m_blinkStandby</a>	21
7.2.5.9	<a href="#">m_cal</a>	21
7.2.5.10	<a href="#">m_channel</a>	21
7.2.5.11	<a href="#">m_currentTemp</a>	21
7.2.5.12	<a href="#">m_hasFocus</a>	21
7.2.5.13	<a href="#">m_heatState</a>	21
7.2.5.14	<a href="#">m_inStandby</a>	21
7.2.5.15	<a href="#">m_ledPin</a>	21
7.2.5.16	<a href="#">m_ledState</a>	21
7.2.5.17	<a href="#">m_nextBlink</a>	21
7.2.5.18	<a href="#">m_nextLowering</a>	21
7.2.5.19	<a href="#">m_nextPass</a>	21
7.2.5.20	<a href="#">m_nextTempStep</a>	21
7.2.5.21	<a href="#">m_pwmPin</a>	21

7.2.5.22	<a href="#">m_pwmValue</a>	21
7.2.5.23	<a href="#">m_ref</a>	21
7.2.5.24	<a href="#">m_sensorPin</a>	21
7.2.5.25	<a href="#">m_targetTemp</a>	21
7.2.5.26	<a href="#">m_tempHasChanged</a>	21
7.2.5.27	<a href="#">PWM_MAX_VALUE</a>	21
7.2.5.28	<a href="#">TEMPERATURE_MAX</a>	21
7.2.5.29	<a href="#">TEMPERATURE_MIN</a>	21
7.2.5.30	<a href="#">TEMPERATURE_STANDBY</a>	22
7.2.5.31	<a href="#">TEMPERATURE_TOLERANCE</a>	22
7.3	<a href="#">aDSChannels Class Reference</a>	22
7.3.1	<a href="#">Member Enumeration Documentation</a>	26
7.3.1.1	<a href="#">Channel_t</a>	26
7.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	26
7.3.2.1	<a href="#">aDSChannels</a>	26
7.3.2.2	<a href="#">~aDSChannels</a>	26
7.3.3	<a href="#">Member Function Documentation</a>	26
7.3.3.1	<a href="#">_backupCalibrationFromEEPROM</a>	26
7.3.3.2	<a href="#">_checkForMagicNumbers</a>	26
7.3.3.3	<a href="#">_clearValue</a>	27
7.3.3.4	<a href="#">_crc8</a>	28
7.3.3.5	<a href="#">_displayBigDigit</a>	28
7.3.3.6	<a href="#">_displayBigDigits</a>	28
7.3.3.7	<a href="#">_enableData</a>	28
7.3.3.8	<a href="#">_enableDataCheck</a>	29
7.3.3.9	<a href="#">_getTempFromEEPROM</a>	29
7.3.3.10	<a href="#">_read</a>	29
7.3.3.11	<a href="#">_restoreCalibrationFromEEPROM</a>	29
7.3.3.12	<a href="#">_scissor</a>	30
7.3.3.13	<a href="#">_setTempToEEPROM</a>	30
7.3.3.14	<a href="#">_showBanner</a>	30
7.3.3.15	<a href="#">_updateDisplay</a>	30
7.3.3.16	<a href="#">_updateField</a>	30
7.3.3.17	<a href="#">_wakeupFromStandby</a>	31
7.3.3.18	<a href="#">_write</a>	31
7.3.3.19	<a href="#">_writeMagicNumbers</a>	31
7.3.3.20	<a href="#">getCalibrationValues</a>	31
7.3.3.21	<a href="#">getOperationMode</a>	32
7.3.3.22	<a href="#">incEncoderPosition</a>	32
7.3.3.23	<a href="#">isInCalibration</a>	32

7.3.3.24	isInStandby	32
7.3.3.25	isJoined	32
7.3.3.26	pingOperationMode	32
7.3.3.27	restoreCalibrationValues	33
7.3.3.28	saveCalibrationValues	33
7.3.3.29	service	33
7.3.3.30	setCalibrationMode	33
7.3.3.31	setCalibrationValues	33
7.3.3.32	setFocusToNextChannel	34
7.3.3.33	setOperationMode	34
7.3.3.34	setup	34
7.3.3.35	syncData	34
7.3.3.36	toggleJoined	35
7.3.3.37	toggleStandbyMode	35
7.3.3.38	updateOperationMode	35
7.3.4	Member Data Documentation	35
7.3.4.1	DATA_CHANNEL1_LED_STATE	35
7.3.4.2	DATA_CHANNEL1_TEMP_READ	35
7.3.4.3	DATA_CHANNEL1_TEMP_SET	35
7.3.4.4	DATA_CHANNEL2_ENABLED	35
7.3.4.5	DATA_CHANNEL2_LED_STATE	35
7.3.4.6	DATA_CHANNEL2_TEMP_READ	35
7.3.4.7	DATA_CHANNEL2_TEMP_SET	36
7.3.4.8	DATA_CHANNELS_JOINED	36
7.3.4.9	DATA_DISPLAY	36
7.3.4.10	DATA_DISPLAY_STANDBY	36
7.3.4.11	DATA_FOCUS	36
7.3.4.12	DATA_IN_CALIBRATION	36
7.3.4.13	DATA_OPERATION	36
7.3.4.14	DATA_STANDBY	36
7.3.4.15	DISPLAY_UPDATE_RATE	36
7.3.4.16	EEPROM_ADDR_CALIBRATION_CHAN_1	36
7.3.4.17	EEPROM_ADDR_CALIBRATION_CHAN_2	36
7.3.4.18	EEPROM_ADDR_CHANNEL_JOINED	37
7.3.4.19	EEPROM_ADDR_MAGIC	37
7.3.4.20	EEPROM_ADDR_TEMP_CHANNEL_ONE	37
7.3.4.21	EEPROM_ADDR_TEMP_CHANNEL_TWO	37
7.3.4.22	EEPROM_CALIBRATION_SIZE	37
7.3.4.23	EEPROM_STORAGE_STARTING	37
7.3.4.24	EEPROM_TEMP_SIZE	37

7.3.4.25	<code>m_channels</code>	37
7.3.4.26	<code>m_datas</code>	37
7.3.4.27	<code>m_isValidEEPROM</code>	37
7.3.4.28	<code>m_lastTempChange</code>	37
7.3.4.29	<code>m_lcd</code>	37
7.3.4.30	<code>m_lcdCols</code>	37
7.3.4.31	<code>m_lcdRows</code>	37
7.3.4.32	<code>m_nextDisplayUpdate</code>	37
7.3.4.33	<code>m_nextMeasureUpdate</code>	37
7.3.4.34	<code>m_operationMode</code>	38
7.3.4.35	<code>m_operationTick</code>	38
7.3.4.36	<code>m_storedToEEPROM</code>	38
7.3.4.37	<code>MEASURE_UPDATE_RATE</code>	38
7.3.4.38	<code>OFFSET_MARKER_LEFT</code>	38
7.3.4.39	<code>OFFSET_MARKER_RIGHT</code>	38
7.3.4.40	<code>OFFSET_VALUE</code>	38
7.3.4.41	<code>OPERATION_SET_TIMEOUT</code>	38
7.3.4.42	<code>TEMP_SETTING_INACTIVITY</code>	38
7.4	<code>aDSEngine</code> Class Reference	38
7.4.1	Constructor & Destructor Documentation	39
7.4.1.1	<code>aDSEngine</code>	39
7.4.1.2	<code>~aDSEngine</code>	39
7.4.2	Member Function Documentation	39
7.4.2.1	<code>_handleSerialInput</code>	39
7.4.2.2	<code>run</code>	39
7.4.2.3	<code>setup</code>	40
7.4.3	Member Data Documentation	40
7.4.3.1	<code>m_channels</code>	40
7.4.3.2	<code>m_datas</code>	40
7.4.3.3	<code>m_encoder</code>	40
7.4.3.4	<code>m_RXbuffer</code>	40
7.4.3.5	<code>m_RXoffset</code>	40
7.4.3.6	<code>m_serialInputTick</code>	40
7.4.3.7	<code>RXBUFFER_MAXLEN</code>	40
7.5	<code>aDSChannel::aPin_t</code> Struct Reference	40
7.5.1	Detailed Description	41
7.5.2	Member Data Documentation	41
7.5.2.1	<code>mask</code>	41
7.5.2.2	<code>outputRegister</code>	41
7.5.2.3	<code>pin</code>	41

7.5.2.4	port . . . . .	41
7.5.2.5	timer . . . . .	41
7.6	aDSChannel::CalibrationData_t Struct Reference . . . . .	41
7.6.1	Detailed Description . . . . .	41
7.6.2	Member Data Documentation . . . . .	42
7.6.2.1	offset . . . . .	42
7.6.2.2	slope . . . . .	42
<b>8</b>	<b>File Documentation</b>	<b>43</b>
8.1	aDSEngine.cpp File Reference . . . . .	43
8.1.1	Detailed Description . . . . .	43
8.1.2	Function Documentation . . . . .	44
8.1.2.1	getNumericalLength . . . . .	44
8.1.2.2	timer1ISR . . . . .	44
8.1.3	Variable Documentation . . . . .	44
8.1.3.1	_bigDigitsBottom . . . . .	44
8.1.3.2	_bigDigitsTop . . . . .	45
8.1.3.3	DIGIT_WIDTH . . . . .	45
8.1.3.4	pEncoder . . . . .	45
8.1.3.5	PROGMEM . . . . .	45
8.2	aDSEngine.h File Reference . . . . .	45
8.2.1	Detailed Description . . . . .	47
8.2.2	Macro Definition Documentation . . . . .	47
8.2.2.1	IS_DATA_ENABLED . . . . .	47
8.2.2.2	SIMU . . . . .	47
8.2.3	Enumeration Type Documentation . . . . .	47
8.2.3.1	OperationMode_t . . . . .	47
8.2.4	Variable Documentation . . . . .	47
8.2.4.1	CHANNEL2_ENABLE_PIN . . . . .	47
8.2.4.2	ENCODER_A_PIN . . . . .	47
8.2.4.3	ENCODER_B_PIN . . . . .	48
8.2.4.4	ENCODER_PB_PIN . . . . .	48
8.2.4.5	ENCODER_STEPS_PER_NOTCH . . . . .	48
8.2.4.6	LCD_COLS . . . . .	48
8.2.4.7	LCD_D4_PIN . . . . .	48
8.2.4.8	LCD_D5_PIN . . . . .	48
8.2.4.9	LCD_D6_PIN . . . . .	48
8.2.4.10	LCD_D7_PIN . . . . .	48
8.2.4.11	LCD_ENABLE_PIN . . . . .	48
8.2.4.12	LCD_ROWS . . . . .	48



8.2.4.13	LCD_RS_PIN	48
8.2.4.14	LED_CHANNEL1_PIN	48
8.2.4.15	LED_CHANNEL2_PIN	49
8.2.4.16	PROGRAM_VERSION_MAJOR	49
8.2.4.17	PROGRAM_VERSION_MINOR	49
8.2.4.18	PWM_CHANNEL1_PIN	49
8.2.4.19	PWM_CHANNEL2_PIN	49
8.2.4.20	TEMP_SENSOR_CHANNEL1_PIN	49
8.2.4.21	TEMP_SENSOR_CHANNEL2_PIN	49
8.3	CDC.cpp File Reference	49
8.4	HardwareSerial.cpp File Reference	49
8.5	HID.cpp File Reference	49
8.6	IPAddress.cpp File Reference	49
8.7	libraries.cpp File Reference	50
8.8	main.cpp File Reference	50
8.9	new.cpp File Reference	50
8.10	Print.cpp File Reference	50
8.11	sketch.cpp File Reference	50
8.11.1	Function Documentation	50
8.11.1.1	loop	50
8.11.1.2	setup	50
8.11.2	Variable Documentation	50
8.11.2.1	engine	50
8.12	Stream.cpp File Reference	51
8.13	Tone.cpp File Reference	51
8.14	USBCore.cpp File Reference	51
8.15	WInterrupts.c File Reference	51
8.16	wiring.c File Reference	51
8.16.1	Macro Definition Documentation	52
8.16.1.1	FRACT_INC	52
8.16.1.2	FRACT_MAX	52
8.16.1.3	MICROSECONDS_PER_TIMER0_OVERFLOW	52
8.16.1.4	MILLIS_INC	52
8.16.2	Function Documentation	52
8.16.2.1	delay	52
8.16.2.2	delayMicroseconds	52
8.16.2.3	if	52
8.16.2.4	init	52
8.16.2.5	micros	52
8.16.2.6	millis	52

8.16.3 Variable Documentation . . . . .	52
8.16.3.1 f . . . . .	52
8.16.3.2 m . . . . .	52
8.16.3.3 timer0_fract . . . . .	52
8.16.3.4 timer0_millis . . . . .	52
8.16.3.5 timer0_overflow_count . . . . .	52
8.17 wiring_analog.c File Reference . . . . .	52
8.18 wiring_digital.c File Reference . . . . .	52
8.19 wiring_pulse.c File Reference . . . . .	52
8.20 wiring_shift.c File Reference . . . . .	53
8.21 WMath.cpp File Reference . . . . .	53
8.22 WString.cpp File Reference . . . . .	53
<b>Index</b>	<b>54</b>

## Chapter 1

# SMD Soldering Station for Weller RT Series Tips

Based on a project from **Martin Kumm** [http://www.martin-kumm.de/wiki/doku.php?id=Projects:SMD\\_Solderstation](http://www.martin-kumm.de/wiki/doku.php?id=Projects:SMD_Solderstation).

