

Manual for aWXIron

1.2

Written by F1RMB, Daniel Caujolle-Bert ©2015

Contents

1	SMD Soldering Station for Weller RT Series Tips	1
2	User Interface overview	3
2.1	Encoder usage	3
2.2	Joined mode	4
2.3	Standby	4
3	Calibration Process	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Class Documentation	11
6.1	aDSChannels::_eepromCalibrationValue_t Union Reference	11
6.1.1	Detailed Description	11
6.1.2	Member Data Documentation	11
6.1.2.1	c	11
6.1.2.2	v	11
6.2	aDSChannel Class Reference	11
6.2.1	Detailed Description	14
6.2.2	Member Enumeration Documentation	14
6.2.2.1	HeatingState_t	14
6.2.3	Constructor & Destructor Documentation	14
6.2.3.1	aDSChannel	14
6.2.3.2	~aDSChannel	14
6.2.4	Member Function Documentation	14
6.2.4.1	_analogRead	14
6.2.4.2	_analogWrite	14
6.2.4.3	_digitalRead	15
6.2.4.4	_digitalWrite	15

6.2.4.5	_turnOffPWM	15
6.2.4.6	_turnPWM	15
6.2.4.7	getADCValue	15
6.2.4.8	getCalibration	16
6.2.4.9	getHeatState	16
6.2.4.10	getLEDState	16
6.2.4.11	getStandbyMode	16
6.2.4.12	getTemperature	16
6.2.4.13	hasFocus	16
6.2.4.14	isPlugged	17
6.2.4.15	isTempHasChanged	17
6.2.4.16	service	17
6.2.4.17	setBrother	17
6.2.4.18	setCalibration	17
6.2.4.19	setFocus	17
6.2.4.20	setStandbyMode	18
6.2.4.21	setTemperature	18
6.2.4.22	setup	18
6.2.4.23	syncTempChange	18
6.2.4.24	updateLEDState	19
6.2.5	Member Data Documentation	20
6.2.5.1	BLINK_UPDATE_RATE	20
6.2.5.2	DEFAULT_TEMPERATURE_OFFSET	20
6.2.5.3	DEFAULT_TEMPERATURE_SLOPE	20
6.2.5.4	m_adcValue	20
6.2.5.5	m_blinkStandby	20
6.2.5.6	m_brother	20
6.2.5.7	m_cal	20
6.2.5.8	m_channel	20
6.2.5.9	m_currentTemp	20
6.2.5.10	m_hasFocus	20
6.2.5.11	m_heatState	20
6.2.5.12	m_inStandby	20
6.2.5.13	m_isPlugged	20
6.2.5.14	m_ledPin	20
6.2.5.15	m_ledState	20
6.2.5.16	m_nextBlink	20
6.2.5.17	m_nextPass	20
6.2.5.18	m_pwmPin	20
6.2.5.19	m_pwmValue	20

6.2.5.20	m_ref	21
6.2.5.21	m_sensorPin	21
6.2.5.22	m_targetTemp	21
6.2.5.23	m_tempHasChanged	21
6.2.5.24	PWM_MAX_VALUE	21
6.2.5.25	TEMPERATURE_MAX	21
6.2.5.26	TEMPERATURE_MIN	21
6.2.5.27	TEMPERATURE_STANDBY	21
6.2.5.28	TEMPERATURE_TOLERANCE	21
6.3	aDSChannels Class Reference	21
6.3.1	Member Enumeration Documentation	25
6.3.1.1	Channel_t	25
6.3.2	Constructor & Destructor Documentation	25
6.3.2.1	aDSChannels	25
6.3.2.2	~aDSChannels	26
6.3.3	Member Function Documentation	26
6.3.3.1	_backupCalibrationFromEEPROM	26
6.3.3.2	_checkForMagicNumbers	26
6.3.3.3	_clearValue	26
6.3.3.4	_crc8	26
6.3.3.5	_displayBigDigit	27
6.3.3.6	_displayBigDigits	27
6.3.3.7	_enableData	27
6.3.3.8	_enableDataCheck	27
6.3.3.9	_getTempFromEEPROM	28
6.3.3.10	_read	28
6.3.3.11	_restoreCalibrationFromEEPROM	28
6.3.3.12	_scissor	28
6.3.3.13	_setTempToEEPROM	29
6.3.3.14	_showBanner	29
6.3.3.15	_updateDisplay	29
6.3.3.16	_updateField	29
6.3.3.17	_wakeupFromStandby	30
6.3.3.18	_write	30
6.3.3.19	_writeMagicNumbers	30
6.3.3.20	getCalibrationValues	30
6.3.3.21	getOperationMode	30
6.3.3.22	incEncoderPosition	30
6.3.3.23	isInCalibration	31
6.3.3.24	isInStandby	31

6.3.3.25	isJoined	31
6.3.3.26	pingOperationMode	31
6.3.3.27	restoreCalibrationValues	31
6.3.3.28	saveCalibrationValues	31
6.3.3.29	service	32
6.3.3.30	setCalibrationMode	32
6.3.3.31	setCalibrationValues	32
6.3.3.32	setFocusToNextChannel	32
6.3.3.33	setOperationMode	32
6.3.3.34	setup	33
6.3.3.35	syncData	33
6.3.3.36	toggleJoined	33
6.3.3.37	toggleStandbyMode	33
6.3.3.38	updateOperationMode	34
6.3.4	Member Data Documentation	34
6.3.4.1	DATA_CHANNEL1_LED_STATE	34
6.3.4.2	DATA_CHANNEL1_TEMP_READ	34
6.3.4.3	DATA_CHANNEL1_TEMP_SET	34
6.3.4.4	DATA_CHANNEL2_ENABLED	34
6.3.4.5	DATA_CHANNEL2_LED_STATE	34
6.3.4.6	DATA_CHANNEL2_TEMP_READ	34
6.3.4.7	DATA_CHANNEL2_TEMP_SET	34
6.3.4.8	DATA_CHANNELS_JOINED	34
6.3.4.9	DATA_DISPLAY	34
6.3.4.10	DATA_DISPLAY_STANDBY	34
6.3.4.11	DATA_FOCUS	35
6.3.4.12	DATA_IN_CALIBRATION	35
6.3.4.13	DATA_OPERATION	35
6.3.4.14	DATA_STANDBY	35
6.3.4.15	DISPLAY_UPDATE_RATE	35
6.3.4.16	EEPROM_ADDR_CALIBRATION_CHAN_1	35
6.3.4.17	EEPROM_ADDR_CALIBRATION_CHAN_2	35
6.3.4.18	EEPROM_ADDR_CHANNEL_JOINED	35
6.3.4.19	EEPROM_ADDR_MAGIC	35
6.3.4.20	EEPROM_ADDR_TEMP_CHANNEL_ONE	35
6.3.4.21	EEPROM_ADDR_TEMP_CHANNEL_TWO	35
6.3.4.22	EEPROM_CALIBRATION_SIZE	36
6.3.4.23	EEPROM_STORAGE_STARTING	36
6.3.4.24	EEPROM_TEMP_SIZE	36
6.3.4.25	m_channels	36

