

Murasaki Class Library

0.3.0

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Preface</b>	<b>1</b>
1.1	Simplified IO . . . . .	1
1.2	Preemptive multi-task . . . . .	2
1.3	Blocking IO . . . . .	2
1.4	Thread safe IO . . . . .	2
1.5	Versatile printf() logger . . . . .	2
1.6	Guard by assertion . . . . .	3
1.7	System Logging . . . . .	3
1.8	Configurable . . . . .	3
<b>2</b>	<b>Target and Environment</b>	<b>5</b>
<b>3</b>	<b>Usage Introduction</b>	<b>7</b>
3.1	Message output . . . . .	7
3.2	Serial communication . . . . .	8
3.3	Debugging with Murasaki. . . . .	8
3.4	Tasking . . . . .	10
3.5	Other peripheral . . . . .	10
3.5.1	I2C Master . . . . .	11
3.5.2	I2C Slave . . . . .	11
3.5.3	SPI Master . . . . .	11
3.5.4	SPI Slave . . . . .	12
3.5.5	GPIO . . . . .	12
3.6	Program flow . . . . .	12
3.6.1	Application flow . . . . .	13
3.6.2	HAL Assertion flow . . . . .	15
3.6.3	Spurious Interrupt flow . . . . .	15
3.6.4	Assertion flow . . . . .	16
3.6.5	General Interrupt flow . . . . .	16
3.6.6	EXTI flow . . . . .	16

<b>4</b>	<b>Porting guide</b>	<b>17</b>
4.1	Directory Structure	17
4.1.1	Src directory	18
4.1.2	Inc directory	18
4.1.3	Src-tp and Inc-tp directory	18
4.1.4	murasaki.hpp	18
4.1.5	template directory	18
4.1.5.1	platform_config.hpp	18
4.1.5.2	platform_defs.hpp	19
4.1.5.3	murasaki_platform.hpp	19
4.1.5.4	murasaki_platform.cpp	19
4.1.6	install script	19
4.2	CubeMX setting	20
4.2.1	Heap Size	20
4.2.2	Stack Size	21
4.2.3	Task stack size of the default task	21
4.2.4	UART peripheral	21
4.2.5	SPI Master peripheral	21
4.2.6	SPI Slave peripheral	21
4.2.7	I2C peripheral	22
4.2.8	EXTI	22
4.3	Configuration	22
4.4	Task Priority and Stack Size	22
4.5	Heap memory consideration	23
4.6	Platform variable	23
4.7	Routing interrupts	25
4.8	Error handling	26
4.9	Summary of the porting	27

<b>5</b>	<b>Step-by-Step Porting Guide</b>	<b>29</b>
5.1	UART configuration . . . . .	30
5.2	CPU, EXTI, and System tick configuration . . . . .	32
5.3	FreeRTOS configuration . . . . .	34
5.4	Clock configuration . . . . .	36
5.5	Project configuration and code generation . . . . .	37
5.6	Clone the Murasaki repository and install . . . . .	39
<b>6</b>	<b>Module Index</b>	<b>41</b>
6.1	Modules . . . . .	41
<b>7</b>	<b>Namespace Index</b>	<b>43</b>
7.1	Namespace List . . . . .	43
<b>8</b>	<b>Hierarchical Index</b>	<b>45</b>
8.1	Class Hierarchy . . . . .	45
<b>9</b>	<b>Class Index</b>	<b>47</b>
9.1	Class List . . . . .	47
<b>10</b>	<b>File Index</b>	<b>49</b>
10.1	File List . . . . .	49
<b>11</b>	<b>Module Documentation</b>	<b>53</b>
11.1	Murasaki Class Collection . . . . .	53
11.1.1	Detailed Description . . . . .	54
11.1.2	Macro Definition Documentation . . . . .	54
11.1.2.1	MURASAKI_ASSERT . . . . .	54
11.1.2.2	MURASAKI_PRINT_ERROR . . . . .	55
11.1.2.3	MURASAKI_SYSLOG . . . . .	55
11.2	Synchronization and Exclusive access . . . . .	57
11.2.1	Detailed Description . . . . .	57
11.3	Third party classes . . . . .	58
11.3.1	Detailed Description . . . . .	58

11.4	Definitions and Configuration	59
11.4.1	Detailed Description	59
11.4.2	Macro Definition Documentation	59
11.4.2.1	MURASAKI_CONFIG_NODEBUG	59
11.4.2.2	PLATFORM_CONFIG_DEBUG_BUFFER_SIZE	59
11.4.2.3	PLATFORM_CONFIG_DEBUG_LINE_SIZE	59
11.4.2.4	PLATFORM_CONFIG_DEBUG_SERIAL_TIMEOUT	60
11.4.2.5	PLATFORM_CONFIG_DEBUG_TASK_PRIORITY	60
11.4.2.6	PLATFORM_CONFIG_DEBUG_TASK_STACK_SIZE	60
11.4.3	Enumeration Type Documentation	60
11.4.3.1	I2cStatus	60
11.4.3.2	SpiClockPhase	61
11.4.3.3	SpiClockPolarity	61
11.4.3.4	SpiStatus	61
11.4.3.5	SyslogFacility	62
11.4.3.6	SyslogSeverity	62
11.4.3.7	UartHardwareFlowControl	63
11.4.3.8	UartStatus	63
11.4.3.9	UartTimeout	63
11.4.3.10	WaitMilliseconds	64
11.5	Application Specific Platform	65
11.5.1	Detailed Description	65
11.5.2	Function Documentation	66
11.5.2.1	CustomAssertFailed(uint8_t *file, uint32_t line)	66
11.5.2.2	CustomDefaultHandler()	67
11.5.2.3	ExecPlatform()	67
11.5.2.4	HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)	67
11.5.2.5	HAL_I2C_ErrorCallback(I2C_HandleTypeDef *hi2c)	68
11.5.2.6	HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)	68
11.5.2.7	HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef *hi2c)	68

11.5.2.8 HAL_SPI_ErrorCallback(SPI_HandleTypeDef *hspi)	69
11.5.2.9 HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi)	69
11.5.2.10 HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)	69
11.5.2.11 HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)	70
11.5.2.12 HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)	70
11.5.2.13 InitPlatform()	70
11.5.3 Variable Documentation	71
11.5.3.1 debugger	71
11.6 Abstract Classes	72
11.6.1 Detailed Description	72
11.7 Helper classes	73
11.7.1 Detailed Description	73
11.7.2 Function Documentation	73
11.7.2.1 operator delete(void *ptr)	73
11.7.2.2 operator delete[](void *ptr)	74
11.7.2.3 operator new(std::size_t size)	74
11.7.2.4 operator new[](std::size_t size)	74
11.8 CMSIS	75
11.8.1 Detailed Description	75
11.9 Stm32f4xx_system	76
11.9.1 Detailed Description	76
11.10 STM32F4xx_System_Private_Includes	77
11.10.1 Detailed Description	77
11.10.2 Macro Definition Documentation	77
11.10.2.1 HSE_VALUE	77
11.10.2.2 HSI_VALUE	77
11.11 STM32F4xx_System_Private_TypesDefinitions	78
11.12 STM32F4xx_System_Private_Defines	79
11.12.1 Detailed Description	79
11.12.2 Macro Definition Documentation	79
11.12.2.1 VECT_TAB_OFFSET	79
11.13 STM32F4xx_System_Private_Macros	80
11.14 STM32F4xx_System_Private_Variables	81
11.14.1 Detailed Description	81
11.15 STM32F4xx_System_Private_FunctionPrototypes	82
11.16 STM32F4xx_System_Private_Functions	83
11.16.1 Detailed Description	83
11.16.2 Function Documentation	83
11.16.2.1 SystemCoreClockUpdate(void)	83
11.16.2.2 SystemInit(void)	84

<b>12 Namespace Documentation</b>	<b>85</b>
12.1 murasaki Namespace Reference	85
12.1.1 Detailed Description	86
12.1.2 Function Documentation	86
12.1.2.1 AddSyslogFacilityToMask(murasaki::SyslogFacility facility)	86
12.1.2.2 AllowedSyslogOut(murasaki::SyslogFacility facility, murasaki::SyslogSeverity severity)	86
12.1.2.3 RemoveSyslogFacilityFromMask(murasaki::SyslogFacility facility)	87
12.1.2.4 SetSyslogFacilityMask(uint32_t mask)	87
12.1.2.5 SetSyslogSererityThreshold(murasaki::SyslogSeverity severity)	87
12.1.3 Variable Documentation	87
12.1.3.1 platform	87
<b>13 Class Documentation</b>	<b>89</b>
13.1 murasaki::Adau1361 Class Reference	89
13.1.1 Constructor & Destructor Documentation	90
13.1.1.1 Adau1361(unsigned int fs, murasaki::I2CMasterStrategy *controler, unsigned int i2c_device_addr)	90
13.1.2 Member Function Documentation	90
13.1.2.1 configure_board(void)=0	90
13.1.2.2 configure_pll(void)=0	91
13.1.2.3 send_command(const uint8_t command[], int size)	91
13.1.2.4 send_command_table(const uint8_t table[][3], int rows)	91
13.1.2.5 set_aux_input_gain(float left_gain, float right_gain, bool mute=false)	91
13.1.2.6 set_hp_output_gain(float left_gain, float right_gain, bool mute=false)	92
13.1.2.7 set_line_input_gain(float left_gain, float right_gain, bool mute=false)	92
13.1.2.8 set_line_output_gain(float left_gain, float right_gain, bool mute=false)	92
13.1.2.9 start(void)	93
13.1.2.10 wait_pll_lock(void)	93
13.2 murasaki::AudioCodecStrategy Class Reference	93
13.2.1 Detailed Description	94
13.2.2 Constructor & Destructor Documentation	94



13.2.2.1	AudioCodecStrategy(unsigned int fs)	94
13.2.3	Member Function Documentation	94
13.2.3.1	set_aux_input_gain(float left_gain, float right_gain, bool mute=false)	94
13.2.3.2	set_hp_output_gain(float left_gain, float right_gain, bool mute=false)	94
13.2.3.3	set_line_input_gain(float left_gain, float right_gain, bool mute=false)	95
13.2.3.4	set_line_output_gain(float left_gain, float right_gain, bool mute=false)	95
13.2.3.5	set_mic_input_gain(float left_gain, float right_gain, bool mute=false)	95
13.2.3.6	start(void)=0	96
13.3	murasaki::AudioStrategy Class Reference	96
13.3.1	Detailed Description	97
13.3.2	Constructor & Destructor Documentation	97
13.3.2.1	AudioStrategy(void *peripheral, unsigned int channel_length, unsigned int num← _phases, unsigned int num_channnels)	97
13.3.3	Member Function Documentation	98
13.3.3.1	TransmitAndReceive(float **tx_channels, float **rx_channels)	98
13.3.3.2	TransmitAndReceive(float *tx_left, float *tx_right, float *rx_left, float *rx_right)	98
13.4	murasaki::BitIn Class Reference	99
13.4.1	Detailed Description	100
13.4.2	Constructor & Destructor Documentation	100
13.4.2.1	BitIn(GPIO_TypeDef *port, uint16_t pin)	100
13.4.3	Member Function Documentation	101
13.4.3.1	Get(void)	101
13.4.3.2	GetPeripheralHandle()	101
13.5	murasaki::BitInStrategy Class Reference	101
13.5.1	Detailed Description	102
13.5.2	Member Function Documentation	102
13.5.2.1	Get(void)=0	102
13.6	murasaki::BitOut Class Reference	103
13.6.1	Detailed Description	104
13.6.2	Constructor & Destructor Documentation	104
13.6.2.1	BitOut(GPIO_TypeDef *port, uint16_t pin)	104

13.6.3	Member Function Documentation	104
13.6.3.1	Get(void)	104
13.6.3.2	GetPeripheralHandle()	104
13.6.3.3	Set(unsigned int state=1)	104
13.7	murasaki::BitOutStrategy Class Reference	105
13.7.1	Detailed Description	106
13.7.2	Member Function Documentation	106
13.7.2.1	Get(void)=0	106
13.7.2.2	Set(unsigned int state=1)=0	106
13.8	murasaki::CriticalSection Class Reference	106
13.8.1	Detailed Description	107
13.8.2	Member Function Documentation	107
13.8.2.1	Enter()	107
13.8.2.2	Leave()	107
13.9	murasaki::Debugger Class Reference	107
13.9.1	Detailed Description	108
13.9.2	Constructor & Destructor Documentation	108
13.9.2.1	Debugger(LoggerStrategy *logger)	108
13.9.3	Member Function Documentation	108
13.9.3.1	AutoRePrint()	108
13.9.3.2	GetchFromTask()	109
13.9.3.3	Printf(const char *fmt,...)	109
13.9.3.4	RePrint()	109
13.9.4	Member Data Documentation	110
13.9.4.1	facility_mask_	110
13.9.4.2	line_	110
13.9.4.3	severity_	110
13.10	murasaki::DebuggerFifo Class Reference	110
13.10.1	Detailed Description	111
13.10.2	Constructor & Destructor Documentation	111

13.10.2.1 DebuggerFifo(unsigned int buffer_size)	111
13.10.3 Member Function Documentation	111
13.10.3.1 Get(uint8_t data[], unsigned int size)	111
13.10.3.2 SetPostMortem()	112
13.11 murasaki::DebuggerUart Class Reference	112
13.11.1 Detailed Description	113
13.11.2 Constructor & Destructor Documentation	114
13.11.2.1 DebuggerUart(UART_HandleTypeDef *uart)	114
13.11.3 Member Function Documentation	114
13.11.3.1 HandleError(void *const ptr)	114
13.11.3.2 Receive(uint8_t *data, unsigned int count, unsigned int *transferred_count, Uart↔ Timeout uart_timeout, WaitMilliseconds timeout_ms)	115
13.11.3.3 ReceiveCompleteCallback(void *const ptr)	115
13.11.3.4 SetHardwareFlowControl(UartHardwareFlowControl control)	116
13.11.3.5 SetSpeed(unsigned int baud_rate)	116
13.11.3.6 Transmit(const uint8_t *data, unsigned int size, WaitMilliseconds timeout_ms)	116
13.11.3.7 TransmitCompleteCallback(void *const ptr)	117
13.12 murasaki::FifoStrategy Class Reference	117
13.12.1 Detailed Description	118
13.12.2 Constructor & Destructor Documentation	118
13.12.2.1 FifoStrategy(unsigned int buffer_size)	118
13.12.3 Member Function Documentation	118
13.12.3.1 Get(uint8_t data[], unsigned int size)	118
13.12.3.2 Put(uint8_t const data[], unsigned int size)	119
13.13 murasaki::GPIO_type Struct Reference	119
13.13.1 Detailed Description	119
13.14 murasaki::I2cMaster Class Reference	120
13.14.1 Detailed Description	121
13.14.2 Constructor & Destructor Documentation	121
13.14.2.1 I2cMaster(I2C_HandleTypeDef *i2c_handle)	121
13.14.3 Member Function Documentation	122

13.14.3.1	HandleError(void *ptr)	122
13.14.3.2	Receive(unsigned int addr, uint8_t *rx_data, unsigned int rx_size, unsigned int *transferred_count, WaitMilliseconds timeout_ms)	122
13.14.3.3	ReceiveCompleteCallback(void *ptr)	123
13.14.3.4	Transmit(unsigned int addr, const uint8_t *tx_data, unsigned int tx_size, unsigned int *transferred_count, WaitMilliseconds timeout_ms)	123
13.14.3.5	TransmitCompleteCallback(void *ptr)	124
13.14.3.6	TransmitThenReceive(unsigned int addr, const uint8_t *tx_data, unsigned int tx_size, uint8_t *rx_data, unsigned int rx_size, unsigned int *tx_transferred_count, unsigned int *rx_transferred_count, WaitMilliseconds timeout_ms)	124
13.15	murasaki::I2CMasterStrategy Class Reference	125
13.15.1	Detailed Description	126
13.15.2	Member Function Documentation	126
13.15.2.1	HandleError(void *ptr)=0	126
13.15.2.2	Receive(unsigned int addr, uint8_t *rx_data, unsigned int rx_size, unsigned int *transferred_count=nullptr, WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely)=0	127
13.15.2.3	ReceiveCompleteCallback(void *ptr)=0	127
13.15.2.4	Transmit(unsigned int addr, const uint8_t *tx_data, unsigned int tx_size, unsigned int *transferred_count=nullptr, WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely)=0	128
13.15.2.5	TransmitCompleteCallback(void *ptr)=0	128
13.15.2.6	TransmitThenReceive(unsigned int addr, const uint8_t *tx_data, unsigned int tx_size, uint8_t *rx_data, unsigned int rx_size, unsigned int *tx_transferred_count=nullptr, unsigned int *rx_transferred_count=nullptr, WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely)=0	128
13.16	murasaki::I2cSlave Class Reference	129
13.16.1	Detailed Description	130
13.16.2	Member Function Documentation	131
13.16.2.1	HandleError(void *ptr)	131
13.16.2.2	Receive(uint8_t *rx_data, unsigned int rx_size, unsigned int *transferred_count, WaitMilliseconds timeout_ms)	131
13.16.2.3	ReceiveCompleteCallback(void *ptr)	132
13.16.2.4	Transmit(const uint8_t *tx_data, unsigned int tx_size, unsigned int *transferred_count, WaitMilliseconds timeout_ms)	132
13.16.2.5	TransmitCompleteCallback(void *ptr)	133

13.17murasaki::I2cSlaveStrategy Class Reference . . . . .	134
13.17.1 Detailed Description . . . . .	135
13.17.2 Member Function Documentation . . . . .	135
13.17.2.1 HandleError(void *ptr)=0 . . . . .	135
13.17.2.2 Receive(uint8_t *rx_data, unsigned int rx_size, unsigned int *transferred← _count=nullptr, murasaki::WaitMilliseconds timeout_ms=murasaki::kwms← Indefinitely)=0 . . . . .	135
13.17.2.3 ReceiveCompleteCallback(void *ptr)=0 . . . . .	136
13.17.2.4 Transmit(const uint8_t *tx_data, unsigned int tx_size, unsigned int *transferred← _count=nullptr, murasaki::WaitMilliseconds timeout_ms=murasaki::kwms← Indefinitely)=0 . . . . .	136
13.17.2.5 TransmitCompleteCallback(void *ptr)=0 . . . . .	136
13.18murasaki::LoggerStrategy Class Reference . . . . .	137
13.18.1 Detailed Description . . . . .	137
13.18.2 Constructor & Destructor Documentation . . . . .	138
13.18.2.1 ~LoggerStrategy() . . . . .	138
13.18.3 Member Function Documentation . . . . .	138
13.18.3.1 DoPostMortem(void *debugger_fifo) . . . . .	138
13.18.3.2 getCharacter()=0 . . . . .	138
13.18.3.3 putMessage(char message[], unsigned int size)=0 . . . . .	138
13.19murasaki::LoggingHelpers Struct Reference . . . . .	139
13.20murasaki::PeripheralStrategy Class Reference . . . . .	139
13.20.1 Detailed Description . . . . .	140
13.21murasaki::Platform Struct Reference . . . . .	140
13.21.1 Detailed Description . . . . .	141
13.22murasaki::SpiMaster Class Reference . . . . .	141
13.22.1 Detailed Description . . . . .	142
13.22.2 Constructor & Destructor Documentation . . . . .	142
13.22.2.1 SpiMaster(SPI_HandleTypeDef *spi_handle) . . . . .	142
13.22.3 Member Function Documentation . . . . .	143
13.22.3.1 HandleError(void *ptr) . . . . .	143

13.22.3.2 TransmitAndReceive(murasaki::SpiSlaveAdapterStrategy *spi_spec, const uint8_t *tx_data, uint8_t *rx_data, unsigned int size, murasaki::WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely) . . . . .	143
13.22.3.3 TransmitAndReceiveCompleteCallback(void *ptr) . . . . .	144
13.23 murasaki::SpiMasterStrategy Class Reference . . . . .	144
13.23.1 Detailed Description . . . . .	145
13.23.2 Member Function Documentation . . . . .	145
13.23.2.1 HandleError(void *ptr)=0 . . . . .	145
13.23.2.2 TransmitAndReceive(murasaki::SpiSlaveAdapterStrategy *spi_spec, const uint8_t *tx_data, uint8_t *rx_data, unsigned int size, murasaki::WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely)=0 . . . . .	146
13.23.2.3 TransmitAndReceiveCompleteCallback(void *ptr)=0 . . . . .	146
13.24 murasaki::SpiSlave Class Reference . . . . .	147
13.24.1 Detailed Description . . . . .	148
13.24.2 Constructor & Destructor Documentation . . . . .	148
13.24.2.1 SpiSlave(SPI_HandleTypeDef *spi_handle) . . . . .	148
13.24.3 Member Function Documentation . . . . .	148
13.24.3.1 HandleError(void *ptr) . . . . .	149
13.24.3.2 TransmitAndReceive(const uint8_t *tx_data, uint8_t *rx_data, unsigned int size, unsigned int *transferred_count, murasaki::WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely) . . . . .	149
13.24.3.3 TransmitAndReceiveCompleteCallback(void *ptr) . . . . .	150
13.25 murasaki::SpiSlaveAdapter Class Reference . . . . .	150
13.25.1 Detailed Description . . . . .	151
13.25.2 Constructor & Destructor Documentation . . . . .	151
13.25.2.1 SpiSlaveAdapter(murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha,::GPIO_TypeDef *port, uint16_t pin) . . . . .	151
13.25.2.2 SpiSlaveAdapter(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin) . . . . .	152
13.25.3 Member Function Documentation . . . . .	152
13.25.3.1 AssertCs() . . . . .	152
13.25.3.2 DeassertCs() . . . . .	152
13.26 murasaki::SpiSlaveAdapterStrategy Class Reference . . . . .	153
13.26.1 Detailed Description . . . . .	153

13.26.2 Constructor & Destructor Documentation . . . . .	153
13.26.2.1 SpiSlaveAdapterStrategy(murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha) . . . . .	153
13.26.2.2 SpiSlaveAdapterStrategy(unsigned int pol, unsigned int pha) . . . . .	154
13.26.3 Member Function Documentation . . . . .	154
13.26.3.1 AssertCs() . . . . .	154
13.26.3.2 DeassertCs() . . . . .	154
13.26.3.3 GetCpha() . . . . .	154
13.26.3.4 GetCpol() . . . . .	155
13.27 murasaki::SpiSlaveStrategy Class Reference . . . . .	155
13.27.1 Detailed Description . . . . .	156
13.27.2 Member Function Documentation . . . . .	156
13.27.2.1 HandleError(void *ptr)=0 . . . . .	156
13.27.2.2 TransmitAndReceive(const uint8_t *tx_data, uint8_t *rx_data, unsigned int size, unsigned int *transferred_count=nullptr, murasaki::WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely)=0 . . . . .	156
13.27.2.3 TransmitAndReceiveCompleteCallback(void *ptr)=0 . . . . .	157
13.28 murasaki::Synchronizer Class Reference . . . . .	157
13.28.1 Detailed Description . . . . .	157
13.28.2 Member Function Documentation . . . . .	157
13.28.2.1 Release() . . . . .	157
13.28.2.2 Wait(WaitMilliseconds timeout_ms=kwmsIndefinitely) . . . . .	157
13.29 murasaki::Task Class Reference . . . . .	158
13.29.1 Detailed Description . . . . .	159
13.29.2 Constructor & Destructor Documentation . . . . .	159
13.29.2.1 Task(const char *task_name, unsigned short stack_depth, UBaseType_t task_priority, const void *task_parameter, void(*task_body_func)(const void *)) . . . . .	159
13.29.3 Member Function Documentation . . . . .	160
13.29.3.1 TaskBody(const void *ptr) . . . . .	160
13.30 murasaki::TaskStrategy Class Reference . . . . .	160
13.30.1 Detailed Description . . . . .	161
13.30.2 Constructor & Destructor Documentation . . . . .	161

13.30.2.1 TaskStrategy(const char *task_name, unsigned short stack_depth, UBaseType↵ _t task_priority, const void *task_parameter) . . . . .	161
13.30.3 Member Function Documentation . . . . .	161
13.30.3.1 GetName() . . . . .	161
13.30.3.2 Launch(void *ptr) . . . . .	161
13.30.3.3 Start() . . . . .	162
13.30.3.4 TaskBody(const void *ptr)=0 . . . . .	162
13.31 murasaki::Uart Class Reference . . . . .	162
13.31.1 Detailed Description . . . . .	164
13.31.2 Constructor & Destructor Documentation . . . . .	164
13.31.2.1 Uart(UART_HandleTypeDef *uart) . . . . .	164
13.31.3 Member Function Documentation . . . . .	165
13.31.3.1 HandleError(void *const ptr) . . . . .	165
13.31.3.2 Receive(uint8_t *data, unsigned int count, unsigned int *transferred_count, Uart↵ Timeout uart_timeout, WaitMilliseconds timeout_ms) . . . . .	165
13.31.3.3 ReceiveCompleteCallback(void *const ptr) . . . . .	166
13.31.3.4 SetHardwareFlowControl(UartHardwareFlowControl control) . . . . .	166
13.31.3.5 SetSpeed(unsigned int baud_rate) . . . . .	167
13.31.3.6 Transmit(const uint8_t *data, unsigned int size, WaitMilliseconds timeout_ms) . . . . .	167
13.31.3.7 TransmitCompleteCallback(void *const ptr) . . . . .	167
13.32 murasaki::UartLogger Class Reference . . . . .	168
13.32.1 Detailed Description . . . . .	169
13.32.2 Constructor & Destructor Documentation . . . . .	169
13.32.2.1 UartLogger(UartStrategy *uart) . . . . .	169
13.32.3 Member Function Documentation . . . . .	169
13.32.3.1 DoPostMortem(void *debugger_fifo) . . . . .	170
13.32.3.2 getCharacter() . . . . .	170
13.32.3.3 putMessage(char message[], unsigned int size) . . . . .	170
13.33 murasaki::UartStrategy Class Reference . . . . .	170
13.33.1 Detailed Description . . . . .	172
13.33.2 Member Function Documentation . . . . .	172
13.33.2.1 HandleError(void *ptr)=0 . . . . .	172
13.33.2.2 Receive(uint8_t *data, unsigned int size, unsigned int *transferred_count=nullptr, UartTimeout uart_timeout=murasaki::kutNoldleTimeout, WaitMilliseconds timeout_ms=murasaki::kwmsIndefinitely)=0 . . . . .	172
13.33.2.3 ReceiveCompleteCallback(void *ptr)=0 . . . . .	173
13.33.2.4 SetHardwareFlowControl(UartHardwareFlowControl control) . . . . .	173
13.33.2.5 SetSpeed(unsigned int speed) . . . . .	173
13.33.2.6 Transmit(const uint8_t *data, unsigned int size, WaitMilliseconds timeout_↵ ms=murasaki::kwmsIndefinitely)=0 . . . . .	173
13.33.2.7 TransmitCompleteCallback(void *ptr)=0 . . . . .	174



<b>14 File Documentation</b>	<b>175</b>
14.1 /home/takemasa/workspace_st/nucleo-f446-sample/Inc/main.h File Reference . . . . .	175
14.1.1 Detailed Description . . . . .	176
14.1.2 Function Documentation . . . . .	176
14.1.2.1 Error_Handler(void) . . . . .	176
14.2 /home/takemasa/workspace_st/nucleo-f446-sample/Inc/murasaki_include_stub.h File Reference . .	177
14.2.1 Detailed Description . . . . .	177
14.3 /home/takemasa/workspace_st/nucleo-f446-sample/Inc/murasaki_platform.hpp File Reference . . .	178
14.3.1 Detailed Description . . . . .	179
14.4 /home/takemasa/workspace_st/nucleo-f446-sample/Inc/platform_config.hpp File Reference . . . .	179
14.4.1 Detailed Description . . . . .	180
14.4.2 Macro Definition Documentation . . . . .	180
14.4.2.1 MURASAKI_CONFIG_NOSYSLOG . . . . .	180
14.5 /home/takemasa/workspace_st/nucleo-f446-sample/Inc/platform_defs.hpp File Reference . . . . .	180
14.5.1 Detailed Description . . . . .	181
14.6 /home/takemasa/workspace_st/nucleo-f446-sample/Inc/stm32f4xx_hal_conf.h File Reference . . .	181
14.6.1 Detailed Description . . . . .	182
14.6.2 Macro Definition Documentation . . . . .	183
14.6.2.1 assert_param . . . . .	183
14.6.2.2 EXTERNAL_CLOCK_VALUE . . . . .	183
14.6.2.3 HSE_STARTUP_TIMEOUT . . . . .	183
14.6.2.4 HSE_VALUE . . . . .	183
14.6.2.5 HSI_VALUE . . . . .	184
14.6.2.6 LSE_STARTUP_TIMEOUT . . . . .	184
14.6.2.7 LSE_VALUE . . . . .	184
14.6.2.8 LSI_VALUE . . . . .	184
14.6.2.9 PHY_AUTONEGO_COMPLETE . . . . .	184
14.6.2.10 PHY_AUTONEGOTIATION . . . . .	184
14.6.2.11 PHY_BCR . . . . .	184
14.6.2.12 PHY_BSR . . . . .	184