The hardware has been redesigned (two channels, 16x2 LCD instead of 7 segments display, etc), and the software rewrote from scratch.

I want to especially thank my friend **Olivier, F5LGJ**, for his great help and support in this project.



## Chapter 2

# User Interface overview

- The Soldering Station control is done using a simple rotary encoder, which integrates a push button.
- The temperature range goes from 100 °C, up to 450 °C.
- Depending of the hardware assembly, it can control one or two soldering irons:
  1. In Single mode, the temperature reading and setting is displayed using double height font.
  2. In dual channels version, both soldering irons can be controled separately, or can be joined:
    - when separate channels mode is used, each channel is independent. Simple click on the encoder push button will set the focus to the next channel. The focused channel temperature will be surrounded by the symbols [ and ]
    - when joined mode is used, the temperature is displayed like in Single channel mode (double height font), both channels share the same settings (target temperature and standby mode).
- LED status decoding:

LED Status	Meaning
ON	the tip is heating
OFF	the tip is cooling
Blinking	the tip has reached his target temperature
Three times blinking	the soldering station is in Standby mode (see <a href="#">Standby</a> )

- The target temperature is stored, for each channel, inside the microcontroller's EEPROM. The values will be restored on the next startup. After a timeout of 60 seconds, a new defined target temperature will be stored into the EEPROM. If in the meantime the user defines a new target temperature, the timeout is resetted
- When the station is in temperature reading mode, the displayed value(s) is left aligned. When the station is in settings mode, the displayed value is right aligned.

## 2.1 Encoder usage

- In any mode (settings or temperature reading), the rotary encoder is used to define the target temperature. Turn the encoder clockwise to increase the target temperature, and anti-clockwise to decrease it.
- When the soldering station is not in settings mode, it displays the soldering tip temperature. A single encoder detents rotation will switch the soldering station into settings mode, and display the target temperature without any change to the target temperature setting.

- When the soldering station is in settings mode, if not action is done using the encoder's rotation within 3 seconds, it will switch back to temperature reading.

- Encoder's push button:

- Single soldering tip version:

Button	Action
Single Click	<i>no effect</i>
Double Click	toggles standby mode (see <a href="#">Standby</a> )
Held	<i>no effect</i>

- Dual soldering tip version:

Button	Action
Single click	change the focus to the next channel (if not in joined mode)
Double click	switch to standby mode (see <a href="#">Standby</a> )
Held	toggles joined mode (see <a href="#">Joined mode</a> )

## 2.2 Joined mode

- With dual channel enabled hardware, it's possible to share the same temperature preset for both soldering tips.

See [Encoder usage](#)

## 2.3 Standby

- A double-click on the encoder brings the soldering station in the standby mode.
- When standby mode is enabled, the target temperature will go down to 150 °C if the temperature setting is set above this point, otherwise it will go down to 100 °C.
- Any encoder action will exit the standby mode.
- When Standby mode is activated, the LEDs blink three times cyclically.

## Chapter 3

# Calibration Process

- **Prerequisites:**

- **Hardware:**

- \* Digital Thermometer (e.g: you multimeter with a K probe)

- **Software:**

- \* A serial terminal emulator (e.g. *"HyperTerminal"* or *"Tera Term"* on Windows, *"minicom"* or *"cute-com"* on Linux).
    - \* The calibration spreadsheet file **aWXIronsCalibration.ods**
    - \* A software able to open the calibration spreadsheet, like *"LibreOffice"*, *"OpenOffice"* and so on.

The serial communication settings are: **57600, 8, N, 1**

- **Why a calibration:**

You have to connect the soldering station to the PC, using a USB cable. The calibration process is necessary get accurate temperature control. To turn the soldering station in calibration, you have to keep the encoder push button pressed while turning the station ON. Once the station is ready to use, the '**CAL**' string is displayed on the top left side of the LCD display. In calibration mode, the readed temperature isn't displayed. Instead, the ADC value is shown. It's really important to start from the lowest temperature setting (100°C), and wait for till the temperature stabilize.

- **Process Description:**

- Blah

**Todo** write me





## Chapter 4

# Todo List

Page **Calibration Process**

write me



## Chapter 5

# Class Index

### 5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">aDSChannels::_eepromCalibrationValue_t</a>	
Union to manipulate float/uint8_t [] calibration values . . . . .	13
<a href="#">aDSChannel</a>	
ADSChannel class . . . . .	13
<a href="#">aDSChannels</a> . . . . .	22
<a href="#">aDSEngine</a> . . . . .	38
<a href="#">aDSChannel::aPin_t</a>	
Our pin structure . . . . .	40
<a href="#">aDSChannel::CalibrationData_t</a>	
Calibration values . . . . .	41



## Chapter 6

# File Index

### 6.1 File List

Here is a list of all files with brief descriptions:

<a href="#">aDSEngine.cpp</a>	43
<a href="#">aDSEngine.h</a>	45
<a href="#">CDC.cpp</a>	49
<a href="#">HardwareSerial.cpp</a>	49
<a href="#">HID.cpp</a>	49
<a href="#">IPAddress.cpp</a>	49
<a href="#">libraries.cpp</a>	50
<a href="#">main.cpp</a>	50
<a href="#">new.cpp</a>	50
<a href="#">Print.cpp</a>	50
<a href="#">sketch.cpp</a>	50
<a href="#">Stream.cpp</a>	51
<a href="#">Tone.cpp</a>	51
<a href="#">USBCore.cpp</a>	51
<a href="#">WInterrupts.c</a>	51
<a href="#">wiring.c</a>	51
<a href="#">wiring_analog.c</a>	52
<a href="#">wiring_digital.c</a>	52
<a href="#">wiring_pulse.c</a>	52
<a href="#">wiring_shift.c</a>	53
<a href="#">WMath.cpp</a>	53
<a href="#">WString.cpp</a>	53



## Chapter 7

# Class Documentation

### 7.1 aDSChannels::\_eepromCalibrationValue\_t Union Reference

Union to manipulate float/uint8\_t [] calibration values.

```
#include <aDSEngine.h>
```

#### Public Attributes

- float [v](#)
- uint8\_t [c](#) [sizeof(float)]

#### 7.1.1 Detailed Description

Union to manipulate float/uint8\_t [] calibration values.

#### 7.1.2 Member Data Documentation

7.1.2.1 uint8\_t aDSChannels::\_eepromCalibrationValue\_t::c[sizeof(float)]

7.1.2.2 float aDSChannels::\_eepromCalibrationValue\_t::v

The documentation for this union was generated from the following file:

- [aDSEngine.h](#)

### 7.2 aDSChannel Class Reference

[aDSChannel](#) class

```
#include <aDSEngine.h>
```

#### Classes

- struct [aPin\\_t](#)  
*Our pin structure.*
- struct [CalibrationData\\_t](#)  
*Calibration values.*

## Public Types

- enum [HeatingState\\_t](#) { [HEATING\\_STATE\\_HEATING](#), [HEATING\\_STATE\\_COOLING](#), [HEATING\\_STATE\\_REACHED](#), [HEATING\\_STATE\\_STANDBY](#) }

*Heating State enumeration.*

## Public Member Functions

- [aDSChannel](#) ()  
*aDSChannel class constructor*
- virtual [~aDSChannel](#) ()  
*aDSChannel destructor*
- void [setup](#) (uint8\_t, uint8\_t, uint8\_t)  
*Setup member function, should be called before any other member.*
- void [setFocus](#) (bool)  
*Set the focus, as display point of view.*
- bool [hasFocus](#) ()  
*Get the focus state.*
- uint16\_t [getTemperature](#) (OperationMode\_t)  
*Get current temperature accordingly from the given mode (SET/READ)*
- bool [setTemperature](#) (OperationMode\_t, int16\_t)  
*Set current temperature accordingly from the given mode (SET/READ)*
- bool [service](#) (unsigned long)  
*This member should be called often, it manage heating/cooling of the Channel.*
- void [setStandbyMode](#) (bool)  
*Enable or disable channel's standby.*
- bool [getStandbyMode](#) ()  
*Get channel's standby enableity.*
- bool [isTempHasChanged](#) ()  
*Is target temperature has changed (use for EEPROM storage)*
- void [syncTempChange](#) ()  
*Reset temperature change flag (use for EEPROM storage)*
- uint8\_t [updateLEDState](#) (unsigned long)  
*Change LED state according for Heating/Cooling/Standby status.*
- uint8\_t [getLEDState](#) ()  
*Get state LED status.*
- [HeatingState\\_t](#) [getHeatState](#) ()  
*Get channel heating state.*
- void [setCalibration](#) (float, float)
- const [CalibrationData\\_t](#) [getCalibration](#) () const  
*Get calibration data values.*
- int16\_t [getADCValue](#) ()  
*Get latest ADC value.*



## Static Public Attributes

- static const int16\_t [TEMPERATURE\\_MIN](#) = 10
- static const int16\_t [TEMPERATURE\\_MAX](#) = 450  
*Maximum temperature.*
- static const int16\_t [TEMPERATURE\\_STANDBY](#) = 150  
*Standby temperature.*
- static const float [ADC\\_TO\\_TEMP\\_GAIN](#) = 0.39
- static const float [ADC\\_TO\\_TEMP\\_OFFSET](#) = 23.9
- static const uint8\_t [CNTRL\\_GAIN](#) = 10
- static const unsigned long [BLINK\\_UPDATE\\_RATE](#) = 400  
*Update rate for LED blinking, in ms.*
- static const int16\_t [TEMPERATURE\\_TOLERANCE](#) = 3  
*Temperature tolerance for REACHED state, +/- 2 °C.*
- static const float [DEFAULT\\_TEMPERATURE\\_SLOPE](#) = 0.3947387545
- static const float [DEFAULT\\_TEMPERATURE\\_OFFSET](#) = 43.8279285472
- static const uint8\_t [PWM\\_MAX\\_VALUE](#) = 150  
*Maximum PWM value.*

## Protected Member Functions

- void [\\_turnOffPWM](#) (aPin\_t)  
*Turns PWM off for the given pin.*
- int8\_t [\\_digitalRead](#) (aPin\_t)  
*Get state of the given pin.*
- void [\\_digitalWrite](#) (aPin\_t, uint8\_t)  
*Set state of the given pin.*
- uint16\_t [\\_analogRead](#) (aPin\_t)  
*Get analog value of the given pin.*
- void [\\_analogWrite](#) (aPin\_t, uint8\_t)  
*Set analog value for the given pin.*

## Private Attributes

- [aPin\\_t](#) m\_pwmPin
- [aPin\\_t](#) m\_sensorPin
- [aPin\\_t](#) m\_ledPin
- bool m\_hasFocus
- int16\_t m\_targetTemp
- int16\_t m\_currentTemp
- int8\_t m\_pwmValue
- int16\_t m\_adcValue
- bool m\_inStandby
- [HeatingState\\_t](#) m\_heatState
- uint8\_t m\_ledState
- unsigned long m\_nextPass
- bool m\_tempHasChanged
- unsigned long m\_nextBlink
- uint8\_t m\_blinkStandby
- uint8\_t m\_ref
- [CalibrationData\\_t](#) m\_cal
- uint8\_t m\_channel
- unsigned long m\_nextTempStep
- unsigned long m\_nextLowering

## 7.2.1 Detailed Description

[aDSChannel](#) class

## 7.2.2 Member Enumeration Documentation

### 7.2.2.1 enum `aDSChannel::HeatingState_t`

Heating State enumeration.

Enumerator

***HEATING\_STATE\_HEATING*** Heating.  
***HEATING\_STATE\_COOLING*** Cooling.  
***HEATING\_STATE\_REACHED*** Target temperature reached.  
***HEATING\_STATE\_STANDBY*** In standby mode.