6.3.4.26	<code>m_datas</code>	36
6.3.4.27	<code>m_isValidEEPROM</code>	36
6.3.4.28	<code>m_lastTempChange</code>	36
6.3.4.29	<code>m_lcd</code>	36
6.3.4.30	<code>m_lcdCols</code>	36
6.3.4.31	<code>m_lcdRows</code>	36
6.3.4.32	<code>m_nextDisplayUpdate</code>	36
6.3.4.33	<code>m_nextMeasureUpdate</code>	36
6.3.4.34	<code>m_operationMode</code>	36
6.3.4.35	<code>m_operationTick</code>	36
6.3.4.36	<code>m_storedToEEPROM</code>	36
6.3.4.37	<code>MEASURE_UPDATE_RATE</code>	36
6.3.4.38	<code>OFFSET_MARKER_LEFT</code>	36
6.3.4.39	<code>OFFSET_MARKER_RIGHT</code>	36
6.3.4.40	<code>OFFSET_VALUE</code>	36
6.3.4.41	<code>OPERATION_SET_TIMEOUT</code>	37
6.3.4.42	<code>TEMP_SETTING_INACTIVITY</code>	37
6.4	<code>aDSEngine</code> Class Reference	37
6.4.1	Constructor & Destructor Documentation	38
6.4.1.1	<code>aDSEngine</code>	38
6.4.1.2	<code>~aDSEngine</code>	38
6.4.2	Member Function Documentation	38
6.4.2.1	<code>_handleSerialInput</code>	38
6.4.2.2	<code>run</code>	38
6.4.2.3	<code>setup</code>	38
6.4.3	Member Data Documentation	38
6.4.3.1	<code>m_channels</code>	38
6.4.3.2	<code>m_datas</code>	38
6.4.3.3	<code>m_encoder</code>	38
6.4.3.4	<code>m_RXbuffer</code>	38
6.4.3.5	<code>m_RXoffset</code>	38
6.4.3.6	<code>m_serialInputTick</code>	39
6.4.3.7	<code>RXBUFFER_MAXLEN</code>	39
6.5	<code>aDSChannel::aPin_t</code> Struct Reference	39
6.5.1	Detailed Description	39
6.5.2	Member Data Documentation	39
6.5.2.1	<code>mask</code>	39
6.5.2.2	<code>outputRegister</code>	39
6.5.2.3	<code>pin</code>	39
6.5.2.4	<code>port</code>	39

6.5.2.5	timer	40
6.6	aDSChannel::CalibrationData_t Struct Reference	40
6.6.1	Detailed Description	40
6.6.2	Member Data Documentation	40
6.6.2.1	offset	40
6.6.2.2	slope	40
7	File Documentation	41
7.1	aDSEngine.cpp File Reference	41
7.1.1	Detailed Description	41
7.1.2	Function Documentation	42
7.1.2.1	getNumericalLength	42
7.1.2.2	timer1ISR	42
7.1.3	Variable Documentation	42
7.1.3.1	_bigDigitsBottom	42
7.1.3.2	_bigDigitsTop	43
7.1.3.3	channelCount	43
7.1.3.4	DIGIT_WIDTH	43
7.1.3.5	pEncoder	43
7.1.3.6	PROGMEM	43
7.2	aDSEngine.h File Reference	43
7.2.1	Detailed Description	45
7.2.2	Macro Definition Documentation	45
7.2.2.1	IS_DATA_ENABLED	45
7.2.3	Enumeration Type Documentation	45
7.2.3.1	OperationMode_t	45
7.2.4	Variable Documentation	45
7.2.4.1	CHANNEL2_ENABLE_PIN	45
7.2.4.2	ENCODER_A_PIN	45
7.2.4.3	ENCODER_B_PIN	46
7.2.4.4	ENCODER_PB_PIN	46
7.2.4.5	ENCODER_STEPS_PER_NOTCH	46
7.2.4.6	LCD_COLS	46
7.2.4.7	LCD_D4_PIN	46
7.2.4.8	LCD_D5_PIN	46
7.2.4.9	LCD_D6_PIN	46
7.2.4.10	LCD_D7_PIN	46
7.2.4.11	LCD_ENABLE_PIN	46
7.2.4.12	LCD_ROWS	46
7.2.4.13	LCD_RS_PIN	46

7.2.4.14	LED_CHANNEL1_PIN	46
7.2.4.15	LED_CHANNEL2_PIN	47
7.2.4.16	PROGRAM_VERSION_MAJOR	47
7.2.4.17	PROGRAM_VERSION_MINOR	47
7.2.4.18	PWM_CHANNEL1_PIN	47
7.2.4.19	PWM_CHANNEL2_PIN	47
7.2.4.20	TEMP_SENSOR_CHANNEL1_PIN	47
7.2.4.21	TEMP_SENSOR_CHANNEL2_PIN	47
7.3	CDC.cpp File Reference	47
7.4	HardwareSerial.cpp File Reference	47
7.5	HID.cpp File Reference	47
7.6	IPAddress.cpp File Reference	47
7.7	libraries.cpp File Reference	48
7.8	main.cpp File Reference	48
7.9	new.cpp File Reference	48
7.10	Print.cpp File Reference	48
7.11	sketch.cpp File Reference	48
7.11.1	Function Documentation	48
7.11.1.1	loop	48
7.11.1.2	setup	48
7.11.2	Variable Documentation	48
7.11.2.1	engine	48
7.12	Stream.cpp File Reference	49
7.13	Tone.cpp File Reference	49
7.14	USBCore.cpp File Reference	49
7.15	WInterrupts.c File Reference	49
7.16	wiring.c File Reference	49
7.16.1	Macro Definition Documentation	50
7.16.1.1	FRACT_INC	50
7.16.1.2	FRACT_MAX	50
7.16.1.3	MICROSECONDS_PER_TIMER0_OVERFLOW	50
7.16.1.4	MILLIS_INC	50
7.16.2	Function Documentation	50
7.16.2.1	delay	50
7.16.2.2	delayMicroseconds	50
7.16.2.3	if	50
7.16.2.4	init	50
7.16.2.5	micros	50
7.16.2.6	millis	50
7.16.3	Variable Documentation	50

7.16.3.1	f	50
7.16.3.2	m	50
7.16.3.3	timer0_fract	50
7.16.3.4	timer0_millis	50
7.16.3.5	timer0_overflow_count	50
7.17	wiring_analog.c File Reference	50
7.18	wiring_digital.c File Reference	50
7.19	wiring_pulse.c File Reference	50
7.20	wiring_shift.c File Reference	51
7.21	WMath.cpp File Reference	51
7.22	WString.cpp File Reference	51
Index		52

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.

The hardware has been redesigned and modified (two channels, 16x2 LCD instead of 7 segments display, etc). The software has also been 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 handle one or two soldering irons:
 1. In Single mode, the temperature reading and setting are displayed using double height font.
 2. In dual channels version, both soldering irons can be controlled separately, or can be joined:
 - when separate channels mode is used, each channel is independent. Simple clicking on the encoder push button will set the focus to the next channel. The focused channel's 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).
- 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 Standby)

- 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 30 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 will be resetted
- When the station is in temperature reading mode, the displayed values are left aligned. When the station is in settings mode, the displayed values are 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's temperature. A single encoder detents rotation will switch the soldering station into settings mode, and displays the target temperature without any change to the target temperature setting.

- When the soldering station is in settings mode, and no 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	switch to standby mode (see Standby)
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 Standby)
Held	toggles joined mode (see Joined mode)

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 exits from *Standby* mode.
- When *Standby* mode is activated, the LEDs blink three times cyclically.

Chapter 3

Calibration Process

- **Prerequisites:**

- **Hardware:**

- * Digital Thermometer (e.g: your digital multimeter with a thermocouple)

- **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:**

The calibration process is necessary get accurate temperature control.