14.6.2.13 PHY_DUPLEX_STATUS . . . . .	184
14.6.2.14 PHY_FULLDUPLEX_100M . . . . .	185
14.6.2.15 PHY_FULLDUPLEX_10M . . . . .	185
14.6.2.16 PHY_HALFDUPLEX_100M . . . . .	185
14.6.2.17 PHY_HALFDUPLEX_10M . . . . .	185
14.6.2.18 PHY_ISOLATE . . . . .	185
14.6.2.19 PHY_JABBER_DETECTION . . . . .	185
14.6.2.20 PHY_LINKED_STATUS . . . . .	185
14.6.2.21 PHY_LOOPBACK . . . . .	185
14.6.2.22 PHY_POWERDOWN . . . . .	185
14.6.2.23 PHY_RESET . . . . .	185
14.6.2.24 PHY_RESTART_AUTONEGOTIATION . . . . .	186
14.6.2.25 PHY_SPEED_STATUS . . . . .	186
14.6.2.26 PHY_SR . . . . .	186
14.6.2.27 TICK_INT_PRIORITY . . . . .	186
14.6.2.28 VDD_VALUE . . . . .	186
14.6.3 Function Documentation . . . . .	186
14.6.3.1 assert_failed(uint8_t *file, uint32_t line) . . . . .	186
14.7 /home/takemasa/workspace_st/nucleo-f446-sample/Inc/stm32f4xx_it.h File Reference . . . . .	187
14.7.1 Detailed Description . . . . .	187
14.8 /home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/tp/adau1361.hpp File Reference . . . . .	187
14.8.1 Detailed Description . . . . .	188
14.9 /home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/audiocodecstrategy.hpp File Reference . . . . .	189
14.9.1 Detailed Description . . . . .	189
14.10/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/audiostrategy.hpp File Reference . . . . .	190
14.10.1 Detailed Description . . . . .	190
14.11/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitin.hpp File Reference . . . . .	191
14.11.1 Detailed Description . . . . .	192
14.12/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitinstrategy.hpp File Reference . . . . .	193
14.12.1 Detailed Description . . . . .	194

14.13/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitout.hpp File Reference . . . .	195
14.13.1 Detailed Description . . . . .	196
14.14/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitoutstrategy.hpp File Reference	197
14.14.1 Detailed Description . . . . .	198
14.15/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/criticalsection.hpp File Reference	199
14.15.1 Detailed Description . . . . .	199
14.16/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/debugger.hpp File Reference . .	200
14.16.1 Detailed Description . . . . .	201
14.17/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/debuggerfifo.hpp File Reference .	202
14.17.1 Detailed Description . . . . .	203
14.18/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/debuggeruart.hpp File Reference	204
14.18.1 Detailed Description . . . . .	205
14.19/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/fifostrategy.hpp File Reference .	206
14.19.1 Detailed Description . . . . .	207
14.20/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cmaster.hpp File Reference . .	207
14.20.1 Detailed Description . . . . .	208
14.21/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cmasterstrategy.hpp File Refer- ence . . . . .	209
14.21.1 Detailed Description . . . . .	210
14.22/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cslave.hpp File Reference . . .	211
14.22.1 Detailed Description . . . . .	212
14.23/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cslavestrategy.hpp File Reference	213
14.23.1 Detailed Description . . . . .	214
14.24/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/loggerstrategy.hpp File Reference	215
14.24.1 Detailed Description . . . . .	216
14.25/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki.hpp File Reference . .	217
14.25.1 Detailed Description . . . . .	218
14.26/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_0_intro.hpp File Refer- ence . . . . .	218
14.26.1 Detailed Description . . . . .	218
14.27/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_1_env.hpp File Reference	218

14.27.1 Detailed Description . . . . .	218
14.28/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_2_ug.hpp File Reference	218
14.28.1 Detailed Description . . . . .	218
14.29/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_3_pg.hpp File Reference	219
14.29.1 Detailed Description . . . . .	219
14.30/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_4_mod.hpp File Reference . . . . .	219
14.30.1 Detailed Description . . . . .	219
14.31/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_5_spg.hpp File Reference . . . . .	219
14.32/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_assert.hpp File Reference . . . . .	219
14.32.1 Detailed Description . . . . .	221
14.33/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_config.hpp File Reference . . . . .	221
14.33.1 Detailed Description . . . . .	222
14.34/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_defs.hpp File Reference	223
14.34.1 Detailed Description . . . . .	223
14.35/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_syslog.hpp File Reference . . . . .	224
14.35.1 Detailed Description . . . . .	225
14.36/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/peripheralstrategy.hpp File Reference . . . . .	225
14.36.1 Detailed Description . . . . .	226
14.37/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spimaster.hpp File Reference . . . . .	226
14.37.1 Detailed Description . . . . .	228
14.38/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spimasterstrategy.hpp File Reference . . . . .	228
14.38.1 Detailed Description . . . . .	229
14.39/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislave.hpp File Reference . . . . .	230
14.39.1 Detailed Description . . . . .	231
14.40/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislaveadapter.hpp File Reference	232
14.40.1 Detailed Description . . . . .	233

14.41/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislaveadapterstrategy.hpp File Reference . . . . .	234
14.41.1 Detailed Description . . . . .	235
14.42/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislavestrategy.hpp File Reference . . . . .	236
14.42.1 Detailed Description . . . . .	237
14.43/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/synchronizer.hpp File Reference . . . . .	238
14.43.1 Detailed Description . . . . .	239
14.44/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/task.hpp File Reference . . . . .	239
14.44.1 Detailed Description . . . . .	240
14.45/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/taskstrategy.hpp File Reference . . . . .	241
14.45.1 Detailed Description . . . . .	242
14.46/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/uart.hpp File Reference . . . . .	242
14.46.1 Detailed Description . . . . .	243
14.47/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/uartlogger.hpp File Reference . . . . .	244
14.47.1 Detailed Description . . . . .	245
14.48/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/uartstrategy.hpp File Reference . . . . .	246
14.48.1 Detailed Description . . . . .	247
14.49/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Src/allocators.cpp File Reference . . . . .	247
14.49.1 Detailed Description . . . . .	248
14.50/home/takemasa/workspace_st/nucleo-f446-sample/Src/main.c File Reference . . . . .	249
14.50.1 Detailed Description . . . . .	249
14.50.2 Function Documentation . . . . .	250
14.50.2.1 assert_failed(uint8_t *file, uint32_t line) . . . . .	250
14.50.2.2 Error_Handler(void) . . . . .	250
14.50.2.3 HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) . . . . .	250
14.50.2.4 main(void) . . . . .	250
14.50.2.5 StartDefaultTask(void const *argument) . . . . .	251
14.50.2.6 SystemClock_Config(void) . . . . .	251
14.50.3 Variable Documentation . . . . .	251
14.50.3.1 hdma_usart2_rx . . . . .	251
14.51/home/takemasa/workspace_st/nucleo-f446-sample/Src/murasaki_platform.cpp File Reference . . . . .	252
14.51.1 Detailed Description . . . . .	252
14.51.2 Function Documentation . . . . .	252
14.51.2.1 HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c) . . . . .	252
14.51.2.2 HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c) . . . . .	253
14.52/home/takemasa/workspace_st/nucleo-f446-sample/Src/stm32f4xx_it.c File Reference . . . . .	253
14.52.1 Detailed Description . . . . .	254
14.52.2 Variable Documentation . . . . .	254
14.52.2.1 hdma_usart2_rx . . . . .	254
14.53/home/takemasa/workspace_st/nucleo-f446-sample/Src/system_stm32f4xx.c File Reference . . . . .	255
14.53.1 Detailed Description . . . . .	256



# Chapter 1

## Preface

Murasaki, is a class library on the STM32Cube HAL and FreeRTOS.

By using Murasaki, you can program STM32 series quickly and easily. You can obtain the source code of the Murasaki Library from the [GitHub repository](#).

Murasaki has following design philosophies:

- [Simplified IO](#)
- [Preemptive multi-task](#)
- [Blocking IO](#)
- [Thread safe IO](#)
- [Versatile printf\(\) logger](#)
- [Guard by assertion](#)
- [System Logging](#)
- [Configurable](#)

### 1.1 Simplified IO

The IO function is packaged by class types. For example, The [murasaki::Uart](#) class can receive a UART handle

```
murasaki::AbstractUart * uart3 = new murasaki::Uart( &huart3 );
```

Where huart3 is a UART port 3 handle generated by the CubeMX.

The STM32Cube HAL is quite rich and flexible. On the other hand, it is quite huge and complex. The classes in Murasaki simplifies it by letting flexibility beside. For example, the [murasaki::Uart](#) class can support only the DMA transfer. The interrupt-based transfer is not supported. By giving up the flexibility, programming with Murasaki is easier than using HAL directly.

## 1.2 Preemptive multi-task

The Murasaki class library is built on FreeRTOS's preemptive configuration. As a result, Murasaki is automatically aware with preemptive multi-task.

That means, Murasaki's classes don't use polling to wait for any event. Then, a task can do some job while other tasks are waiting for some event.

The multi-task programming helps to divide a bigger program to sub-units. This is a good way to develop a large program easier. And the more important point, it is easier to maintain.

## 1.3 Blocking IO

The blocking IO is one of the most important features of Murasaki.

The peripheral wrapping class like `murasaki::Uart` provides a set of member functions to do the data transmission/receiving. Such the member functions are programmed as "blocking" IO.

The blocking IO function doesn't return until each IO function finished completely. For example, if you transmit 10bytes through the UART, the IO member function transmits the 10bytes data, and then, return.

Note: Sometimes, the "completion" means the end of the DMA transfer session, rather than the true transmission of the last byte. In this case, system generates a completion interrupt while the data is still in FIFO of the peripheral. This is a hardware issue.

To provide the blocking IO, some member functions are restricted to use only in the task context.

## 1.4 Thread safe IO

The blocking IO and the preemptive multi-task provide easier programming. In the other hand, there is a possibility that two different task accesses one peripheral simultaneously. This kind of access messes the peripheral's behavior.

To prevent this condition, each peripheral wrapping class has exclusive access mechanism by mutex.

By this mechanism, if two tasks try to transmit through one peripheral, one task is kept waiting until the other finished to transmit.

## 1.5 Versatile printf() logger

Logging or "printf debug" is a strong tool in the embedded system development.

Murasaki has three levels of the printf debugging mechanism. One is the `murasaki::debugger->Printf()`, the second is `MURASAKI_ASSERT` macro. In addition to these two, `MURASAKI_SYSLOG` macro is available.

The `murasaki::debugger->Printf()` is flexible output mechanism which has several good features :

- printf() compatible parameters.
- Task/interrupt bi-context operation
- None-blocking logging by internal buffer.
- User configurable output port

These features allow a programmer to do the printf() debug not only in the task context but also in the interrupt context.



## 1.6 Guard by assertion

In addition to the `murasaki::debugger->Printf()`, programmer can use `MURASAKI_ASSERT` macro. This allows easy assertion and logging. This macro uses the `murasaki::debugger->Printf()` internally.

This assertion is used inside Murasaki class library. As a result, the wrong context, wrong parameter, etc will be reported to the debugger output.

## 1.7 System Logging

`MURASAKI_SYSLOG` provides the message output based on the level and filtering. This mechanism is intended to help the Murasaki library development. But also application can use this mechanism.

## 1.8 Configurable

Murasaki is configurable from the two point of view.

First, Musaraki's modules enable only when the relevant peripheral is generated by CubeMX. This allows you set the CubeMX to generate only the used peripheral's source code. Such the setting makes total source code smaller. In the other hand, all unused drivers are invisible. For example, if you don't enable the I2C pins on CubeMX, Murasaki cannot see such the module.

Murasaki can adopt such the situation. The source code of Murasaki relevant to the peripheral which is not generated, will be disabled by `ifdef` control.

The Second part of the configurable characteristics is Murasaki itself. The programmer can customize the Murasaki for example, task stack size.



## Chapter 2

# Target and Environment

Murasaki library was originally developed with following environment:

- Nucleo F746ZG ( STM32F746ZG )
- STM32CubeMX 5.0
- SW4STM32 1.16.0.201807130628 ( with eclipse 4.6.3 )
- Ubuntu 16.04.03 (64bit)

And then, confirmed portability with following boards :

- Nucleo F746ZG ( STM32F746ZG : Cortex-M7 )
- Nucleo F722ZE ( STM32F722ZE : Cortex-M7 ),
- Nucleo F303K8 ( STM32F303K8 : Cortex-M4 )
- Nucleo L152RE ( STM32L152RE : Cortex-M3 )
- Nucleo F091RC ( STM32F091RC : Cortex-M0 )



## Chapter 3

# Usage Introduction

In this introduction, we see how to use Murasaki class library in the STM32 program.

In this section, we see following issues :

- [Message output](#)
- [Serial communication](#)
- [Debugging with Murasaki.](#)
- [Tasking](#)
- [Other peripheral](#)
- [Program flow](#)

For the easy-to-understand description, we assume several things on the application skeleton which we are going to use Murasaki :

- The application skeleton is generated by [CubeMX](#)
- The application skeleton is configured to use FreeRTOS
- UART3 is configured to work with DMA.

### 3.1 Message output

The Murasaki library has a Printf() like message output mechanism.

This mechanism is an easy way to display a message from an embedded microcomputer to the terminal simulator like kermit on a host computer. Murasaki's Printf() is based on the standard C language formatting library. So, programmer can output a message as like standard printf().

As usual, let's start from "hello, world".

```
murasaki::debugger->Printf("Hello, world!\n");
```

In Murasaki manner, the `Printf()` is not a global function. This is a method of `murasaki::Debugger` class. The `murasaki::debugger` variable is a one of two Murasaki's global variable. And it provide an easy to use message output.

The end-of-line character is depend on the terminal. In the above sample, the terminator is `\n`. This is for the linux based kermit. Other terminal system may need other end-of-line character.

Because the `Printf()` works as like standard `printf()`, you can also use the format string.

```
murasaki::debugger->Printf("count is %d\n", count);
```

The `Printf()` is designed as debugger message output for an embeded realtime system. Thenk this function is :

- Thread safe
- Blocking
- Buffered

In the other word, you can use this function in either task or interrupt handler without bothering the real time process.

## 3.2 Serial communication

`murasaki::Uart` is the asynchronous serial communication.

The initial baud rate, parity and data size are defined by CubeMX. So, there is no need to initialize the communication parameter in application program. User can transmit data by just passing its address and size.

```
uint8_t data[5] = { 1, 2, 3, 4, 5 };
murasaki::UartStatus stat;

stat = murasaki::platform.uart->Transmit(
    data,
    5);
```

Beside of transmit, also `Receive()` member function exists.

## 3.3 Debugging with Murasaki.

As we saw, Murasaki has a simple messaging output for real-time debugging.

This feature is typically used as UART serial output, but configurable by the programmer.

The `murasaki::debugger` is the useful variable to output the debugging message. `murasaki::debugger->prntf()` has several good feature.

- Versatile `printf()` style format string.
- Can call from both task and interrupt context
- Non-blocking

These features help the programmer to display the message in the real-time, multi-task application.

In addition to this simple debugging variable, a programmer can use `assert_failure()` function of the STM32 HA. The STM32Cube HAL has `assert_failure()` to check the parameter on the fly. By default, this function is disabled. To use this function, programmer have to make it enable, and add function to receive the debug information.

To enable the `assert_failuer()`, edit the `stm32fxx_hal_conf.h` in the `Inc` directory. This file is generated by CubeMX. You can find `USE_FULL_ASSERT` macro as comment out. By declaring this macro, `assert_failure` is enabled.

```
#define USE_FULL_ASSERT    1
```

And then, you should modify `assert_failure()` in `main.c`, to call output function (Note, this modification is altered by the install script. See [Clone the Murasaki repository and install](#) of the [Step-by-Step Porting Guide](#). Still `USE_FULL_ASSERT` macro is a responsibility of the porting programmer ).

```
void assert_failed(uint8_t* file, uint32_t line)
{
    CustomAssertFailed(file, line); // debugging stub.
}
```

This hook calls `CustomAssertFailed()` function.

```
// Hook for the assert_failure() in main.c
void CustomAssertFailed(uint8_t* file, uint32_t line)
{
    murasaki::debugger->Printf("Wrong parameters value: file %s on line %d\n", file
    , line);
}
```

Once above programming is done, you can watch the integrity of the HAL parameter by reading the console output.

Above debugging mechanism redirects all HAL assertion, Murasaki assertion and application debug message to the specified logging port. That logging port is able to customize. In the case of the User's Guide, logging is done through the UART port.

Time by time, you may not want to connect a serial terminal to the board unless you have a problem. That means when you find a problem and connect your serial terminal, the assertion message is already transmitted ( and lost ).

Murasaki can save this problem. By adding the following code after creating `murasaki::Debugger` instance, you can use history functionality.

```
murasaki::debugger->AutoHistory();
```

The `murasaki::Debugger::AutoHistory()` creates a dedicated task for auto history function. This task watch the input from the logging port. Again, in this User's guide it is UART. Once any character is received from the logging port ( terminal ), previously transmitted message is sent again. So you can read the last tens of messages.

The auto history is handy, but it blocks all input from the terminal. If you want to have your own console program through the debug port input, do not you the auto history. Alternatively, you can send the previously transmitted message again, by calling `murasaki::Debugger::PrintHistory()` explicitly.

Murasaki also have post-mortem debugging feature which helps to analyze severe error. Murasaki adds a hook into the `Default_Handler` of the `startup_stm32****.s` file.

```
.section .text.Default_Handler,"ax",%progbits
.global CustomDefaultHandler
Default_Handler:
    bl CustomDefaultHandler
Infinite_Loop:
    b Infinite_Loop
```

The inserted `bl` instruction supersedes the infinite loop at spurious interrupt handler. Alternatively, `CustomDefaultHandler()` is called. The `CustomDefaultHandler()` stops entire Debugger process, and get into the polling mode serial operation with auto history.

That mean, once spurious interrupt happen, you can read the messages in the debug message FIFO by pressing any key. This feature helps to analyze the assertion message just before the trouble.

## 3.4 Tasking

`murasaki::Task` is a type of the task of the FreeRTOS.

By using `murasaki::Task`, a programmer can easily create a task object. This object encapsulate the task of the FreeRTOS.

First of all, you must define a task body function. Any function name is acceptable, Only the return type and parameter type is specified.

```
// Task body of the murasaki::platform.task1
void TaskBodyFunction(const void* ptr)
{
    while (true)    // dummy loop
    {
        murasaki::platform.led2->Toggle(); // toggling LED
        murasaki::Sleep(static_cast<murasaki::WaitMilliseconds>(700));
    }
}
```

Then, create a Task object.

There are several parameter to pass for the constructor. The first parameter is the name of the task in FreeRTOS. The second one is the task stack size. This size is depend on the task body function. The third one is the priority of the new task. This bigger value is the higher priority. The fourth one is the pointer to the task parameter. This parameter is passed to the task function body. And then, the last one is the pointer to the task body function.

```
// For demonstration of FreeRTOS task.
murasaki::platform.task1 = new murasaki::Task(
    "Master",
    256,
    (( configMAX_PRIORITIES > 1) ? 1 : 0),
    nullptr,
    &TaskBodyFunction
);
```

Once task object is created, you must call `Start()` member function to start the task.

```
murasaki::platform.task1->Start();
```

Then, task you can call `Start()` member function to run.

## 3.5 Other peripheral

This section shows samples of the other peripherals.

- [I2C Master](#)
- [I2C Slave](#)
- [SPI Master](#)
- [SPI Slave](#)
- [GPIO](#)



### 3.5.1 I2C Master

`murasaki::I2cMaster` class provides the serial communication

The I2C master is easy to use. To send a message to the slave device, you need to specify the slave address in 7bits, pointer to data and data size in byte.

```
uint8_t data[5] = { 1, 2, 3, 4, 5 };
murasaki::I2cStatus stat;

stat = murasaki::platform.i2cMaster->Transmit(
    127,
    data,
    5);
```

In addition to the `Transmit()`, `murasaki::I2cMaster` class has `Receive()`, and `TransmitThenReceive()` member function.

### 3.5.2 I2C Slave

`murasaki::I2cSlave` class provides the I2C slave function.

The I2C slave is much easier than master, because it doesn't need to specify the slave address. The I2C slave device address is given by CubeMX.

```
uint8_t data[5];
murasaki::I2cStatus stat;

stat = murasaki::platform.i2cSlave->Receive(
    data,
    5);
```

In addition to the `Transmit()`, `murasaki::I2cSlave` class has `Receive()` member function.

### 3.5.3 SPI Master

`murasaki::SpiMaster` is the SPI master class of Murasaki.

This class is more complicated than other peripherals, because of flexibility. The SPI master controller must adapt to the several variation of the SPI communication.

- CPOL configuration
- CPHA configuration
- GPIO port configuration to select a slave

The flexibility to above configurations need special mechanism. In Murasaki, this flexibility is responsibility of the `murasaki::SpiSlaveAdapter` class. This class holds these configuration. Then, passed to the master class.

So, you must create a `murasaki::SpiSlaveAdapter` class object, at first.

```
// Create a slave adapter. This object specify the protocol and slave select pin
murasaki::SpiSlaveAdapterStrategy * slave_spec;
slave_spec = new murasaki::SpiSlaveAdapter(
    murasaki::kspoFallThenRise,
    murasaki::ksphLatchThenShift,
    SPI_SLAVE_SEL_GPIO_Port,
    SPI_SLAVE_SEL_Pin
);
```

Then, you can pass the `SpiSlaveAdapter` class object to the `murasaki::SpiMaster::TransmitAndRecieve()` function.

```
// Transmit and receive data through SPI
uint8_t tx_data[5] = { 1, 2, 3, 4, 5 };
uint8_t rx_data[5];
murasaki::SpiStatus stat;
stat = murasaki::platform.spiMaster->TransmitAndReceive(
    slave_spec,
    tx_data,
    rx_data,
    5);
```

### 3.5.4 SPI Slave

[murasaki::SpiSlave](#) class provides the SPI slave functionality.

This class encapsulate the SPI slave function.

```
// Transmit and receive data through SPI
uint8_t tx_data[5] = { 1, 2, 3, 4, 5 };
uint8_t rx_data[5];
murasaki::SpiStatus stat;
stat = murasaki::platform.spiSlave->TransmitAndReceive(
    tx_data,
    rx_data,
    5);
```

### 3.5.5 GPIO

[murasaki::BitOut](#) and [murasaki::BitIn](#) provides the GPIO functionality

Following is the example of the [murasaki::BitOut](#) class.

```
// Toggle LED.
murasaki::platform.led->Toggle();
```

In addition to the Toggle(), BitIn has Set() and Clear() member function.

## 3.6 Program flow

In this section, we see the program flow of a Murasaki application.

Murasaki has 3 program flows. The start point of these flows are always inside CubeMX generated code. 2 out of 3 flows are for debugging. Only 1 flow have to be understood well by an application programmer.

- [Application flow](#)
- [HAL Assertion flow](#)
- [Spurious Interrupt flow](#)
- [Assertion flow](#)
- [General Interrupt flow](#)
- [EXTI flow](#)

### 3.6.1 Application flow

The application program flow is the main flow of a Murasaki application.

This program flow starts from the `StartDefaultTask()` in the `Src/main.c`. The `StartDefaultTask()` is a default and first task created by CubeMX. In the other words, this task is automatically created without configuration.

From this function, two Murasaki function is called. One is `InitPlatform()`. The other is `ExecPlatform()`. Note that both function calls are inserted by installer. See [Clone the Murasaki repository and install](#) of the [Step-by-Step Porting Guide](#) for details.

```
void StartDefaultTask(void const * argument)
{
    // USER CODE BEGIN 5
    InitPlatform();
    ExecPlatform();
    // Infinite loop
    for(;;)
    {
        osDelay(1);
    }
    // USER CODE END 5
}
```

The `InitPlatform()` function is defined in the `Src/murasaki_platform.cpp`. Because the file extension is `.cpp`, the `murasaki_platform.cpp` is compiled by C++ compiler while the `main.c` is compiled by C compiler. This allows programmer uses C++ language. Thus, the `InitPlatform()` is the good place to initialize the class based variables.

As the name suggests, `InitPlatform()` is where programmer initialize the platform variables `murasaki::platform` and `murasaki::debugger`.

```
void InitPlatform()
{
    // UART device setting for console interface.
    // On Nucleo, the port connected to the USB port of ST-Link is
    // referred here.
    murasaki::platform.uart_console = new
        murasaki::DebuggerUart(&huart3);
    // UART is used for logging port.
    // At least one logger is needed to run the debugger class.
    murasaki::platform.logger = new murasaki::UartLogger(
        murasaki::platform.uart_console);
    // Setting the debugger
    murasaki::debugger = new murasaki::Debugger(
        murasaki::platform.logger);
    // Set the debugger as AutoRePrint mode, for the easy operation.
    murasaki::debugger->AutoRePrint(); // type any key to show history.

    // For demonstration, one GPIO LED port is reserved.
    // The port and pin names are fined by CubeMX.
    murasaki::platform.led = new murasaki::BitOut(LD2_GPIO_Port,
        LD2_Pin);

    // For demonstration of master and slave I2C
    murasaki::platform.i2c_master = new murasaki::I2cMaster(&hi2c1);

    murasaki::platform.sync_with_button = new
        murasaki::Synchronizer();

    // For demonstration of FreeRTOS task.
    murasaki::platform.task1 = new murasaki::Task(
        "Master",
        256,
        1,
        nullptr,
        &TaskBodyFunction
    );

    // the Following block is just for sample.
}
```

In this sample, the first half of the `InitPlatform()` is building a `murasaki::debugger` variable. Because this variable is utilized for the debugging of the entire application, there is a value to make it at first.

Probably the most critical statement in this part is the creation of the `DebuggerUart` class object.

```
murasaki::platform.uart_console = new
    murasaki::DebuggerUart (&huart3);
```

In this statement, the `DebuggerUart` receives the pointer to the `huart3` as a parameter. The `huart3` is a handle variable of the UART3 generated by CubeMx. Let's remind the UART3 is utilized as communication path through the USB. So, in this sample code, we are making debugging console through the USB-serial line of the Nucleo F722ZE board.

Because the `huart3` is generated into the `main.c` directory, we have to declare this variable as an external variable. You can find the declaration around the top of the `Src/murasaki_platform.cpp`.

```
extern UART_HandleTypeDef huart3;
```

Note that the UART port number varies among the different Nucleo board. So, the porting programmer have a responsibility to refer the right UART.

The second half of the `InitPlatform()` is the creation part of the other peripheral object. This part fully depends on the application. A programmer can define any object, by modifying the `murasaki::Platform` struct in the `Inc/platform_↵  
defs.hpp`.

The second function called from the `StartDefaultTask()` is the `ExecPlatform()`. This function is also defined in the `Src/murasaki_platform.cpp`.

```
void ExecPlatform()
{
    murasaki::platform.task1->Start();

    // print a message with counter value to the console.
    murasaki::debugger->Printf("Push user button to display the I2C slave device \n
    ");

    // Loop forever
    while (true) {
        murasaki::platform.sync_with_button->Wait();
        I2cSearch(murasaki::platform.i2c_master);
    }
}
```

This function is the body of application. So, you can read GPIO, ADC other peripherals. And output to the DAC, GPIO, and other peripherals from here.

### 3.6.2 HAL Assertion flow

HAL Assertion is a STM32Cube HAL's programming help mechanism.

STM32Cube HAL provides a run-time parameter check. This parameter check is enabled by un-comment the `USE_FULL_ASSERT` macro inside `stm32xxx_hal_conf.h` file. See "Run-time checking" of the HAL manual for detail.

Assertion is defined in `Src/main.c`. As `assert_failed()` function. This function is empty at first. The murasaki install script fills by `CustomAssertFailed()` calling statement.

```
void assert_failed(uint8_t *file, uint32_t line)
{
    // USER CODE BEGIN 6
    CustomAssertFailed(file, line);
    // USER CODE END 6
}
```

If a HAL API received wrong parameter, the `assert_failed()` function is called with its filename and line number. Then, `assert_failed()` call `CustomAssertFailed()` function in the `Src/murasaki_platform.cpp` file.