## 7.2.3 Constructor & Destructor Documentation

### 7.2.3.1 `aDSChannel::aDSChannel ( )`

[aDSChannel](#) class constructor

### 7.2.3.2 `aDSChannel::~~aDSChannel ( )` [virtual]

[aDSChannel](#) destructor

## 7.2.4 Member Function Documentation

### 7.2.4.1 `uint16_t aDSChannel::_analogRead ( aPin_t pin )` [protected]

Get analog value of the given pin.

Took and Hacked from Arduino [wiring\\_analog.c](#)

Parameters

<i>pin</i>	<a href="#">aPin_t</a> : pin
------------	------------------------------

Returns

`uint8_t`: analog value (0..255)

### 7.2.4.2 `void aDSChannel::_analogWrite ( aPin_t pin, uint8_t val )` [protected]

Set analog value for the given pin.

Took and Hacked from Arduino [wiring\\_analog.c](#)

Parameters

<i>pin</i>	<a href="#">aPin_t</a> : pin
<i>val</i>	uint8_t : analog value (0..255)

## Returns

void

**7.2.4.3** int8\_t aDSChannel::\_digitalRead ( aPin\_t *pin* ) [protected]

Get state of the given pin.

Took and Hacked from Arduino [wiring\\_digital.c](#)

## Parameters

<i>pin</i>	<a href="#">aPin_t</a> : pin
------------	------------------------------

## Returns

int8\_t : HIGH or LOW

**7.2.4.4** void aDSChannel::\_digitalWrite ( aPin\_t *pin*, uint8\_t *val* ) [protected]

Set state of the given pin.

Took and Hacked from Arduino [wiring\\_digital.c](#)

## Parameters

<i>pin</i>	<a href="#">aPin_t</a> : pin
<i>val</i>	uint8_t : state (HIGH or LOW)

## Returns

void

**7.2.4.5** void aDSChannel::\_turnOffPWM ( aPin\_t *pin* ) [protected]

Turns PWM off for the given pin.

Took and Hacked from Arduino [wiring\\_digital.c](#)

## Parameters

<i>pin</i>	<a href="#">aPin_t</a> : pin
------------	------------------------------

## Returns

void

**7.2.4.6** int16\_t aDSChannel::getADCValue ( )

Get latest ADC value.

## Returns

int16\_t : ADC value

#### 7.2.4.7 `const aDSChannel::CalibrationData_t aDSChannel::getCalibration ( ) const`

Get calibration data values.

##### Returns

`const CalibrationData_t` : Calibration data

#### 7.2.4.8 `aDSChannel::HeatingState_t aDSChannel::getHeatState ( )`

Get channel heating state.

##### Returns

`aDSChannel::HeatingState_t` : heating state

#### 7.2.4.9 `uint8_t aDSChannel::getLEDState ( )`

Get state LED status.

##### Returns

`uint8_t` : HIGH or LOW (LED on or off, accordingly)

#### 7.2.4.10 `bool aDSChannel::getStandbyMode ( )`

Get channel's standby enableity.

##### Returns

`bool`

#### 7.2.4.11 `uint16_t aDSChannel::getTemperature ( OperationMode_t mode )`

Get current temperature accordingly from the given mode (SET/READ)

##### Parameters

<i>mode</i>	<code>OperationMode_t</code> : operation mode
-------------	---

##### Returns

`uint16_t` : temperature, in Celcius

#### 7.2.4.12 `bool aDSChannel::hasFocus ( )`

Get the focus state.

##### Returns

`bool` : focus state

**7.2.4.13 bool aDSChannel::isTempHasChanged ( )**

Is target temperature has changed (use for EEPROM storage)

**Returns**

bool : return true if target temperature has changed, otherwise false

**7.2.4.14 bool aDSChannel::service ( unsigned long *m* )**

This member should be called often, it manage heating/cooling of the Channel.

**Parameters**

<i>m</i>	unsigned long : current timestamp
----------	-----------------------------------

**Returns**

bool : return true if readed temperature has changed since last call, otherwise false

**7.2.4.15 void aDSChannel::setCalibration ( float *slope*, float *offset* )****7.2.4.16 void aDSChannel::setFocus ( bool *v* )**

Set the focus, as display point of view.

**Parameters**

<i>v</i>	bool : focus state
----------	--------------------

**Returns**

void

**7.2.4.17 void aDSChannel::setStandbyMode ( bool *enable* )**

Enable or disable channel's standby.

**Parameters**

<i>enable</i>	bool : enability
---------------	------------------

**Returns**

void

**7.2.4.18 bool aDSChannel::setTemperature ( **OperationMode\_t** *mode*, **int16\_t** *temp* )**

Set current temperature accordingly from the given mode (SET/READ)

**Parameters**

<i>mode</i>	OperationMode_t : operation mode
<i>temp</i>	int16_t : temperature, in Celcius

**Returns**

bool : true if temperature has been changed, otherwise false

#### 7.2.4.19 void aDSChannel::setup ( uint8\_t *pwmPin*, uint8\_t *sensorPin*, uint8\_t *ledPin* )

Setup member function, should be called before any other member.

Pins will be embedded into [aPin\\_t](#) object, timer, mask, port and output register will be set also here, preventing using Arduino analog/digital{Read/Write}() calls, which are quite slow.

**Parameters**

<i>pwmPin</i>	uint8_t : PWM pin, used to drive the output MosFET
<i>sensorPin</i>	uint8_t : Sensor pin, used to get analog temperature value.
<i>ledPin</i>	uint8_t : LED pin, used to reflect Heating/Cooling state

**Returns**

void

#### 7.2.4.20 void aDSChannel::syncTempChange ( )

Reset temperature change flag (use for EEPROM storage)

**Returns**

void

#### 7.2.4.21 uint8\_t aDSChannel::updateLEDState ( unsigned long *m* )

Change LED state according for Heating/Cooling/Standby status.

**Parameters**

<i>m</i>	unsigned long : timestamp
----------	---------------------------

**Returns**

uint8\_t : HIGH or LOW (LED on or off, accordingly)

### 7.2.5 Member Data Documentation

7.2.5.1 const float aDSChannel::ADC\_TO\_TEMP\_GAIN = 0.39 [static]

7.2.5.2 const float aDSChannel::ADC\_TO\_TEMP\_OFFSET = 23.9 [static]

7.2.5.3 const unsigned long aDSChannel::BLINK\_UPDATE\_RATE = 400 [static]

Update rate for LED blinking, in ms.

- 7.2.5.4 `const uint8_t aDSChannel::CNTRL_GAIN = 10` [static]
- 7.2.5.5 `const float aDSChannel::DEFAULT_TEMPERATURE_OFFSET = 43.8279285472` [static]
- 7.2.5.6 `const float aDSChannel::DEFAULT_TEMPERATURE_SLOPE = 0.3947387545` [static]
- 7.2.5.7 `int16_t aDSChannel::m_adcValue` [private]
- 7.2.5.8 `uint8_t aDSChannel::m_blinkStandby` [private]
- 7.2.5.9 `CalibrationData_t aDSChannel::m_cal` [private]
- 7.2.5.10 `uint8_t aDSChannel::m_channel` [private]
- 7.2.5.11 `int16_t aDSChannel::m_currentTemp` [private]
- 7.2.5.12 `bool aDSChannel::m_hasFocus` [private]
- 7.2.5.13 `HeatingState_t aDSChannel::m_heatState` [private]
- 7.2.5.14 `bool aDSChannel::m_inStandby` [private]
- 7.2.5.15 `aPin_t aDSChannel::m_ledPin` [private]
- 7.2.5.16 `uint8_t aDSChannel::m_ledState` [private]
- 7.2.5.17 `unsigned long aDSChannel::m_nextBlink` [private]
- 7.2.5.18 `unsigned long aDSChannel::m_nextLowering` [private]
- 7.2.5.19 `unsigned long aDSChannel::m_nextPass` [private]
- 7.2.5.20 `unsigned long aDSChannel::m_nextTempStep` [private]
- 7.2.5.21 `aPin_t aDSChannel::m_pwmPin` [private]
- 7.2.5.22 `int8_t aDSChannel::m_pwmValue` [private]
- 7.2.5.23 `uint8_t aDSChannel::m_ref` [private]
- 7.2.5.24 `aPin_t aDSChannel::m_sensorPin` [private]
- 7.2.5.25 `int16_t aDSChannel::m_targetTemp` [private]
- 7.2.5.26 `bool aDSChannel::m_tempHasChanged` [private]
- 7.2.5.27 `const uint8_t aDSChannel::PWM_MAX_VALUE = 150` [static]  
Maximum PWM value.
- 7.2.5.28 `const int16_t aDSChannel::TEMPERATURE_MAX = 450` [static]  
Maximum temperature.
- 7.2.5.29 `const int16_t aDSChannel::TEMPERATURE_MIN = 10` [static]

7.2.5.30 `const int16_t aDSChannel::TEMPERATURE_STANDBY = 150` [static]

Standby temperature.

7.2.5.31 `const int16_t aDSChannel::TEMPERATURE_TOLERANCE = 3` [static]

Temperature tolerance for REACHED state, +/- 2 °C.

The documentation for this class was generated from the following files:

- [aDSEngine.h](#)
- [aDSEngine.cpp](#)

## 7.3 aDSChannels Class Reference

```
#include <aDSEngine.h>
```

### Classes

- union [\\_eepromCalibrationValue\\_t](#)  
*Union to manipulate float/uint8\_t [] calibration values.*

### Public Types

- enum [Channel\\_t](#) { [CHANNEL\\_ONE](#), [CHANNEL\\_TWO](#), [CHANNEL\\_MAX](#) }  
*Channels enumeration.*

### Public Member Functions

- [aDSChannels](#) (uint8\_t, uint8\_t, uint8\_t, uint8\_t, uint8\_t, uint8\_t)  
*aDSChannels constructor*
- virtual [~aDSChannels](#) ()  
*aDSChannels destructor*
- void [setup](#) (uint8\_t, uint8\_t, uint8\_t, uint8\_t, uint8\_t, uint8\_t, uint8\_t, uint8\_t, uint8\_t)  
*Setup member function, should be called before any other member.*
- void [setOperationMode](#) ([OperationMode\\_t](#))  
*Set operation mode (SET/READ)*
- [OperationMode\\_t](#) [getOperationMode](#) ()  
*Get current operation mode.*
- void [updateOperationMode](#) ()  
*Update operation mode.*
- void [pingOperationMode](#) ()  
*Reset timeout for OPERATION\_MODE\_SET mode.*
- void [syncData](#) (uint16\_t)  
*Reset given bit inside bitfield, regardless of its state.*
- void [incEncoderPosition](#) (uint16\_t)  
*Increment or decrement encoder position.*
- void [service](#) ()  
*This member should be called often, it manage channels, LCD and EEPROM.*
- void [toggleJoined](#) ()



- Toggle channels joining.*
  - bool `isJoinded ()`
- Get channel joining state.*
  - void `setFocusToNextChannel ()`
- Set the focus to the next channel, if any.*
  - void `toggleStandbyMode ()`
- Toggle standby mode.*
  - bool `isInStandby ()`
- Get standby mode.*
  - void `setCalibrationValues (Channel_t, aDSChannel::CalibrationData_t)`
- Set calibration values for given channel.*
  - `aDSChannel::CalibrationData_t` `getCalibrationValues (Channel_t)`
- Get calibration value for given channel.*
  - void `restoreCalibrationValues ()`
- Restore calibration values from EEPROM.*
  - void `saveCalibrationValues (Channel_t)`
- Save calibration value of given into EEPROM.*
  - bool `isInCalibration ()`
- Get calibration mode enableity.*
  - void `setCalibrationMode (bool)`
- Set calibration mode.*