Some average values are used by default, but that won't gives you accurate temperature control.

- **Process Description:**

- **Step 1: Calibration Mode**

You have to open the soldering station's box and connect the soldering station to the PC, using a USB cable.

To turn the soldering station in calibration, you have to keep the encoder's push button pressed while turning ON the station. 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 anymore. Instead, the ADC value is shown.

The station will self set the target temperature to 100 °C. You have to wait until the temperature stabilizes.

- **Step 2: Calibrate Channel 1 or 2**

Select matching **Channel** tab in the calibration spreadsheet file.

Set the temperature target using the pseudo temperature value in the first column. Wait until LED blinks and the displayed ADC value stabilize.

Apply the thermocouple to the soldering tip, then write down the readed temperature to the column named '**Temp °C**'.

If needed, adjust the value in the '**ADCread**' column, accordingly to the one displayed on the station's LCD.

Apply the same procedure for all spreadsheet's rows.

Once you completed the array, down the chart, the '**Calibration String**' cell contains the string you have to copy and paste to the serial terminal emulator, e.g:

```
:CAL:1:0.3757498594,51.7808993467
```

This string starts with **:CAL:**, followed by the channel's name, then two floating point values, comma separated.

Once the string entered and validated with the **[RETURN]** key, you should get a **:OK:** acknowledge message.

In case you get **:ERR:**, double check the calibration string you pasted.

If you own a Dual Channel soldering station, **repeat this step for the second Channel.**

– **Last Step: Backup**

Once the full calibration is done, you **HAVE** to store the new values into the EEPROM, using the following command:

```
:CAL:SAVE
```

As usual, you should get a **:OK:** acknowledge message.

Once you validate the calibration with this command, you leave the calibration mode.

You can unplug the USB cable, close the box and use your soldering station now.

• **Other available calibration commands:**

- **:CAL:OFF** Cancels the calibration process, restoring previous values.
- **:CAL:DUMP** Displays the calibration values in the serial terminal emulator

Warning

If you own a Dual Channel soldering station, you **HAVE** to calibrate both channels, or at least enter the old calibration string for the channel you won't calibrate.

Chapter 4

Class Index

4.1 Class List

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

aDSChannels::_eepromCalibrationValue_t	
Union to manipulate float/uint8_t [] calibration values	11
aDSChannel	
ADSCchannel class	11
aDSChannels	21
aDSEngine	37
aDSChannel::aPin_t	
Our pin structure	39
aDSChannel::CalibrationData_t	
Calibration values	40

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

aDSEngine.cpp	41
aDSEngine.h	43
CDC.cpp	47
HardwareSerial.cpp	47
HID.cpp	47
IPAddress.cpp	47
libraries.cpp	48
main.cpp	48
new.cpp	48
Print.cpp	48
sketch.cpp	48
Stream.cpp	49
Tone.cpp	49
USBCore.cpp	49
WInterrupts.c	49
wiring.c	49
wiring_analog.c	50
wiring_digital.c	50
wiring_pulse.c	50
wiring_shift.c	51
WMath.cpp	51
WString.cpp	51

Chapter 6

Class Documentation

6.1 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)]

6.1.1 Detailed Description

Union to manipulate float/uint8_t [] calibration values.

6.1.2 Member Data Documentation

6.1.2.1 uint8_t aDSChannels::_eepromCalibrationValue_t::c[sizeof(float)]

6.1.2.2 float aDSChannels::_eepromCalibrationValue_t::v

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

- [aDSEngine.h](#)

6.2 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.
- bool [isPlugged](#) ()
Get is tip plugged state.
- void [setCalibration](#) (float, float)
Set calibration data values.
- const [CalibrationData_t](#) [getCalibration](#) () const
Get calibration data values.
- int16_t [getADCValue](#) ()
Get latest ADC value.
- void [setBrother](#) ([aDSChannel](#) *)
Set relationship.

Static Public Attributes

- static const int16_t [TEMPERATURE_MIN](#) = 100
Minimum temperature.
- static const int16_t [TEMPERATURE_MAX](#) = 450
Maximum temperature.
- static const int16_t [TEMPERATURE_STANDBY](#) = 150
Standby temperature.
- 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
Default slope value, used for ADC to Temp correction.
- static const float [DEFAULT_TEMPERATURE_OFFSET](#) = 43.8279285472
Default offset value, used for ADC to Temp correction.
- static const int16_t [PWM_MAX_VALUE](#) = 150
Maximum PWM value.

Protected Member Functions

- void [_turnOffPWM](#) (aPin_t)
Turns PWM off for the given pin.
- void [_turnPWM](#) (bool)
- 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](#)
- bool [m_isPlugged](#)
- [aDSChannel * m_brother](#)

6.2.1 Detailed Description

[aDSChannel](#) class

6.2.2 Member Enumeration Documentation

6.2.2.1 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.

6.2.3 Constructor & Destructor Documentation

6.2.3.1 `aDSChannel::aDSChannel ()`

[aDSChannel](#) class constructor

6.2.3.2 `aDSChannel::~~aDSChannel ()` [virtual]

[aDSChannel](#) destructor

6.2.4 Member Function Documentation

6.2.4.1 `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>	<code>aPin_t</code> : pin
------------	---------------------------

Returns

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

6.2.4.2 `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>	aPin_t : pin
<i>val</i>	uint8_t : analog value (0..255)

Returns

void

6.2.4.3 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>	aPin_t : pin
------------	------------------------------

Returns

int8_t : HIGH or LOW

6.2.4.4 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>	aPin_t : pin
<i>val</i>	uint8_t : state (HIGH or LOW)

Returns

void

6.2.4.5 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>	aPin_t : pin
------------	------------------------------

Returns

void

6.2.4.6 void aDSChannel::_turnPWM (bool enable) [protected]**6.2.4.7 int16_t aDSChannel::getADCValue ()**

Get latest ADC value.

Returns

int16_t : ADC value

6.2.4.8 `const aDSChannel::CalibrationData_t aDSChannel::getCalibration () const`

Get calibration data values.

Returns

`const aDSChannel::CalibrationData_t` : Calibration data

6.2.4.9 `aDSChannel::HeatingState_t aDSChannel::getHeatState ()`

Get channel heating state.

Returns

`aDSChannel::HeatingState_t` : heating state

6.2.4.10 `uint8_t aDSChannel::getLEDState ()`

Get state LED status.

Returns

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

6.2.4.11 `bool aDSChannel::getStandbyMode ()`

Get channel's standby enableity.

Returns

`bool`

6.2.4.12 `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

6.2.4.13 `bool aDSChannel::hasFocus ()`

Get the focus state.

Returns

`bool` : focus state

6.2.4.14 bool aDSChannel::isPlugged ()

Get is tip plugged state.

Returns

bool : plugged state

6.2.4.15 bool aDSChannel::isTempHasChanged ()

Is target temperature has changed (use for EEPROM storage)

Returns

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

6.2.4.16 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

6.2.4.17 void aDSChannel::setBrother (aDSChannel * *p*)

Set relationship.

Parameters

<i>p</i>	aDSChannel* : pointer to other aDSChannel
----------	---

Returns

void

6.2.4.18 void aDSChannel::setCalibration (float *slope*, float *offset*)

Set calibration data values.