The `CustomAssertFailed()` print the filename and line number with message.

```
void CustomAssertFailed(uint8_t* file, uint32_t line) {
    murasaki::debugger->Printf(
        "Wrong parameters value: file %s on line %d\n",
        file,
        line);
}
```

### 3.6.3 Spurious Interrupt flow

Murasaki provides a mechanism to catch a spurious interrupt.

`Default_handler` is the entry point of the spurious interrupt handler. This is defined in `startup/startup_stm32xxxx.s`.

The install script modify this handler to call the pref `CustomDefaultHandler()` in the `Src/murasaki_platform.cpp`.

```
.section .text.Default_Handler,"ax",%progbits
.global CustomDefaultHandler
Default_Handler:
    bl CustomDefaultHandler
Infinite_Loop:
    b Infinite_Loop
```

`CustomDefaultHandler()` put the debugger to the post mortem state which can work without the debug helper tasks. This function keep watching UART and if any input is found, it flushes the entire data of debug FIFO.

Thus, programmer can see the last messages before triggering spurious interrupt.

```
void CustomDefaultHandler() {
    // Call debugger's post mortem processing. Never return again.
    murasaki::debugger->DoPostMortem();
}
```

### 3.6.4 Assertion flow

The assertion flow is similar to the Spurious Interrupt flow.

Once assertion is raised, assertion macro raised Hard Fault exception. The Hard Fault exception handler in the Src/st32\*\*\*\*\_it.c calls CustomDefaultHandler.

```
void HardFault_Handler(void)
{
    CustomDefaultHandler();
    while (1)
    {
    }
}
```

### 3.6.5 General Interrupt flow

As described in the HAL manual, STM32Cube HAL handles all peripheral related interrupt, and then, call corresponding callback function.

These call backs are optional from the view point of the peripheral hardware, but essential hook to sync with software.

Murasaki is using these callback to notify the end of processing, to the peripheral class objects. For example, following is the sample of callback.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef * huart)
{
    // Poll all uart rx related interrupt receivers.
    // If hit, return. If not hit, check next.
    if (murasaki::platform.uart_console->ReceiveCompleteCallback(huart))
        return;
}
```

This callback is called from HAL, after the end of peripheral interrupt processing. And calling the ReceiveCompleteCallback() of the UART object in the platform. Note that Murasaki object returns true, if the callback member function parameter matches with its own hardware handle. Then, the function can return if the return value is true.

Note that forwarding this call back to all the relevant peripheral is a Responsibility of the porting programmer. To forward the callback to the multiple objects, you can call like this.

```
if (murasaki::platform.uart_console->ReceiveCompleteCallback(huart))
    return;
if (murasaki::platform.uart_1->ReceiveCompleteCallback(huart))
    return;
if (murasaki::platform.uart_2->ReceiveCompleteCallback(huart))
    return;
```

### 3.6.6 EXTI flow

EXTI flow is very similar to the [General Interrupt flow](#) except its timing.

While other peripheral raises interrupt after the peripheral instance are created, EXTI peripheral may raise the interrupt before the platform peripherals are ready.

Then, EXTI call back has guard to avoid the null pointer access.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if ( USER_Btn_Pin == GPIO_Pin) {
        // release the waiting task
        if (murasaki::platform.sync_with_button != nullptr)
            murasaki::platform.sync_with_button->Release();
    }
}
```

## Chapter 4

# Porting guide

This porting guide introduces murasaki class library porting step by step.

In this guide, user will study the library porting to the STM32 microcomputer system working with STM32Cube HAL.

Followings are the contents of this porting guide :

- [Directory Structure](#)
- [CubeMX setting](#)
- [Configuration](#)
- [Task Priority and Stack Size](#)
- [Heap memory consideration](#)
- [Platform variable](#)
- [Routing interrupts](#)
- [Error handling](#)
- [Summary of the porting](#)

There are some other manuals of murasaki class library :

- [Preface](#)
- [Usage Introduction](#)
- [Murasaki Class Collection](#)

### 4.1 Directory Structure

Murasaki has four main directory and several user-modifiable files.

This page describes these directories and files.

### 4.1.1 Src directory

Almost files of the Murasaki source code are stored in this directory. Basically, there is no need to edit the files inside this directory, except the development of Murasaki itself. The project setting must refer this directory as the source directory.

### 4.1.2 Inc directory

This directory contains the include files, the project setting must refer this directory as an include directory.

### 4.1.3 Src-tp and Inc-tp directory

The class collection of the third party peripherals. The "third party" means, the outside of the microprocessor.

Currently these directories are not utilized.

### 4.1.4 murasaki.hpp

Usually, the [murasaki.hpp](#) include file is the only one to include from an application program. By including this file, an application can refer all the definition of the Murasaki

This file is stored in the Inc directory.

### 4.1.5 template directory

#### 4.1.5.1 platform\_config.hpp

The [platform\\_config.hpp](#) file is a collection of the build configuration. By defining a macro, a programmer can change the behavior of the Murasaki.

There are mainly two types of the configuration in this file.

One type of configuration is to override the [murasaki\\_config.hpp](#) file. All contents of the [murasaki\\_config.hpp](#) are macros. These macros are defined to control the Murasaki, for example: the task priority, the task stack size or the timeout period, described in the [Definitions and Configuration](#).

The other configuration type is the assertion inside Murasaki. See [MURASAKI\\_CONFIG\\_NODEBUG](#) for details.

The [platform\\_config.hpp](#) is better to be copied in the /Inc directory of the application. The [install script](#) will copy this file to /Src directory of application for programmer.



#### 4.1.5.2 platform\_defs.hpp

As same as [platform\\_config.hpp](#), the [platform\\_defs.hpp](#) is not the core part of the Murasaki class library. This include file has a definition of the [murasaki::platform](#) which provide "nice looking" aggregation of the class objects.

The application programmer can define the [murasaki::Platform](#) type freely. There is no limitation or requirement what you put into unless compiler reports an error message.

On the other hand, a programmer may find that adding the peripheral-based class variables and middleware based class variables into the [murasaki::Platform](#) type is reasonable. Actually, the independent devices ( ie:I2C connected LCD controller ) may be better to be a member variable of the [mruasaki::Platform](#) type.

The [platform\\_defs.hpp](#) is better to be copied in the /Inc directory of the application. The [install script](#) will copy this file to /Src directory of application for programmer.

See [Application Specific Platform](#) as usage sample.

#### 4.1.5.3 murasaki\_platform.hpp

A header file of the [murasaki\\_platform.cpp](#). This file is better to be copied in the /Inc directory of the application. The [install script](#) will copy this file to /Src directory of application for programmer.

#### 4.1.5.4 murasaki\_platform.cpp

The [murasaki\\_platform.cpp](#) is the interface between the application and the HAL/RTOS. This file has variables / functions which user needs to program at porting time.

- [murasaki::platform](#) variable
- [murasaki::debugger](#) variable
- [InitPlatform\(\)](#) to initialize the platform variable
- [ExecPlatform\(\)](#) to execute the platform algorithm
- [Interrupt routing functions](#)
- [HAL assertion function and Custome default exception handler](#)

The [murasaki\\_platform.cpp](#) is better to be copied in the /Src directory of the application. The [install script](#) will copy this file to /Src directory of application for programmer.

#### 4.1.6 install script

The install script have mainly 4 tasks.

- Copy template files to the appropriate application directories from [template directory](#)
- Modify [main.c](#) to call the [InitPlatform\(\)](#) and [ExecPlatform\(\)](#) from the default task.
- Modify [main.c](#) to call the [CustomAssertFailed\(\)](#) from the HAL assertion
- Modify the hard fault handler to call the [CustomDefaultHandler\(\)](#)
- Generate [murasaki\\_include\\_stub.h](#) to let the Murasaki library to include HAL headers.

Last one is little tricky to do it manually. Refer [murasaki\\_include\\_stub.h](#) for details.

## 4.2 CubeMX setting

There is several required CubeMX setting.

- [Heap Size](#)
- [Stack Size](#)
- [Task stack size of the default task](#)
- [UART peripheral](#)
- [SPI Master peripheral](#)
- [SPI Slave peripheral](#)
- [I2C peripheral](#)
- [EXTI](#)

### 4.2.1 Heap Size

Heap is very important in the application with murasaki.

First, class instances are created inside heap region by new operator often. And second, [murasaki::Debugger](#) allocates a huge size of FIFO buffer. This buffer stays in between the [murasaki::Debugger::Printf\(\)](#) function and the logger task. The size of this FIFO buffer is defined by [PLATFORM\\_CONFIG\\_DEBUG\\_BUFFER\\_SIZE](#). The default is 4KB.

Usually, the heap is simply called "heap", without precise definition of terminology. But let's call it "system heap" here. The system heap is the one which is managed by new and delete operators by default.

In addition to the system heap, FreeRTOS has its own heap. This heap is managed separately from the system heap. This management includes the heap size watching and returning error. And this heap is thread safe while the system heap is not.

Using two heap is not easy. And definitely, the FreeRTOS heap is better than the system heap in the embedded application. So, in murasaki, the new and the delete operators are overloaded and redirected to the FreeRTOS heap. See [Heap memory consideration](#) for detail.

To avoid the heap allocation problem, it is better to have more than 8kB FreeRTOS heap. The FreeRTOS heap size can be changed by CubeMX :

```
Tab => Pinout & Configuration => Middleware => FreeRTOS => Config Parameters Tab => TOTAL_HEAP_SIZE
```

On the other hand, the system heap size can be smaller like 128 Byte because we don't use it..

Note that to know the minimum requirement of the system heap size, you must investigate how much allocations are done before entering FreeRTOS. Because murasaki application doesn't use any system heap, only very small management memory should be required in system heap.

The system Heap size can be set by following place.

```
Tab => Project Manager => Code Generator => Linker Settings
```

### 4.2.2 Stack Size

In this section, the stack means the interrupt stack.

The interrupt stack is used only when the interrupt is accepted. Then, it is basically small.

By the way, murasaki uses its assertion often. Once assertion fails, a message is created by `snprintf()` function and transmitted through FIFO. These operations consume stack. And assertion can be happen also in the ISR context.

The debugging in the ISR is not easy without assertion and `printf()`. To make them always possible, it is better to set the interrupt stack size bigger than 256 Bytes. The interrupt stack size can be changed by CubeMX :

Tab => Project Manager => Code Generator => Linker Settings

### 4.2.3 Task stack size of the default task

The default task has very small stack ( 128 Bytes )

This is not enough to use murasaki and its debugger output functionality. It should be increased at smallest 256 Bytes.

It can be changed by CubeMX:

Tab => Pinout & Configuration => Middleware => FreeRTOS => Config Parameters Tab => MINIMAL\_STACK\_SIZE

### 4.2.4 UART peripheral

UART/USART peripheral have to be configured as Asynchronous mode.

The DMA have to be enabled for both TX and RX. Both DMA must be normal mode.

All 3 of the NVIC interrupt have to be enabled.

### 4.2.5 SPI Master peripheral

SPI Master peripheral have to be configured as Full-Duplex Master mode. The NSS must be disabled.

The DMA have to be enabled for both TX and RX. Both DMA must be normal mode.

All 3 of the NVIC interrupt have to be enabled.

### 4.2.6 SPI Slave peripheral

SPI Slave peripheral have to be configured as Full-Duplex Slave mode. The NSS must be input signal.

The DMA have to be enabled for both TX and RX. Both DMA must be normal mode.

All 3 of the NVIC interrupt have to be enabled.

### 4.2.7 I2C peripheral

I2C have to be configured as "I2" mode.

The NVIC interrupt have to be enabled.

To configure as I2C device, the primary slave address have to be configured.

### 4.2.8 EXTI

The corresponding interrupt have to be enabled by NVIC.

## 4.3 Configuration

Murasaki has configurable parameters.

These parameters control mainly the task size and task priority.

One of the special configurations is [MURASAKI\\_CONFIG\\_NODEBUG](#) macro. This macro controls whether assertion inside Murasaki source code works or ignored.

To customize the configuration, define the configuration macro with the desired value in the [platform\\_config.hpp](#) file. This definition will override the Murasaki default configuration.

For the detail of each macro, see [Definitions and Configuration](#).

## 4.4 Task Priority and Stack Size

The FreeRTOS task priority is allowed from 1 to configMAX\_PRIORITIES.

Where configMAX\_PRIORITIES is porting dependent. The task with priority == configMAX\_PRIORITIES will run with the highest priority among all tasks.

At the initial state, the Murasaki has two hidden tasks inside. Both are running for the [murasaki::Debugger](#) class, and both task's priority are defined as [PLATFORM\\_CONFIG\\_DEBUG\\_TASK\\_PRIORITY](#). By default, the value of [PLATFORM\\_CONFIG\\_DEBUG\\_TASK\\_PRIORITY](#) is configMAX\_PRIORITIES - 1. That means, debug tasks priority is very high.

The debug tasks should have priority as high as possible. Otherwise, another task may block the debugging message.

Unlike the task priority, the interrupt priority is easy. Usually, it is not so sensitive because the ISR is very short in the good designed RTOS application design. In this case, all ISR can be a same priority.

In the bad designed RTOS application, there are very few things we can do.

## 4.5 Heap memory consideration

In Murasaki, there is a re-definition of `operator new` and `operator delete` inside `allocators.cpp`.

This re-definition let the `pvPortMalloc()` allocate a fragment of memory for the `operator new`.

This changes converges all allocation to the FreeRTOS's heap. There is some merit of the convergence:

- The FreeRTOS heap is thread safe while the system heap in SW4STM32 is not thread-safe
- The FreeRTOS heap is checking the heap size limitation and return an error, while the system heap behavior in SW4STM32 is not clear.
- The heap size calculation is easier if we integrate the memory allocation activity into one heap.

On the other hand, FreeRTOS heap is not able to allocate/deallocate in the ISR context. And it is impossible to use the FreeRTOS heap before starting up the FreeRTOS. Then, we have to follow the rules here :

- C++ new / delete operators have to be called after FreeRTOS started.
- C++ new / delete operators have to be called in the task context.

## 4.6 Platform variable

The `murasaki::platform` and the `murasaki::debugger` have to be initialized by the `InitPlatform()` function.

The programming of this function is a responsibility of the porting programmer.

First of all, the porting programmer has to make the peripheral handles as visible from the `murasaki_platform.cpp`.

For example, CubeMx generate the `huart2` for Nucleo L152RE for the serial communication over the ST-LINK USB connection. `huart2` is defined in `main.c` as like below:

```
UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_rx;
DMA_HandleTypeDef hdma_usart2_tx;
```

To use this handle, the porting programmer has to declare the same name as an external variable, in the `murasaki_↵_platform.cpp` :

```
extern UART_HandleTypeDef huart2;
```

After these preparations, the porting programmer can program the `InitPlatform()` :

```

void InitPlatform()
{
    // UART device setting for console interface.
    // On Nucleo, the port connected to the USB port of ST-Link is
    // referred here.
    murasaki::platform.uart_console = new
    murasaki::Uart(&huart2);
    // UART is used for logging port.
    // At least one logger is needed to run the debugger class.
    murasaki::platform.logger = new murasaki::UartLogger(
    murasaki::platform.uart_console);
    // Setting the debugger
    murasaki::debugger = new murasaki::Debugger(
    murasaki::platform.logger);
    // Set the debugger as AutoRePrint mode, for the easy operation.
    murasaki::debugger->AutoRePrint(); // type any key to show history.

    // For demonstration, one GPIO LED port is reserved.
    // The port and pin names are fined by CubeMX.
    murasaki::platform.led = new murasaki::BitOut(LD2_GPIO_Port,
    LD2_Pin);
}

```

In this sample, we initialize the `uart_console` member variable which is `AbstractUart` class. The applicaiton programmer control the UART2 over this `uart_console` member variable.

In the second step, we pass this `uart_cosome` to the logger member variable. This member variable is an essential stub for the `murasaki::debugger`. In this example, we assign the UART2 port as interface for the debugging output.

After the logger becomes ready, we initialize the `murasaki::debugger`. As we already discussed, this debugger receives a logger object as a parameter. The debugger output all messages through this logger.

The last step is optional. We invoke the `murasaki::Debugger::AutoRePrint()` member function. By calling this function, logger re-print the old data in the FIFO again whenever the end-user type any key of the keyboard.

This "auto re-print by any key" is convenient in the small system. But for the large system which has its own command line shell, this input-interruption is harmful. For such the system, programmer want to call `murasaki::Debugger::RePrint()` member function, by certain customer command.

Once the debugger is ready to use, we create the led member variable as a general purpose output port of the application .

The `ExecPlatform()` function implements the actual algorithm of application. In the example below, the application is blinking a LED and printing a messages on the console output.

```

void ExecPlatform()
{
    // counter for the demonstration.
    static int count = 0;

    // Loop forever
    while (true) {
        // Toggle LED.
        murasaki::platform.led->Toggle();

        // print a message with counter value to the console.
        murasaki::debugger->Printf("Hello %d \n", count);

        // update the counter value.
        count++;

        // wait for a while
        murasaki::Sleep(static_cast<murasaki::WaitMilliseconds>(500));
    }
}

```

Finally, above two functions have to be called from `StartDefaultTask` of the `main.c`. Also, `main.c` must include the `murasaki_platform.hpp` to read the prototype of these functions.

Following is the sample of the `StartDefaultTask()`. The actual code have a comment to work together the code generator of the CubeMX. But this sample remove them because of the documenattion tool ( doxygen ) limitation.

```
void StartDefaultTask(void const * argument)
{
    InitPlatform();
    ExecPlatform();

    for(;;)
    {
        osDelay(1);
    }
}
```

## 4.7 Routing interrupts

The `murasaki_platform.cpp` has skeletons of HAL callback.

These callbacks are pre-defined inside HAL as receptors of interrupt. These definitions inside HAL are "weak" binding. Thus, these skeletons in `murasaki_platform.cpp` overrides the definition. The porting programmer have to program these skeletons correctly.

In the Murasaki manner, the skeletons have to call the relevant callback member function of platform variables. For example, this is the typical programming of the call back :

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart)
{
    if (murasaki::platform.uart_console->TransmitCompleteCallback(huart))
        return;
}
```

In this sample, the `TxCpltCallback()` calls `murasaki::platform.uart_console->TransmitCompleteCallback()` member function. And then return if that member function returns true. Note that all the callbacks in the Murasaki class returns true if the given peripheral handle matches with its internal handle. Thus, this is good way to poll all the UART peripheral inside this callback function.

Following is the list of the interrupts which application have to route to the peripheral class variables.

- void `HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart);`
- void `HAL_UART_RxCpltCallback(UART_HandleTypeDef * huart);`
- void `HAL_UART_ErrorCallback(UART_HandleTypeDef *huart);`
- void `HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi);`
- void `HAL_SPI_ErrorCallback(SPI_HandleTypeDef * hspi);`
- void `HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef * hi2c);`
- void `HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef * hi2c);`
- void `HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef * hi2c);`
- void `HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef * hi2c);`
- void `HAL_I2C_ErrorCallback(I2C_HandleTypeDef * hi2c);`
- void `HAL_GPIO_EXTI_Callback(uint16_t GPIO_P);`

## 4.8 Error handling

The `murasaki_platform.cpp` has two error handling functions.

These functions are pre-programmed from the first. And usually its enough to use the pre-programmed version. In the other hand the porting programmer have to modify the application program to call these error handling functions at appropriate situation. Otherwise, these error handling functions will be never called.

The `CustomAssertFailed()` function should be called from the `assert_failed()` function. The `assert_failed()` function is located in the `main.c`. Modifying the `assert_failed()` is the responsibility of the porting programmer.

```
void assert_failed(uint8_t* file, uint32_t line)
{
    CustomAssertFailed(file, line);
}
```

To enable the `assert_failed()`, the porting programmer have to uncomment the `USE_FULL_ASSERT` macro inside `stm32xxx_hal_conf.h`. The file name is depend on the target microprocessor. Thus, the porting programmer have to search the all files inside project.

At the time of 2019/May, this definition is in the one for the following files :

- `stm32f0xx_hal_conf.h`
- `stm32f3xx_hal_conf.h`
- `stm32f7xx_hal_conf.h`
- `stm32l1xx_hal_conf.h`

The `CustomDefaultHandler()` function should be called from the default exception routine. But the system default exception handler ( `Default_Handler` ) doesn't do anything by default. To maximize the information to the JTAG debugger, this is programmed as very simple eternal loop.

The default exception handler can be programmed or left untouched as porting programmer want. It is up to the system policy. If it is re-programmed to call the `CustomDefaultHandler()`, `murasaki::debugger` object take the control of the debug message FIFO at the exception handler context.

If the exception happened and the `CustomDefaultHandler` is called, the end user can see the entire messages in the debug FIFO by typing any key of the keyboard. This is useful to see the last message from the assertion. The last message usually represent the cause of the exception. The end user can debug the application program based on this last assertion message.

The HAL default exception routine is programmed at startup/startup\_stm32xxxx.s by assembly language.

The porting programmer can modify it as below, to call the `CustomDefaultHandler()`;