### Static Public Attributes

- static const uint8\_t `OFFSET_VALUE` = 2  
*Value column LCD offset.*
- static const uint8\_t `OFFSET_MARKER_LEFT` = 0  
*Column LCD offset for left marker '['.*
- static const uint8\_t `OFFSET_MARKER_RIGHT` = 10  
*Column LCD offset for right marker ']'.*
- static const unsigned long `OPERATION_SET_TIMEOUT` = 3000  
*Automatic toggle settings->reading timeout (3 seconds), in ms.*
- static const unsigned long `DISPLAY_UPDATE_RATE` = 200  
*Display update rate, in ms.*
- static const unsigned long `MEASURE_UPDATE_RATE` = 200  
*Measurement (for `aDSChannel`) rate, in ms.*
- static const unsigned long `TEMP_SETTING_INACTIVITY` = 60000  
*Timeout in ms, after which the new target temperature will be stored in the EEPROM.*
- static const uint16\_t `DATA_CHANNEL2_ENABLED` = 1  
*Bitfield: Channel 2 is enabled.*
- static const uint16\_t `DATA_CHANNELS_JOINED` = 1 << 1  
*Bitfield: Channel 1 & 2 are joinded.*
- static const uint16\_t `DATA_OPERATION` = 1 << 2  
*Bitfield: Operation mode has changed.*
- static const uint16\_t `DATA_CHANNEL1_TEMP_SET` = 1 << 3  
*Bitfield: Target temperature of channel 1 has changed.*
- static const uint16\_t `DATA_CHANNEL1_TEMP_READ` = 1 << 4  
*Bitfield: Readed temperature of channel 1 has changed.*
- static const uint16\_t `DATA_CHANNEL1_LED_STATE` = 1 << 5  
*Bitfield: LED state of channel 1 has changed.*

- static const uint16\_t [DATA\\_CHANNEL2\\_TEMP\\_SET](#) = 1 << 6  
*Bitfield: Target temperature of channel 1 has changed.*
- static const uint16\_t [DATA\\_CHANNEL2\\_TEMP\\_READ](#) = 1 << 7  
*Bitfield: Readed temperature of channel 1 has changed.*
- static const uint16\_t [DATA\\_CHANNEL2\\_LED\\_STATE](#) = 1 << 8  
*Bitfield: LED state of channel 1 has changed.*
- static const uint16\_t [DATA\\_DISPLAY](#) = 1 << 9  
*Bitfield: Display should be refreshed.*
- static const uint16\_t [DATA\\_STANDBY](#) = 1 << 10  
*Bitfield: Standby state.*
- static const uint16\_t [DATA\\_DISPLAY\\_STANDBY](#) = 1 << 11  
*Bitfield: Standby state has changed.*
- static const uint16\_t [DATA\\_FOCUS](#) = 1 << 12  
*Bitfield: Focus has changed.*
- static const uint16\_t [DATA\\_IN\\_CALIBRATION](#) = 1 << 13  
*Bitfield: in Calibration.*

## Protected Member Functions

- void [\\_enableData](#) (uint16\_t, bool)  
*Enable a bit, regardless of its state, inside bitfield m\_datas.*
- void [\\_enableDataCheck](#) (uint16\_t, bool)  
*Enable a bit, if it's not already set, inside bitfield m\_datas.*
- void [\\_updateDisplay](#) ()  
*Update LCD display, if needed.*
- void [\\_displayBigDigit](#) (uint8\_t, uint8\_t, uint8\_t=0)  
*Display a big digit to given position.*
- void [\\_displayBigDigits](#) (uint16\_t, uint8\_t, uint8\_t=0)  
*Display a big digits number to given position.*
- void [\\_clearValue](#) (uint8\_t, int=0)  
*Clear numerical value field (in non big digit mode) on LCD.*
- void [\\_updateField](#) (OperationMode\_t, int16\_t, uint8\_t)  
*Update value on LCD from given mode and row.*
- void [\\_wakeUpFromStandby](#) ()  
*Wake up from standby mode.*
- void [\\_showBanner](#) ()  
*Display a banner on the LCD.*
- bool [\\_checkForMagicNumbers](#) ()  
*Check for the magic number in the EEPROM.*
- void [\\_writeMagicNumbers](#) ()  
*Write magic numbers into EEPROM.*
- uint8\_t [\\_crc8](#) (const uint8\_t \*, uint8\_t)  
*CRC8 computation.*
- template<typename T >  
bool [\\_write](#) (T const, int16\_t &)  
*Template to write a value into EEPROM, at given address.*
- template<typename T >  
bool [\\_read](#) (T &, int16\_t &)  
*Template to read a value from the EEPROM, at given address.*
- template<typename T >  
void [\\_scissor](#) (T v, uint8\_t \*, size\_t &)

- Template to decompose the value into an array of uint8\_t (used for CRC8 computation)*

  - `bool _getTempFromEEPROM (int16_t, uint16_t &)`
  - Helper to read the stored temperature inside EEPROM at given address.*
  - `void _setTempToEEPROM (int16_t, uint16_t)`
  - Helper to write a temperature inside EEPROM at given address.*
  - `void _restoreCalibrationFromEEPROM (int16_t, aDSChannel &)`
  - Restore calibration value for given channel.*
  - `void _backupCalibrationFromEEPROM (int16_t, aDSChannel &)`
  - Backup calibration value for given channel.*

### Static Protected Attributes

- `static const int16_t EEPROM_ADDR_MAGIC = 0`
- EEPROM offset storage start for magic numbers (0xDEAD)*
- `static const int16_t EEPROM_STORAGE_STARTING = 5`
- EEPROM starting address for program datas.*
- `static const int16_t EEPROM_TEMP_SIZE = sizeof(uint16_t) + sizeof(uint8_t)`
- EEPROM temperature size (temperature + crc)*
- `static const int16_t EEPROM_ADDR_CHANNEL_JOINED = EEPROM_STORAGE_STARTING + 1`
- Channels are joined.*
- `static const int16_t EEPROM_ADDR_TEMP_CHANNEL_ONE = EEPROM_ADDR_CHANNEL_JOINED + EEPROM_TEMP_SIZE`
- Target temp for Channel 1.*
- `static const int16_t EEPROM_ADDR_TEMP_CHANNEL_TWO = EEPROM_ADDR_TEMP_CHANNEL_ONE + EEPROM_TEMP_SIZE`
- Target temp for Channel 2.*
- `static const int16_t EEPROM_CALIBRATION_SIZE = (sizeof(float) * 2) + sizeof(uint8_t)`
- EEPROM calibration size: 2 float (slope & offset), and one uint8\_t for crc.*
- `static const int16_t EEPROM_ADDR_CALIBRATION_CHAN_1 = EEPROM_ADDR_TEMP_CHANNEL_TWO + EEPROM_TEMP_SIZE`
- EEPROM start offset for Channel 1 calibration values.*
- `static const int16_t EEPROM_ADDR_CALIBRATION_CHAN_2 = EEPROM_ADDR_CALIBRATION_CHAN_1 + EEPROM_CALIBRATION_SIZE`
- EEPROM start offset for Channel 2 calibration values.*

### Private Attributes

- LiquidCrystal `m_lcd`
- `aDSChannel m_channels [CHANNEL_MAX]`
- `OperationMode_t m_operationMode`
- `unsigned long m_operationTick`
- `uint16_t m_datas`
- `uint8_t m_lcdCols`
- `uint8_t m_lcdRows`
- `unsigned long m_nextDisplayUpdate`
- `unsigned long m_nextMeasureUpdate`
- `unsigned long m_lastTempChange`
- `bool m_isValidEEPROM`
- `bool m_storedToEEPROM`

### 7.3.1 Member Enumeration Documentation

#### 7.3.1.1 enum aDSChannels::Channel\_t

Channels enumeration.

Enumerator

***CHANNEL\_ONE***

***CHANNEL\_TWO***

***CHANNEL\_MAX***

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 aDSChannels::aDSChannels ( uint8\_t rs, uint8\_t e, uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7 )

[aDSChannels](#) constructor

Parameters

<i>rs</i>	uint8_t : LCD RS pin
<i>e</i>	uint8_t : LCD Enable pin
<i>d4</i>	uint8_t : LCD D4 pin
<i>d5</i>	uint8_t : LCD D5 pin
<i>d6</i>	uint8_t : LCD D6 pin
<i>d7</i>	uint8_t : LCD D7 pin

#### 7.3.2.2 aDSChannels::~~aDSChannels ( ) [virtual]

[aDSChannels](#) destructor

### 7.3.3 Member Function Documentation

#### 7.3.3.1 void aDSChannels::\_backupCalibrationFromEEPROM ( int16\_t startAddr, aDSChannel & channel ) [protected]

Backup calibration value for given channel.

Parameters

<i>startAddr</i>	int16_t : EEPROM start address
<i>channel</i>	<a href="#">aDSChannel&amp;</a> : channel

Returns

void

#### 7.3.3.2 bool aDSChannels::\_checkForMagicNumbers ( ) [protected]

Check for the magic number in the EEPROM.

Returns

bool : true if magic numbers has been found, otherwise false

7.3.3.3 void aDSChannels::\_clearValue ( uint8\_t row, int *destMinus* = 0 ) [protected]

Clear numerical value field (in non big digit mode) on LCD.

## Parameters

<i>row</i>	uint8_t : LCD row position
<i>destMinus</i>	int : right offset sub

## Returns

void

7.3.3.4 uint8\_t aDSChannels::\_crc8 ( const uint8\_t\* *addr*, uint8\_t *len* ) [protected]

CRC8 computation.

Code took from [http://www.pjrc.com/teensy/td\\_libs\\_OneWire.html](http://www.pjrc.com/teensy/td_libs_OneWire.html)

## Parameters

<i>addr</i>	const uint8_t* : <b>Data source</b>
<i>len</i>	uint8_t : <b>Data source length</b>

## Returns

uint8\_t : **CRC**

7.3.3.5 void aDSChannels::\_displayBigDigit ( uint8\_t *digit*, uint8\_t *position*, uint8\_t *offset* = 0 ) [protected]

Display a big digit to given position.

## Parameters

<i>digit</i>	uint8_t : offset in _bigDigit{Top/Bottom} array
<i>position</i>	uint8_t : LCD position
<i>offset</i>	uint8_t : LCD offset position

## Returns

void

7.3.3.6 void aDSChannels::\_displayBigDigits ( uint16\_t *value*, uint8\_t *position*, uint8\_t *offset* = 0 ) [protected]

Display a big digits number to given position.

## Parameters

<i>value</i>	uint16_t : value to display, DIGIT_WIDTH max length
<i>position</i>	uint8_t : LCD position
<i>offset</i>	uint8_t : LCD offset position

## Returns

void

7.3.3.7 void aDSChannels::\_enableData ( uint16\_t *bit*, bool *enable* ) [protected]

Enable a bit, regardless of its state, inside bitfield m\_datas.

## Parameters

<i>bit</i>	uint16_t : bit to enable/disable
<i>enable</i>	bool : bit enableity

## Returns

void

**7.3.3.8** void aDSChannels::\_enableDataCheck ( uint16\_t *bit*, bool *enable* ) [protected]

Enable a bit, if it's not already set, inside bitfield m\_datas.

## Parameters

<i>bit</i>	uint16_t : bit to enable/disable
<i>enable</i>	bool : bit enableity

## Returns

void

**7.3.3.9** bool aDSChannels::\_getTempFromEEPROM ( int16\_t *startAddr*, uint16\_t & *temp* ) [protected]

Helper to read the stored temperature inside EEPROM at given address.

## Parameters

<i>startAddr</i>	int16_t : start address
<i>temp</i>	uint16_t& : temperature

## Returns

bool : true if the CRCs matches

**7.3.3.10** template<typename T> bool aDSChannels::\_read ( T & *v*, int16\_t & *addr* ) [protected]

Template to read a value from the EEPROM, at given address.

## Parameters

<i>v</i>	T& : readed value
<i>addr</i>	int16_t& : start address

## Returns

template &lt;typename T&gt; bool : true on read success, otherwise false

**7.3.3.11** void aDSChannels::\_restoreCalibrationFromEEPROM ( int16\_t *startAddr*, aDSChannel & *channel* )  
[protected]

Restore calibration value for given channel.

## Parameters

<i>startAddr</i>	int16_t : EEPROM start address
<i>channel</i>	aDSChannel& : channel

## Returns

void

7.3.3.12 `template<typename T> void aDSChannels::_scissor ( T v, uint8_t* dest, size_t & offset )` [protected]

Template to decompose the value into an array of uint8\_t (used for CRC8 computation)

## Parameters

<i>v</i>	T : value
<i>dest</i>	uint8_t* : pointer to the destination array
<i>offset</i>	size_t& : start offset of the array

## Returns

template &lt;typename T&gt; void

7.3.3.13 `void aDSChannels::_setTempToEEPROM ( int16_t startAddr, uint16_t temp )` [protected]

Helper to write a temperature inside EEPROM at given address.

## Parameters

<i>startAddr</i>	int16_t : start address
<i>temp</i>	uint16_t : temperature

## Returns

void

7.3.3.14 `void aDSChannels::_showBanner ( )` [protected]

Display a banner on the LCD.

## Returns

void

7.3.3.15 `void aDSChannels::_updateDisplay ( )` [protected]

Update LCD display, if needed.