Parameters

<i>slope</i>	float : slope value
<i>offset</i>	float : offset value

Returns

void

6.2.4.19 void aDSChannel::setFocus (bool *v*)

Set the focus, as display point of view.

Parameters

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

Returns

void

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

Enable or disable channel's standby.

Parameters

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

Returns

void

6.2.4.21 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

6.2.4.22 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

6.2.4.23 void aDSChannel::syncTempChange ()

Reset temperature change flag (use for EEPROM storage)

Returns

void

6.2.4.24 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)

6.2.5 Member Data Documentation

6.2.5.1 `const unsigned long aDSChannel::BLINK_UPDATE_RATE = 400` [static]

Update rate for LED blinking, in ms.

6.2.5.2 `const float aDSChannel::DEFAULT_TEMPERATURE_OFFSET = 43.8279285472` [static]

Default offset value, used for ADC to Temp correction.

6.2.5.3 `const float aDSChannel::DEFAULT_TEMPERATURE_SLOPE = 0.3947387545` [static]

Default slope value, used for ADC to Temp correction.

6.2.5.4 `int16_t aDSChannel::m_adcValue` [private]

6.2.5.5 `uint8_t aDSChannel::m_blinkStandby` [private]

6.2.5.6 `aDSChannel* aDSChannel::m_brother` [private]

6.2.5.7 `CalibrationData_t aDSChannel::m_cal` [private]

6.2.5.8 `uint8_t aDSChannel::m_channel` [private]

6.2.5.9 `int16_t aDSChannel::m_currentTemp` [private]

6.2.5.10 `bool aDSChannel::m_hasFocus` [private]

6.2.5.11 `HeatingState_t aDSChannel::m_heatState` [private]

6.2.5.12 `bool aDSChannel::m_inStandby` [private]

6.2.5.13 `bool aDSChannel::m_isPlugged` [private]

6.2.5.14 `aPin_t aDSChannel::m_ledPin` [private]

6.2.5.15 `uint8_t aDSChannel::m_ledState` [private]

6.2.5.16 `unsigned long aDSChannel::m_nextBlink` [private]

6.2.5.17 `unsigned long aDSChannel::m_nextPass` [private]

6.2.5.18 `aPin_t aDSChannel::m_pwmPin` [private]

6.2.5.19 `int8_t aDSChannel::m_pwmValue` [private]

6.2.5.20 `uint8_t aDSChannel::m_ref` [private]

6.2.5.21 `aPin_t aDSChannel::m_sensorPin` [private]

6.2.5.22 `int16_t aDSChannel::m_targetTemp` [private]

6.2.5.23 `bool aDSChannel::m_tempHasChanged` [private]

6.2.5.24 `const int16_t aDSChannel::PWM_MAX_VALUE = 150` [static]

Maximum PWM value.

6.2.5.25 `const int16_t aDSChannel::TEMPERATURE_MAX = 450` [static]

Maximum temperature.

6.2.5.26 `const int16_t aDSChannel::TEMPERATURE_MIN = 100` [static]

Minimum temperature.

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

Standby temperature.

6.2.5.28 `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](#)

6.3 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 enability.
- 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** = 30000
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 joined.
- 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](#) (int16_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`

6.3.1 Member Enumeration Documentation

6.3.1.1 enum aDSChannels::Channel_t

Channels enumeration.

Enumerator

`CHANNEL_ONE`
`CHANNEL_TWO`
`CHANNEL_MAX`

6.3.2 Constructor & Destructor Documentation

6.3.2.1 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

6.3.2.2 aDSChannels::~~aDSChannels () [virtual]

[aDSChannels](#) destructor

6.3.3 Member Function Documentation

6.3.3.1 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>	aDSChannel& : channel

Returns

void

6.3.3.2 bool aDSChannels::_checkForMagicNumbers () [protected]

Check for the magic number in the EEPROM.

Returns

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

6.3.3.3 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

6.3.3.4 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

Parameters

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

Returns

uint8_t : **CRC**

6.3.3.5 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

6.3.3.6 void aDSChannels::_displayBigDigits (int16_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

6.3.3.7 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

6.3.3.8 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 enability

Returns

void

6.3.3.9 `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

6.3.3.10 `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 <typename T> bool : true on read success, otherwise false

6.3.3.11 `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

6.3.3.12 `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 <typename T> void

6.3.3.13 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

6.3.3.14 void aDSChannels::_showBanner () [protected]

Display a banner on the LCD.

Returns

void

6.3.3.15 void aDSChannels::_updateDisplay () [protected]

Update LCD display, if needed.

Returns

void

6.3.3.16 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

6.3.3.17 void aDSChannels::_wakeupFromStandby () [protected]

Wake up from standby mode.

Returns

void

6.3.3.18 template<typename T> bool aDSChannels::_write (T const v, int16_t & 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 <typename T> bool : true on write success, otherwise false

6.3.3.19 void aDSChannels::_writeMagicNumbers () [protected]

Write magic numbers into EEPROM.

Returns

void

6.3.3.20 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

6.3.3.21 OperationMode_t aDSChannels::getOperationMode ()

Get current operation mode.

Returns

OperationMode_t : operation mode

6.3.3.22 void aDSChannels::incEncoderPosition (uint16_t v)

Increment or decrement encoder position.

Parameters

<i>v</i>	uint16_t : increment value (signed)
----------	-------------------------------------

Returns

void

6.3.3.23 bool aDSChannels::isInCalibration ()

Get calibration mode enable.

Returns

bool : true if in calibration mode, otherwise false

6.3.3.24 bool aDSChannels::isInStandby ()

Get standby mode.

Returns

bool : true if in standby mode, otherwise false

6.3.3.25 bool aDSChannels::isJoined ()

Get channel joining state.

Returns

bool : true if joined, otherwise false

6.3.3.26 void aDSChannels::pingOperationMode ()

Reset timeout for OPERATION_MODE_SET mode.

Returns

void

6.3.3.27 void aDSChannels::restoreCalibrationValues ()

Restore calibration values from EEPROM.

Returns

void

6.3.3.28 void aDSChannels::saveCalibrationValues (Channel_t chan)

Save calibration value of given into EEPROM.

Parameters

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

Returns

void

6.3.3.29 void aDSChannels::service ()

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

Returns

void

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

Set calibration mode.

Parameters

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

Returns

void

6.3.3.31 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>	aDSChannel::CalibrationData_t : calibration values

Returns

void

6.3.3.32 void aDSChannels::setFocusToNextChannel ()

Set the focus to the next channel, if any.

Returns

void

6.3.3.33 void aDSChannels::setOperationMode (OperationMode_t *m*)

Set operation mode (SET/READ)

Parameters

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

Returns

void

6.3.3.34 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 enable 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

6.3.3.35 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

6.3.3.36 void aDSChannels::toggleJoined ()

Toggle channels joining.

Returns

void

6.3.3.37 void aDSChannels::toggleStandbyMode ()

Toggle standby mode.

Returns

void

6.3.3.38 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

6.3.4 Member Data Documentation

6.3.4.1 const uint16_t aDSChannels::DATA_CHANNEL1_LED_STATE = 1 << 5 [static]

Bitfield: LED state of channel 1 has changed.

6.3.4.2 const uint16_t aDSChannels::DATA_CHANNEL1_TEMP_READ = 1 << 4 [static]

Bitfield: Readed temperature of channel 1 has changed.

6.3.4.3 const uint16_t aDSChannels::DATA_CHANNEL1_TEMP_SET = 1 << 3 [static]

Bitfield: Target temperature of channel 1 has changed.