```
Default_Handler:
Infinite_Loop:
    bl CustomDefaultHandler

    b Infinite_Loop
.size Default_Handler, .-Default_Handler
```



## 4.9 Summary of the porting

Following is the porting steps :

- Adjust heap size and stack size as described in the [CubeMX setting](#)
- Generate an application skeleton from CubeMX.
- Checkout Murasaki repository into your project.
- Copy the template files as described in the [Directory Structure](#) .
- Configure Muraaski as described in the [Configuration](#) and the [Task Priority and Stack Size](#)
- Call [InitPlatform\(\)](#) and [ExecPlatform\(\)](#) as described [Platform variable](#).
- Route the interrupts as described [Routing interrupts](#).
- Route the error handling as described [Error handling](#)



## Chapter 5

# Step-by-Step Porting Guide

This chapter goes through the actual operation of the CubeMX and SW4STM32 to create an empty application with Murasaki.

To develop your own application, you should create the platform with Murasaki by yourself. In this chapter, we will see the procedure to create a sample application for the Nucleo F722ZE board.

This chapter is written based on the following software and hardware.

- CubeMX ver 5.0.1
- System Workbench for STM32 ver 1.17.0.201812190825
- Ubuntu 16.04 LTS
- Nucleo F722ZE

Followings are the contents of this chapter.

- [UART configuration](#)
- [CPU, EXTI, and System tick configuration](#)
- [FreeRTOS configuration](#)
- [Clock configuration](#)
- [Project configuration and code generation](#)
- [Clone the Murasaki repository and install](#)

## 5.1 UART configuration

In this section, we configure the UART communication parameter, DMA and interrupts.

Once you select the Nucleo F722ZE on the CubeMX, let's start to modify it from the UART configuration. Nucleo F722ZE board utilizes the USART3 peripheral as UART3. And this UART3 port is connected with ST-Link board. Thus we can communicate with the application through the USB by terminal software.

The Murasaki library support this communication by [murasaki::Debugger](#) class. To use the Debugger class, we have to configure the UART port correctly.

To configure the UART, select the UART3 peripheral inside the Connectivity category of the Pinout & Configuration tab. The default tab is the Parameter and Setting tab. In this tab, we will configure the Basic Parameters like Baud rate, word length, etc...

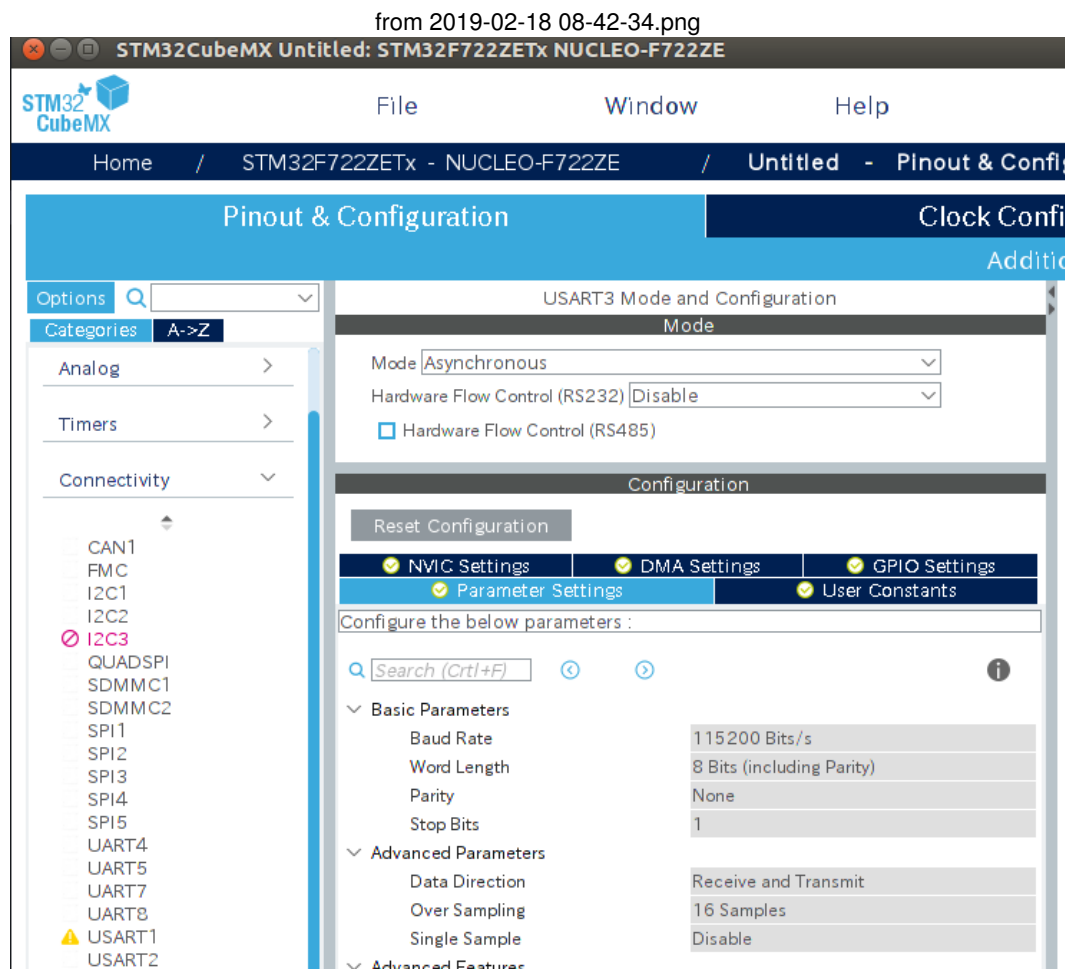
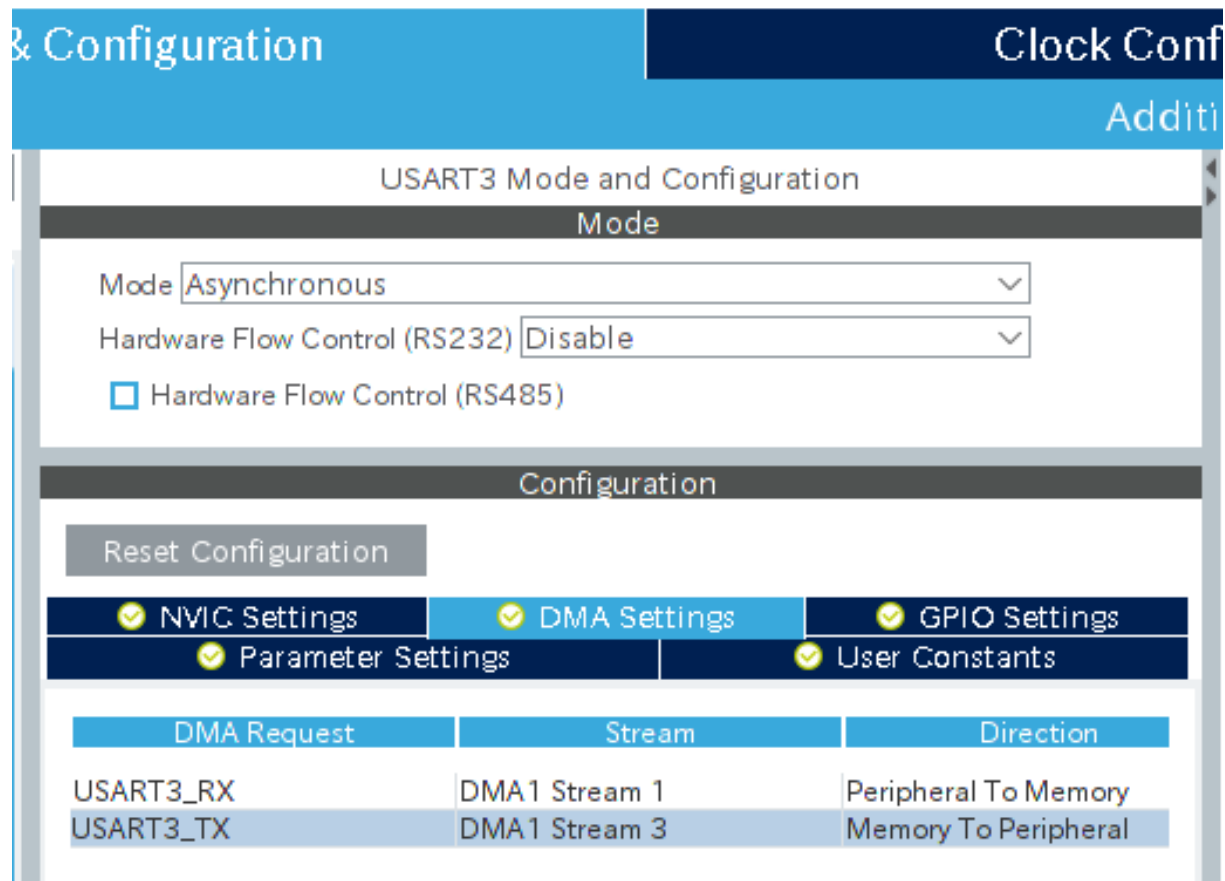


Figure 5.1 CubeMX UART panel

And then, we configure the DMA. The [murasaki::Uart](#) class uses the DMA transfer for both TX and RX. To enable DMA, click the DMA Settings tab and add DMAs. The default state of the DMA configuration after clicking Add button is undetermined. Then, select the TX and RX DMA channel.

from 2019-02-18 08-43-41.png



**Figure 5.2 UART DMA Settings tab**

Finally, we configure the interrupt by NVIC Settings tab. Check all checkboxes.

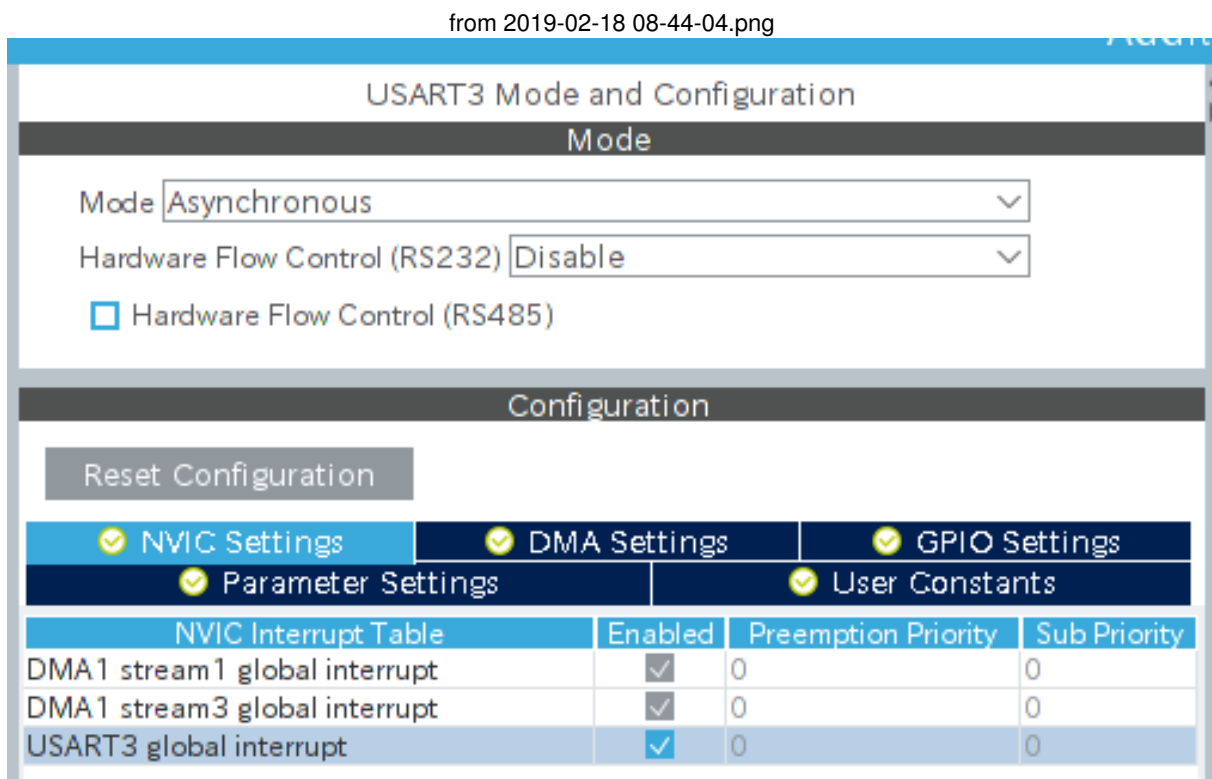


Figure 5.3 UART NIC Settings tab

By the way, we don't use the USB OTG of the Nucleo F722ZE in this demo. So let's disable it. This is optional. There is no side effect to enable USB except memory usage.

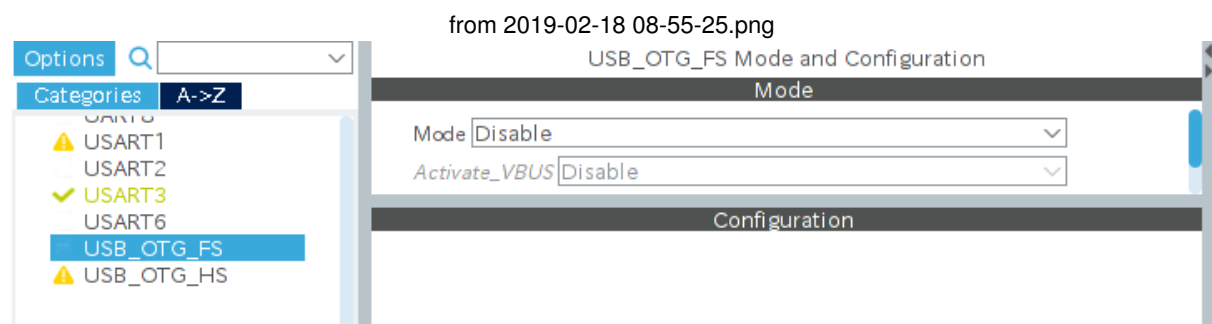


Figure 5.4 USB\_OTG\_FS Mode and Configuration

## 5.2 CPU, EXTI, and System tick configuration

In this section, we configure the CPU, EXTI, and System tick timer.

By default, CubeMX doesn't configure the CPU core. As a result, all caches and flash accelerator are disabled. Enabling these features accelerates the code execution speed.

Select CORETEX\_M7 tab of the System Core category. Then, enable these items.

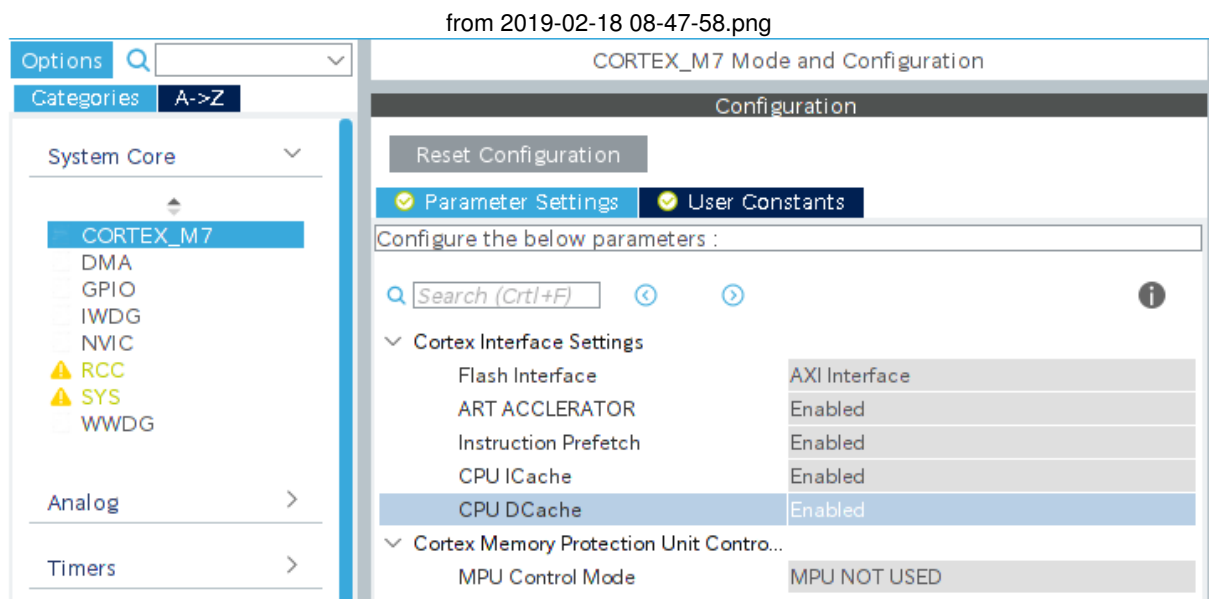


Figure 5.5 CORTEX\_M7 Mode and Configuration

Nucleo F722ZE board uses PC13 pin as user button (Blue button on the board). In this demo we will use this button as an interrupt source. Then, we have to configure this PC13 pin as interruptible.

Select GPIO in the System Core category and set the PC13 GPIO mode to External Interrupt.

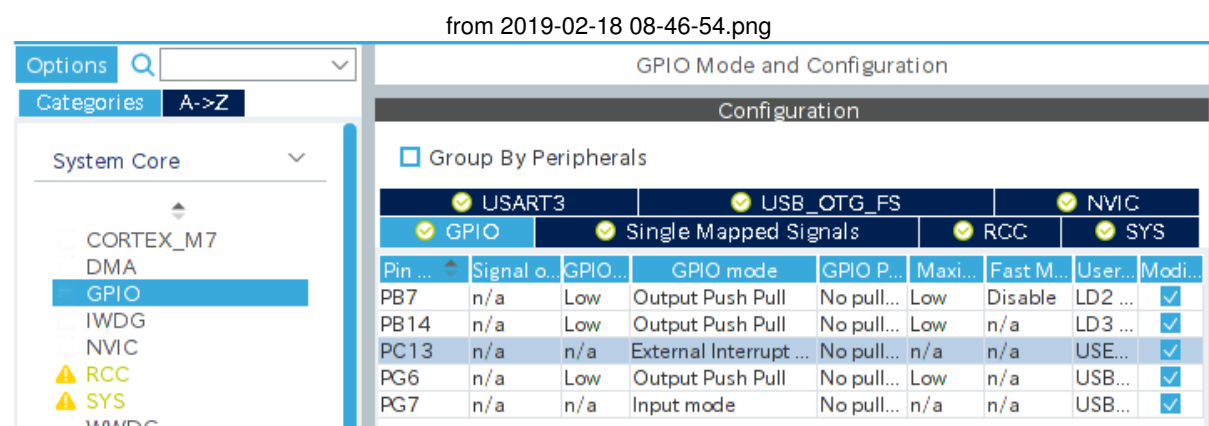


Figure 5.6 GPIO Mode and Configuration

Select NVIC tab to enable the EXTI line input.

from 2019-02-18 08-47-07.png

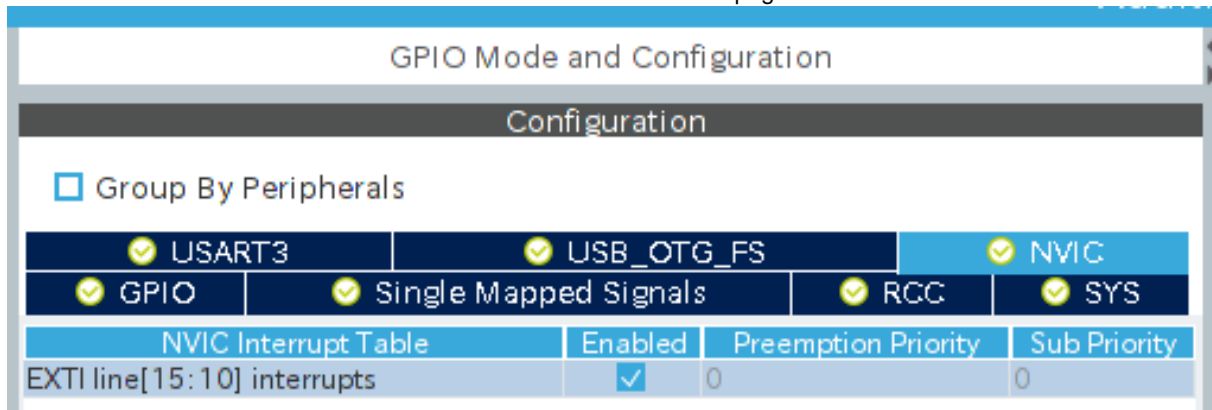


Figure 5.7 GPIO NVIC(EXTI)

Then set the Timebase source. This is timer selection for the system tick. FreeRTOS recommend using the GP timer as system tick source. So, the select one of the unused timer. In the figure below we are choosing TIM14.

from 2019-02-18 08-48-58.png

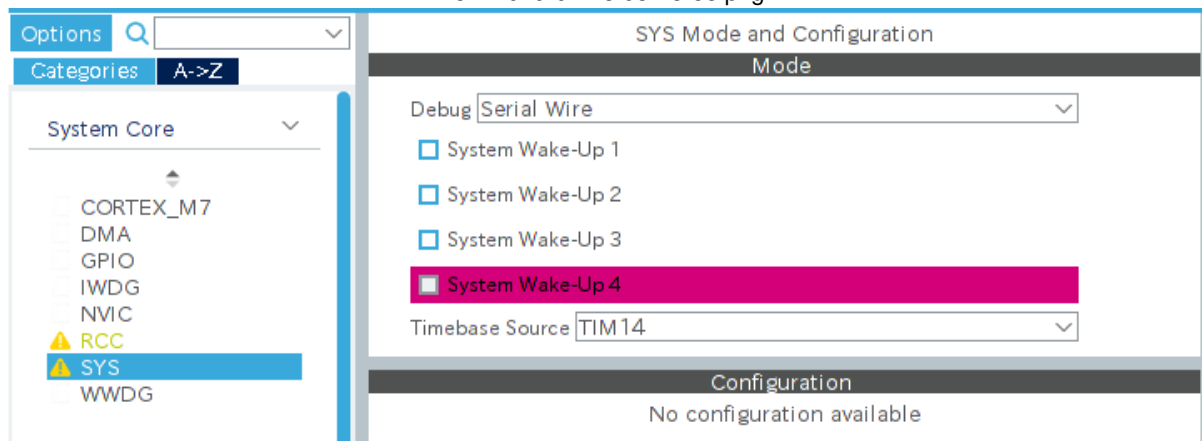


Figure 5.8 Sys Mode and Configuration

### 5.3 FreeRTOS configuration

To run a FreeRTOS application, the heap memory size, and the default stack size have to be configured.

FreeRTOS is the important part of the Murasaki platform. To run the FreeRTOS, we have to configure at least two parameters.

At first, we have to increase MINIMAL\_STACK\_SIZE. This is the stack size of the first task created by CubeMX. See the [Task stack size of the default task](#) for detail. The default value is 128 Byte. It should be at least 256 Byte.



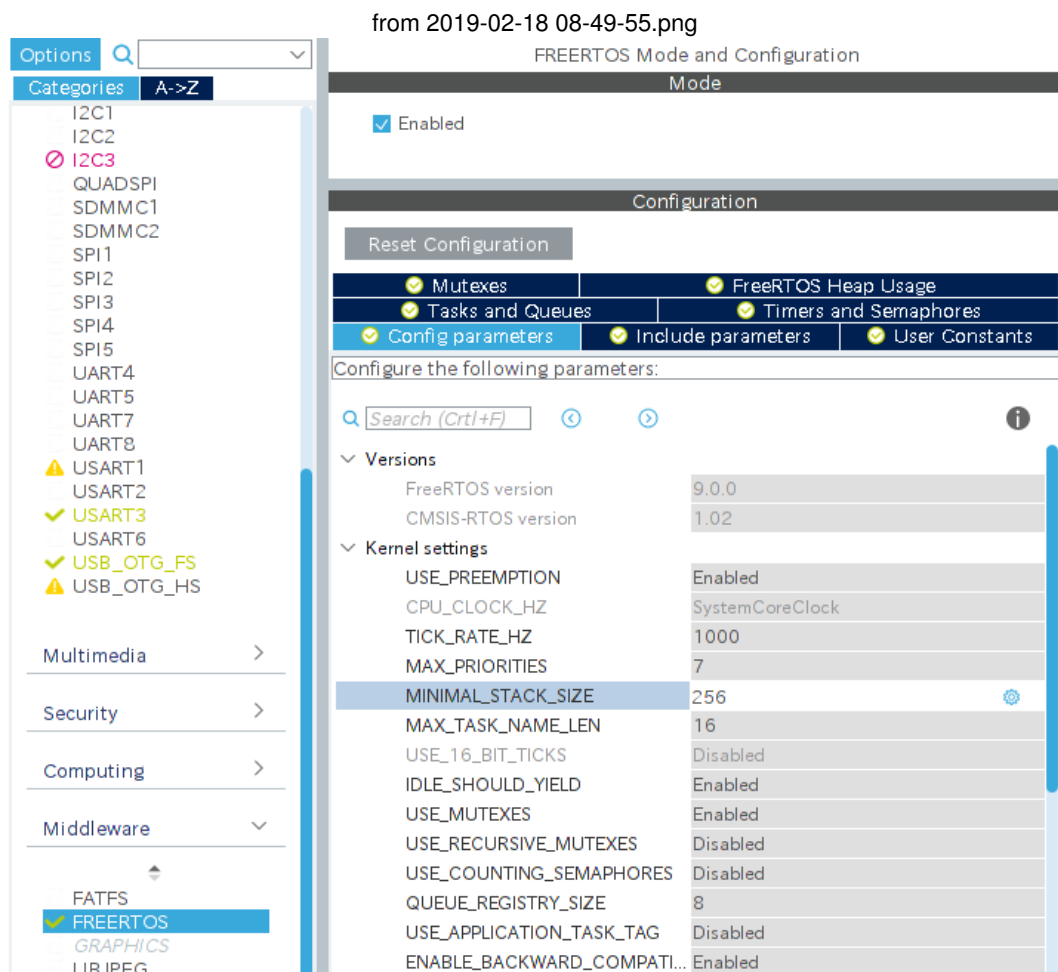


Figure 5.9 FREERTOS Mode and Configuration: MINIMAL\_STACK\_SIZE

Another important parameter is TOTAL\_HEAP\_SIZE. This is the size of the heap under the FreeRTOS management. See the [Heap Size](#) for detail.

16kB is a little bit smaller. 32kB and greater is preferable.

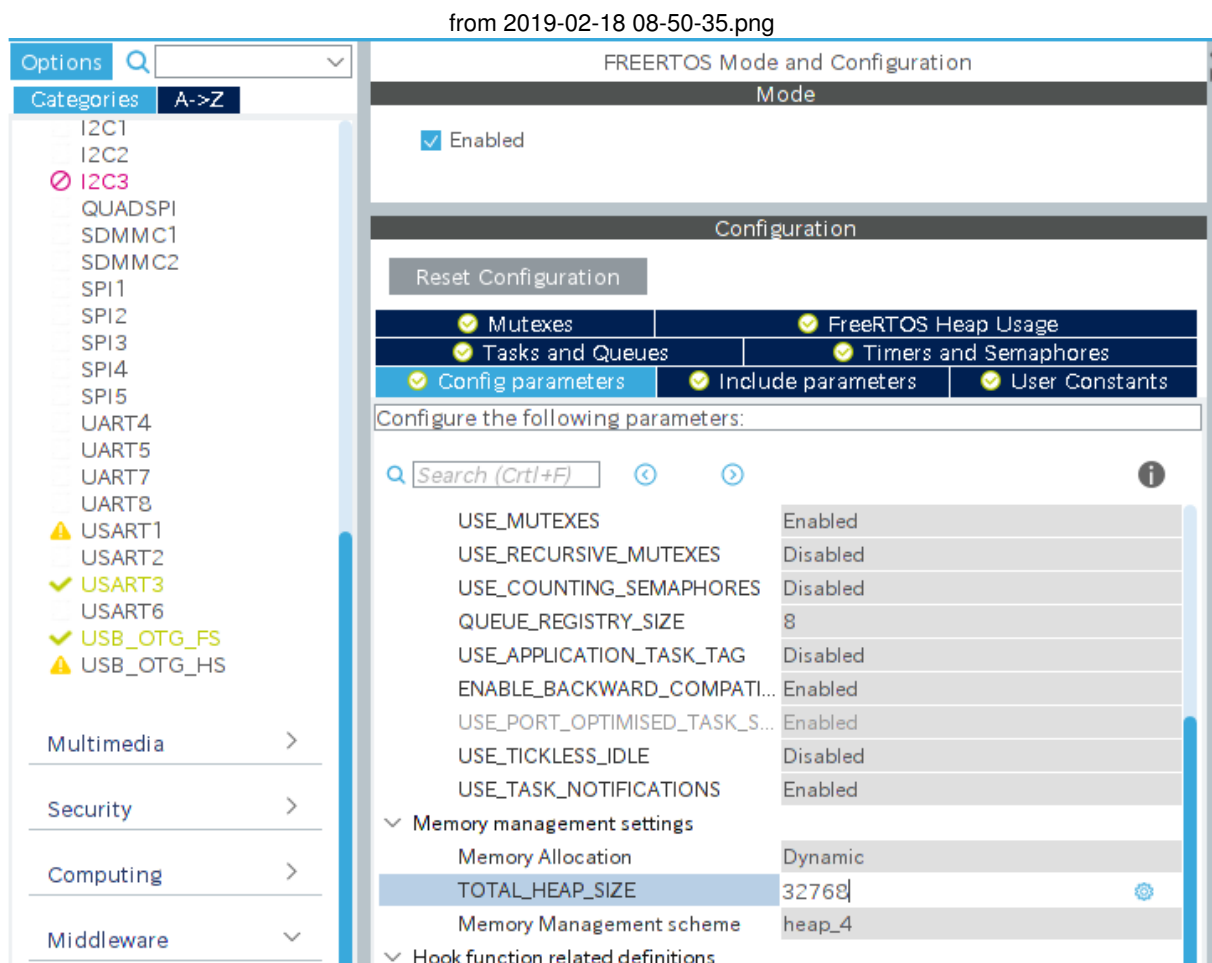


Figure 5.10 FREERTOS Mode and Configuration: TOTAL\_HEAP\_SIZE

## 5.4 Clock configuration

At the Feb/2019, CubeMX has a bug on the Nucleo F722ZE clock setting.

The Nucleo F722ZE board has 8MHz as input clock (HSE). But CubeMX default setting is 25MHz. So, we have to fix this bug by hand.

Select the Clock Configuration tab. The HSE input frequency is at the left end of the clock chain . Change this frequency from 25 to 8.

Once you change, CubeMX adjusts the entire clock but that is still not enough. CPU clock (HCLK) is too low. Then, we should modify by hand again.

The HCLK is located at the right end of the figure below. Change it to 216 MHz which is the Maximum operation frequency.

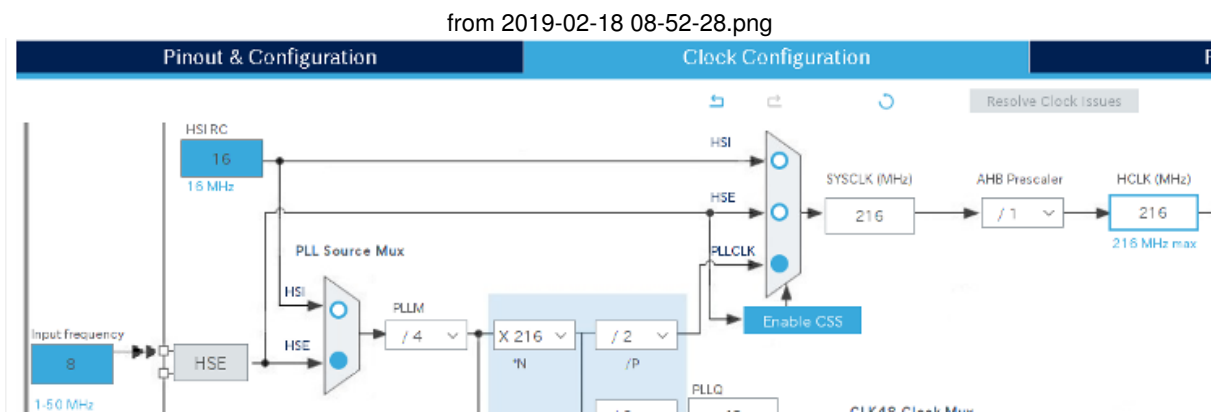


Figure 5.11 Clock Configuration

## 5.5 Project configuration and code generation

At last, we configure the SW4STM32 project and generate a skeleton code.

CubeMX's board setting is done. Now, we have to configure the project itself. In this chapter, we will define the project type and location. Note that to start the following procedure, we must create a workspace of the SW4STM32 and keep it open by SW4STM32 during the following configuration.

At first, select the Project Manager tab.

Because Murasaki Class library target is the SW4STM32, we must set the tool chaine to SW4STM32. And then, browse the SW4STM32 workspace which is open by SW4STM32. Then type a preferred project name. In this guide, we choose ;

- murasaki\_demo as project name.
- workspace\_murasaki\_sample as workspace directory name.

from 2019-02-18 08-57-07.png

The Project Manager settings form includes the following fields and options:

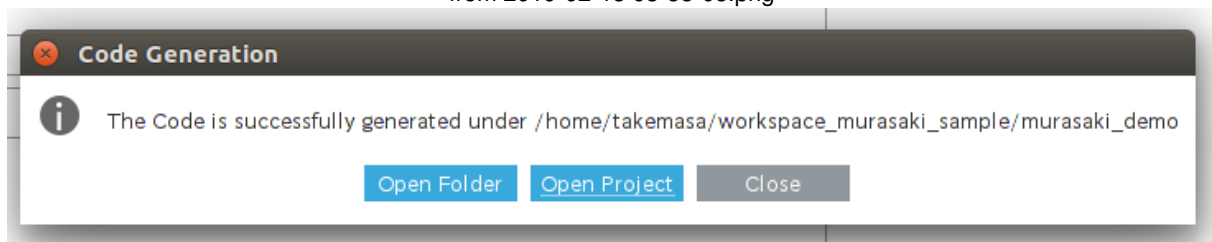
- Project Name:** murasaki\_demo
- Project Location:** /home/takemasa/workspace\_murasaki\_sample/ (with a Browse button)
- Application Structure:** Basic (with a dropdown arrow) and ☐ Do not generate the main()
- Toolchain Folder Location:** /home/takemasa/workspace\_murasaki\_sample/murasaki\_demo/
- Toolchain / IDE:** SW4STM32 (with a dropdown arrow) and ☒ Generate Under Root

Figure 5.12 Project Manager

Now, we are ready to generate the code. Click the "GENERATE CODE" button near the right upper corner.

Once code regeneration is finished, CubeMX shows the Code Generation dialog. Click "Open Project".

from 2019-02-18 08-58-05.png

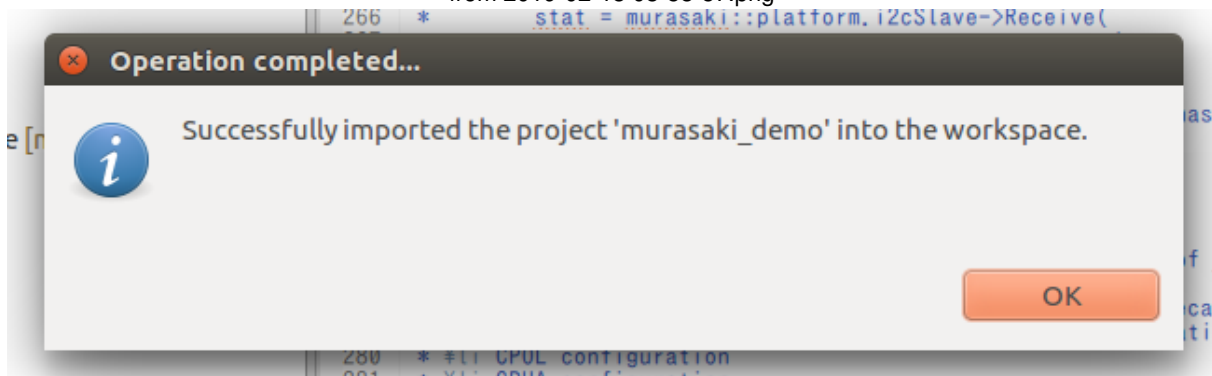


**Figure 5.13 Code Generation**

Then, CubeMX let the SW4STM32 import the generated project into the workspace (This is tricky part. Generating the code into workspace is not enough. We have to import that project to the workspace. "Open Project" button let the SW4STM32 import it).

SW4STM32 import the project and show a dialog.

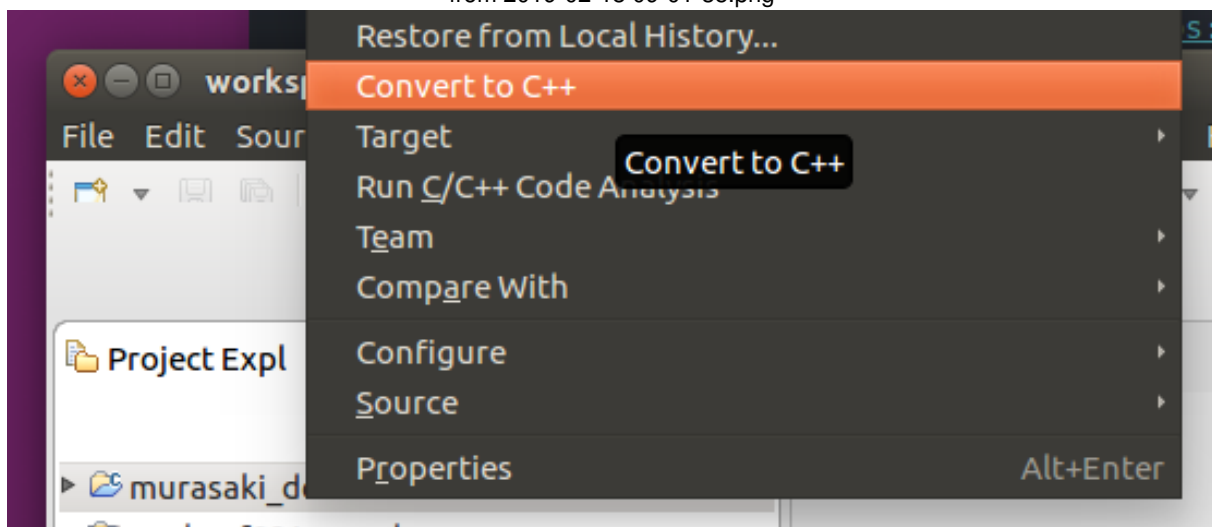
from 2019-02-18 08-58-57.png



**Figure 5.14 Successfully imported**

Now, the project is ready to build. But to go to next step, we must convert the Project to the C++ project. The generated code is C project. But we use the class library inside application. Thus, this conversion is essential.

from 2019-02-18 09-01-35.png



**Figure 5.15 Convert to C++ Project**

## 5.6 Clone the Murasaki repository and install

At last, we configure the SW4STM32 project and generate a code

The project is still as is after CubeMX code generation. Remember, our workspace and project was like this :

- murasaki\_demo as project name.
- workspace\_murasaki\_sample as workspace directory name.

So, the directory is workspace\_murasaki\_sample/murasaki\_demo. Let's open a shell window, and execute following command :