## Returns

void

7.3.3.16 `void aDSChannels::_updateField ( OperationMode_t mode, int16_t value, uint8_t row )` [protected]

Update value on LCD from given mode and row.



## Parameters

<i>mode</i>	OperationMode_t : operation mode (SET/READ)
<i>value</i>	int16_t : value to display
<i>row</i>	uint8_t : LCD row

## Returns

void

## 7.3.3.17 void aDSChannels::\_wakeupFromStandby ( ) [protected]

Wake up from standby mode.

## Returns

void

## 7.3.3.18 template&lt;typename T&gt; bool aDSChannels::\_write ( T const v, int16\_t &amp; addr ) [protected]

Template to write a value into EEPROM, at given address.

## Parameters

<i>v</i>	T const : value
<i>addr</i>	int16_t& : start address

## Returns

template &lt;typename T&gt; bool : true on write success, otherwise false

## 7.3.3.19 void aDSChannels::\_writeMagicNumbers ( ) [protected]

Write magic numbers into EEPROM.

## Returns

void

## 7.3.3.20 aDSChannel::CalibrationData\_t aDSChannels::getCalibrationValues ( Channel\_t chan )

Get calibration value for given channel.

## Parameters

<i>chan</i>	Channel_t : channel
-------------	---------------------

## Returns

aDSChannel::CalibrationData\_t : calibration values

**7.3.3.21 OperationMode\_t aDSChannels::getOperationMode ( )**

Get current operation mode.

**Returns**

OperationMode\_t : operation mode

**7.3.3.22 void aDSChannels::incEncoderPosition ( uint16\_t v )**

Increment or decrement encoder position.

**Parameters**

v	uint16_t : increment value (signed)
---	-------------------------------------

**Returns**

void

**7.3.3.23 bool aDSChannels::isInCalibration ( )**

Get calibration mode enableity.

**Returns**

bool : true if in calibration mode, otherwise false

**7.3.3.24 bool aDSChannels::isInStandby ( )**

Get standby mode.

**Returns**

bool : true if in standby mode, otherwise false

**7.3.3.25 bool aDSChannels::isJoinded ( )**

Get channel joining state.

**Returns**

bool : true if joinded, otherwise false

**7.3.3.26 void aDSChannels::pingOperationMode ( )**

Reset timeout for OPERATION\_MODE\_SET mode.

**Returns**

void

**7.3.3.27 void aDSChannels::restoreCalibrationValues ( )**

Restore calibration values from EEPROM.

Returns

void

**7.3.3.28 void aDSChannels::saveCalibrationValues ( Channel\_t chan )**

Save calibration value of given into EEPROM.

Parameters

<i>chan</i>	Channel_t : channel
-------------	---------------------

Returns

void

**7.3.3.29 void aDSChannels::service ( )**

This member should be called often, it manage channels, LCD and EEPROM.

Returns

void

**7.3.3.30 void aDSChannels::setCalibrationMode ( bool enable )**

Set calibration mode.

Parameters

<i>enable</i>	bool : enability
---------------	------------------

Returns

void

**7.3.3.31 void aDSChannels::setCalibrationValues ( Channel\_t chan, aDSChannel::CalibrationData\_t cal )**

Set calibration values for given channel.

Parameters

<i>chan</i>	Channel_t : channel
<i>cal</i>	<a href="#">aDSChannel::CalibrationData_t</a> : calibration values

Returns

void

### 7.3.3.32 void aDSChannels::setFocusToNextChannel ( )

Set the focus to the next channel, if any.

#### Returns

void

### 7.3.3.33 void aDSChannels::setOperationMode ( OperationMode\_t m )

Set operation mode (SET/READ)

#### Parameters

<i>m</i>	OperationMode_t : new operation mode
----------	--------------------------------------

#### Returns

void

### 7.3.3.34 void aDSChannels::setup ( uint8\_t cols, uint8\_t rows, uint8\_t pwmChan1, uint8\_t sensChan1, uint8\_t ledChan1, uint8\_t chkChan2, uint8\_t pwmChan2, uint8\_t sensChan2, uint8\_t ledChan2 )

Setup member function, should be called before any other member.

#### Parameters

<i>cols</i>	uint8_t : LCD number of columns
<i>rows</i>	uint8_t : LCD number of rows
<i>pwmChan1</i>	uint8_t : Channel 1 PWM pin
<i>sensChan1</i>	uint8_t : Channel 1 Temperature Sensor pin
<i>ledChan1</i>	uint8_t : Channel 1 LED pin
<i>chkChan2</i>	uint8_t : Channel 2 enableity pin
<i>pwmChan2</i>	uint8_t : Channel 2 PWM pin
<i>sensChan2</i>	uint8_t : Channel 2 Temperature Sensor pin
<i>ledChan2</i>	uint8_t : Channel 2 LED pin

#### Returns

void

### 7.3.3.35 void aDSChannels::syncData ( uint16\_t bit )

Reset given bit inside bitfield, regardless of its state.

#### Parameters

<i>bit</i>	uint16_t : bit to reset
------------	-------------------------

#### Returns

void

#### 7.3.3.36 void aDSChannels::toggleJoined ( )

Toggle channels joining.

##### Returns

void

#### 7.3.3.37 void aDSChannels::toggleStandbyMode ( )

Toggle standby mode.

##### Returns

void

#### 7.3.3.38 void aDSChannels::updateOperationMode ( )

Update operation mode.

If operation mode is currently set to OPERATION\_MODE\_SET, and OPERATION\_SET\_TIMEOUT timeout is triggered, operation mode will be switched to OPERATION\_MODE\_READ

##### Returns

void

### 7.3.4 Member Data Documentation

#### 7.3.4.1 const uint16\_t aDSChannels::DATA\_CHANNEL1\_LED\_STATE = 1 << 5 [static]

Bitfield: LED state of channel 1 has changed.

#### 7.3.4.2 const uint16\_t aDSChannels::DATA\_CHANNEL1\_TEMP\_READ = 1 << 4 [static]

Bitfield: Readed temperature of channel 1 has changed.

#### 7.3.4.3 const uint16\_t aDSChannels::DATA\_CHANNEL1\_TEMP\_SET = 1 << 3 [static]

Bitfield: Target temperature of channel 1 has changed.

#### 7.3.4.4 const uint16\_t aDSChannels::DATA\_CHANNEL2\_ENABLED = 1 [static]

Bitfield: Channel 2 is enabled.

#### 7.3.4.5 const uint16\_t aDSChannels::DATA\_CHANNEL2\_LED\_STATE = 1 << 8 [static]

Bitfield: LED state of channel 1 has changed.

#### 7.3.4.6 const uint16\_t aDSChannels::DATA\_CHANNEL2\_TEMP\_READ = 1 << 7 [static]

Bitfield: Readed temperature of channel 1 has changed.

7.3.4.7 `const uint16_t aDSChannels::DATA_CHANNEL2_TEMP_SET = 1 << 6` [static]

Bitfield: Target temperature of channel 1 has changed.

7.3.4.8 `const uint16_t aDSChannels::DATA_CHANNELS_JOINED = 1 << 1` [static]

Bitfield: Channel 1 & 2 are joined.

7.3.4.9 `const uint16_t aDSChannels::DATA_DISPLAY = 1 << 9` [static]

Bitfield: Display should be refreshed.

7.3.4.10 `const uint16_t aDSChannels::DATA_DISPLAY_STANDBY = 1 << 11` [static]

Bitfield: Standby state has changed.

7.3.4.11 `const uint16_t aDSChannels::DATA_FOCUS = 1 << 12` [static]

Bitfield: Focus has changed.

7.3.4.12 `const uint16_t aDSChannels::DATA_IN_CALIBRATION = 1 << 13` [static]

Bitfield: in Calibration.

7.3.4.13 `const uint16_t aDSChannels::DATA_OPERATION = 1 << 2` [static]

Bitfield: Operation mode has changed.

7.3.4.14 `const uint16_t aDSChannels::DATA_STANDBY = 1 << 10` [static]

Bitfield: Standby state.

7.3.4.15 `const unsigned long aDSChannels::DISPLAY_UPDATE_RATE = 200` [static]

Display update rate, in ms.

7.3.4.16 `const int16_t aDSChannels::EEPROM_ADDR_CALIBRATION_CHAN_1 = EEPROM_ADDR_TEMP_CHANNEL_T-  
WO + EEPROM_TEMP_SIZE` [static], [protected]

EEPROM start offset for Channel 1 calibration values.

7.3.4.17 `const int16_t aDSChannels::EEPROM_ADDR_CALIBRATION_CHAN_2 = EEPROM_AD-  
DR_CALIBRATION_CHAN_1 + EEPROM_CALIBRATION_SIZE` [static],  
[protected]

EEPROM start offset for Channel 2 calibration values.

**7.3.4.18** `const int16_t aDSChannels::EEPROM_ADDR_CHANNEL_JOINED = EEPROM_STORAGE_STARTING + 1`  
`[static], [protected]`

Channels are joined.

**7.3.4.19** `const int16_t aDSChannels::EEPROM_ADDR_MAGIC = 0` `[static], [protected]`

EEPROM offset storage start for magic numbers (0xDEAD)

**7.3.4.20** `const int16_t aDSChannels::EEPROM_ADDR_TEMP_CHANNEL_ONE = EEPROM_ADDR_CHANNEL_JOINED + EEPROM_TEMP_SIZE` `[static], [protected]`

Target temp for Channel 1.

**7.3.4.21** `const int16_t aDSChannels::EEPROM_ADDR_TEMP_CHANNEL_TWO = EEPROM_ADDR_TEMP_CHANNEL_ONE + EEPROM_TEMP_SIZE` `[static], [protected]`

Target temp for Channel 2.

**7.3.4.22** `const int16_t aDSChannels::EEPROM_CALIBRATION_SIZE = (sizeof(float) * 2) + sizeof(uint8_t)` `[static], [protected]`

EEPROM calibration size: 2 float (slope & offset), and one uint8\_t for crc.

**7.3.4.23** `const int16_t aDSChannels::EEPROM_STORAGE_STARTING = 5` `[static], [protected]`

EEPROM starting address for program datas.

**7.3.4.24** `const int16_t aDSChannels::EEPROM_TEMP_SIZE = sizeof(uint16_t) + sizeof(uint8_t)` `[static], [protected]`

EEPROM temperature size (temperature + crc)

**7.3.4.25** `aDSChannel aDSChannels::m_channels[CHANNEL_MAX]` `[private]`

**7.3.4.26** `uint16_t aDSChannels::m_dats` `[private]`

**7.3.4.27** `bool aDSChannels::m_isValidEEPROM` `[private]`

**7.3.4.28** `unsigned long aDSChannels::m_lastTempChange` `[private]`

**7.3.4.29** `LiquidCrystal aDSChannels::m_lcd` `[private]`

**7.3.4.30** `uint8_t aDSChannels::m_lcdCols` `[private]`

**7.3.4.31** `uint8_t aDSChannels::m_lcdRows` `[private]`

**7.3.4.32** `unsigned long aDSChannels::m_nextDisplayUpdate` `[private]`

**7.3.4.33** `unsigned long aDSChannels::m_nextMeasureUpdate` `[private]`

7.3.4.34 `OperationMode_t aDSChannels::m_operationMode` [private]

7.3.4.35 `unsigned long aDSChannels::m_operationTick` [private]

7.3.4.36 `bool aDSChannels::m_storedToEEPROM` [private]

7.3.4.37 `const unsigned long aDSChannels::MEASURE_UPDATE_RATE = 200` [static]

Measurement (for [aDSChannel](#)) rate, in ms.

7.3.4.38 `const uint8_t aDSChannels::OFFSET_MARKER_LEFT = 0` [static]

Column LCD offset for left marker '['.

7.3.4.39 `const uint8_t aDSChannels::OFFSET_MARKER_RIGHT = 10` [static]

Column LCD offset for right marker ']'.

7.3.4.40 `const uint8_t aDSChannels::OFFSET_VALUE = 2` [static]

Value column LCD offset.

7.3.4.41 `const unsigned long aDSChannels::OPERATION_SET_TIMEOUT = 3000` [static]

Automatic toggle settings->reading timeout (3 seconds), in ms.

7.3.4.42 `const unsigned long aDSChannels::TEMP_SETTING_INACTIVITY = 60000` [static]

Timeout in ms, after which the new target temperature will be stored in the EEPROM.