6.3.4.4 const uint16_t aDSChannels::DATA_CHANNEL2_ENABLED = 1 [static]

Bitfield: Channel 2 is enabled.

6.3.4.5 const uint16_t aDSChannels::DATA_CHANNEL2_LED_STATE = 1 << 8 [static]

Bitfield: LED state of channel 1 has changed.

6.3.4.6 const uint16_t aDSChannels::DATA_CHANNEL2_TEMP_READ = 1 << 7 [static]

Bitfield: Readed temperature of channel 1 has changed.

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

Bitfield: Target temperature of channel 1 has changed.

6.3.4.8 const uint16_t aDSChannels::DATA_CHANNELS_JOINED = 1 << 1 [static]

Bitfield: Channel 1 & 2 are joined.

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

Bitfield: Display should be refreshed.

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

Bitfield: Standby state has changed.

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

Bitfield: Focus has changed.

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

Bitfield: in Calibration.

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

Bitfield: Operation mode has changed.

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

Bitfield: Standby state.

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

Display update rate, in ms.

6.3.4.16 `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.

6.3.4.17 `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.

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

Channels are joined.

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

EEPROM offset storage start for magic numbers (0xDEAD)

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

Target temp for Channel 1.

6.3.4.21 `const int16_t aDSChannels::EEPROM_ADDR_TEMP_CHANNEL_TWO = EEPROM_ADDR_TEMP_CHANNEL_O-
NE + EEPROM_TEMP_SIZE` [static], [protected]

Target temp for Channel 2.

6.3.4.22 `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.

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

EEPROM starting address for program datas.

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

EEPROM temperature size (temperature + crc)

6.3.4.25 `aDSChannel aDSChannels::m_channels[CHANNEL_MAX] [private]`

6.3.4.26 `uint16_t aDSChannels::m_datas [private]`

6.3.4.27 `bool aDSChannels::m_isValidEEPROM [private]`

6.3.4.28 `unsigned long aDSChannels::m_lastTempChange [private]`

6.3.4.29 `LiquidCrystal aDSChannels::m_lcd [private]`

6.3.4.30 `uint8_t aDSChannels::m_lcdCols [private]`

6.3.4.31 `uint8_t aDSChannels::m_lcdRows [private]`

6.3.4.32 `unsigned long aDSChannels::m_nextDisplayUpdate [private]`

6.3.4.33 `unsigned long aDSChannels::m_nextMeasureUpdate [private]`

6.3.4.34 `OperationMode_t aDSChannels::m_operationMode [private]`

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

6.3.4.36 `bool aDSChannels::m_storedToEEPROM [private]`

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

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

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

Column LCD offset for left marker '['.

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

Column LCD offset for right marker ']'.

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

Value column LCD offset.

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

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

6.3.4.42 `const unsigned long aDSChannels::TEMP_SETTING_INACTIVITY = 30000` `[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](#)

6.4 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
USB input communication buffer's max size.

6.4.1 Constructor & Destructor Documentation

6.4.1.1 `aDSEngine::aDSEngine ()`

`aDSEngine` constructor

6.4.1.2 `aDSEngine::~~aDSEngine ()` `[virtual]`

`aDSEngine` destructor

6.4.2 Member Function Documentation

6.4.2.1 `void aDSEngine::_handleSerialInput ()` `[protected]`

Handle serial input, in calibration mode only.

Returns

void

6.4.2.2 `void aDSEngine::run ()`

Main loop.

Returns

void

6.4.2.3 `void aDSEngine::setup ()`

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

Returns

void

6.4.3 Member Data Documentation

6.4.3.1 `aDSChannels aDSEngine::m_channels` `[private]`

6.4.3.2 `uint16_t aDSEngine::m_datas` `[private]`

6.4.3.3 `ClickEncoder aDSEngine::m_encoder` `[private]`

6.4.3.4 `uint8_t aDSEngine::m_RXbuffer[RXBUFFER_MAXLEN]` `[private]`

USB rx buffer.

6.4.3.5 `uint8_t aDSEngine::m_RXoffset` `[private]`

USB rx buffer offset counter.

6.4.3.6 unsigned long aDSEngine::m_serialInputTick [private]

6.4.3.7 const uint8_t aDSEngine::RXBUFFER_MAXLEN = 64 [static], [private]

USB input communication buffer's max size.

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

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

6.5 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.

6.5.1 Detailed Description

Our pin structure.

6.5.2 Member Data Documentation

6.5.2.1 uint8_t aDSChannel::aPin_t::mask

Bit mask of the pin.

6.5.2.2 volatile uint8_t* aDSChannel::aPin_t::outputRegister

Output register of the pin.

6.5.2.3 uint8_t aDSChannel::aPin_t::pin

"Arduino" pin

6.5.2.4 uint8_t aDSChannel::aPin_t::port

Port of the pin.

6.5.2.5 uint8_t aDSChannel::aPin_t::timer

Timer of the pin.

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

- [aDSEngine.h](#)

6.6 aDSChannel::CalibrationData_t Struct Reference

Calibration values.

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

Public Attributes

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

6.6.1 Detailed Description

Calibration values.

Contains Slope and Offset float values

6.6.2 Member Data Documentation

6.6.2.1 float aDSChannel::CalibrationData_t::offset

Offset value.

6.6.2.2 float aDSChannel::CalibrationData_t::slope

Slope value.

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

- [aDSEngine.h](#)

Chapter 7

File Documentation

7.1 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 [12][[DIGIT_WIDTH](#)]
_glyphs[] offsets
- static const uint8_t _bigDigitsBottom [12][[DIGIT_WIDTH](#)]
_glyphs[] offsets
- ClickEncoder * [pEncoder](#) = NULL
Global pointer to ClickEncoder object, used inside [timer1ISR\(\)](#) function.
- uint8_t [channelCount](#) = 0

7.1.1 Detailed Description

Copyright

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

7.1.2 Function Documentation

7.1.2.1 `static int8_t getNumericalLength (int16_t n)` [static]

Return numerical character length of argument.

Parameters

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

Returns

int8_t : length

7.1.2.2 `void timer1ISR (void)`

Timer1 class ISR function.

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

Returns

void

7.1.3 Variable Documentation

7.1.3.1 `const uint8_t _bigDigitsBottom[12][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 },
    { 0, 0, 0 }
}
```

`_glyphs[]` offsets

0..9 + ' ' bottom characters matrix

7.1.3.2 `const uint8_t _bigDigitsTop[12][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 },
  { 1, 1, 1 }
}

```

`_glyphs[]` offsets

0..9 + ' ' top characters matrix

7.1.3.3 `uint8_t channelCount = 0`7.1.3.4 `const uint8_t DIGIT_WIDTH = 3` `[static]`

Max numerical length of temperature (used with big digits)

7.1.3.5 `ClickEncoder* pEncoder = NULL`Global pointer to ClickEncoder object, used inside `timer1ISR()` function.7.1.3.6 `const uint8_t _glyphs[][8] PROGMEM` `[static]`

LCD glyphs (for big digits and LED)

7.2 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 [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.*

7.2.1 Detailed Description

Copyright

Copyright (C) 2015 F1RMB, Daniel Caujolle-Bert 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

7.2.2 Macro Definition Documentation

7.2.2.1 `#define IS_DATA_ENABLED(bit) ((m_datas & bit))`

7.2.3 Enumeration Type Documentation

7.2.3.1 `enum OperationMode_t`

Operation Mode enumeration.