```
cd workspace_murasaki_sample/murasaki_demo
git clone https://github.com/suikan4github/murasaki.git
cd murasaki
./install
```

That's it. The Murasaki source tree is integrated into your project, and the installer script embed the essential codes into several files generated by CubeMX.

Let's go back to SW4STM and refresh the project ( Type F5 ). Without refreshing, you cannot see the Murasaki directory inside your project.

Now, we are at the final stage. Open the project property, expand the C/C++ General, choose the Paths and Symbols, select the include tab, and click the GNU C++. This is the include path lists. Click Add button and type "murasaki/Inc". Then check the Add to all configurations..

Click ok if the directory is correctly typed.

from 2019-02-20 18-20-48.png

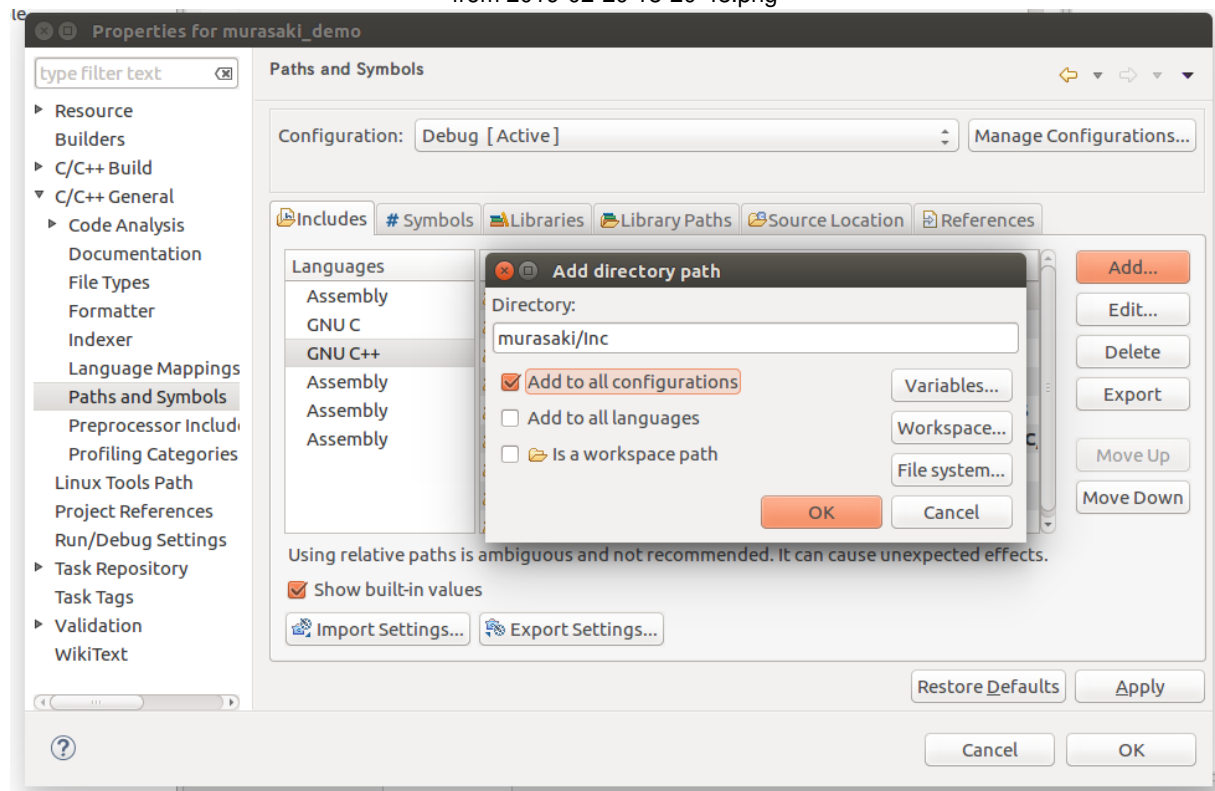


Figure 5.16 Add Murasaki include path

Next, click the Source Location tab, and add "murasaki/Src" .

from 2019-02-18 09-12-04.png

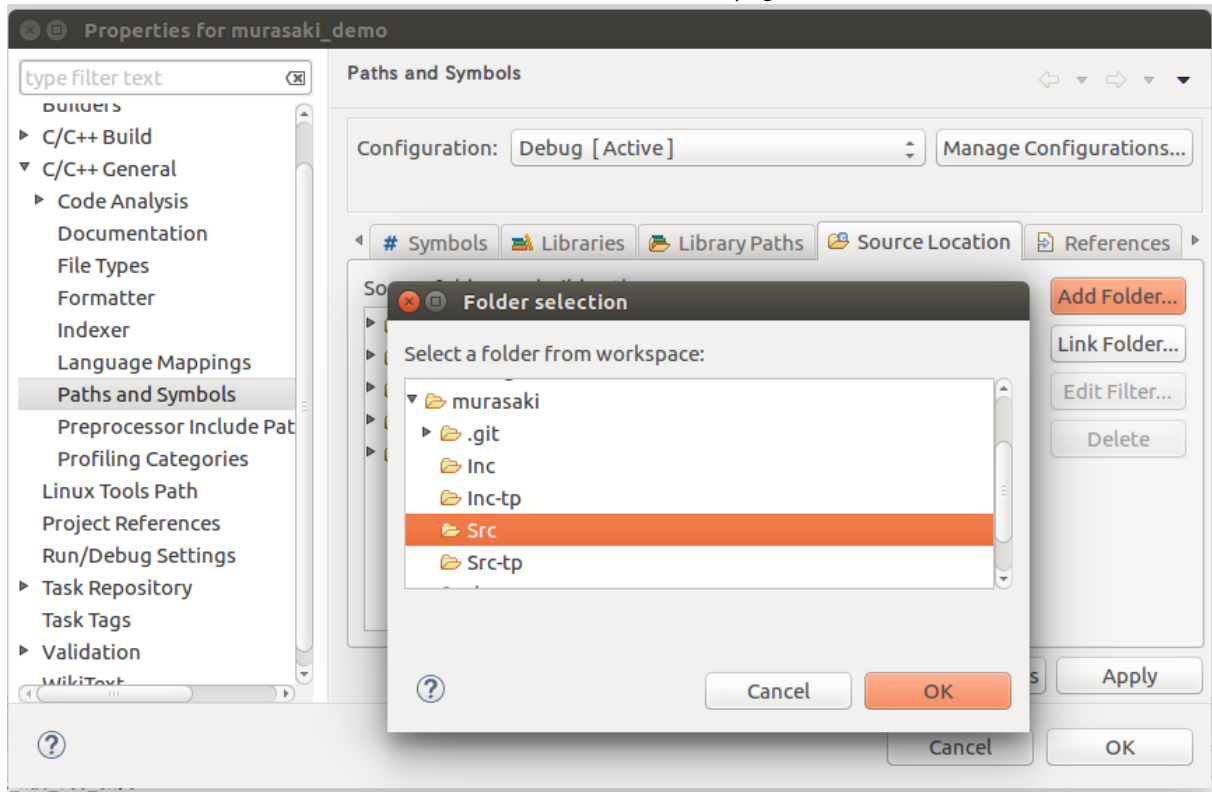


Figure 5.17 Add Murasaki source path

## Chapter 6

# Module Index

### 6.1 Modules

Here is a list of all modules:

Murasaki Class Collection . . . . .	53
Synchronization and Exclusive access . . . . .	57
Third party classes . . . . .	58
Definitions and Configuration . . . . .	59
Application Specific Platform . . . . .	65
Abstract Classes . . . . .	72
Helper classes . . . . .	73
CMSIS . . . . .	75
Stm32f4xx_system . . . . .	76
STM32F4xx_System_Private_Includes . . . . .	77
STM32F4xx_System_Private_TypesDefinitions . . . . .	78
STM32F4xx_System_Private_Defines . . . . .	79
STM32F4xx_System_Private_Macros . . . . .	80
STM32F4xx_System_Private_Variables . . . . .	81
STM32F4xx_System_Private_FunctionPrototypes . . . . .	82
STM32F4xx_System_Private_Functions . . . . .	83





## Chapter 7

# Namespace Index

### 7.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">murasaki</a>	Personal <a href="#">Platform</a> parts collection . . . . .	<a href="#">85</a>
--------------------------	--	--------------------



## Chapter 8

# Hierarchical Index

### 8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

murasaki::AudioCodecStrategy . . . . .	93
murasaki::Adau1361 . . . . .	89
murasaki::CriticalSection . . . . .	106
murasaki::Debugger . . . . .	107
murasaki::FifoStrategy . . . . .	117
murasaki::DebuggerFifo . . . . .	110
murasaki::GPIO_type . . . . .	119
murasaki::LoggerStrategy . . . . .	137
murasaki::UartLogger . . . . .	168
murasaki::LoggingHelpers . . . . .	139
murasaki::PeripheralStrategy . . . . .	139
murasaki::AudioStrategy . . . . .	96
murasaki::BitInStrategy . . . . .	101
murasaki::BitIn . . . . .	99
murasaki::BitOutStrategy . . . . .	105
murasaki::BitOut . . . . .	103
murasaki::I2CMasterStrategy . . . . .	125
murasaki::I2cMaster . . . . .	120
murasaki::I2cSlaveStrategy . . . . .	134
murasaki::I2cSlave . . . . .	129
murasaki::SpiMasterStrategy . . . . .	144
murasaki::SpiMaster . . . . .	141
murasaki::SpiSlaveStrategy . . . . .	155
murasaki::SpiSlave . . . . .	147
murasaki::UartStrategy . . . . .	170
murasaki::DebuggerUart . . . . .	112
murasaki::Uart . . . . .	162
murasaki::Platform . . . . .	140
murasaki::SpiSlaveAdapterStrategy . . . . .	153
murasaki::SpiSlaveAdapter . . . . .	150
murasaki::Synchronizer . . . . .	157
murasaki::TaskStrategy . . . . .	160
murasaki::Task . . . . .	158



## Chapter 9

# Class Index

### 9.1 Class List

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

<a href="#">murasaki::Adau1361</a>	
Audio Codec LSI class . . . . .	89
<a href="#">murasaki::AudioCodecStrategy</a>	
Abstract audio codec controller . . . . .	93
<a href="#">murasaki::AudioStrategy</a>	
Stereo Audio is served by the descendants of this class . . . . .	96
<a href="#">murasaki::BitIn</a>	
General purpose bit input . . . . .	99
<a href="#">murasaki::BitInStrategy</a>	
Definition of the root class of bit input . . . . .	101
<a href="#">murasaki::BitOut</a>	
General purpose bit output . . . . .	103
<a href="#">murasaki::BitOutStrategy</a>	
Definition of the root class of bit output . . . . .	105
<a href="#">murasaki::CriticalSection</a>	
A critical section for task context . . . . .	106
<a href="#">murasaki::Debugger</a>	
Debug class. Provides printf() style output for both task and ISR context . . . . .	107
<a href="#">murasaki::DebuggerFifo</a>	
FIFO with thread safe . . . . .	110
<a href="#">murasaki::DebuggerUart</a>	
Logging dedicated UART class . . . . .	112
<a href="#">murasaki::FifoStrategy</a>	
Basic FIFO without thread safe . . . . .	117
<a href="#">murasaki::GPIO_type</a>	
A structure to en-group the GPIO port and GPIO pin . . . . .	119
<a href="#">murasaki::I2cMaster</a>	
Thread safe, blocking IO. Encapsulating I2C master. Based on STM32Cube HAL driver and FreeRTOS . . . . .	120
<a href="#">murasaki::I2CMasterStrategy</a>	
Definition of the root class of I2C master . . . . .	125
<a href="#">murasaki::I2cSlave</a>	
Thread safe, blocking IO. Encapsulating I2C slave. Based on STM32Cube HAL driver and FreeRTOS . . . . .	129
<a href="#">murasaki::I2cSlaveStrategy</a>	
Definition of the root class of I2C Slave . . . . .	134

<a href="#">murasaki::LoggerStrategy</a>	
Abstract class for logging . . . . .	137
<a href="#">murasaki::LoggingHelpers</a>	
A stracture to engroup the logging tools . . . . .	139
<a href="#">murasaki::PeripheralStrategy</a>	
Mother of all peripheral class . . . . .	139
<a href="#">murasaki::Platform</a>	
Custom aggregation struct for user platform . . . . .	140
<a href="#">murasaki::SpiMaster</a>	
Thread safe, blocking IO. Encapsulating SPI master. Based on STM32Cube HAL driver and FreeRTOS . . . . .	141
<a href="#">murasaki::SpiMasterStrategy</a>	
Root class of the SPI master . . . . .	144
<a href="#">murasaki::SpiSlave</a>	
Thread safe, blocking IO. Encapsulating SPI slave. Based on STM32Cube HAL driver and FreeRTOS . . . . .	147
<a href="#">murasaki::SpiSlaveAdapter</a>	
A speficier of SPI slave . . . . .	150
<a href="#">murasaki::SpiSlaveAdapterStrategy</a>	
Definition of the root class of SPI slave adapter . . . . .	153
<a href="#">murasaki::SpiSlaveStrategy</a>	
Root class of the SPI slave . . . . .	155
<a href="#">murasaki::Synchronizer</a>	
Synchronization class between a task and interrupt. This class provide the synchronization between a task and interrupt . . . . .	157
<a href="#">murasaki::Task</a>	
An easy to use task class . . . . .	158
<a href="#">murasaki::TaskStrategy</a>	
A mother of all tasks . . . . .	160
<a href="#">murasaki::Uart</a>	
Concrete implementation of UART controller. Based on the STM32Cube HAL DMA Transfer . . . . .	162
<a href="#">murasaki::UartLogger</a>	
Logging through an UART port . . . . .	168
<a href="#">murasaki::UartStrategy</a>	
Definition of the root class of UART . . . . .	170

## Chapter 10

# File Index

### 10.1 File List

Here is a list of all documented files with brief descriptions:

/home/takemasa/workspace_st/nucleo-f446-sample/Inc/main.h	
: Header for <a href="#">main.c</a> file. This file contains the common defines of the application	175
/home/takemasa/workspace_st/nucleo-f446-sample/Inc/murasaki_include_stub.h	
Stub to include the HAL headers	177
/home/takemasa/workspace_st/nucleo-f446-sample/Inc/murasaki_platform.hpp	
An interface for the applicaiton from murasaki library to <a href="#">main.c</a>	178
/home/takemasa/workspace_st/nucleo-f446-sample/Inc/platform_config.hpp	
Application dependent configuration	179
/home/takemasa/workspace_st/nucleo-f446-sample/Inc/platform_defs.hpp	
Murasaki platform customize file	180
/home/takemasa/workspace_st/nucleo-f446-sample/Inc/stm32f4xx_hal_conf.h	
HAL configuration file	181
/home/takemasa/workspace_st/nucleo-f446-sample/Inc/stm32f4xx_it.h	
This file contains the headers of the interrupt handlers	187
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc-tp/adau1361.hpp	187
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/audiocodecstrategy.hpp	189
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/audiostrategy.hpp	
Root class of the stereo audio	190
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitin.hpp	
GPIO bit in class	191
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitinstrategy.hpp	
Abstract class of the GPIO bit in	193
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitout.hpp	
GPIO bit out class	195
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitoutstrategy.hpp	
Abstract class of GPIO bit out	197
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/criticalsection.hpp	
Class to protect a certain section from the interference	199
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/debugger.hpp	
Debug print class. For both ISR and task	200
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/debuggerfifo.hpp	
Dedicated FIFO to logging the debug message	202
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/debuggeruart.hpp	
UART. Thread safe and blocking IO	204
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/fifostrategy.hpp	
Abstract class of FIFO	206

/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cmaster.hpp	
I2C master. Thread safe, blocking IO	207
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cmasterstrategy.hpp	
Root class definition of the I2C Master	209
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cslave.hpp	
I2C slave. Thread safe, blocking IO	211
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cslavestrategy.hpp	
Root class definition of the I2C Slave	213
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/loggerstrategy.hpp	
Simplified logging function	215
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki.hpp	
Application include file for Murasaki class library	217
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_0_intro.hpp	
Doxygen document file. No need to include	218
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_1_env.hpp	
Doxygen document file. No need to include	218
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_2_ug.hpp	
Doxygen document file. No need to include	218
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_3_pg.hpp	
Porting Guide	219
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_4_mod.hpp	
Module definition	219
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_5_spg.hpp	
Step by Step Porting guide	219
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_assert.hpp	
Assertion definition	219
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_config.hpp	
Configuration file for platform	221
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_defs.hpp	
Common definition of the platform	223
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/murasaki_syslog.hpp	
Syslog definition	224
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/peripheralstrategy.hpp	
Mother of All peripheral	225
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spimaster.hpp	
SPI Master. Thread safe and blocking IO	226
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spimasterstrategy.hpp	
SPI master root class	228
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislave.hpp	
SPI Slave. Thread safe and blocking IO	230
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislaveadapter.hpp	
STM32 SPI slave speifire	232
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislaveadapterstrategy.hpp	
Abstract class of SPI slave specification	234
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislavestrategy.hpp	
SPI master root class	236
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/synchronizer.hpp	
Synchronization between a Task and interrupt	238
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/task.hpp	
Simplified Task class	239
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/taskstrategy.hpp	
Mother of All Tasks	241
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/uart.hpp	
UART. Thread safe and blocking IO	242
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/uartlogger.hpp	
Logging to Uart	244
/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/uartstrategy.hpp	
Root class definition of the UART driver	246



/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Src/ <a href="#">allocators.cpp</a>	
Alternative memory allocators . . . . .	247
/home/takemasa/workspace_st/nucleo-f446-sample/Src/ <a href="#">main.c</a>	
: Main program body . . . . .	249
/home/takemasa/workspace_st/nucleo-f446-sample/Src/ <a href="#">murasaki_platform.cpp</a>	
A glue file between the user application and HAL/RTOS . . . . .	252
/home/takemasa/workspace_st/nucleo-f446-sample/Src/ <a href="#">stm32f4xx_it.c</a>	
Interrupt Service Routines . . . . .	253
/home/takemasa/workspace_st/nucleo-f446-sample/Src/ <a href="#">system_stm32f4xx.c</a>	
CMSIS Cortex-M4 Device Peripheral Access Layer System Source File . . . . .	255

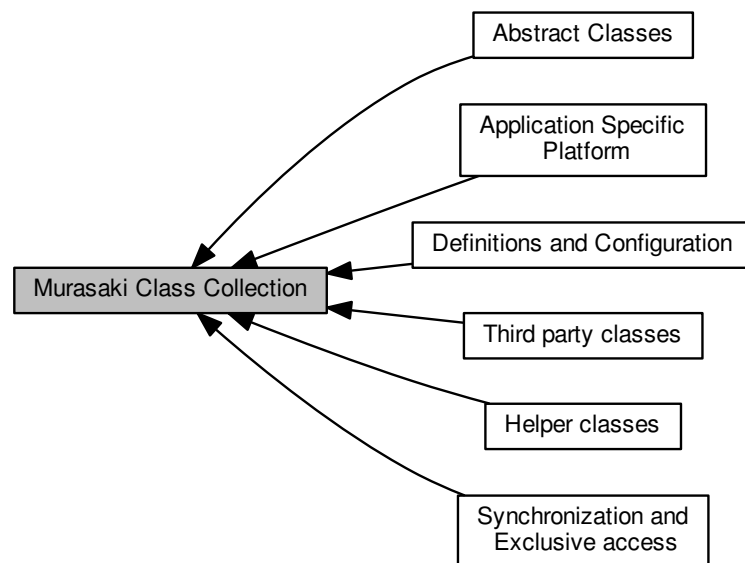


# Chapter 11

## Module Documentation

### 11.1 Murasaki Class Collection

Collaboration diagram for Murasaki Class Collection:



#### Modules

- [Synchronization and Exclusive access](#)
- [Third party classes](#)
- [Definitions and Configuration](#)
- [Application Specific Platform](#)
- [Abstract Classes](#)
- [Helper classes](#)

## Classes

- class [murasaki::BitIn](#)
- struct [murasaki::GPIO\\_type](#)
- class [murasaki::BitOut](#)
- class [murasaki::Debugger](#)
- class [murasaki::I2cMaster](#)
- class [murasaki::I2cSlave](#)
- class [murasaki::SpiMaster](#)
- class [murasaki::SpiSlave](#)
- class [murasaki::SpiSlaveAdapter](#)
- class [murasaki::Task](#)
- class [murasaki::Uart](#)
- class [murasaki::UartLogger](#)

## Macros

- `#define MURASAKI_ASSERT(COND)`
- `#define MURASAKI_PRINT_ERROR(ERR)`
- `#define MURASAKI_SYSLOG(FACILITY, SEVERITY, FORMAT, ...)`

### 11.1.1 Detailed Description

This is a reference guide of murasaki class library. This guide describes class by class and cover entire library. It is not recommended to read the reference for the first time user.

Alternatively, the [Usage Introduction](#) is provided to study step by step.

### 11.1.2 Macro Definition Documentation

#### 11.1.2.1 `#define MURASAKI_ASSERT( COND )`

##### Value:

```
if ( ! (COND) )\
{
    murasaki::debugger->Printf("-----\n");
    murasaki::debugger->Printf(MURASAKI_ASSERT_MSG, __func__, __LINE__,
    ,__MURASAKI_FILE__ );\
    murasaki::debugger->Printf("Fail expression : %s\n", #COND);\
    { void (*foo)(void) = (void (*)())1; foo(); }\
}
```

Assert the COND is true.

##### Parameters

<i>COND</i>	Condition as bool type.
-------------	-------------------------

Print the COND expression to the logging port if COND is false. Do nothing if CODN is true.

After printing the assertion failure message, this aspersion triggers the Hard Fault exception. The Hard Fault Exception is caught by [HardFault\\_Handler\(\)](#) and eventually invoke the [murasaki::debugger->DoPostMortem\(\)](#), to put the system into the post mortem debug mode.

Following code in the macro definition calls a non-existing function located address 1. Such the access causes a hard fault execution.

```
1 { void (*foo)(void) = (void (*)())1; foo(); }\
```

This assertion do nothing if programmer defines [MURASAKI\\_CONFIG\\_NODEBUG](#) macro as true. This macro is defined in the file [platform\\_config.hpp](#).

### 11.1.2.2 #define MURASAKI\_PRINT\_ERROR( ERR )

**Value:**

```
if ( (ERR) ) \
{ \
    murasaki::debugger->Printf(MURASAKI_ERROR_MSG, __func__, __LINE__, \
    __MURASAKI__FILE__, #ERR ); \
}
```

Print ERR if ERR is true.

**Parameters**

<i>ERR</i>	Condition as bool type.
------------	-------------------------

Print the ERR expression to the logging port if COND is true. Do nothing if ERR is true.

This assertion do nothing if programmer defines [MURASAKI\\_CONFIG\\_NODEBUG](#) macro as true. This macro is defined in the file [platform\\_config.hpp](#).

For example, following code is typical usage of this macro. ERROR macro is copied from STM32Cube HAL source code.