The documentation for this class was generated from the following files:

- [aDSEngine.h](#)
- [aDSEngine.cpp](#)

## 7.4 aDSEngine Class Reference

```
#include <aDSEngine.h>
```

### Public Member Functions

- [aDSEngine](#) ()  
*aDSEngine constructor*
- virtual [~aDSEngine](#) ()  
*aDSEngine destructor*
- void [setup](#) ()  
*Setup member function, should be called before any other member.*
- void [run](#) ()  
*Main loop.*



## Protected Member Functions

- void [\\_handleSerialInput](#) ( )  
*Handle serial input, in calibration mode only.*

## Private Attributes

- [aDSChannels](#) [m\\_channels](#)
- ClickEncoder [m\\_encoder](#)
- [uint16\\_t](#) [m\\_datas](#)
- [uint8\\_t](#) [m\\_RXbuffer](#) [RXBUFFER\_MAXLEN]  
*USB rx buffer.*
- [uint8\\_t](#) [m\\_RXoffset](#)  
*USB rx buffer offset counter.*
- unsigned long [m\\_serialInputTick](#)

## Static Private Attributes

- static const [uint8\\_t](#) [RXBUFFER\\_MAXLEN](#) = 64

### 7.4.1 Constructor & Destructor Documentation

#### 7.4.1.1 [aDSEngine::aDSEngine](#) ( )

[aDSEngine](#) constructor

#### 7.4.1.2 [aDSEngine::~~aDSEngine](#) ( ) [virtual]

[aDSEngine](#) destructor

### 7.4.2 Member Function Documentation

#### 7.4.2.1 void [aDSEngine::\\_handleSerialInput](#) ( ) [protected]

Handle serial input, in calibration mode only.

Returns

void

#### 7.4.2.2 void [aDSEngine::run](#) ( )

Main loop.

Returns

void

#### 7.4.2.3 void aDSEngine::setup ( )

Setup member function, should be called before any other member.

Returns

void

### 7.4.3 Member Data Documentation

7.4.3.1 aDSChannels aDSEngine::m\_channels [private]

7.4.3.2 uint16\_t aDSEngine::m\_datas [private]

7.4.3.3 ClickEncoder aDSEngine::m\_encoder [private]

7.4.3.4 uint8\_t aDSEngine::m\_RXbuffer[RXBUFFER\_MAXLEN] [private]

USB rx buffer.

7.4.3.5 uint8\_t aDSEngine::m\_RXoffset [private]

USB rx buffer offset counter.

7.4.3.6 unsigned long aDSEngine::m\_serialInputTick [private]

7.4.3.7 const uint8\_t aDSEngine::RXBUFFER\_MAXLEN = 64 [static],[private]

The documentation for this class was generated from the following files:

- [aDSEngine.h](#)
- [aDSEngine.cpp](#)

## 7.5 aDSChannel::aPin\_t Struct Reference

Our pin structure.

```
#include <aDSEngine.h>
```

### Public Attributes

- uint8\_t [pin](#)  
*"Arduino" pin*
- uint8\_t [timer](#)  
*Timer of the pin.*
- uint8\_t [mask](#)  
*Bit mask of the pin.*
- uint8\_t [port](#)  
*Port of the pin.*
- volatile uint8\_t \* [outputRegister](#)  
*Output register of the pin.*

### 7.5.1 Detailed Description

Our pin structure.

### 7.5.2 Member Data Documentation

#### 7.5.2.1 uint8\_t aDSChannel::aPin\_t::mask

Bit mask of the pin.

#### 7.5.2.2 volatile uint8\_t\* aDSChannel::aPin\_t::outputRegister

Output register of the pin.

#### 7.5.2.3 uint8\_t aDSChannel::aPin\_t::pin

"Arduino" pin

#### 7.5.2.4 uint8\_t aDSChannel::aPin\_t::port

Port of the pin.

#### 7.5.2.5 uint8\_t aDSChannel::aPin\_t::timer

Timer of the pin.

The documentation for this struct was generated from the following file:

- [aDSEngine.h](#)

## 7.6 aDSChannel::CalibrationData\_t Struct Reference

Calibration values.

```
#include <aDSEngine.h>
```

### Public Attributes

- float [slope](#)  
*Slope value.*
- float [offset](#)  
*Offset value.*

### 7.6.1 Detailed Description

Calibration values.

Contains Slope and Offset float values

## 7.6.2 Member Data Documentation

### 7.6.2.1 float aDSChannel::CalibrationData\_t::offset

Offset value.

### 7.6.2.2 float aDSChannel::CalibrationData\_t::slope

Slope value.

The documentation for this struct was generated from the following file:

- [aDSEngine.h](#)

## Chapter 8

# File Documentation

### 8.1 aDSEngine.cpp File Reference

```
#include "aDSEngine.h"  
#include <wiring_private.h>
```

#### Functions

- static int8\_t [getNumericalLength](#) (int16\_t n)  
*Return numerical character length of argument.*
- void [timer1ISR](#) (void)  
*Timer1 class ISR function.*

#### Variables

- static const uint8\_t [DIGIT\\_WIDTH](#) = 3  
*Max numerical length of temperature (used with big digits)*
- static const uint8\_t [\\_glyphs](#)[][8] [PROGMEM](#)  
*LCD glyphs (for big digits and LED)*
- static const uint8\_t [\\_bigDigitsTop](#) [11][[DIGIT\\_WIDTH](#)]  
*\_glyphs[] offsets*
- static const uint8\_t [\\_bigDigitsBottom](#) [11][[DIGIT\\_WIDTH](#)]  
*\_glyphs[] offsets*
- ClickEncoder \* [pEncoder](#) = NULL  
*Global pointer to ClickEncoder object, used inside [timer1ISR\(\)](#) function.*

#### 8.1.1 Detailed Description

##### Copyright

Copyright (C) 2015 F1RMB, Daniel Caujolle-Bert [f1rmb.daniel@gmail.com](mailto:f1rmb.daniel@gmail.com)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

#### Author

F1RMB, Daniel Caujolle-Bert [f1rmb.daniel@gmail.com](mailto:f1rmb.daniel@gmail.com)

## 8.1.2 Function Documentation

### 8.1.2.1 static int8\_t getNumericalLength ( int16\_t *n* ) [static]

Return numerical character length of argument.

#### Parameters

<i>n</i>	int16_t : value to get length from
----------	------------------------------------

#### Returns

int8\_t : length

### 8.1.2.2 void timer1ISR ( void )

Timer1 class ISR function.

This function is periodically called from Timer1 class, which handles encoder events

#### Returns

void

## 8.1.3 Variable Documentation

### 8.1.3.1 const uint8\_t \_bigDigitsBottom[11][DIGIT\_WIDTH] [static]

#### Initial value:

```
=
{
    { 3, 1, 3 },
    { 1, 3, 1 },
    { 3, 1, 1 },
    { 1, 1, 3 },
    { 32, 32, 3 },
    { 1, 1, 3 },
    { 3, 1, 3 },
    { 32, 32, 3 },
    { 3, 1, 3 },
    { 1, 1, 3 },
    { 32, 32, 32 }
}
```

\_glyphs[] offsets

0..9 + ' ' bottom characters matrix

### 8.1.3.2 `const uint8_t _bigDigitsTop[11][DIGIT_WIDTH]` `[static]`

Initial value:

```
=
{
  { 3, 0, 3 },
  { 0, 3, 32 },
  { 2, 2, 3 },
  { 0, 2, 3 },
  { 3, 1, 3 },
  { 3, 2, 2 },
  { 3, 2, 2 },
  { 0, 0, 3 },
  { 3, 2, 3 },
  { 3, 2, 3 },
  { 32, 32, 32 }
}
```

`_glyphs[]` offsets

0..9 + ' ' top characters matrix

### 8.1.3.3 `const uint8_t DIGIT_WIDTH = 3` `[static]`

Max numerical length of temperature (used with big digits)

### 8.1.3.4 `ClickEncoder* pEncoder = NULL`

Global pointer to ClickEncoder object, used inside `timer1ISR()` function.

### 8.1.3.5 `const uint8_t _glyphs[][8] PROGMEM` `[static]`

LCD glyphs (for big digits and LED)

## 8.2 aDSEngine.h File Reference

```
#include <Arduino.h>
#include <LiquidCrystal.h>
#include <EEPROM.h>
#include "TimerOne.h"
#include "ClickEncoder.h"
```

### Classes

- class [aDSChannel](#)  
*aDSChannel class*
- struct [aDSChannel::CalibrationData\\_t](#)  
*Calibration values.*
- struct [aDSChannel::aPin\\_t](#)  
*Our pin structure.*
- class [aDSChannels](#)
- union [aDSChannels::\\_eepromCalibrationValue\\_t](#)  
*Union to manipulate float/uint8\_t [] calibration values.*
- class [aDSEngine](#)

## Macros

- `#define SIMU 1`
- `#define IS_DATA_ENABLED(bit) ((m_datas & bit))`

## Enumerations

- enum `OperationMode_t` { `OPERATION_MODE_READ`, `OPERATION_MODE_SET`, `OPERATION_MODE_UNKNOWN` }
- Operation Mode enumeration.*

## Variables

- static const uint8\_t `CHANNEL2_ENABLE_PIN` = 13  
*Pin to check from if channel 2 is wired.*
- static const uint8\_t `LCD_RS_PIN` = 7  
*LCD RS pin.*
- static const uint8\_t `LCD_ENABLE_PIN` = 8  
*LCD Enable pin.*
- static const uint8\_t `LCD_D4_PIN` = 9  
*LCD D4 pin.*
- static const uint8\_t `LCD_D5_PIN` = 10  
*LCD D5 pin.*
- static const uint8\_t `LCD_D6_PIN` = 11  
*LCD D6 pin.*
- static const uint8\_t `LCD_D7_PIN` = 12  
*LCD D7 pin.*
- static const uint8\_t `LCD_COLS` = 16  
*LCD columns.*
- static const uint8\_t `LCD_ROWS` = 2  
*LCD rows.*
- static const uint8\_t `ENCODER_A_PIN` = 2  
*Encoder A pin.*
- static const uint8\_t `ENCODER_B_PIN` = 3  
*Encoder B pin.*
- static const uint8\_t `ENCODER_PB_PIN` = 4  
*Encoder push button pin.*
- static const uint8\_t `ENCODER_STEPS_PER_NOTCH` = 4  
*Number of steps per notch (indent)*
- static const uint8\_t `PWM_CHANNEL1_PIN` = 5  
*PWM pin of channel 1.*
- static const uint8\_t `PWM_CHANNEL2_PIN` = 6  
*PWM pin of channel 2.*
- static const uint8\_t `TEMP_SENSOR_CHANNEL1_PIN` = A1  
*Temp sensor pin of channel 1.*
- static const uint8\_t `TEMP_SENSOR_CHANNEL2_PIN` = A0  
*Temp sensor pin of channel 2.*
- static const uint8\_t `LED_CHANNEL1_PIN` = A2  
*LED pin of channel 1.*
- static const uint8\_t `LED_CHANNEL2_PIN` = A3  
*LED pin of channel 2.*



- static const uint8\_t `PROGRAM_VERSION_MAJOR` = 1  
*Major program version.*
- static const uint8\_t `PROGRAM_VERSION_MINOR` = 2  
*Minor program version.*

### 8.2.1 Detailed Description

#### Copyright

Copyright (C) 2015 F1RMB, Daniel Caujolle-Bert [flrmb.daniel@gmail.com](mailto:flrmb.daniel@gmail.com)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

#### Author

F1RMB, Daniel Caujolle-Bert [flrmb.daniel@gmail.com](mailto:flrmb.daniel@gmail.com)

### 8.2.2 Macro Definition Documentation

8.2.2.1 `#define IS_DATA_ENABLED( bit ) ((m_dmas & bit))`

8.2.2.2 `#define SIMU 1`

### 8.2.3 Enumeration Type Documentation

8.2.3.1 `enum OperationMode_t`

Operation Mode enumeration.

#### Enumerator

**`OPERATION_MODE_READ`** Reading values.

**`OPERATION_MODE_SET`** Settings values.

**`OPERATION_MODE_UNKNOWN`** Unset (internal)

### 8.2.4 Variable Documentation

8.2.4.1 `const uint8_t CHANNEL2_ENABLE_PIN = 13` `[static]`

Pin to check from if channel 2 is wired.

8.2.4.2 `const uint8_t ENCODER_A_PIN = 2` `[static]`

Encoder A pin.

**8.2.4.3** `const uint8_t ENCODER_B_PIN = 3` `[static]`

Encoder B pin.

**8.2.4.4** `const uint8_t ENCODER_PB_PIN = 4` `[static]`

Encoder push button pin.