Enumerator

`OPERATION_MODE_READ` Reading values.

`OPERATION_MODE_SET` Settings values.

`OPERATION_MODE_UNKNOWN` Unset (internal)

7.2.4 Variable Documentation

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

Pin to check from if channel 2 is wired.

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

Encoder A pin.

7.2.4.3 `const uint8_t ENCODER_B_PIN = 3` `[static]`

Encoder B pin.

7.2.4.4 `const uint8_t ENCODER_PB_PIN = 4` `[static]`

Encoder push button pin.

7.2.4.5 `const uint8_t ENCODER_STEPS_PER_NOTCH = 4` `[static]`

Number of steps per notch (indent)

7.2.4.6 `const uint8_t LCD_COLS = 16` `[static]`

LCD columns.

7.2.4.7 `const uint8_t LCD_D4_PIN = 9` `[static]`

LCD D4 pin.

7.2.4.8 `const uint8_t LCD_D5_PIN = 10` `[static]`

LCD D5 pin.

7.2.4.9 `const uint8_t LCD_D6_PIN = 11` `[static]`

LCD D6 pin.

7.2.4.10 `const uint8_t LCD_D7_PIN = 12` `[static]`

LCD D7 pin.

7.2.4.11 `const uint8_t LCD_ENABLE_PIN = 8` `[static]`

LCD Enable pin.

7.2.4.12 `const uint8_t LCD_ROWS = 2` `[static]`

LCD rows.

7.2.4.13 `const uint8_t LCD_RS_PIN = 7` `[static]`

LCD RS pin.

7.2.4.14 `const uint8_t LED_CHANNEL1_PIN = A2` `[static]`

LED pin of channel 1.

7.2.4.15 `const uint8_t LED_CHANNEL2_PIN = A3` `[static]`

LED pin of channel 2.

7.2.4.16 `const uint8_t PROGRAM_VERSION_MAJOR = 1` `[static]`

Major program version.

7.2.4.17 `const uint8_t PROGRAM_VERSION_MINOR = 2` `[static]`

Minor program version.

7.2.4.18 `const uint8_t PWM_CHANNEL1_PIN = 5` `[static]`

PWM pin of channel 1.

7.2.4.19 `const uint8_t PWM_CHANNEL2_PIN = 6` `[static]`

PWM pin of channel 2.

7.2.4.20 `const uint8_t TEMP_SENSOR_CHANNEL1_PIN = A1` `[static]`

Temp sensor pin of channel 1.

7.2.4.21 `const uint8_t TEMP_SENSOR_CHANNEL2_PIN = A0` `[static]`

Temp sensor pin of channel 2.

7.3 CDC.cpp File Reference

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

7.4 HardwareSerial.cpp File Reference

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

7.5 HID.cpp File Reference

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

7.6 IPAddress.cpp File Reference

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

7.7 libraries.cpp File Reference

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

7.8 main.cpp File Reference

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

7.9 new.cpp File Reference

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

7.10 Print.cpp File Reference

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

7.11 sketch.cpp File Reference

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

Functions

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

Variables

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

7.11.1 Function Documentation

7.11.1.1 void [loop](#) ()

7.11.1.2 void [setup](#) ()

7.11.2 Variable Documentation

7.11.2.1 [aDSEngine](#)* [engine](#)

7.12 Stream.cpp File Reference

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

7.13 Tone.cpp File Reference

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

7.14 USBCore.cpp File Reference

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

7.15 WInterrupts.c File Reference

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

7.16 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`

7.16.1 Macro Definition Documentation

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

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

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

7.16.1.4 `#define MILLIS_INC (MICROSECONDS_PER_TIMER0_OVERFLOW / 1000)`

7.16.2 Function Documentation

7.16.2.1 `void delay (unsigned long ms)`

7.16.2.2 `void delayMicroseconds (unsigned int us)`

7.16.2.3 `if (f >= FRACT_MAX)`

7.16.2.4 `void init ()`

7.16.2.5 `unsigned long micros ()`

7.16.2.6 `unsigned long millis ()`

7.16.3 Variable Documentation

7.16.3.1 `f = timer0_fract`

7.16.3.2 `m = MILLIS_INC`

7.16.3.3 `timer0_fract = 0` `[static]`

7.16.3.4 `timer0_millis = 0`

7.16.3.5 `timer0_overflow_count = 0`

7.17 wiring_analog.c File Reference

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

7.18 wiring_digital.c File Reference

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

7.19 wiring_pulse.c File Reference

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

7.20 wiring_shift.c File Reference

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

7.21 WMath.cpp File Reference

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

7.22 WString.cpp File Reference

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

Index

- ~aDSChannel
 - aDSChannel, [14](#)
- ~aDSChannels
 - aDSChannels, [26](#)
- ~aDSEngine
 - aDSEngine, [38](#)
- _analogRead
 - aDSChannel, [14](#)
- _analogWrite
 - aDSChannel, [14](#)
- _backupCalibrationFromEEPROM
 - aDSChannels, [26](#)
- _bigDigitsBottom
 - aDSEngine.cpp, [42](#)
- _bigDigitsTop
 - aDSEngine.cpp, [42](#)
- _checkForMagicNumbers
 - aDSChannels, [26](#)
- _clearValue
 - aDSChannels, [26](#)
- _crc8
 - aDSChannels, [26](#)
- _digitalRead
 - aDSChannel, [15](#)
- _digitalWrite
 - aDSChannel, [15](#)
- _displayBigDigit
 - aDSChannels, [27](#)
- _displayBigDigits
 - aDSChannels, [27](#)
- _enableData
 - aDSChannels, [27](#)
- _enableDataCheck
 - aDSChannels, [27](#)
- _getTempFromEEPROM
 - aDSChannels, [28](#)
- _handleSerialInput
 - aDSEngine, [38](#)
- _read
 - aDSChannels, [28](#)
- _restoreCalibrationFromEEPROM
 - aDSChannels, [28](#)
- _scissor
 - aDSChannels, [28](#)
- _setTempToEEPROM
 - aDSChannels, [29](#)
- _showBanner
 - aDSChannels, [29](#)
- _turnOffPWM

- aDSChannel, [15](#)
- _turnPWM
 - aDSChannel, [15](#)
- _updateDisplay
 - aDSChannels, [29](#)
- _updateField
 - aDSChannels, [29](#)
- _wakeupFromStandby
 - aDSChannels, [29](#)
- _write
 - aDSChannels, [30](#)
- _writeMagicNumbers
 - aDSChannels, [30](#)
- aDSChannel
 - HEATING_STATE_COOLING, [14](#)
 - HEATING_STATE_HEATING, [14](#)
 - HEATING_STATE_REACHED, [14](#)
 - HEATING_STATE_STANDBY, [14](#)
- aDSChannels
 - CHANNEL_MAX, [25](#)
 - CHANNEL_ONE, [25](#)
 - CHANNEL_TWO, [25](#)
- aDSEngine.h
 - OPERATION_MODE_READ, [45](#)
 - OPERATION_MODE_SET, [45](#)
 - OPERATION_MODE_UNKNOWN, [45](#)
- aDSChannel, [11](#)
 - ~aDSChannel, [14](#)
 - _analogRead, [14](#)
 - _analogWrite, [14](#)
 - _digitalRead, [15](#)
 - _digitalWrite, [15](#)
 - _turnOffPWM, [15](#)
 - _turnPWM, [15](#)
 - aDSChannel, [14](#)
 - aDSChannel, [14](#)
 - BLINK_UPDATE_RATE, [20](#)
 - getADCValue, [15](#)
 - getCalibration, [15](#)
 - getHeatState, [16](#)
 - getLEDState, [16](#)
 - getStandbyMode, [16](#)
 - getTemperature, [16](#)
 - hasFocus, [16](#)
 - HeatingState_t, [14](#)
 - isPlugged, [16](#)
 - isTempHasChanged, [17](#)
 - m_adcValue, [20](#)
 - m_blinkStandby, [20](#)