```
1 bool Uart::HandleError(void* const ptr)
2 {
3     MURASAKI_ASSERT(nullptr != ptr)
4
5     if (peripheral_ == ptr) {
6         // Check error, and print if exist.
7         MURASAKI_PRINT_ERROR(peripheral_>ErrorCode & HAL_UART_ERROR_DMA);
8         MURASAKI_PRINT_ERROR(peripheral_>ErrorCode & HAL_UART_ERROR_PE);
9         MURASAKI_PRINT_ERROR(peripheral_>ErrorCode & HAL_UART_ERROR_NE);
10        MURASAKI_PRINT_ERROR(peripheral_>ErrorCode & HAL_UART_ERROR_FE);
11        MURASAKI_PRINT_ERROR(peripheral_>ErrorCode & HAL_UART_ERROR_ORE);
12        MURASAKI_PRINT_ERROR(peripheral_>ErrorCode & HAL_UART_ERROR_DMA);
13        return true;    // report the ptr matched
14    }
15    else {
16        return false;    // report the ptr doesn't match
17    }
18 }
```

### 11.1.2.3 #define MURASAKI\_SYSLOG( FACILITY, SEVERITY, FORMAT, ... )

output The debug message

## Parameters

<i>FACILITY</i>	Specify which facility makes this log. Choose from <a href="#">murasaki::SyslogFacility</a>
<i>SEVERITY</i>	Specify how message is severe. Choose from <a href="#">murasaki::SyslogSeverity</a>
<i>FORMAT</i>	Message format as printf style.

Output the debug message to debug console output.

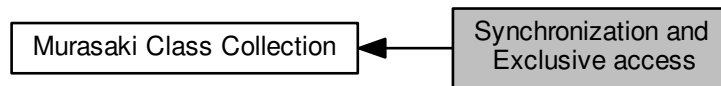
The output message is filtered by the internal threshold set by [murasaki::SetSyslogSererityThreshold](#), [murasaki::SetSyslogFacilityMask](#) and [murasaki::AddSyslogFacilityToMask](#). See these function's document to understand how filter works.

There is recommendation in the SEVERITY parameter :

- [murasaki::kseDebug](#) for Development/Debug message for tracing normal operation.
- [murasaki::kseWarning](#) for relatively severe condition which need abnormal action, or cannot handle.
- [murasaki::kseError](#) for faulty condition from HAL or hardware.
- [murasaki::kseEmergency](#) for software logic error like assert fail

## 11.2 Synchronization and Exclusive access

Collaboration diagram for Synchronization and Exclusive access:



### Classes

- class [murasaki::CriticalSection](#)
- class [murasaki::Synchronizer](#)

### 11.2.1 Detailed Description

These classes are used as parts of the other classes.

## 11.3 Third party classes

Collaboration diagram for Third party classes:



### Classes

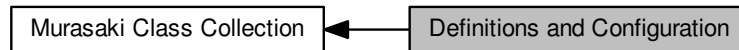
- class [murasaki::Adau1361](#)

### 11.3.1 Detailed Description



## 11.4 Definitions and Configuration

Collaboration diagram for Definitions and Configuration:



- `#define PLATFORM_CONFIG_DEBUG_LINE_SIZE 256`
- `#define PLATFORM_CONFIG_DEBUG_BUFFER_SIZE 4096`
- `#define PLATFORM_CONFIG_DEBUG_SERIAL_TIMEOUT (murasaki::kwmsIndefinitely)`
- `#define PLATFORM_CONFIG_DEBUG_TASK_STACK_SIZE 256`
- `#define PLATFORM_CONFIG_DEBUG_TASK_PRIORITY (( configMAX_PRIORITIES-1 > 0 ) ? configMAX_PRIORITIES-1 : 0)`
- `#define MURASAKI_CONFIG_NODEBUG false`

### 11.4.1 Detailed Description

### 11.4.2 Macro Definition Documentation

#### 11.4.2.1 `#define MURASAKI_CONFIG_NODEBUG false`

Suppress `MURASAKI_ASSERT` macro.

Set this macro to true, to discard the assertion `MURASAKI_ASSERT`. Set this macro false, to use the assertion.

To override the definition here, define same macro inside `platform_config.hpp`.

#### 11.4.2.2 `#define PLATFORM_CONFIG_DEBUG_BUFFER_SIZE 4096`

Size[byte] of the circular buffer to be transmitted through the serial port.

The circular buffer array length to copy the formatted strings before transmitting through the uart.

To override the definition here, define same macro inside `platform_config.hpp`.

#### 11.4.2.3 `#define PLATFORM_CONFIG_DEBUG_LINE_SIZE 256`

Size of one line[byte] in the debug printf.

The array length to store the formatted string. Note that this array is a private instance variable. Then, it will occupy the memory where the class is instantiated. For example, if an object is instantiated in the heap, this line buffer will be reserved in the heap.

If the class is instantiated on the stack, the buffer will be reserved in the stack.

To override the definition here, define same macro inside `platform_config.hpp`.

#### 11.4.2.4 `#define PLATFORM_CONFIG_DEBUG_SERIAL_TIMEOUT (murasaki::kwmsIndefinitely)`

Timeout of the serial port to transmit the string through the Debug class.

By default, there is no timeout. Wait for eternally.

To override the definition here, define same macro inside [platform\\_config.hpp](#).

#### 11.4.2.5 `#define PLATFORM_CONFIG_DEBUG_TASK_PRIORITY (( configMAX_PRIORITIES-1 > 0 ) ? configMAX_PRIORITIES-1 : 0)`

The task priority of the debug task.

The priority of the murasaki::Debugger internal task. To output the logging data as fast as possible, the debug task have to have relatively high priority. In other hand, to yield the CPU to the critical tasks, it's priority have to be smaller than the max priority.

To override the definition here, define same macro inside [platform\\_config.hpp](#).

#### 11.4.2.6 `#define PLATFORM_CONFIG_DEBUG_TASK_STACK_SIZE 256`

Size[Byte] of the task inside Debug class.

The murasaki::Debugger class has internal task to handle its FIFO buffer.

To override the definition here, define same macro inside [platform\\_config.hpp](#).

### 11.4.3 Enumeration Type Documentation

#### 11.4.3.1 `enum murasaki::I2cStatus`

Return status of the I2C classes.

This enums represents the return status from the I2C class method.

In a single master controller system, you need to care only `ki2csNak` and `ki2csTimeOut`. Other error may be caused by multiple master system.

The `ki2csNak` is returned when one of two happens :

- The slave device terminated transfer.
- No slave device responded to the address specified by master device.

The `ki2csTimeOut` is returned when slave device stretched transfer too long.

The `ki2csArbitrationLost` is returned when another master won the arbitration. Usually, the master have to re-try the transfer after certain waiting period.

The `ki2csBussError` is fatal condition. In the master mode, it could be problem of other device. The root cause is not deterministic. Probably it is hardware problem.

Enumerator

**`ki2csOK`** `ki2csOK`

**`ki2csTimeOut`** Master mode error. No response from device.

**`ki2csNak`** Master mode error. Device answers NAK.

**`ki2csBussError`** Master&Slave mode error. START/STOP condition at irregular location.

**`ki2csArbitrationLost`** Master&Slave mode error. Lost arbitration against other master device.

**`ki2csOverrun`** Slave mode error. Overrun or Underrun was detected.

**`ki2csDMA`** Some error detected in DMA module.

**`ki2csUnknown`** Unknown error.

11.4.3.2 enum `murasaki::SpiClockPhase`

SPI clock configuration for master.

This enum represents the setting of the SPI PHA bit of the master configuration. The PHA setting 0 and 1 is LatchThenShift and ShiftThenLatch respectively.

Enumerator

***kspgLatchThenShift*** `kscgLatchThenShift PHA=0`. The first edge is latching. The second edge is shifting.

***kspGShiftThenLatch*** `kscGShiftThenLatch PHA = 1`. The first edge is shifting. The second edge is latching.

11.4.3.3 enum `murasaki::SpiClockPolarity`

SPI clock configuration for Master.

This enum represents the setting of the SPI POL bit of the master configuration. The POL setting 0/1 is RiseThenFall and Fall thenRise respectively.

Enumerator

***kspoRiseThenFall*** `kscpRiseThenFall POL = 0`

***kspoFallThenRise*** `kscpFallThenrise POL = 1`

11.4.3.4 enum `murasaki::SpiStatus`

Return status of the SPI classes.

This enums represents the return status of from the SPI class method.

`kspisModeFault` is returned when the NSS pins are asserted. Note that the Murasaki library doesn't support the Multi master SPI operation. So, this is fatal condition.

`kpisOverflow` and the `kpisDMA` are fatal condition. These can be the problem of the lower driver problem.

Enumerator

***kspisOK*** `ki2csOK`

***kspisTimeOut*** Master mode error. No response from device.

***kspisModeFault*** SPI mode fault error. Two master collision.

***kspisModeCRC*** CRC protocol error.

***kspisOverflow*** Over run.

***kspisFrameError*** Error on TI frame mode.

***kspisDMA*** DMA error.

***kspisErrorFlag*** Other error flag.

***kspisAbort*** Problem in abort process. No way to recover.

***kspisUnknown*** Unknown error.

#### 11.4.3.5 enum murasaki::SyslogFacility

Category to filter the Syslog output.

These are independent facilities to filter the Syslog message output. Each module should specify appropriate facility.

Internally, these value will be used as bit position in mask.

##### Enumerator

**kfaKernel** kfaKernel is specified when the message is bound with the kernel issue.

**kfaSerial** kfaSerial is specified when the message is from the serial module.

**kfaSpiMaster** kfaSpi is specified when the message is from the SPI master module

**kfaSpiSlave** kfaSpi is specified when the message is from the SPI slave module

**kfaI2cMaster** kfaI2c is specified when the message is from the I2C master module.

**kfaI2cSlave** kfaI2c is specified when the message is from the I2C slave module.

**kfaI2s** kfaI2s is specified when the message is from the I2S module

**kfaSai** kfaSai is specified when the message is from the SAI module.

**kfaLog** kfaLog is specified when the message is from the logger and debugger module.

**kfaNone** Disable all facility.

**kfaAll** Enable all facility.

**kfaUser0** User defined facility.

**kfaUser1** User defined facility.

**kfaUser2** User defined facility.

**kfaUser3** User defined facility.

**kfaUser4** User defined facility.

**kfaUser5** User defined facility.

**kfaUser6** User defined facility.

**kfaUser7** User defined facility.

#### 11.4.3.6 enum murasaki::SyslogSeverity

Message severity level.

The lower value is the more serious condition.

##### Enumerator

**kseEmergency** kseEmergency means the system is unusable.

**kseAlert** kseAlert means some acution must be taken immediately.

**kseCritical** kseCritical means critical condition.

**kseError** kseError means error conditions.

**kseWarning** kseWarning means warning condition.

**kseNotice** kseNotice means normal but significant condition.

**kseInfomational** kseInfomational means infomational message.

**kseDebug** kseDebug means debug-level message

#### 11.4.3.7 enum `murasaki::UartHardwareFlowControl`

Attribute of the UART Hardware Flow Control.

This is dedicated to the [UartStrategy](#) class.

Enumerator

- kuhfcNone*** No hardware flow control.
- kuhfcCts*** Control CTS, but RTS.
- kuhfcRts*** Control RTS, but CTS.
- kuhfcCtsRts*** Control Both CTS and RTS.

#### 11.4.3.8 enum `murasaki::UartStatus`

Return status of the UART classes.

The Parity error and the Frame error may occur when user connects DCT/DTE by different communication setting.

The Noise error may cause by the noise on the line.

The overrun may cause when the DMA is too slow or hand shake is not working well.

The DMA error may cause some problem inside HAL.

Enumerator

- kursOK*** No error.
- kursTimeOut*** Time out during transmission / receive.
- kursParity*** Parity error.
- kursNoise*** Error by Noise.
- kursFrame*** Frame error.
- kursOverrun*** Overrun error.
- kursDMA*** Error inside DMA module.

#### 11.4.3.9 enum `murasaki::UartTimeout`

This is specific enum for the `AbstractUart::Receive()` to specify the use of idle line timeout.

The idle line time out is dedicated function of the STM32 peripherals. The interrupt happens when the receive data is discontinued certain time.

Enumerator

- kutNoldleTimeout*** `kutNoldleTimeout` is specified when API should has normal timeout.
- kutIdleTimeout*** `kutIdleTimeout` is specified when API should time out by Idle line

#### 11.4.3.10 enum `murasaki::WaitMilliseconds` : `uint32_t`

Wait time by milliseconds. For the function which has "wait" or "timeout" parameter.

An `uint32_t` derived type for specifying wait duration. The integer value represents the waiting duration by milliseconds. Usually a value of this type is passed to some functions as parameter. There are two special cases.

`kwmsPolling` means function will return immediately regardless of waited event. In other word, with this parameter, function causes time out immediately. Some function may provides the way to know what was the status of the waited event. But some may not.

`kwmsIndefinitely` means function will will not cause time out.

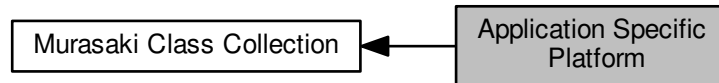
##### Enumerator

***kwmsPolling*** Not waiting. Immediate timeout.

***kwmsIndefinitely*** Wait forever.

## 11.5 Application Specific Platform

Collaboration diagram for Application Specific Platform:



### Classes

- struct `murasaki::Platform`

### Functions

- void `InitPlatform ()`
- void `ExecPlatform ()`
- void `CustomAssertFailed (uint8_t *file, uint32_t line)`
- void `CustomDefaultHandler ()`
- void `HAL_UART_TxCpltCallback (UART_HandleTypeDef *huart)`
- void `HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart)`
- void `HAL_UART_ErrorCallback (UART_HandleTypeDef *huart)`
- void `HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef *hspi)`
- void `HAL_SPI_ErrorCallback (SPI_HandleTypeDef *hspi)`
- void `HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef *hi2c)`
- void `HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef *hi2c)`
- void `HAL_I2C_ErrorCallback (I2C_HandleTypeDef *hi2c)`
- void `HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)`

### Variables

- Debugger \* `murasaki::debugger`

#### 11.5.1 Detailed Description

Typical usage of these variables can be seen below. First of all, an .cpp file have to include `murasaki.hpp`.

```
#include "murasaki.hpp"
```

And then, define the `murasaki::debugger` in the global context. Note that this is essential to use certain debug macros.

The definition of the `murasaki::platform` is optional. But it is recommended to declare for the ease of reading.

```

murasaki::Debugger * murasaki::debugger;
murasaki::Platform * murasaki::platform;

```

Finally, initialize the `murasaki::debugger` and `murasaki::platform`. Again, the `murasaki::debugger` is essential to use the debug macro. The debug macros are used inside murasaki class library. Then, it is mandatory to initialize the debugger member variable.

The following code fragment initialize only the debugger related member variables. Also, the `murasaki::Platform` variable is refereed.

The platform.uart\_console member variable hooks a murasaki::AbstractUart class variable. In this sample, The `murasaki::Uart` class is instantiated. The Uart constructor receives the pointer to the UART\_HandleTypeDef. Usually, the UART\_HandleTypeDef variable is generated by CubeMX. For example, "huart3" variable in the `main.c` file.

The platform.logger member variable hooks a murasaki::AbstractLogger variable. In this example, `murasaki::UartLogger` class variable is instantiated.

Finally, the debugger variable is initialized. The `murasaki::Debugger` constructor receives murasaki::AbstractLogger \* type.

```

void InitPlatform(UART_HandleTypeDef * uart_handle)
{
    murasaki::platform.uart_console = new murasaki::Uart(uart_handle);
    murasaki::platform.logger = new murasaki::UartLogger(murasaki::platform.
        uart_console);

    murasak::debugger = new murasaki::Debugger(murasaki::platform.logger
        );
}

```

## 11.5.2 Function Documentation

### 11.5.2.1 void CustomAssertFailed ( uint8\_t \* file, uint32\_t line )

Hook for the assert\_failure() in `main.c`.

#### Parameters

<i>file</i>	Name of the source file where assertion happen
<i>line</i>	Number of the line where assertion happen

This routine provides a custom hook for the assertion inside STM32Cube HAL. All assertion raised in HAL will be redirected here.

```

1 void assert_failed(uint8_t* file, uint32_t line)
2 {
3     CustomAssertFailed(file, line);
4 }

```

By default, this routine output a message with location informaiton to the debugger console.



### 11.5.2.2 void CustomDefaultHandler ( )

Hook for the default exception handler. Never return.

This routine is invoked from the default handler of the start up file. The modification to the startup file is user's responsibility.

For example, the start up code for the Nucleo-L152RE is startup\_stm152xe.s. This file is generated by CubeMX. This file has default handler as like this:

```
1 .section .text.Default_Handler,"ax",%progbits
2     Default_Handler:
3 Infinite_Loop:
4     b Infinite_Loop
```

This code can be modified to call CustomDefaultHanler as like this :

```
1 .global CustomDefaultHandler
2 .section .text.Default_Handler,"ax",%progbits
3 Default_Handler:
4     bl CustomDefaultHandler
5 Infinite_Loop:
6     b Infinite_Loop
```

### 11.5.2.3 void ExecPlatform ( )

The body of the real application.

The body function of the murasaki application. Usually this function is called from the [StartDefaultTask\(\)](#) of the [main.c](#).

This function is invoked only once, and never return. See [InitPlatform\(\)](#) as calling sample.

By default, it toggles LED as sample program. Inside this function can be customized freely.

### 11.5.2.4 void HAL\_GPIO\_EXTI\_Callback ( uint16\_t GPIO\_Pin )

Optional interrupt handling of EXTI.

#### Parameters

<i>GPIO_Pin</i>	Pin number from 0 to 31
-----------------	-------------------------

This is called from inside of HAL when an EXTI is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

The GPIO\_Pin is the number of Pin. For example, if a programmer set the pin name by CubeMX as FOO, the macro to identify that EXTI is FOO\_Pin

#### 11.5.2.5 void HAL\_I2C\_ErrorCallback ( I2C\_HandleTypeDef \* hi2c )

Optional error handling of I2C.

##### Parameters

<i>hi2c</i>	
-------------	--

This is called from inside of HAL when an I2C error interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

In this call back, the uart device handle have to be passed to the `murasaki::I2c::HandleError()` function.

#### 11.5.2.6 void HAL\_I2C\_MasterTxCpltCallback ( I2C\_HandleTypeDef \* hi2c )

Essential to sync up with I2C.

##### Parameters

<i>hi2c</i>	
-------------	--

This is called from inside of HAL when an I2C transmission done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX interrupt call back.

In this call back, the uart device handle have to be passed to the `murasaki::I2c::TransmitCompleteCallback()` function.

#### 11.5.2.7 void HAL\_I2C\_SlaveTxCpltCallback ( I2C\_HandleTypeDef \* hi2c )

Essential to sync up with I2C.

##### Parameters

<i>hi2c</i>	
-------------	--

This is called from inside of HAL when an I2C transmission done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX interrupt call back.

In this call back, the I2C slave device handle have to be passed to the [murasaki::I2cSlave::TransmitCompleteCallback\(\)](#) function.

#### 11.5.2.8 void HAL\_SPI\_ErrorCallback ( SPI\_HandleTypeDef \* hspi )

Optional error handling of SPI.

##### Parameters

<i>hspi</i>	
-------------	--

This is called from inside of HAL when an SPI error interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

In this call back, the uart device handle have to be passed to the [murasaki::Uart::HandleError\(\)](#) function.

#### 11.5.2.9 void HAL\_SPI\_TxRxCpltCallback ( SPI\_HandleTypeDef \* hspi )

Essential to sync up with SPI.

##### Parameters

<i>hspi</i>	
-------------	--

This is called from inside of HAL when an SPI transfer done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX/RX interrupt call back.

In this call back, the SPI device handle have to be passed to the [murasaki::Spi::TransmitAndReceiveCompleteCallback\(\)](#) function.

#### 11.5.2.10 void HAL\_UART\_ErrorCallback ( UART\_HandleTypeDef \* huart )

Optional error handling of UART.

##### Parameters

<i>huart</i>	
--------------	--

This is called from inside of HAL when an UART error interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

In this call back, the uart device handle have to be passed to the [murasaki::Uart::HandleError\(\)](#) function.

#### 11.5.2.11 void HAL\_UART\_RxCpltCallback ( UART\_HandleTypeDef \* *huart* )

Essential to sync up with UART.

##### Parameters

<i>huart</i>	
--------------	--

This is called from inside of HAL when an UART receive done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default RX interrupt call back.

In this call back, the uart device handle have to be passed to the [murasaki::Uart::ReceiveCompleteCallback\(\)](#) function.

#### 11.5.2.12 void HAL\_UART\_TxCpltCallback ( UART\_HandleTypeDef \* *huart* )

Essential to sync up with UART.

##### Parameters

<i>huart</i>	
--------------	--

This is called from inside of HAL when an UART transmission done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX interrupt call back.

In this call back, the uart device handle have to be passed to the [murasaki::Uart::TransmissionCompleteCallback\(\)](#) function.

#### 11.5.2.13 void InitPlatform ( )

Initialize the platform variables.

The [murasaki::platform](#) variable is an interface between the application program and HAL / RTOS. To use it correctly, the initialization is needed before any activity of murasaki client.

```
1 void StartDefaultTask(void const * argument)
2 {
3     InitPlatform();
4     ExecPlatform();
5 }
```

This function have to be invoked from the [StartDefaultTask\(\)](#) of the [main.c](#) only once to initialize the platform variable.

### 11.5.3 Variable Documentation

#### 11.5.3.1 `murasaki::Debugger` \* `murasaki::debugger`

Global variable to provide the debugging function.

This variable is declared by murasaki platform. But not instantiated. To make it happen, programmer have to make an variable and initialize it explicitly. Otherwise, Certain debug utility/macro may cause link error, because `murasaki::debugger` is referred by these utility/macros.

## 11.6 Abstract Classes

Collaboration diagram for Abstract Classes:



### Classes

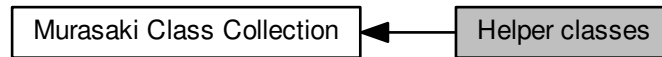
- class [murasaki::AudioCodecStrategy](#)
- class [murasaki::BitInStrategy](#)
- class [murasaki::BitOutStrategy](#)
- class [murasaki::FifoStrategy](#)
- class [murasaki::I2CMasterStrategy](#)
- class [murasaki::I2cSlaveStrategy](#)
- class [murasaki::LoggerStrategy](#)
- class [murasaki::PeripheralStrategy](#)
- class [murasaki::SpiMasterStrategy](#)
- class [murasaki::SpiSlaveAdapterStrategy](#)
- class [murasaki::SpiSlaveStrategy](#)
- class [murasaki::TaskStrategy](#)
- class [murasaki::UartStrategy](#)

### 11.6.1 Detailed Description

Usually, application doesn't instantiate these classes. But pointer may be declared as abstract class as generic placeholder.

## 11.7 Helper classes

Collaboration diagram for Helper classes:



### Classes

- class `murasaki::DebuggerFifo`
- struct `murasaki::LoggingHelpers`
- class `murasaki::DebuggerUart`

### Functions

- void \* `operator new` (std::size\_t size)
- void \* `operator new[]` (std::size\_t size)
- void `operator delete` (void \*ptr)
- void `operator delete[]` (void \*ptr)

#### 11.7.1 Detailed Description

These classes are not used by customer.

#### 11.7.2 Function Documentation

##### 11.7.2.1 void operator delete ( void \* *ptr* )

Deallocate the given memory.

##### Parameters

<i>ptr</i>	Pointer to the memory to deallocate
------------	-------------------------------------

##### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.

### 11.7.2.2 void operator delete[] ( void \* *ptr* )

Deallocate the given memory.

#### Parameters

<i>ptr</i>	Pointer to the memory to deallocate
------------	-------------------------------------

#### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.

### 11.7.2.3 void\* operator new ( std::size\_t *size* )

Allocate a memory piece with given size.

#### Parameters

<i>size</i>	Size of the memory to allocate [byte]
-------------	---------------------------------------

#### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.

### 11.7.2.4 void\* operator new[] ( std::size\_t *size* )

Allocate a memory piece with given size.

#### Parameters

<i>size</i>	Size of the memory to allocate [byte]
-------------	---------------------------------------

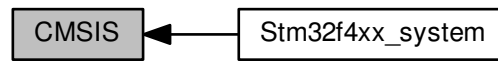
#### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.



## 11.8 CMSIS

Collaboration diagram for CMSIS:



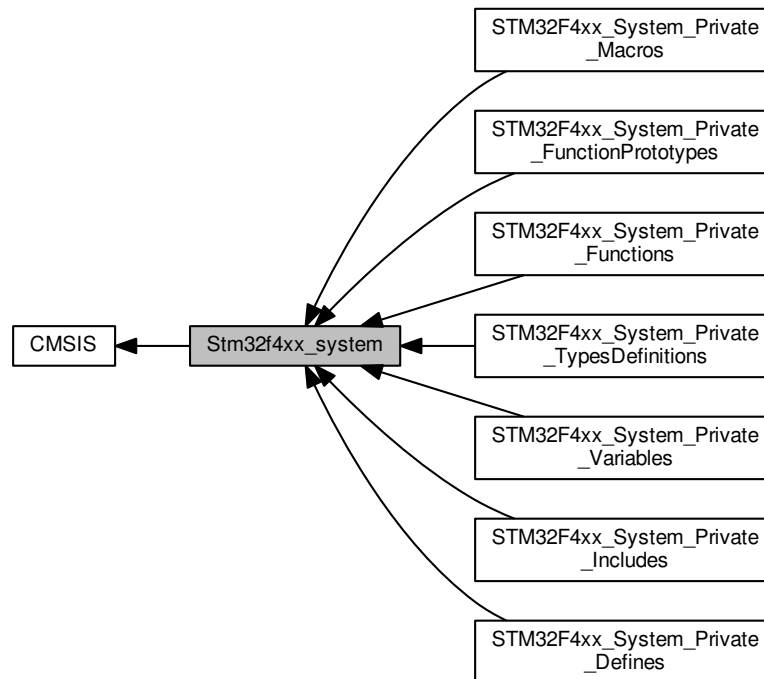
### Modules

- [Stm32f4xx\\_system](#)

### 11.8.1 Detailed Description

## 11.9 Stm32f4xx\_system

Collaboration diagram for Stm32f4xx\_system:



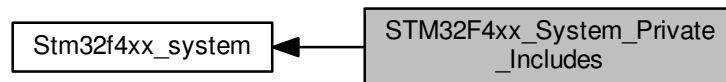
### Modules

- [STM32F4xx\\_System\\_Private\\_Includes](#)
- [STM32F4xx\\_System\\_Private\\_TypeDefinitions](#)
- [STM32F4xx\\_System\\_Private\\_Defines](#)
- [STM32F4xx\\_System\\_Private\\_Macros](#)
- [STM32F4xx\\_System\\_Private\\_Variables](#)
- [STM32F4xx\\_System\\_Private\\_FunctionPrototypes](#)
- [STM32F4xx\\_System\\_Private\\_Functions](#)

### 11.9.1 Detailed Description

## 11.10 STM32F4xx\_System\_Private\_Includes

Collaboration diagram for STM32F4xx\_System\_Private\_Includes:



### Macros

- `#define HSE_VALUE ((uint32_t)25000000)`
- `#define HSI_VALUE ((uint32_t)16000000)`

#### 11.10.1 Detailed Description

#### 11.10.2 Macro Definition Documentation

##### 11.10.2.1 `#define HSE_VALUE ((uint32_t)25000000)`

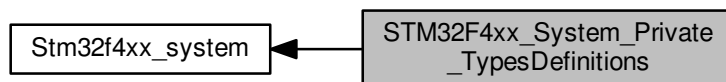
Default value of the External oscillator in Hz

##### 11.10.2.2 `#define HSI_VALUE ((uint32_t)16000000)`

Value of the Internal oscillator in Hz

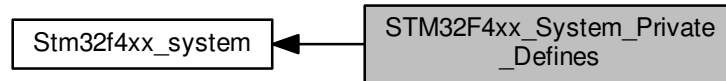
## 11.11 STM32F4xx\_System\_Private\_TypesDefinitions

Collaboration diagram for STM32F4xx\_System\_Private\_TypesDefinitions:



## 11.12 STM32F4xx\_System\_Private\_Defines

Collaboration diagram for STM32F4xx\_System\_Private\_Defines:



### Macros

- `#define VECT_TAB_OFFSET 0x00`

#### 11.12.1 Detailed Description

#### 11.12.2 Macro Definition Documentation

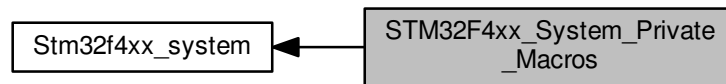
##### 11.12.2.1 `#define VECT_TAB_OFFSET 0x00`

< Uncomment the following line if you need to use external SRAM or SDRAM as data memory

< Uncomment the following line if you need to relocate your vector Table in Internal SRAM. Vector Table base offset field. This value must be a multiple of 0x200.

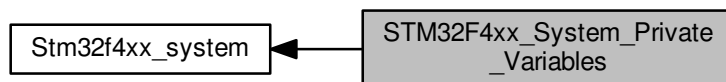
## 11.13 STM32F4xx\_System\_Private\_Macros

Collaboration diagram for STM32F4xx\_System\_Private\_Macros:



## 11.14 STM32F4xx\_System\_Private\_Variables

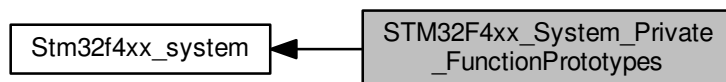
Collaboration diagram for STM32F4xx\_System\_Private\_Variables:



### 11.14.1 Detailed Description

## 11.15 STM32F4xx\_System\_Private\_FunctionPrototypes

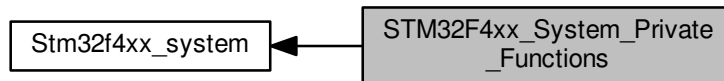
Collaboration diagram for STM32F4xx\_System\_Private\_FunctionPrototypes:





## 11.16 STM32F4xx\_System\_Private\_Functions

Collaboration diagram for STM32F4xx\_System\_Private\_Functions:



### Functions

- void [SystemInit](#) (void)
- void [SystemCoreClockUpdate](#) (void)

#### 11.16.1 Detailed Description

#### 11.16.2 Function Documentation

##### 11.16.2.1 void SystemCoreClockUpdate ( void )

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

#### Note

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the [HSI\\_VALUE\(\\*\)](#)
- If SYSCLK source is HSE, SystemCoreClock will contain the [HSE\\_VALUE\(\\*\\*\)](#)
- If SYSCLK source is PLL, SystemCoreClock will contain the [HSE\\_VALUE\(\\*\\*\)](#) or [HSI\\_VALUE\(\\*\)](#) multiplied/divided by the PLL factors.

(\*) HSI\_VALUE is a constant defined in [stm32f4xx\\_hal\\_conf.h](#) file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.

(\*\*) HSE\_VALUE is a constant defined in [stm32f4xx\\_hal\\_conf.h](#) file (its value depends on the application requirements), user has to ensure that HSE\_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.

- The result of this function could be not correct when using fractional value for HSE crystal.

**Parameters**

<i>None</i>	
-------------	--

**Return values**

<i>None</i>	
-------------	--

**11.16.2.2 void SystemInit ( void )**

Setup the microcontroller system Initialize the FPU setting, vector table location and External memory configuration.

**Parameters**

<i>None</i>	
-------------	--

**Return values**

<i>None</i>	
-------------	--

## Chapter 12

# Namespace Documentation

### 12.1 murasaki Namespace Reference

#### Classes

- class [Adau1361](#)
- class [AudioCodecStrategy](#)
- class [AudioStrategy](#)
- class [BitIn](#)
- class [BitInStrategy](#)
- class [BitOut](#)
- class [BitOutStrategy](#)
- class [CriticalSection](#)
- class [Debugger](#)
- class [DebuggerFifo](#)
- class [DebuggerUart](#)
- class [FifoStrategy](#)
- struct [GPIO\\_type](#)
- class [I2cMaster](#)
- class [I2cMasterStrategy](#)
- class [I2cSlave](#)
- class [I2cSlaveStrategy](#)
- class [LoggerStrategy](#)
- struct [LoggingHelpers](#)
- class [PeripheralStrategy](#)
- struct [Platform](#)
- class [SpiMaster](#)
- class [SpiMasterStrategy](#)
- class [SpiSlave](#)
- class [SpiSlaveAdapter](#)
- class [SpiSlaveAdapterStrategy](#)
- class [SpiSlaveStrategy](#)
- class [Synchronizer](#)
- class [Task](#)
- class [TaskStrategy](#)
- class [Uart](#)
- class [UartLogger](#)
- class [UartStrategy](#)

## Functions

- void [SetSyslogSererityThreshold](#) ([murasaki::SyslogSeverity](#) severity)
- void [SetSyslogFacilityMask](#) (uint32\_t mask)
- void [AddSyslogFacilityToMask](#) ([murasaki::SyslogFacility](#) facility)
- void [RemoveSyslogFacilityFromMask](#) ([murasaki::SyslogFacility](#) facility)
- bool [AllowedSyslogOut](#) ([murasaki::SyslogFacility](#) facility, [murasaki::SyslogSeverity](#) severity)

## Variables

- [Debugger](#) \* [debugger](#)
- [Platform](#) [platform](#)

### 12.1.1 Detailed Description

This name space encloses personal collections of the software parts to create a "platform" of the software development. This specific collection is based on the STM32Cube HAL and FreeRTOS, both are generated by CubeMX.

### 12.1.2 Function Documentation

#### 12.1.2.1 void [murasaki::AddSyslogFacilityToMask](#) ( [murasaki::SyslogFacility](#) *facility* )

Add Syslog facility to the filter mask.

##### Parameters

<i>facility</i>	Allow this facility to output
-----------------	-------------------------------

See [AllowedSyslogOut](#) to understand when the message is out.

#### 12.1.2.2 bool [murasaki::AllowedSyslogOut](#) ( [murasaki::SyslogFacility](#) *facility*, [murasaki::SyslogSeverity](#) *severity* )

Check if given facility and severity message is allowed to output.

##### Parameters

<i>facility</i>	Message facility
<i>severity</i>	Message seveirty

##### Returns

True if the message is allowed to out. False if not allowed.

By comapring internal seveiry threshold and facility mask, decide whether the message can be out or not.

If severity is higher than or equal to `kseError`, message is allowed to out.

If the severity is lower than `kseError`, the message is allowed to out only when :

- The severity is higher than or equal to the internal threshold
- The facility is "1" in the corresponding bit of the internal facility mask.

#### 12.1.2.3 void murasaki::RemoveSyslogFacilityFromMask ( murasaki::SyslogFacility *facility* )

Remove Syslog facility to the filter mask.

##### Parameters

<i>facility</i>	Deny this facility to output
-----------------	------------------------------

See [AllowedSyslogOut](#) to understand when the message is out.

#### 12.1.2.4 void murasaki::SetSyslogFacilityMask ( uint32\_t *mask* )

Set the syslog facility mask.

##### Parameters

<i>mask</i>	Facility bit mask. "1" allows output of the corresponding facility
-------------	--

The parameter is not the facility. A bit mask. By default, the bit mask is 0xFFFFFFFF which allows all facility.

See [AllowedSyslogOut](#) to understand when the message is out.

#### 12.1.2.5 void murasaki::SetSyslogSeverityThreshold ( murasaki::SyslogSeverity *severity* )

Set the syslog severity threshold.

##### Parameters

<i>severity</i>	
-----------------	--

Set the severity threshold. The message below this levels are ignored.

### 12.1.3 Variable Documentation

#### 12.1.3.1 murasaki::Platform murasaki::platform

Global variable to provide the access to the platform component.

This variable is declared by murasaki platform. But not instantiated. To make it happen, programmer have to make an variable and initilize it explicitly.

Note that the instantiation of this variable is optional. This is provided just of ease of read.

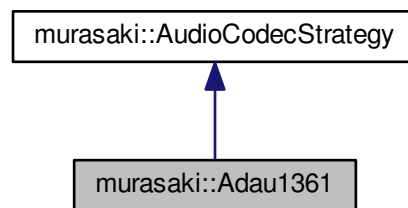
## Chapter 13

# Class Documentation

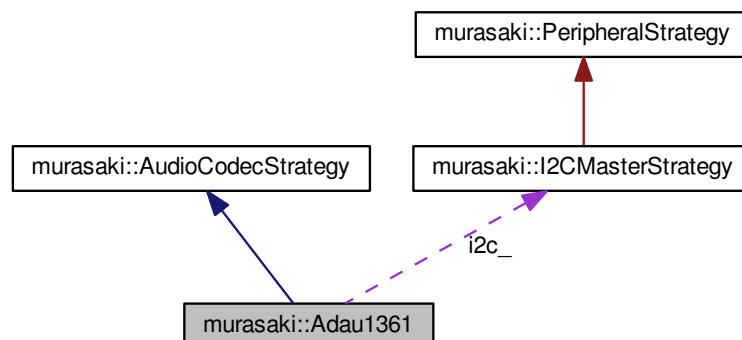
### 13.1 murasaki::Adu1361 Class Reference