**8.2.4.5** `const uint8_t ENCODER_STEPS_PER_NOTCH = 4` `[static]`

Number of steps per notch (indent)

**8.2.4.6** `const uint8_t LCD_COLS = 16` `[static]`

LCD columns.

**8.2.4.7** `const uint8_t LCD_D4_PIN = 9` `[static]`

LCD D4 pin.

**8.2.4.8** `const uint8_t LCD_D5_PIN = 10` `[static]`

LCD D5 pin.

**8.2.4.9** `const uint8_t LCD_D6_PIN = 11` `[static]`

LCD D6 pin.

**8.2.4.10** `const uint8_t LCD_D7_PIN = 12` `[static]`

LCD D7 pin.

**8.2.4.11** `const uint8_t LCD_ENABLE_PIN = 8` `[static]`

LCD Enable pin.

**8.2.4.12** `const uint8_t LCD_ROWS = 2` `[static]`

LCD rows.

**8.2.4.13** `const uint8_t LCD_RS_PIN = 7` `[static]`

LCD RS pin.

**8.2.4.14** `const uint8_t LED_CHANNEL1_PIN = A2` `[static]`

LED pin of channel 1.

**8.2.4.15** `const uint8_t LED_CHANNEL2_PIN = A3` `[static]`

LED pin of channel 2.

**8.2.4.16** `const uint8_t PROGRAM_VERSION_MAJOR = 1` `[static]`

Major program version.

**8.2.4.17** `const uint8_t PROGRAM_VERSION_MINOR = 2` `[static]`

Minor program version.

**8.2.4.18** `const uint8_t PWM_CHANNEL1_PIN = 5` `[static]`

PWM pin of channel 1.

**8.2.4.19** `const uint8_t PWM_CHANNEL2_PIN = 6` `[static]`

PWM pin of channel 2.

**8.2.4.20** `const uint8_t TEMP_SENSOR_CHANNEL1_PIN = A1` `[static]`

Temp sensor pin of channel 1.

**8.2.4.21** `const uint8_t TEMP_SENSOR_CHANNEL2_PIN = A0` `[static]`

Temp sensor pin of channel 2.

## 8.3 CDC.cpp File Reference

```
#include <CDC.cpp>
```

## 8.4 HardwareSerial.cpp File Reference

```
#include <HardwareSerial.cpp>
```

## 8.5 HID.cpp File Reference

```
#include <HID.cpp>
```

## 8.6 IPAddress.cpp File Reference

```
#include <IPAddress.cpp>
```

## 8.7 libraries.cpp File Reference

```
#include <Arduino.h>
#include "ClickEncoder.cpp"
#include "TimerOne.cpp"
```

## 8.8 main.cpp File Reference

```
#include <main.cpp>
```

## 8.9 new.cpp File Reference

```
#include <new.cpp>
```

## 8.10 Print.cpp File Reference

```
#include <Print.cpp>
```

## 8.11 sketch.cpp File Reference

```
#include <Arduino.h>
#include <LiquidCrystal.h>
#include "aDSEngine.h"
```

### Functions

- void [setup](#) ()
- void [loop](#) ()

### Variables

- [aDSEngine](#) \* [engine](#)

### 8.11.1 Function Documentation

8.11.1.1 void [loop](#) ( )

8.11.1.2 void [setup](#) ( )

### 8.11.2 Variable Documentation

8.11.2.1 [aDSEngine](#)\* [engine](#)

## 8.12 Stream.cpp File Reference

```
#include <Stream.cpp>
```

## 8.13 Tone.cpp File Reference

```
#include <Tone.cpp>
```

## 8.14 USBCore.cpp File Reference

```
#include <USBCore.cpp>
```

## 8.15 WInterrupts.c File Reference

```
#include <WInterrupts.c>
```

## 8.16 wiring.c File Reference

```
#include "wiring_private.h"
```

### Macros

- `#define MICROSECONDS_PER_TIMER0_OVERFLOW` (clockCyclesToMicroseconds(PRESCALE\_FACTOR \* 256))
- `#define MILLIS_INC` (MICROSECONDS\_PER\_TIMER0\_OVERFLOW / 1000)
- `#define FRACT_INC` ((MICROSECONDS\_PER\_TIMER0\_OVERFLOW % 1000) >> 3)
- `#define FRACT_MAX` (1000 >> 3)

### Functions

- `if (f >= FRACT_MAX)`
- unsigned long `millis` ()
- unsigned long `micros` ()
- void `delay` (unsigned long ms)
- void `delayMicroseconds` (unsigned int us)
- void `init` ()

### Variables

- volatile unsigned long `timer0_overflow_count` = 0
- volatile unsigned long `timer0_millis` = 0
- static unsigned char `timer0_fract` = 0
- unsigned char `f` = `timer0_fract`
- `m` = `MILLIS_INC`

### 8.16.1 Macro Definition Documentation

8.16.1.1 `#define FRACT_INC ((MICROSECONDS_PER_TIMER0_OVERFLOW % 1000) >> 3)`

8.16.1.2 `#define FRACT_MAX (1000 >> 3)`

8.16.1.3 `#define MICROSECONDS_PER_TIMER0_OVERFLOW (clockCyclesToMicroseconds(PRESCALE_FACTOR * 256))`

8.16.1.4 `#define MILLIS_INC (MICROSECONDS_PER_TIMER0_OVERFLOW / 1000)`

### 8.16.2 Function Documentation

8.16.2.1 `void delay ( unsigned long ms )`

8.16.2.2 `void delayMicroseconds ( unsigned int us )`

8.16.2.3 `if ( f >= FRACT_MAX )`

8.16.2.4 `void init ( )`

8.16.2.5 `unsigned long micros ( )`

8.16.2.6 `unsigned long millis ( )`

### 8.16.3 Variable Documentation

8.16.3.1 `f = timer0_fract`

8.16.3.2 `m = MILLIS_INC`

8.16.3.3 `timer0_fract = 0` `[static]`

8.16.3.4 `timer0_millis = 0`

8.16.3.5 `timer0_overflow_count = 0`

## 8.17 wiring\_analog.c File Reference

```
#include <wiring_analog.c>
```

## 8.18 wiring\_digital.c File Reference

```
#include <wiring_digital.c>
```

## 8.19 wiring\_pulse.c File Reference

```
#include <wiring_pulse.c>
```

## 8.20 wiring\_shift.c File Reference

```
#include <wiring_shift.c>
```

## 8.21 WMath.cpp File Reference

```
#include <WMath.cpp>
```

## 8.22 WString.cpp File Reference

```
#include <WString.cpp>
```

# Index

- ~aDSChannel
  - aDSChannel, [16](#)
- ~aDSChannels
  - aDSChannels, [26](#)
- ~aDSEngine
  - aDSEngine, [39](#)
- \_analogRead
  - aDSChannel, [16](#)
- \_analogWrite
  - aDSChannel, [16](#)
- \_backupCalibrationFromEEPROM
  - aDSChannels, [26](#)
- \_bigDigitsBottom
  - aDSEngine.cpp, [44](#)
- \_bigDigitsTop
  - aDSEngine.cpp, [44](#)
- \_checkForMagicNumbers
  - aDSChannels, [26](#)
- \_clearValue
  - aDSChannels, [26](#)
- \_crc8
  - aDSChannels, [28](#)
- \_digitalRead
  - aDSChannel, [17](#)
- \_digitalWrite
  - aDSChannel, [17](#)
- \_displayBigDigit
  - aDSChannels, [28](#)
- \_displayBigDigits
  - aDSChannels, [28](#)
- \_enableData
  - aDSChannels, [28](#)
- \_enableDataCheck
  - aDSChannels, [29](#)
- \_getTempFromEEPROM
  - aDSChannels, [29](#)
- \_handleSerialInput
  - aDSEngine, [39](#)
- \_read
  - aDSChannels, [29](#)
- \_restoreCalibrationFromEEPROM
  - aDSChannels, [29](#)
- \_scissor
  - aDSChannels, [30](#)
- \_setTempToEEPROM
  - aDSChannels, [30](#)
- \_showBanner
  - aDSChannels, [30](#)
- \_turnOffPWM

- aDSChannel, [17](#)
- \_updateDisplay
  - aDSChannels, [30](#)
- \_updateField
  - aDSChannels, [30](#)
- \_wakeupFromStandby
  - aDSChannels, [31](#)
- \_write
  - aDSChannels, [31](#)
- \_writeMagicNumbers
  - aDSChannels, [31](#)
- aDSChannel
  - HEATING\_STATE\_COOLING, [16](#)
  - HEATING\_STATE\_HEATING, [16](#)
  - HEATING\_STATE\_REACHED, [16](#)
  - HEATING\_STATE\_STANDBY, [16](#)
- aDSChannels
  - CHANNEL\_MAX, [26](#)
  - CHANNEL\_ONE, [26](#)
  - CHANNEL\_TWO, [26](#)
- aDSEngine.h
  - OPERATION\_MODE\_READ, [47](#)
  - OPERATION\_MODE\_SET, [47](#)
  - OPERATION\_MODE\_UNKNOWN, [47](#)
- ADC\_TO\_TEMP\_GAIN
  - aDSChannel, [20](#)
- aDSChannel, [13](#)
  - ~aDSChannel, [16](#)
  - \_analogRead, [16](#)
  - \_analogWrite, [16](#)
  - \_digitalRead, [17](#)
  - \_digitalWrite, [17](#)
  - \_turnOffPWM, [17](#)
  - ADC\_TO\_TEMP\_GAIN, [20](#)
  - aDSChannel, [16](#)
  - aDSChannel, [16](#)
  - BLINK\_UPDATE\_RATE, [20](#)
  - CNTRL\_GAIN, [20](#)
  - getADCValue, [17](#)
  - getCalibration, [17](#)
  - getHeatState, [18](#)
  - getLEDState, [18](#)
  - getStandbyMode, [18](#)
  - getTemperature, [18](#)
  - hasFocus, [18](#)
  - HeatingState\_t, [16](#)
  - isTempHasChanged, [18](#)
  - m\_adcValue, [21](#)
  - m\_blinkStandby, [21](#)



- m\_cal, 21
- m\_channel, 21
- m\_currentTemp, 21
- m\_hasFocus, 21
- m\_heatState, 21
- m\_inStandby, 21
- m\_ledPin, 21
- m\_ledState, 21
- m\_nextBlink, 21
- m\_nextLowering, 21
- m\_nextPass, 21
- m\_nextTempStep, 21
- m\_pwmPin, 21
- m\_pwmValue, 21
- m\_ref, 21
- m\_sensorPin, 21
- m\_targetTemp, 21
- m\_tempHasChanged, 21
- PWM\_MAX\_VALUE, 21
- service, 19
- setCalibration, 19
- setFocus, 19
- setStandbyMode, 19
- setTemperature, 19
- setup, 20
- syncTempChange, 20
- TEMPERATURE\_MAX, 21
- TEMPERATURE\_MIN, 21
- updateLEDState, 20
- aDSChannel::CalibrationData\_t, 41
  - offset, 42
  - slope, 42
- aDSChannel::aPin\_t, 40
  - mask, 41
  - outputRegister, 41
  - pin, 41
  - port, 41
  - timer, 41
- aDSChannels, 22
  - ~aDSChannels, 26
  - \_backupCalibrationFromEEPROM, 26
  - \_checkForMagicNumbers, 26
  - \_clearValue, 26
  - \_crc8, 28
  - \_displayBigDigit, 28
  - \_displayBigDigits, 28
  - \_enableData, 28
  - \_enableDataCheck, 29
  - \_getTempFromEEPROM, 29
  - \_read, 29
  - \_restoreCalibrationFromEEPROM, 29
  - \_scissor, 30
  - \_setTempToEEPROM, 30
  - \_showBanner, 30
  - \_updateDisplay, 30
  - \_updateField, 30
  - \_wakeupFromStandby, 31
  - \_write, 31
  - \_writeMagicNumbers, 31
- aDSChannels, 26
- aDSChannels, 26
- Channel\_t, 26
- DATA\_DISPLAY, 36
- DATA\_FOCUS, 36
- DATA\_OPERATION, 36
- DATA\_STANDBY, 36
- EEPROM\_ADDR\_MAGIC, 37
- EEPROM\_TEMP\_SIZE, 37
- getCalibrationValues, 31
- getOperationMode, 31
- incEncoderPosition, 32
- isInCalibration, 32
- isInStandby, 32
- isJoined, 32
- m\_channels, 37
- m\_datas, 37
- m\_isValidEEPROM, 37
- m\_lastTempChange, 37
- m\_lcd, 37
- m\_lcdCols, 37
- m\_lcdRows, 37
- m\_nextDisplayUpdate, 37
- m\_nextMeasureUpdate, 37
- m\_operationMode, 37
- m\_operationTick, 38
- m\_storedToEEPROM, 38
- OFFSET\_VALUE, 38
- pingOperationMode, 32
- restoreCalibrationValues, 32
- saveCalibrationValues, 33
- service, 33
- setCalibrationMode, 33
- setCalibrationValues, 33
- setFocusToNextChannel, 33
- setOperationMode, 34
- setup, 34
- syncData, 34
- toggleJoined, 34
- toggleStandbyMode, 35
- updateOperationMode, 35
- aDSChannels::\_eepromCalibrationValue\_t, 13
  - c, 13
  - v, 13
- aDSEngine, 38
  - ~aDSEngine, 39
  - \_handleSerialInput, 39
  - aDSEngine, 39
  - aDSEngine, 39
  - m\_RXbuffer, 40
  - m\_RXoffset, 40
  - m\_channels, 40
  - m\_datas, 40
  - m\_encoder, 40
  - m\_serialInputTick, 40
  - RXBUFFER\_MAXLEN, 40
  - run, 39