- m_brother, 20
- m_cal, 20
- m_channel, 20
- m_currentTemp, 20
- m_hasFocus, 20
- m_heatState, 20
- m_inStandby, 20
- m_isPlugged, 20
- m_ledPin, 20
- m_ledState, 20
- m_nextBlink, 20
- m_nextPass, 20
- m_pwmPin, 20
- m_pwmValue, 20
- m_ref, 20
- m_sensorPin, 21
- m_targetTemp, 21
- m_tempHasChanged, 21
- PWM_MAX_VALUE, 21
- service, 17
- setBrother, 17
- setCalibration, 17
- setFocus, 17
- setStandbyMode, 18
- setTemperature, 18
- setup, 18
- syncTempChange, 18
- TEMPERATURE_MAX, 21
- TEMPERATURE_MIN, 21
- updateLEDState, 18
- aDSChannel::CalibrationData_t, 40
 - offset, 40
 - slope, 40
- aDSChannel::aPin_t, 39
 - mask, 39
 - outputRegister, 39
 - pin, 39
 - port, 39
 - timer, 39
- aDSChannels, 21
 - ~aDSChannels, 26
 - _backupCalibrationFromEEPROM, 26
 - _checkForMagicNumbers, 26
 - _clearValue, 26
 - _crc8, 26
 - _displayBigDigit, 27
 - _displayBigDigits, 27
 - _enableData, 27
 - _enableDataCheck, 27
 - _getTempFromEEPROM, 28
 - _read, 28
 - _restoreCalibrationFromEEPROM, 28
 - _scissor, 28
 - _setTempToEEPROM, 29
 - _showBanner, 29
 - _updateDisplay, 29
 - _updateField, 29
 - _wakeupFromStandby, 29
 - _write, 30
 - _writeMagicNumbers, 30
 - aDSChannels, 25
 - aDSChannels, 25
 - Channel_t, 25
 - DATA_DISPLAY, 34
 - DATA_FOCUS, 34
 - DATA_OPERATION, 35
 - DATA_STANDBY, 35
 - EEPROM_ADDR_MAGIC, 35
 - EEPROM_TEMP_SIZE, 36
 - getCalibrationValues, 30
 - getOperationMode, 30
 - incEncoderPosition, 30
 - isInCalibration, 31
 - isInStandby, 31
 - isJoined, 31
 - m_channels, 36
 - m_datas, 36
 - m_isValidEEPROM, 36
 - m_lastTempChange, 36
 - m_lcd, 36
 - m_lcdCols, 36
 - m_lcdRows, 36
 - m_nextDisplayUpdate, 36
 - m_nextMeasureUpdate, 36
 - m_operationMode, 36
 - m_operationTick, 36
 - m_storedToEEPROM, 36
 - OFFSET_VALUE, 36
 - pingOperationMode, 31
 - restoreCalibrationValues, 31
 - saveCalibrationValues, 31
 - service, 32
 - setCalibrationMode, 32
 - setCalibrationValues, 32
 - setFocusToNextChannel, 32
 - setOperationMode, 32
 - setup, 33
 - syncData, 33
 - toggleJoined, 33
 - toggleStandbyMode, 33
 - updateOperationMode, 33
- aDSChannels::_eepromCalibrationValue_t, 11
 - c, 11
 - v, 11
- aDSEngine, 37
 - ~aDSEngine, 38
 - _handleSerialInput, 38
 - aDSEngine, 38
 - aDSEngine, 38
 - m_RXbuffer, 38
 - m_RXoffset, 38
 - m_channels, 38
 - m_datas, 38
 - m_encoder, 38
 - m_serialInputTick, 38
 - RXBUFFER_MAXLEN, 39

- run, 38
- setup, 38
- aDSEngine.cpp, 41
 - _bigDigitsBottom, 42
 - _bigDigitsTop, 42
 - channelCount, 43
 - DIGIT_WIDTH, 43
 - getNumericalLength, 42
 - pEncoder, 43
 - PROGMEM, 43
 - timer1ISR, 42
- aDSEngine.h, 43
 - ENCODER_A_PIN, 45
 - ENCODER_B_PIN, 45
 - ENCODER_PB_PIN, 46
 - IS_DATA_ENABLED, 45
 - LCD_COLS, 46
 - LCD_D4_PIN, 46
 - LCD_D5_PIN, 46
 - LCD_D6_PIN, 46
 - LCD_D7_PIN, 46
 - LCD_ENABLE_PIN, 46
 - LCD_ROWS, 46
 - LCD_RS_PIN, 46
 - LED_CHANNEL1_PIN, 46
 - LED_CHANNEL2_PIN, 46
 - OperationMode_t, 45
 - PWM_CHANNEL1_PIN, 47
 - PWM_CHANNEL2_PIN, 47
- BLINK_UPDATE_RATE
 - aDSChannel, 20
- c
 - aDSChannels::_eepromCalibrationValue_t, 11
- CHANNEL_MAX
 - aDSChannels, 25
- CHANNEL_ONE
 - aDSChannels, 25
- CHANNEL_TWO
 - aDSChannels, 25
- CDC.cpp, 47
- CHANNEL2_ENABLE_PIN
 - aDSEngine.h, 45
- Channel_t
 - aDSChannels, 25
- channelCount
 - aDSEngine.cpp, 43
- DATA_DISPLAY
 - aDSChannels, 34
- DATA_FOCUS
 - aDSChannels, 34
- DATA_OPERATION
 - aDSChannels, 35
- DATA_STANDBY
 - aDSChannels, 35
- DIGIT_WIDTH
 - aDSEngine.cpp, 43
- delay
 - wiring.c, 50
- delayMicroseconds
 - wiring.c, 50
- EEPROM_ADDR_MAGIC
 - aDSChannels, 35
- EEPROM_TEMP_SIZE
 - aDSChannels, 36
- ENCODER_A_PIN
 - aDSEngine.h, 45
- ENCODER_B_PIN
 - aDSEngine.h, 45
- ENCODER_PB_PIN
 - aDSEngine.h, 46
- engine
 - sketch.cpp, 48
- f
 - wiring.c, 50
- FRACT_INC
 - wiring.c, 50
- FRACT_MAX
 - wiring.c, 50
- getADCValue
 - aDSChannel, 15
- getCalibration
 - aDSChannel, 15
- getCalibrationValues
 - aDSChannels, 30
- getHeatState
 - aDSChannel, 16
- getLEDState
 - aDSChannel, 16
- getNumericalLength
 - aDSEngine.cpp, 42
- getOperationMode
 - aDSChannels, 30
- getStandbyMode
 - aDSChannel, 16
- getTemperature
 - aDSChannel, 16
- HEATING_STATE_COOLING
 - aDSChannel, 14
- HEATING_STATE_HEATING
 - aDSChannel, 14
- HEATING_STATE_REACHED
 - aDSChannel, 14
- HEATING_STATE_STANDBY
 - aDSChannel, 14
- HID.cpp, 47
- HardwareSerial.cpp, 47
- hasFocus
 - aDSChannel, 16
- HeatingState_t
 - aDSChannel, 14
- IPAddress.cpp, 47