```
#include <adau1361.hpp>
```

Inheritance diagram for murasaki::Adu1361:



Collaboration diagram for murasaki::Adu1361:



## Public Member Functions

- [Adu1361](#) (unsigned int fs, [murasaki::I2CMasterStrategy](#) \*controller, unsigned int i2c\_device\_addr)
- virtual void [start](#) (void)
- virtual void [set\\_line\\_input\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)
- virtual void [set\\_aux\\_input\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)
- virtual void [set\\_line\\_output\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)
- virtual void [set\\_hp\\_output\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)

## Protected Member Functions

- virtual void [configure\\_pll](#) (void)=0
- virtual void [configure\\_board](#) (void)=0
- virtual void [send\\_command](#) (const uint8\_t command[], int size)
- virtual void [send\\_command\\_table](#) (const uint8\_t table[][3], int rows)
- virtual void [wait\\_pll\\_lock](#) (void)

### 13.1.1 Constructor & Destructor Documentation

13.1.1.1 **murasaki::Adu1361::Adu1361** ( unsigned int *fs*, [murasaki::I2CMasterStrategy](#) \* *controller*, unsigned int *i2c\_device\_addr* )

constructor.

#### Parameters

<i>fs</i>	Sampling frequency.
<i>controller</i>	Pass the I2C controller object.
<i>i2c_device_addr</i>	I2C device address. value range is from 0 to 127

initialize the internal variables.

### 13.1.2 Member Function Documentation

13.1.2.1 **virtual void murasaki::Adu1361::configure\_board** ( void ) [protected],[pure virtual]

configuration of the ADAU1361 for the codec board

A pure virtual function.

This member function must be overridden by inherited class. Before the calling of this function, the codec is initialized as default state except PLL. PLL is set by [configure\\_pll\(\)](#) method before calling this function.

This member function must configure the ADAU1361 registered based on the board circuit. For example, internal signal pass or bias.



13.1.2.2 `virtual void murasaki::Adau1361::configure_pll ( void ) [protected], [pure virtual]`

configuration of PLL for the desired core clock

A pure virtual function.

This member function must be overridden by inherited class. Before the call of this function, R0 is initialized as 0 and then, set the clock source is PLL.

This member function must configure the PLL correctly, confirm the PLL lock status. And then set the SRC.

Note that the setting SRC before PLL lock may fail.

13.1.2.3 `virtual void murasaki::Adau1361::send_command ( const uint8_t command[], int size ) [protected], [virtual]`

send one command to ADAU1361.

Service function for the ADAU1361 board implementer.

#### Parameters

<i>command</i>	command data array. It have to have register address of ADAU1361 in first two bytes.
<i>size</i>	number of bytes in the command, including the register address.

Send one complete command to ADAU1361 by I2C.

13.1.2.4 `virtual void murasaki::Adau1361::send_command_table ( const uint8_t table[][3], int rows ) [protected], [virtual]`

send one command to ADAU1361.

#### Parameters

<i>table</i>	command table. All commands are stored in one row. Each row has only 1 byte data after reg address.
<i>rows</i>	number of the rows in the table.

Service function for the ADAU1361 board implementer.

Send a list of command to ADAU1361. All commands has 3 bytes length. That mean, after two byte register address, only 1 byte data payload is allowed. Commands are sent by I2C

13.1.2.5 `virtual void murasaki::Adau1361::set_aux_input_gain ( float left_gain, float right_gain, bool mute = false ) [virtual]`

Set the aux input gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

Other input lines are not killed. To kill it, user have to mute them explicitly.

Reimplemented from [murasaki::AudioCodecStrategy](#).

```
13.1.2.6 virtual void murasaki::Adau1361::set_hp_output_gain ( float left_gain, float right_gain, bool mute = false )
[virtual]
```

Set the headphone output gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

Other out line like line in are not killed. To kill it, user have to mute them explicitly.

Reimplemented from [murasaki::AudioCodecStrategy](#).

```
13.1.2.7 virtual void murasaki::Adau1361::set_line_input_gain ( float left_gain, float right_gain, bool mute = false )
[virtual]
```

Set the line input gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

As same as [start\(\)](#), this gain control function uses the single-end negative input only. Other input signal of the line in like positive signal or diff signal are killed.

Other input line like aux are not killed. To kill it, user have to mute them explicitly.

Reimplemented from [murasaki::AudioCodecStrategy](#).

```
13.1.2.8 virtual void murasaki::Adau1361::set_line_output_gain ( float left_gain, float right_gain, bool mute = false )
[virtual]
```

Set the line output gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

Other output lines are not killed. To kill it, user have to mute them explicitly.

Reimplemented from [murasaki::AudioCodecStrategy](#).

### 13.1.2.9 virtual void murasaki::Adau1361::start ( void ) [virtual]

Set up the ADAU1361 codec, and then, start the codec.

This method starts the ADAU1361 AD/DA conversion and I2S communication.

The line in is configured to use the Single-End negative input. This is funny but ADAU1361 datasheet specifies to do it. The positive in and diff in are killed. All biases are set as "normal".

The CODEC is configured as master mode. That mean, bclk and WS are given from ADAU1361 to the micro processor.

Implements [murasaki::AudioCodecStrategy](#).

### 13.1.2.10 virtual void murasaki::Adau1361::wait\_pll\_lock ( void ) [protected],[virtual]

wait until PLL locks.

Service function for the ADAu1361 board implementer.

Read the PLL status and repeat it until the PLL locks.

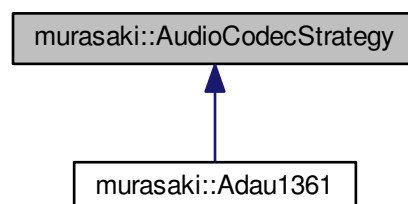
The documentation for this class was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc-tp/adau1361.hpp](/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc-tp/adau1361.hpp)

## 13.2 murasaki::AudioCodecStrategy Class Reference

```
#include <audiocodecstrategy.hpp>
```

Inheritance diagram for murasaki::AudioCodecStrategy:



## Public Member Functions

- [AudioCodecStrategy](#) (unsigned int fs)
- virtual void [start](#) (void)=0
- virtual void [set\\_line\\_input\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)
- virtual void [set\\_aux\\_input\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)
- virtual void [set\\_mic\\_input\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)
- virtual void [set\\_line\\_output\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)
- virtual void [set\\_hp\\_output\\_gain](#) (float left\_gain, float right\_gain, bool mute=false)

### 13.2.1 Detailed Description

This class is template for all codec classes

### 13.2.2 Constructor & Destructor Documentation

13.2.2.1 `murasaki::AudioCodecStrategy::AudioCodecStrategy ( unsigned int fs ) [inline]`

constructor.

Parameters

<i>fs</i>	Sampling frequency.
-----------	---------------------

initialize the internal variables.

### 13.2.3 Member Function Documentation

13.2.3.1 `virtual void murasaki::AudioCodecStrategy::set_aux_input_gain ( float left_gain, float right_gain, bool mute = false ) [inline],[virtual]`

Set the aux input gain and enable the relevant mixer.

Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

Reimplemented in [murasaki::Adau1361](#).

13.2.3.2 `virtual void murasaki::AudioCodecStrategy::set_hp_output_gain ( float left_gain, float right_gain, bool mute = false ) [inline],[virtual]`

Set the headphone output gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

Reimplemented in [murasaki::Adau1361](#).

**13.2.3.3** `virtual void murasaki::AudioCodecStrategy::set_line_input_gain ( float left_gain, float right_gain, bool mute = false ) [inline],[virtual]`

Set the line input gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

Reimplemented in [murasaki::Adau1361](#).

**13.2.3.4** `virtual void murasaki::AudioCodecStrategy::set_line_output_gain ( float left_gain, float right_gain, bool mute = false ) [inline],[virtual]`

Set the line output gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

Reimplemented in [murasaki::Adau1361](#).

**13.2.3.5** `virtual void murasaki::AudioCodecStrategy::set_mic_input_gain ( float left_gain, float right_gain, bool mute = false ) [inline],[virtual]`

Set the mic input gain and enable the relevant mixer.

## Parameters

<i>left_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>right_gain</i>	Gain by dB. The gain value outside of the acceptable range will be saturated.
<i>mute</i>	set true to mute

13.2.3.6 `virtual void murasaki::AudioCodecStrategy::start ( void ) [pure virtual]`

Actual initializer.

Initialize the codec itself and start the conversion process. and configure for given parameter.

Finally, set the input gain to 0dB.

Implemented in [murasaki::Adau1361](#).

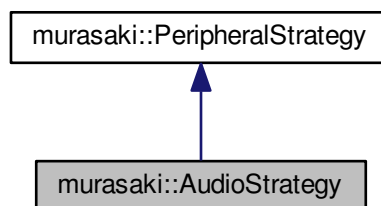
The documentation for this class was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/audiocodecstrategy.hpp](#)

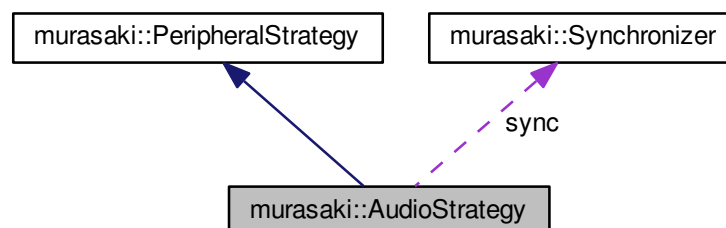
### 13.3 murasaki::AudioStrategy Class Reference

```
#include <audiostrategy.hpp>
```

Inheritance diagram for murasaki::AudioStrategy:



Collaboration diagram for murasaki::AudioStrategy:



## Public Member Functions

- [AudioStrategy](#) (void \*peripheral, unsigned int channel\_length, unsigned int num\_phases, unsigned int num\_channels)
- void [TransmitAndReceive](#) (float \*\*tx\_channels, float \*\*rx\_channels)
- void [TransmitAndReceive](#) (float \*tx\_left, float \*tx\_right, float \*rx\_left, float \*rx\_right)

### 13.3.1 Detailed Description

This class provides an interface to the audio peripheral. Also the internal buffer allocation, multi-phase buffering, and synchronization are provided. The features are :

- Stereo to multi-ch audio
- 32bit floating point buffer as interface with application.
- data range is [-1.0, 1.0) as interface with application.
- blocking and synchronous API
- Internal DMA operation.

Internally, this class provide a multi-buffer DMA operation between the audio peripheral and caller algorithm. The key API is the [TransmitAndReceive](#) member function. This function provide the several key operations

- Multiple-buffer operation to allow a background DMA transfer during caller is processing data.
- Data conversion and scaling between caller's floating point data and DMA's integer data.
- Synchronization between [TransmitAndReceive\(\)](#) and [ReceiveCallback\(\)](#)
- Exclusive access to peripheral

Thus, user doesn't need these things.

The multi-buffer DMA transfer is so called double-buffer or ripple buffer transfer. In Murasaki, "double", "triple" are named as phase. Programmer can specify the phase freely through the constructor.

### 13.3.2 Constructor & Destructor Documentation

13.3.2.1 `murasaki::AudioStrategy::AudioStrategy ( void * peripheral, unsigned int channel_length, unsigned int num_phases, unsigned int num_channels )`

Constructor.

#### Parameters

<i>peripheral</i>	The pointer to the peripheral management variable.
<i>channel_length</i>	Specify how many data are in one channel buffer.
<i>num_phases</i>	2 for double-buffers, 3 for triple-buffers.
<i>num_channels</i>	2 for stereo data. 6 for 5.1ch data. Must be aligned with CODEC configuration.

Initialize the internal variables and allocate the buffer based on the given parameters.

The `channel_length` peripheral specify the number of the data in one channel. Where channel is the independent audio data stream. For example, a stereo data has 2 channel named left and right.

### 13.3.3 Member Function Documentation

#### 13.3.3.1 `void murasaki::AudioStrategy::TransmitAndReceive ( float ** tx_channels, float ** rx_channels )`

Multi channel audio transmission/receiving.

##### Parameters

<i>tx_channels</i>	Array of pointers. The number of the array element have to be same with the number of channel. Each pointer points the TX channel buffers.
<i>rx_channels</i>	Array of pointers. The number of the array element have to be same with the number of channel. Each pointer points the RX channel buffers.

Blocking and synchronous API. Given `tx_channels` buffers are scaled and copied to the DMA buffer. Inside this member function, wait for the complete of the RX data transfer by waiting for the `ReceiveCallback()`. And then, scale the data in DMA buffer and copy to `rx_channels` buffers.

```
#define NUM_CH 8
#define CH_LEN 48

float * tx_channels_array[NUM_CH];
float * rx_channels_array[NUM_CH];

tx_channles_array[0] = new float[CH_LEN];
tx_channles_array[1] = new float[CH_LEN];
tx_channles_array[2] = new float[CH_LEN];
...
tx_channles_array[NUM_CH-1] = new float[CH_LEN];

rx_channles_array[0] = new float[CH_LEN];
rx_channles_array[1] = new float[CH_LEN];
rx_channles_array[2] = new float[CH_LEN];
...
rx_channles_array[NUM_CH-1] = new float[CH_LEN];

while(1)
{
    murasaki::platform.audio->TransmitAndReceive(
        tx_channels_array,
        rx_channels_array );

    // process RX data in rx_channels_array
    ...
    // prepare TX data into rx_channlels_array.
    ...
}
```

#### 13.3.3.2 `void murasaki::AudioStrategy::TransmitAndReceive ( float * tx_left, float * tx_right, float * rx_left, float * rx_right )`

Stereo audio transmission/receiving.

##### Parameters

<i>tx_left</i>	Pointer to the left channel TX buffer
<i>tx_right</i>	Pointer to the right channel TX buffer
<i>rx_left</i>	Pointer to the left channel RX buffer
<i>rx_right</i>	Pointer to the right channel RX buffer



Blocking and synchronous API. Given tx\_channels buffers are scaled and copied to the DMA buffer. Inside this member function, wait for the complete of the RX data transfer by waiting for the ReceiveCallback(). And then, scale the data in DMA buffer and copy to rx\_channels buffers.

```
#define CH_LEN 48

float tx_left_ch_buf[NUM_CH];
float tx_right_ch_buf[NUM_CH];
float rx_left_ch_buf[NUM_CH];
float rx_right_ch_buf[NUM_CH];

while(1)
{
    murasaki::platform.audio->TransmitAndReceive(
        tx_left_ch_buf,
        tx_right_ch_buf,
        rx_left_ch_buf,
        rx_right_ch_buf );

    // process RX data in rx_left_ch_buf and rx_right_ch_buf
    ...
    // prepare TX data into tx_left_ch_buf and tx_right_ch_buf
    ...
}
```

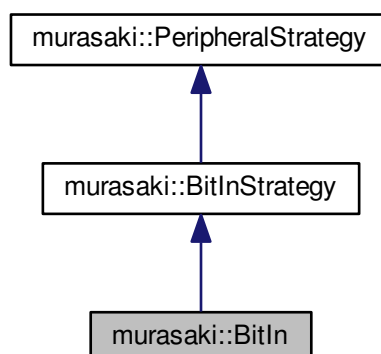
The documentation for this class was generated from the following file:

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/[audiostrategy.hpp](#)

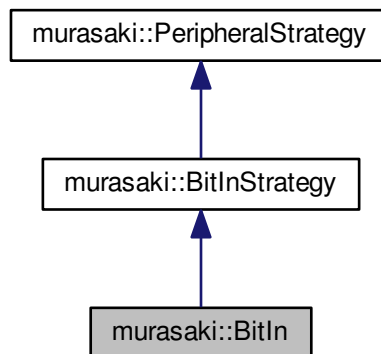
## 13.4 murasaki::BitIn Class Reference

```
#include <bitin.hpp>
```

Inheritance diagram for murasaki::BitIn:



Collaboration diagram for `murasaki::BitIn`:



## Public Member Functions

- `BitIn` (`GPIO_TypeDef *port`, `uint16_t pin`)
- virtual unsigned int `Get` (void)
- virtual void \* `GetPeripheralHandle` ()

### 13.4.1 Detailed Description

The `BitIn` class is the wrapper of the GPIO controller. To use the `BitIn` class, make an instance with `GPIO_TypeDef * type` pointer. For example, to create an instance for a switch peripheral:

```
my_switc = new murasaki::BitIn(sw_port, sw_pin);
```

Where `sw_port` and `sw_pin` are the macro generated by CubeMX for GPIO pin. the GPIO peripheral have to be configured to be right direction.

### 13.4.2 Constructor & Destructor Documentation

#### 13.4.2.1 `murasaki::BitIn::BitIn ( GPIO_TypeDef * port, uint16_t pin )`

Constructor.

#### Parameters

<i>port</i>	Pinter to the port strict.
<i>pin</i>	Number of the pin to input.

### 13.4.3 Member Function Documentation

#### 13.4.3.1 unsigned int murasaki::BitIn::Get ( void ) [virtual]

Get a status of the output pin.

##### Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" output state, respectively.

Implements [murasaki::BitInStrategy](#).

#### 13.4.3.2 void \* murasaki::BitIn::GetPeripheralHandle ( ) [virtual]

pass the raw peripheral handler

##### Returns

pointer to the [GPIO\\_type](#) variable hidden in a class.

Implements [murasaki::PeripheralStrategy](#).

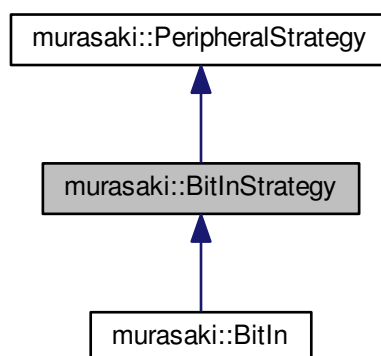
The documentation for this class was generated from the following files:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/bitin.hpp](#)
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Src/bitin.cpp](#)

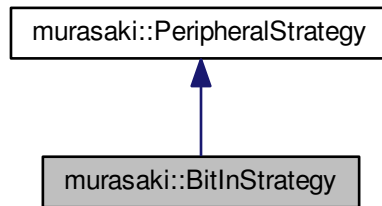
## 13.5 murasaki::BitInStrategy Class Reference

```
#include <bitinstrategy.hpp>
```

Inheritance diagram for murasaki::BitInStrategy:



Collaboration diagram for `murasaki::BitInStrategy`:



## Public Member Functions

- virtual unsigned int [Get](#) (void)=0

### 13.5.1 Detailed Description

A prototype of the general purpose bit input class

### 13.5.2 Member Function Documentation

13.5.2.1 virtual unsigned int `murasaki::BitInStrategy::Get ( void )` [pure virtual]

Get a status of the input pin.

#### Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" input state, respectively.

Implemented in [murasaki::BitIn](#).

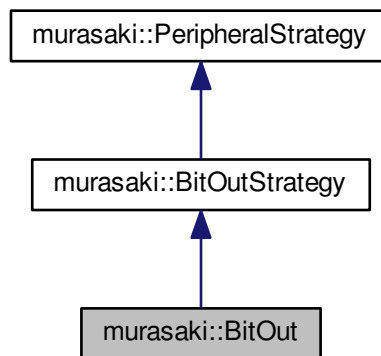
The documentation for this class was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/bitinstrategy.hpp](#)

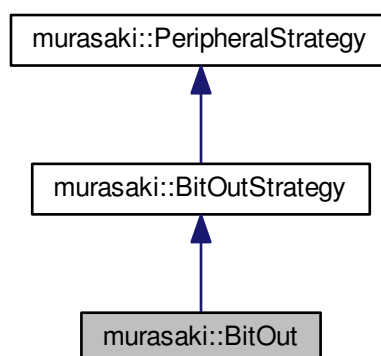
## 13.6 murasaki::BitOut Class Reference

```
#include <bitout.hpp>
```

Inheritance diagram for murasaki::BitOut:



Collaboration diagram for murasaki::BitOut:



### Public Member Functions

- `BitOut` (GPIO\_TypeDef \*port, uint16\_t pin)
- virtual void `Set` (unsigned int state=1)
- virtual unsigned int `Get` (void)
- virtual void \* `GetPeripheralHandle` ()

### 13.6.1 Detailed Description

The [BitOut](#) class is the wrapper of the GPIO controller. To use the [BitOut](#) class, make an instance with `GPIO_TypeDef *` type pointer. For example, to create an instance for the a peripheral:

```
my_LED = new murasaki::BitOut(LED_port, LED_pin);
```

Where `LED_port` and `LED_pin` are the macro generated by CubeMX for GPIO pin. the GPIO peripheral have to be configured to be right direction.

### 13.6.2 Constructor & Destructor Documentation

#### 13.6.2.1 `murasaki::BitOut::BitOut ( GPIO_TypeDef * port, uint16_t pin )`

Constructor.

Parameters

<i>port</i>	Pinter to the port strict.
<i>pin</i>	Number of the pin to output.

### 13.6.3 Member Function Documentation

#### 13.6.3.1 `unsigned int murasaki::BitOut::Get ( void ) [virtual]`

Get a status of the output pin.

Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" output state, respectively.

Implements [murasaki::BitOutStrategy](#).

#### 13.6.3.2 `void * murasaki::BitOut::GetPeripheralHandle ( ) [virtual]`

pass the raw peripheral handler

Returns

pointer to the [GPIO\\_type](#) variable hidden in a class.

Implements [murasaki::PeripheralStrategy](#).

#### 13.6.3.3 `void murasaki::BitOut::Set ( unsigned int state = 1 ) [virtual]`

Set a status of the output pin.

## Parameters

<i>state</i>	Set "H" if the value is none zero, vice versa.
--------------	--

Implements [murasaki::BitOutStrategy](#).

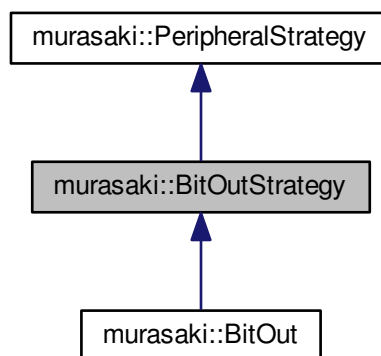
The documentation for this class was generated from the following files:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/bitout.hpp](#)
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Src/bitout.cpp](#)

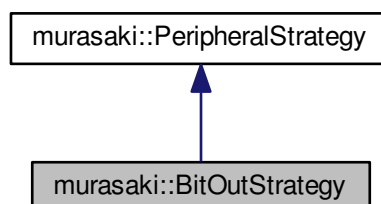
## 13.7 murasaki::BitOutStrategy Class Reference

```
#include <bitoutstrategy.hpp>
```

Inheritance diagram for murasaki::BitOutStrategy:



Collaboration diagram for murasaki::BitOutStrategy:



## Public Member Functions

- virtual void [Set](#) (unsigned int state=1)=0
- virtual unsigned int [Get](#) (void)=0

### 13.7.1 Detailed Description

A prototype of the general purpose bit out class

### 13.7.2 Member Function Documentation

#### 13.7.2.1 virtual unsigned int `murasaki::BitOutStrategy::Get ( void )` [pure virtual]

Get a status of the output pin.

##### Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" output state, respectively.

Implemented in [murasaki::BitOut](#).

#### 13.7.2.2 virtual void `murasaki::BitOutStrategy::Set ( unsigned int state = 1 )` [pure virtual]

Set a status of the output pin.

##### Parameters

<code>state</code>	Set "H" if the value is none zero, vice versa.
--------------------	--

Implemented in [murasaki::BitOut](#).

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

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/bitoutstrategy.hpp](/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/bitoutstrategy.hpp)

## 13.8 `murasaki::CriticalSection` Class Reference