- setup, [39](#)
- aDSEngine.cpp, [43](#)
  - \_bigDigitsBottom, [44](#)
  - \_bigDigitsTop, [44](#)
  - DIGIT\_WIDTH, [45](#)
  - getNumericalLength, [44](#)
  - pEncoder, [45](#)
  - PROGMEM, [45](#)
  - timer1ISR, [44](#)
- aDSEngine.h, [45](#)
  - ENCODER\_A\_PIN, [47](#)
  - ENCODER\_B\_PIN, [47](#)
  - ENCODER\_PB\_PIN, [48](#)
  - IS\_DATA\_ENABLED, [47](#)
  - LCD\_COLS, [48](#)
  - LCD\_D4\_PIN, [48](#)
  - LCD\_D5\_PIN, [48](#)
  - LCD\_D6\_PIN, [48](#)
  - LCD\_D7\_PIN, [48](#)
  - LCD\_ENABLE\_PIN, [48](#)
  - LCD\_ROWS, [48](#)
  - LCD\_RS\_PIN, [48](#)
  - LED\_CHANNEL1\_PIN, [48](#)
  - LED\_CHANNEL2\_PIN, [48](#)
  - OperationMode\_t, [47](#)
  - PWM\_CHANNEL1\_PIN, [49](#)
  - PWM\_CHANNEL2\_PIN, [49](#)
  - SIMU, [47](#)
- BLINK\_UPDATE\_RATE
  - aDSChannel, [20](#)
- c
  - aDSChannels::\_eepromCalibrationValue\_t, [13](#)
- CHANNEL\_MAX
  - aDSChannels, [26](#)
- CHANNEL\_ONE
  - aDSChannels, [26](#)
- CHANNEL\_TWO
  - aDSChannels, [26](#)
- CDC.cpp, [49](#)
- CHANNEL2\_ENABLE\_PIN
  - aDSEngine.h, [47](#)
- CNTRL\_GAIN
  - aDSChannel, [20](#)
- Channel\_t
  - aDSChannels, [26](#)
- DATA\_DISPLAY
  - aDSChannels, [36](#)
- DATA\_FOCUS
  - aDSChannels, [36](#)
- DATA\_OPERATION
  - aDSChannels, [36](#)
- DATA\_STANDBY
  - aDSChannels, [36](#)
- DIGIT\_WIDTH
  - aDSEngine.cpp, [45](#)
- delay
  - wiring.c, [52](#)
- delayMicroseconds
  - wiring.c, [52](#)
- EEPROM\_ADDR\_MAGIC
  - aDSChannels, [37](#)
- EEPROM\_TEMP\_SIZE
  - aDSChannels, [37](#)
- ENCODER\_A\_PIN
  - aDSEngine.h, [47](#)
- ENCODER\_B\_PIN
  - aDSEngine.h, [47](#)
- ENCODER\_PB\_PIN
  - aDSEngine.h, [48](#)
- engine
  - sketch.cpp, [50](#)
- f
  - wiring.c, [52](#)
- FRACT\_INC
  - wiring.c, [52](#)
- FRACT\_MAX
  - wiring.c, [52](#)
- getADCValue
  - aDSChannel, [17](#)
- getCalibration
  - aDSChannel, [17](#)
- getCalibrationValues
  - aDSChannels, [31](#)
- getHeatState
  - aDSChannel, [18](#)
- getLEDState
  - aDSChannel, [18](#)
- getNumericalLength
  - aDSEngine.cpp, [44](#)
- getOperationMode
  - aDSChannels, [31](#)
- getStandbyMode
  - aDSChannel, [18](#)
- getTemperature
  - aDSChannel, [18](#)
- HEATING\_STATE\_COOLING
  - aDSChannel, [16](#)
- HEATING\_STATE\_HEATING
  - aDSChannel, [16](#)
- HEATING\_STATE\_REACHED
  - aDSChannel, [16](#)
- HEATING\_STATE\_STANDBY
  - aDSChannel, [16](#)
- HID.cpp, [49](#)
- HardwareSerial.cpp, [49](#)
- hasFocus
  - aDSChannel, [18](#)
- HeatingState\_t
  - aDSChannel, [16](#)
- IPAddress.cpp, [49](#)

- IS\_DATA\_ENABLED
  - aDSEngine.h, 47
- if
  - wiring.c, 52
- incEncoderPosition
  - aDSChannels, 32
- init
  - wiring.c, 52
- isInCalibration
  - aDSChannels, 32
- isInStandby
  - aDSChannels, 32
- isJoinded
  - aDSChannels, 32
- isTempHasChanged
  - aDSChannel, 18
- LCD\_COLS
  - aDSEngine.h, 48
- LCD\_D4\_PIN
  - aDSEngine.h, 48
- LCD\_D5\_PIN
  - aDSEngine.h, 48
- LCD\_D6\_PIN
  - aDSEngine.h, 48
- LCD\_D7\_PIN
  - aDSEngine.h, 48
- LCD\_ENABLE\_PIN
  - aDSEngine.h, 48
- LCD\_ROWS
  - aDSEngine.h, 48
- LCD\_RS\_PIN
  - aDSEngine.h, 48
- LED\_CHANNEL1\_PIN
  - aDSEngine.h, 48
- LED\_CHANNEL2\_PIN
  - aDSEngine.h, 48
- libraries.cpp, 50
- loop
  - sketch.cpp, 50
- m
  - wiring.c, 52
- m\_RXbuffer
  - aDSEngine, 40
- m\_RXoffset
  - aDSEngine, 40
- m\_adcValue
  - aDSChannel, 21
- m\_blinkStandby
  - aDSChannel, 21
- m\_cal
  - aDSChannel, 21
- m\_channel
  - aDSChannel, 21
- m\_channels
  - aDSChannels, 37
  - aDSEngine, 40
- m\_currentTemp
  - aDSChannel, 21
- m\_datas
  - aDSChannels, 37
  - aDSEngine, 40
- m\_encoder
  - aDSEngine, 40
- m\_hasFocus
  - aDSChannel, 21
- m\_heatState
  - aDSChannel, 21
- m\_inStandby
  - aDSChannel, 21
- m\_isValidEEPROM
  - aDSChannels, 37
- m\_lastTempChange
  - aDSChannels, 37
- m\_lcd
  - aDSChannels, 37
- m\_lcdCols
  - aDSChannels, 37
- m\_lcdRows
  - aDSChannels, 37
- m\_ledPin
  - aDSChannel, 21
- m\_ledState
  - aDSChannel, 21
- m\_nextBlink
  - aDSChannel, 21
- m\_nextDisplayUpdate
  - aDSChannels, 37
- m\_nextLowering
  - aDSChannel, 21
- m\_nextMeasureUpdate
  - aDSChannels, 37
- m\_nextPass
  - aDSChannel, 21
- m\_nextTempStep
  - aDSChannel, 21
- m\_operationMode
  - aDSChannels, 37
- m\_operationTick
  - aDSChannels, 38
- m\_pwmPin
  - aDSChannel, 21
- m\_pwmValue
  - aDSChannel, 21
- m\_ref
  - aDSChannel, 21
- m\_sensorPin
  - aDSChannel, 21
- m\_serialInputTick
  - aDSEngine, 40
- m\_storedToEEPROM
  - aDSChannels, 38
- m\_targetTemp
  - aDSChannel, 21
- m\_tempHasChanged
  - aDSChannel, 21

- MILLIS\_INC
  - wiring.c, 52
- main.cpp, 50
- mask
  - aDSChannel::aPin\_t, 41
- micros
  - wiring.c, 52
- millis
  - wiring.c, 52
- new.cpp, 50
- OPERATION\_MODE\_READ
  - aDSEngine.h, 47
- OPERATION\_MODE\_SET
  - aDSEngine.h, 47
- OPERATION\_MODE\_UNKNOWN
  - aDSEngine.h, 47
- OFFSET\_MARKER\_LEFT
  - aDSChannels, 38
- OFFSET\_VALUE
  - aDSChannels, 38
- offset
  - aDSChannel::CalibrationData\_t, 42
- OperationMode\_t
  - aDSEngine.h, 47
- outputRegister
  - aDSChannel::aPin\_t, 41
- pEncoder
  - aDSEngine.cpp, 45
- PROGMEM
  - aDSEngine.cpp, 45
- PWM\_CHANNEL1\_PIN
  - aDSEngine.h, 49
- PWM\_CHANNEL2\_PIN
  - aDSEngine.h, 49
- PWM\_MAX\_VALUE
  - aDSChannel, 21
- pin
  - aDSChannel::aPin\_t, 41
- pingOperationMode
  - aDSChannels, 32
- port
  - aDSChannel::aPin\_t, 41
- Print.cpp, 50
- RXBUFFER\_MAXLEN
  - aDSEngine, 40
- restoreCalibrationValues
  - aDSChannels, 32
- run
  - aDSEngine, 39
- SIMU
  - aDSEngine.h, 47
- saveCalibrationValues
  - aDSChannels, 33
- service
  - aDSChannel, 19
  - aDSChannels, 33
- setCalibration
  - aDSChannel, 19
- setCalibrationMode
  - aDSChannels, 33
- setCalibrationValues
  - aDSChannels, 33
- setFocus
  - aDSChannel, 19
- setFocusToNextChannel
  - aDSChannels, 33
- setOperationMode
  - aDSChannels, 34
- setStandbyMode
  - aDSChannel, 19
- setTemperature
  - aDSChannel, 19
- setup
  - aDSChannel, 20
  - aDSChannels, 34
  - aDSEngine, 39
  - sketch.cpp, 50
- sketch.cpp, 50
  - engine, 50
  - loop, 50
  - setup, 50
- slope
  - aDSChannel::CalibrationData\_t, 42
- Stream.cpp, 51
- syncData
  - aDSChannels, 34
- syncTempChange
  - aDSChannel, 20
- TEMPERATURE\_MAX
  - aDSChannel, 21
- TEMPERATURE\_MIN
  - aDSChannel, 21
- TEMPERATURE\_STANDBY
  - aDSChannel, 21
- timer
  - aDSChannel::aPin\_t, 41
- timer0\_fract
  - wiring.c, 52
- timer0\_millis
  - wiring.c, 52
- timer0\_overflow\_count
  - wiring.c, 52
- timer1ISR
  - aDSEngine.cpp, 44
- toggleJoined
  - aDSChannels, 34
- toggleStandbyMode
  - aDSChannels, 35
- Tone.cpp, 51
- USBCore.cpp, 51
- updateLEDState

- aDSChannel, [20](#)
- updateOperationMode
  - aDSChannels, [35](#)
- v
  - aDSChannels::\_eepromCalibrationValue\_t, [13](#)
- WInterrupts.c, [51](#)
- WMath.cpp, [53](#)
- WString.cpp, [53](#)
- wiring.c, [51](#)
  - delay, [52](#)
  - delayMicroseconds, [52](#)
  - f, [52](#)
  - FRACT\_INC, [52](#)
  - FRACT\_MAX, [52](#)
  - if, [52](#)
  - init, [52](#)
  - m, [52](#)
  - MILLIS\_INC, [52](#)
  - micros, [52](#)
  - millis, [52](#)
  - timer0\_fract, [52](#)
  - timer0\_millis, [52](#)
  - timer0\_overflow\_count, [52](#)
- wiring\_analog.c, [52](#)
- wiring\_digital.c, [52](#)
- wiring\_pulse.c, [52](#)
- wiring\_shift.c, [53](#)