- IS_DATA_ENABLED
 - aDSEngine.h, 45
- if
 - wiring.c, 50
- incEncoderPosition
 - aDSChannels, 30
- init
 - wiring.c, 50
- isInCalibration
 - aDSChannels, 31
- isInStandby
 - aDSChannels, 31
- isJoinded
 - aDSChannels, 31
- isPlugged
 - aDSChannel, 16
- isTempHasChanged
 - aDSChannel, 17
- LCD_COLS
 - aDSEngine.h, 46
- LCD_D4_PIN
 - aDSEngine.h, 46
- LCD_D5_PIN
 - aDSEngine.h, 46
- LCD_D6_PIN
 - aDSEngine.h, 46
- LCD_D7_PIN
 - aDSEngine.h, 46
- LCD_ENABLE_PIN
 - aDSEngine.h, 46
- LCD_ROWS
 - aDSEngine.h, 46
- LCD_RS_PIN
 - aDSEngine.h, 46
- LED_CHANNEL1_PIN
 - aDSEngine.h, 46
- LED_CHANNEL2_PIN
 - aDSEngine.h, 46
- libraries.cpp, 48
- loop
 - sketch.cpp, 48
- m
 - wiring.c, 50
- m_RXbuffer
 - aDSEngine, 38
- m_RXoffset
 - aDSEngine, 38
- m_adcValue
 - aDSChannel, 20
- m_blinkStandby
 - aDSChannel, 20
- m_brother
 - aDSChannel, 20
- m_cal
 - aDSChannel, 20
- m_channel
 - aDSChannel, 20
- m_channels
 - aDSChannels, 36
- aDSEngine, 38
- m_currentTemp
 - aDSChannel, 20
- m_datas
 - aDSChannels, 36
- aDSEngine, 38
- m_encoder
 - aDSEngine, 38
- m_hasFocus
 - aDSChannel, 20
- m_heatState
 - aDSChannel, 20
- m_inStandby
 - aDSChannel, 20
- m_isPlugged
 - aDSChannel, 20
- m_isValidEEPROM
 - aDSChannels, 36
- m_lastTempChange
 - aDSChannels, 36
- m_lcd
 - aDSChannels, 36
- m_lcdCols
 - aDSChannels, 36
- m_lcdRows
 - aDSChannels, 36
- m_ledPin
 - aDSChannel, 20
- m_ledState
 - aDSChannel, 20
- m_nextBlink
 - aDSChannel, 20
- m_nextDisplayUpdate
 - aDSChannels, 36
- m_nextMeasureUpdate
 - aDSChannels, 36
- m_nextPass
 - aDSChannel, 20
- m_operationMode
 - aDSChannels, 36
- m_operationTick
 - aDSChannels, 36
- m_pwmPin
 - aDSChannel, 20
- m_pwmValue
 - aDSChannel, 20
- m_ref
 - aDSChannel, 20
- m_sensorPin
 - aDSChannel, 21
- m_serialInputTick
 - aDSEngine, 38
- m_storedToEEPROM
 - aDSChannels, 36
- m_targetTemp
 - aDSChannel, 21

- m_tempHasChanged
 - aDSChannel, 21
- MILLIS_INC
 - wiring.c, 50
- main.cpp, 48
- mask
 - aDSChannel::aPin_t, 39
- micros
 - wiring.c, 50
- millis
 - wiring.c, 50
- new.cpp, 48
- OPERATION_MODE_READ
 - aDSEngine.h, 45
- OPERATION_MODE_SET
 - aDSEngine.h, 45
- OPERATION_MODE_UNKNOWN
 - aDSEngine.h, 45
- OFFSET_MARKER_LEFT
 - aDSChannels, 36
- OFFSET_VALUE
 - aDSChannels, 36
- offset
 - aDSChannel::CalibrationData_t, 40
- OperationMode_t
 - aDSEngine.h, 45
- outputRegister
 - aDSChannel::aPin_t, 39
- pEncoder
 - aDSEngine.cpp, 43
- PROGMEM
 - aDSEngine.cpp, 43
- PWM_CHANNEL1_PIN
 - aDSEngine.h, 47
- PWM_CHANNEL2_PIN
 - aDSEngine.h, 47
- PWM_MAX_VALUE
 - aDSChannel, 21
- pin
 - aDSChannel::aPin_t, 39
- pingOperationMode
 - aDSChannels, 31
- port
 - aDSChannel::aPin_t, 39
- Print.cpp, 48
- RXBUFFER_MAXLEN
 - aDSEngine, 39
- restoreCalibrationValues
 - aDSChannels, 31
- run
 - aDSEngine, 38
- saveCalibrationValues
 - aDSChannels, 31
- service
 - aDSChannel, 17
 - aDSChannels, 32
- setBrother
 - aDSChannel, 17
- setCalibration
 - aDSChannel, 17
- setCalibrationMode
 - aDSChannels, 32
- setCalibrationValues
 - aDSChannels, 32
- setFocus
 - aDSChannel, 17
- setFocusToNextChannel
 - aDSChannels, 32
- setOperationMode
 - aDSChannels, 32
- setStandbyMode
 - aDSChannel, 18
- setTemperature
 - aDSChannel, 18
- setup
 - aDSChannel, 18
 - aDSChannels, 33
 - aDSEngine, 38
 - sketch.cpp, 48
- sketch.cpp, 48
 - engine, 48
 - loop, 48
 - setup, 48
- slope
 - aDSChannel::CalibrationData_t, 40
- Stream.cpp, 49
- syncData
 - aDSChannels, 33
- syncTempChange
 - aDSChannel, 18
- TEMPERATURE_MAX
 - aDSChannel, 21
- TEMPERATURE_MIN
 - aDSChannel, 21
- TEMPERATURE_STANDBY
 - aDSChannel, 21
- timer
 - aDSChannel::aPin_t, 39
- timer0_fract
 - wiring.c, 50
- timer0_millis
 - wiring.c, 50
- timer0_overflow_count
 - wiring.c, 50
- timer1ISR
 - aDSEngine.cpp, 42
- toggleJoined
 - aDSChannels, 33
- toggleStandbyMode
 - aDSChannels, 33
- Tone.cpp, 49

- USBCore.cpp, [49](#)
- updateLEDState
 - aDSChannel, [18](#)
- updateOperationMode
 - aDSChannels, [33](#)
- v
 - aDSChannels::_eepromCalibrationValue_t, [11](#)
- WInterrupts.c, [49](#)
- WMath.cpp, [51](#)
- WString.cpp, [51](#)
- wiring.c, [49](#)
 - delay, [50](#)
 - delayMicroseconds, [50](#)
 - f, [50](#)
 - FRACT_INC, [50](#)
 - FRACT_MAX, [50](#)
 - if, [50](#)
 - init, [50](#)
 - m, [50](#)
 - MILLIS_INC, [50](#)
 - micros, [50](#)
 - millis, [50](#)
 - timer0_fract, [50](#)
 - timer0_millis, [50](#)
 - timer0_overflow_count, [50](#)
- wiring_analog.c, [50](#)
- wiring_digital.c, [50](#)
- wiring_pulse.c, [50](#)
- wiring_shift.c, [51](#)