```
#include <criticalsection.hpp>
```

## Public Member Functions

- void [Enter](#) ()
- void [Leave](#) ()



### 13.8.1 Detailed Description

The critical section prevent other task to preempt that critical section. So, a task can modify the shared variable safely inside critical section.

This class provide a critical section for the task context only. This critical section is not protected from the ISR.

The critical section have to start by [CriticalSection::Enter\(\)](#) and quit by [CriticalSection::Leave\(\)](#).

### 13.8.2 Member Function Documentation

#### 13.8.2.1 void murasaki::CriticalSection::Enter ( )

Entering critical section.

Entering critical section in task context. No other task can preemptive the task inside critical section.

#### 13.8.2.2 void murasaki::CriticalSection::Leave ( )

Leaving critical section.

All critical section started by [CriticalSection::Enter\(\)](#) have to be quit by this member function.

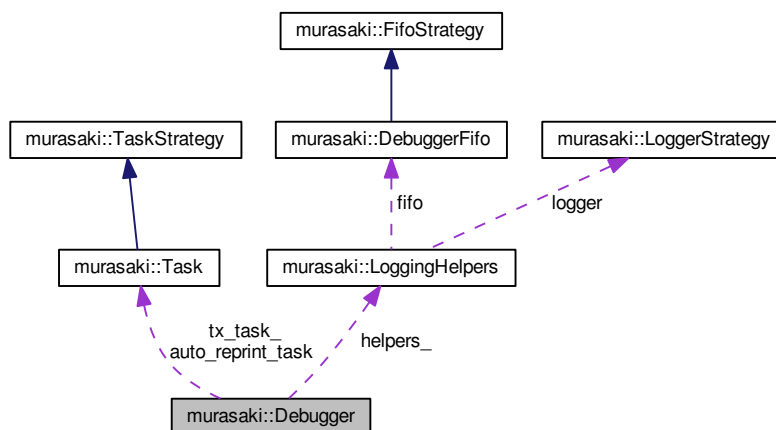
The documentation for this class was generated from the following files:

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/criticalsection.hpp
- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Src/criticalsection.cpp

## 13.9 murasaki::Debugger Class Reference

```
#include <debugger.hpp>
```

Collaboration diagram for murasaki::Debugger:



## Public Member Functions

- [Debugger](#) ([LoggerStrategy](#) \*logger)
- void [Printf](#) (const char \*fmt,...)
- char [GetchFromTask](#) ()
- void [RePrint](#) ()
- void [AutoRePrint](#) ()

## Protected Attributes

- char [line\\_](#) [[PLATFORM\\_CONFIG\\_DEBUG\\_LINE\\_SIZE](#)]
- [murasaki::SyslogSeverity](#) [severity\\_](#)
- [uint32\\_t](#) [facility\\_mask\\_](#)

### 13.9.1 Detailed Description

Wrapper class to help the printf debug. The printf() method can be called from both task context and ISR context.

There are several configurable parameters of this class:

- [PLATFORM\\_CONFIG\\_DEBUG\\_BUFFER\\_SIZE](#)
- [PLATFORM\\_CONFIG\\_DEBUG\\_LINE\\_SIZE](#)
- [PLATFORM\\_CONFIG\\_DEBUG\\_TASK\\_STACK\\_SIZE](#)
- [PLATFORM\\_CONFIG\\_DEBUG\\_TASK\\_PRIORITY](#)
- [PLATFORM\\_CONFIG\\_DEBUG\\_SERIAL\\_TIMEOUT](#)

See [Application Specific Platform](#) as example this class.

### 13.9.2 Constructor & Destructor Documentation

#### 13.9.2.1 `murasaki::Debugger::Debugger ( LoggerStrategy * logger )`

Constructor. Create internal variable.

##### Parameters

<i>logger</i>	The pointer to the <a href="#">LoggerStrategy</a> wrapper class variable.
---------------	---

### 13.9.3 Member Function Documentation

#### 13.9.3.1 `void murasaki::Debugger::AutoRePrint ( )`

Print history automatically.

Once this member function is called, internally new task is created. This new task watches input by [GetchFromTask\(\)](#) and for each input char is received, trigger the [RePrint\(\)](#).

This auto reprint function is exclusive and irreversible. Once auto reprint is triggered, there is no way to stop the auto reprint. The second call for the AutoHistory may be ignored

This member function have to be called from task context.

#### 13.9.3.2 char murasaki::Debugger::GetchFromTask ( )

Receive one character from serial port.

##### Returns

Received character.

A blocking function which returns received character. The receive is done on the UART which is passed to the constructor.

This is thread safe and task context dedicated function. Never call from ISR.

Becareful, this is blocking while the Debug::Printf() non-blocking.

#### 13.9.3.3 void murasaki::Debugger::Printf ( const char \* *fmt*, ... )

Debug output function.

##### Parameters

<i>fmt</i>	Format string
...	optional parameters

The printf() compatible method. This method can be called from both task context and ISR context. This method internally calls sprintf() variant. So, the parameter processing is fully compatible with with printf().

The formatted string is stored in the internal circular buffer. And data inside buffer is transmitted through the uart which is passed by constructor. If the buffer is overflowed, this method streos as possible, and discard the rest of string. That mean, this method is not blocking.

This member function is non-blocking, thread safe and re-entrant.

Be careful, this is non-blocking while the Debug::getchFromTask() is blocking.

At 2018/Jan/14 measurement, task stack was consumed 49bytes.

#### 13.9.3.4 void murasaki::Debugger::RePrint ( )

Print the old data again.

Must call from task context. For each time this member function is called, old data in the buffer is re-sent again.

The data to be re-setn is the one in the data in side circular buffer. Then, the resent size is same as [PLATFORM\\_CONFIG\\_DEBUG\\_BUFFER\\_SIZE](#) .

### 13.9.4 Member Data Documentation

#### 13.9.4.1 `uint32_t` `murasaki::Debugger::facility_mask_` `[protected]`

Syslog facility filter mask.

If certain bit is "1", the corresponding Syslog facility is allowed to output. By default the value is 0xFFFF ( equivalent to SyslogAllowAllFacilities(0xFFFFFFFF) )

#### 13.9.4.2 `char` `murasaki::Debugger::line_[PLATFORM_CONFIG_DEBUG_LINE_SIZE]` `[protected]`

as receiver for the `snprintf()`

This variable can be local variable of the `printf()` member function. In this case, the implementation of the `printf()` is much easier. In the other hand, each task must have enough depth on its task stack.

Probably, having bigger task for each task doesn't pay, and it may cause stack overflow bug at the debug or assertion. This is not preferable.

#### 13.9.4.3 `murasaki::SyslogSeverity` `murasaki::Debugger::severity_` `[protected]`

Syslog severity threshold.

All severity level lower than this value will be ignored by `Syslog()` function. Note that `murasaki::kseEmergency` is the highest and `murasaki::kseDebug` is the lowest severity.

By default, the severity level threshold is `murasaki::kseError`. That means, the weaker severity than `kseError` is ignored.

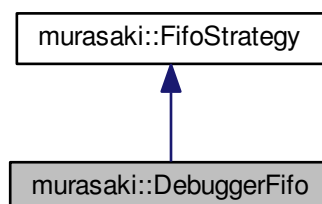
The documentation for this class was generated from the following files:

- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/debugger.hpp`
- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Src/debugger.cpp`

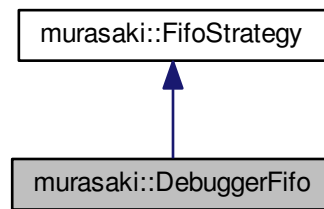
## 13.10 `murasaki::DebuggerFifo` Class Reference

```
#include <debuggerfifo.hpp>
```

Inheritance diagram for `murasaki::DebuggerFifo`:



Collaboration diagram for murasaki::DebuggerFifo:



## Public Member Functions

- `DebuggerFifo` (unsigned int *buffer\_size*)
- virtual unsigned int `Get` (uint8\_t *data*[], unsigned int *size*)
- virtual void `SetPostMortem` ()

### 13.10.1 Detailed Description

Non blocking , thread safe FIFO

The Put member function returns with "copied" data count. If the internal buffer is full, it returns without copy data. This is thread safe and ISR/Task bi-modal.

The Get member function returns with "copied" data count and data. If the internal buffer is empty, it returns without copy data.

### 13.10.2 Constructor & Destructor Documentation

#### 13.10.2.1 murasaki::DebuggerFifo::DebuggerFifo ( unsigned int *buffer\_size* )

Create an internal buffer.

##### Parameters

<i>buffer_size</i>	Size of the internal buffer to be allocated [byte]
--------------------	--

Allocate the internal buffer with given *buffer\_size*. The buffer contents is initialized by blank.

### 13.10.3 Member Function Documentation

#### 13.10.3.1 unsigned int murasaki::DebuggerFifo::Get ( uint8\_t *data*[], unsigned int *size* ) [virtual]

Get the data from the internal buffer. This is thread safe function. Do not call from ISR.

**Parameters**

<i>data</i>	Data buffer to receive from the internal buffer
<i>size</i>	Size of the data parameter.

**Returns**

The count of copied data. 0, if the internal buffer is empty

Reimplemented from [murasaki::FifoStrategy](#).

**13.10.3.2 void murasaki::DebuggerFifo::SetPostMortem ( ) [virtual]**

Transit to the post mortem mode.

In this mode, FIFO doesn't sync between the put and get method. Actually, this mode assumes nobody send message by [Put\(\)](#)

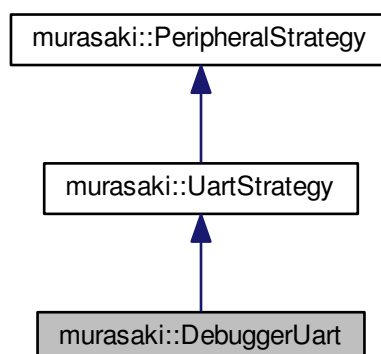
The documentation for this class was generated from the following files:

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/debuggerfifo.hpp
- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Src/debuggerfifo.cpp

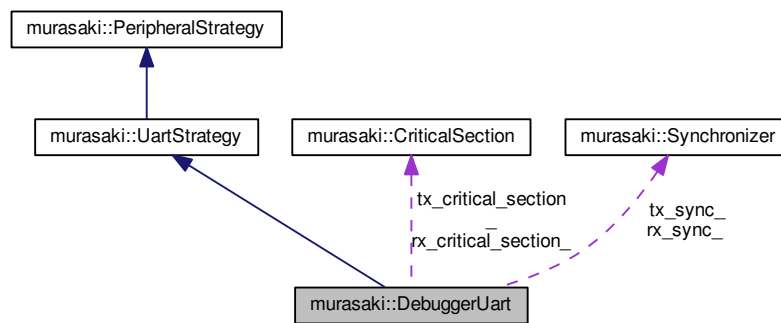
**13.11 murasaki::DebuggerUart Class Reference**

```
#include <debuggeruart.hpp>
```

Inheritance diagram for murasaki::DebuggerUart:



Collaboration diagram for murasaki::DebuggerUart:



## Public Member Functions

- [DebuggerUart](#) (UART\_HandleTypeDef \*uart)
- virtual void [SetHardwareFlowControl](#) ([UartHardwareFlowControl](#) control)
- virtual void [SetSpeed](#) (unsigned int baud\_rate)
- virtual [murasaki::UartStatus](#) [Transmit](#) (const uint8\_t \*data, unsigned int size, [WaitMilliseconds](#) timeout\_ms)
- virtual [murasaki::UartStatus](#) [Receive](#) (uint8\_t \*data, unsigned int count, unsigned int \*transferred\_count, [UartTimeout](#) uart\_timeout, [WaitMilliseconds](#) timeout\_ms)
- virtual bool [TransmitCompleteCallback](#) (void \*const ptr)
- virtual bool [ReceiveCompleteCallback](#) (void \*const ptr)
- virtual bool [HandleError](#) (void \*const ptr)

### 13.11.1 Detailed Description

The [Uart](#) class is the wrapper of the UART controller. To use the [DebuggerUart](#) class, make an instance with UART\_HandleTypeDef \* type pointer. For example, to create an instance for the UART3 peripheral :

```
my_uart3 = new murasaki::DebuggerUart(&huart3);
```

Where huart3 is the handle generated by CubeMX for UART3 peripheral. To use this class, the UART peripheral have to be configured to use the DMA functionality. The baud rate, length and flow control should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart)
{
    my_uart3->TransmitCompleteCallback(huart);
}
```

Where HAL\_UART\_TxCpltCallback is a predefined name of the UART interrupt handler. This is invoked by system whenever a DMA baed UART transmission is complete. Becuase the default function is weakly bound, above definition will overwrite the default one.

Note that above callback is invoked for any UARTn where n is 1, 2, 3... To avoid the confusion, [Uart::Transmit↔CompleteCallback\(\)](#) method chckes whether given parameter matches with its UART\_HandleTypeDef \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

As same as Tx, RX needs [HAL\\_UART\\_TxCpltCallback\(\)](#).

Once the instance and callbacks are correctly prepared, we can use the Tx/Rx member function.

The [Uart::Transmit\(\)](#) member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

The [Uart::Receive\(\)](#) member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

## 13.11.2 Constructor & Destructor Documentation

### 13.11.2.1 `murasaki::DebuggerUart::DebuggerUart ( UART_HandleTypeDef * uart )`

Constructor.

#### Parameters

<i>uart</i>	Pointer to a UART control struct. This device have to be configured to use DMA and interrupt for both Tx and Rx.
-------------	--

Store the given uart pointer into the internal variable. This pointer is passed to the STM32Cube HAL UART functions when needed.

## 13.11.3 Member Function Documentation

### 13.11.3.1 `bool murasaki::DebuggerUart::HandleError ( void *const ptr ) [virtual]`

Error handling.

#### Parameters

<i>ptr</i>	Pointer to UART_HandleTypeDef struct.
------------	---------------------------------------

#### Returns

true: ptr matches with UART device and handle the error. false : doesn't match.

A handle to print out the error message.



Checks whether handle has error and if there is, print appropriate error. Then return.

Implements [murasaki::UartStrategy](#).

**13.11.3.2** `murasaki::UartStatus murasaki::DebuggerUart::Receive ( uint8_t * data, unsigned int count, unsigned int * transferred_count, UartTimeout uart_timeout, WaitMilliseconds timeout_ms )` [virtual]

Receive raw data through an UART by blocking mode.

#### Parameters

<i>data</i>	Data buffer to place the received data..
<i>count</i>	The count of the data ( byte ) to be transfered. Must be smaller than 65536
<i>transferred_count</i>	This parameter is ignored.
<i>uart_timeout</i>	This parameter is ignored
<i>timeout_ms</i>	Time out limit by milliseconds.

#### Returns

Always returns OK

Receive to given data buffer through an UART device.

The receiving mode is blocking. That means, function returns when specified number of data has been received, except timeout. Passing [murasaki::kwmsIndefinitely](#) to the parameter *timeout\_ms* orders not to return until complete receiving. Other value of *timeout\_ms* parameter specifies the time out by millisecond. If time out happen, function returns false. If not happen, it returns true.

This function is exclusive. Internally this function is guarded by mutex. Then this function is thread safe. This function is forbidden to call from ISR.

Implements [murasaki::UartStrategy](#).

**13.11.3.3** `bool murasaki::DebuggerUart::ReceiveCompleteCallback ( void *const ptr )` [virtual]

Call back for entire block transfer completion.

#### Parameters

<i>ptr</i>	Pointer to UART_HandleTypeDef struct.
------------	---------------------------------------

#### Returns

true: *ptr* matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based receiving. The context have to be interrupt.

This member function checks whether the given *ptr* parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from [HAL\\_UART\\_RxCpltCallback\(\)](#). See STM32F7 HAL manual for detail

Implements [murasaki::UartStrategy](#).

**13.11.3.4** `void murasaki::DebuggerUart::SetHardwareFlowControl ( UartHardwareFlowControl control ) [virtual]`

Set the behavior of the hardware flow control.

#### Parameters

<i>control</i>	The control mode.
----------------	-------------------

Before calling this method, all transmission and receive activities have to be finished. This is responsibility of the programmer.

Note this method is NOT re-entrant. In other word, this member function can be called from both task and interrupt context.

Reimplemented from [murasaki::UartStrategy](#).

**13.11.3.5** `void murasaki::DebuggerUart::SetSpeed ( unsigned int baud_rate ) [virtual]`

Set the BAUD rate.

#### Parameters

<i>baud_rate</i>	BAUD rate ( 110, 300,... 57600,... )
------------------	--------------------------------------

Before calling this method, all transmission and receive activities have to be finished. This is responsibility of the programmer.

Note this method is NOT re-entrant. In other word, this member function can be called from both task and interrupt context.

Reimplemented from [murasaki::UartStrategy](#).

**13.11.3.6** `murasaki::UartStatus murasaki::DebuggerUart::Transmit ( const uint8_t* data, unsigned int size, WaitMilliseconds timeout_ms ) [virtual]`

Transmit raw data through an UART by blocking mode.

#### Parameters

<i>data</i>	Data buffer to be transmitted.
<i>size</i>	The count of the data ( byte ) to be transferred. Must be smaller than 65536
<i>timeout_ms</i>	Time out limit by milliseconds.

**Returns**

Always returns OK

Transmit given data buffer through an UART device.

The transmission mode is blocking. That means, function returns when all data has been transmitted, except timeout. Passing [murasaki::kwmsIndefinitely](#) to the parameter `timeout_ms` orders not to return until complete transmission. Other value of `timeout_ms` parameter specifies the time out by millisecond. If time out happen, function returns false. If not happen, it returns true.

This function is exclusive. Internally the function is guarded by mutex. Then this function is thread safe. This function is forbidden to call from ISR.

Implements [murasaki::UartStrategy](#).

**13.11.3.7 bool murasaki::DebuggerUart::TransmitCompleteCallback ( void \*const ptr ) [virtual]**

Call back for entire block transfer completion.

**Parameters**

<i>ptr</i>	Pointer to UART_HandleTypeDef struct.
------------	---------------------------------------

**Returns**

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based transmission. The context have to be interrupt.

This member function checks whether the given ptr parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from [HAL\\_UART\\_TxCpltCallback\(\)](#). See STM32F7 HAL manual for detail

Implements [murasaki::UartStrategy](#).

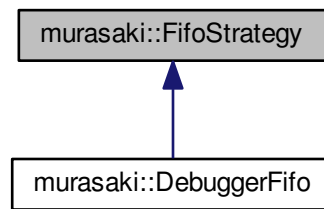
The documentation for this class was generated from the following files:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/debuggeruart.hpp](#)
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Src/debuggeruart.cpp](#)

**13.12 murasaki::FifoStrategy Class Reference**

```
#include <fifostrategy.hpp>
```

Inheritance diagram for `murasaki::FifoStrategy`:



## Public Member Functions

- [FifoStrategy](#) (unsigned int `buffer_size`)
- virtual unsigned int [Put](#) (uint8\_t const data[], unsigned int size)
- virtual unsigned int [Get](#) (uint8\_t data[], unsigned int size)

### 13.12.1 Detailed Description

Foundemental FIFO. No blocking , not thread safe.

The Put member function returns with "copied" data count. If the internal buffer is full, it returns without copy data.

The Get member funciton returns with "copied" data count and data. If the internal buffer is empty, it returns without copy data.

### 13.12.2 Constructor & Destructor Documentation

#### 13.12.2.1 `murasaki::FifoStrategy::FifoStrategy ( unsigned int buffer_size )`

Create an internal buffer.

##### Parameters

<i>buffer_size</i>	Size of the internal buffer to be allocated [byte]
--------------------	--

Allocate the internal buffer with given `buffer_size`. The contents is not initialized.

### 13.12.3 Member Function Documentation

#### 13.12.3.1 `unsigned int murasaki::FifoStrategy::Get ( uint8_t data[], unsigned int size )` [virtual]

Get the data from the internal buffer.

**Parameters**

<i>data</i>	Data buffer to receive from the internal buffer
<i>size</i>	Size of the data parameter.

**Returns**

The count of copied data. 0, if the internal buffer is empty

Reimplemented in [murasaki::DebuggerFifo](#).

**13.12.3.2** unsigned int murasaki::FifoStrategy::Put ( uint8\_t const *data*[], unsigned int *size* ) [virtual]

Put the data into the internal buffer.

**Parameters**

<i>data</i>	Data to be copied to the internal buffer
<i>size</i>	Data count to be copied

**Returns**

The count of copied data. 0, if the internal buffer is full.

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

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/fifostrategy.hpp](#)
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Src/fifostrategy.cpp](#)

## 13.13 murasaki::GPIO\_type Struct Reference

```
#include <bitout.hpp>
```

### 13.13.1 Detailed Description

This struct is used in the [BitIn](#) class and [BitOut](#) class. These classes returns a pointer to the variable of this type, as return value of the GetPeripheralHandle() member function.

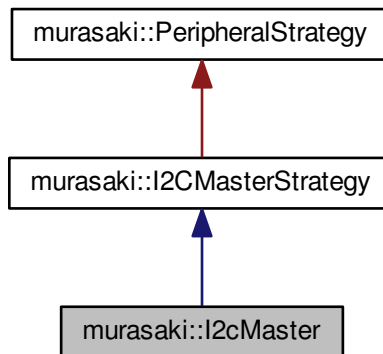
The documentation for this struct was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/bitout.hpp](#)

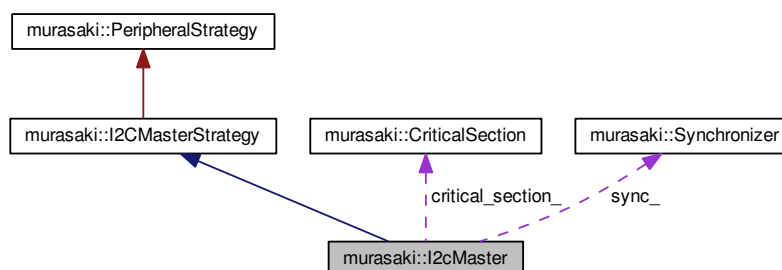
## 13.14 murasaki::I2cMaster Class Reference

```
#include <i2cmaster.hpp>
```

Inheritance diagram for murasaki::I2cMaster:



Collaboration diagram for murasaki::I2cMaster:



### Public Member Functions

- [I2cMaster](#) (`I2C_HandleTypeDef *i2c_handle`)
- virtual [murasaki::I2cStatus Transmit](#) (`unsigned int addr, const uint8_t *tx_data, unsigned int tx_size, unsigned int *transferred_count, WaitMilliSeconds timeout_ms`)
- virtual [murasaki::I2cStatus Receive](#) (`unsigned int addr, uint8_t *rx_data, unsigned int rx_size, unsigned int *transferred_count, WaitMilliSeconds timeout_ms`)
- virtual [murasaki::I2cStatus TransmitThenReceive](#) (`unsigned int addr, const uint8_t *tx_data, unsigned int tx_size, uint8_t *rx_data, unsigned int rx_size, unsigned int *tx_transferred_count, unsigned int *rx_transferred_count, WaitMilliSeconds timeout_ms`)
- virtual bool [TransmitCompleteCallback](#) (`void *ptr`)
- virtual bool [ReceiveCompleteCallback](#) (`void *ptr`)
- virtual bool [HandleError](#) (`void *ptr`)

### 13.14.1 Detailed Description

The [I2cMaster](#) class is the wrapper of the I2C controller. To use the [I2cMaster](#) class, make an instance with [I2C\\_HandleTypeDef \\* type pointer](#). For example, to create an instance for the I2C3 peripheral :

```
my_i2c3 = new murasaki::I2cMaster(&hi2c3);
```

Where hi2c3 is the handle generated by CubeMX for I2C3 peripheral. To use this class, the I2C peripheral have to be configured to use the interrupt functionality without DMA. The bitrate should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_I2C_TxCpltCallback(I2C_HandleTypeDef * hi2c)
{
    my_i2c3->TransmitCompleteCallback(hi2c);
}
```

Where HAL\_I2C\_TxCpltCallback is a predefined name of the I2C interrupt handler. This is invoked by system whenever a interrupt baed I2C transmission is complete. Becuase the default function is weakly bound, above definition will overwrite the default one.

Note that above callback is invoked for any I2Cn where n is 1, 2, 3... To avoid the confusion, [I2cMaster::TransmitCompleteCallback\(\)](#) method chckes whether given parameter matches with its [I2C\\_HandleTypeDef \\* pointer](#) ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

As same as Tx, RX needs HAL\_I2C\_TxCpltCallback().

Once the instance and callback are correctly prepared, we can use the Tx/Rx member function.

The [I2cMaster::Transmit\(\)](#) member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by [kwmsIndefinitely](#) which specifes never time out.

The [I2cMaster::Receive\(\)](#) member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by [kwmsIndefinitely](#) which species never time out.

The [I2cMaster::TransmitThenReceive\(\)](#) member function is blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by [kwmsIndefinitely](#) which species never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

Note : In case an time out occurs during transmit / receive, this implementation calls [HAL\\_I2C\\_MASTER\\_ABORT\\_IT\(\)](#). But it is unknown whether this is right thing to do. The HAL reference of the STM32F7 is not clear for this case. For example, it doesn't tell what programmer do to stop the transfer at the middle. And also, it doesn't tell what's happen if the [HAL\\_I2C\\_MASTER\\_ABORT\\_IT\(\)](#) is called.

According to the source code of the [HAL\\_I2C\\_MASTER\\_ABORT\\_IT\(\)](#), no interrupt will be raised by this API call.

### 13.14.2 Constructor & Destructor Documentation

#### 13.14.2.1 murasaki::I2cMaster::I2cMaster ( I2C\_HandleTypeDef \* i2c\_handle )

Constructor.

## Parameters

<i>i2c_handle</i>	Peripheral handle created by CubeMx
-------------------	-------------------------------------

## 13.14.3 Member Function Documentation

13.14.3.1 `bool murasaki::I2cMaster::HandleError ( void * ptr ) [virtual]`

Error handling.

## Parameters

<i>ptr</i>	Pointer to I2C_HandleTypeDef struct.
------------	--------------------------------------

## Returns

true: ptr matches with device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements [murasaki::I2CMasterStrategy](#).

13.14.3.2 `murasaki::I2cStatus murasaki::I2cMaster::Receive ( unsigned int addr, uint8_t * rx_data, unsigned int rx_size, unsigned int * transferred_count, WaitMilliseconds timeout_ms ) [virtual]`

Thread safe, blocking receiving over I2C.

## Parameters

<i>addr</i>	7bit address of the I2C device.
<i>rx_data</i>	Data array to transmit.
<i>rx_size</i>	Data counts[bytes] to transmit. Must be smaller than 65536
<i>transferred_count</i>	( Currently, Just ignored) the count of the bytes transferred during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

## Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code :

- [murasaki::ki2csOK](#) : All Receive completed.



- [murasaki::ki2csNak](#) : Receive terminated by NAK receiving.
- [murasaki::ki2csArbitrationLost](#) : Receive terminated by an arbitration error of the multi-master.
- [murasaki::ki2csBussError](#) : Receive terminated by bus error
- [murasaki::ki2csTimeOut](#) : Receive abort by timeout.
- other value : Unhandled error. I2C device are re-initialized.

Implements [murasaki::I2CMasterStrategy](#).

**13.14.3.3** `bool murasaki::I2cMaster::ReceiveCompleteCallback ( void * ptr ) [virtual]`

Call back to be called for entire block transfer is complete.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

#### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements [murasaki::I2CMasterStrategy](#).

**13.14.3.4** `murasaki::I2cStatus murasaki::I2cMaster::Transmit ( unsigned int addr, const uint8_t * tx_data, unsigned int tx_size, unsigned int * transferred_count, WaitMilliseconds timeout_ms ) [virtual]`

Thread safe, blocking transmission over I2C.

#### Parameters

<i>addr</i>	7bit address of the I2C device.
<i>tx_data</i>	Data array to transmit.
<i>tx_size</i>	Data counts[bytes] to transmit. Must be smaller than 65536
<i>transferred_count</i>	( Currently, Just ignored) the count of the bytes transfered during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

#### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code :

- [murasaki::ki2csOK](#) : All transmission completed.
- [murasaki::ki2csNak](#) : Transmission terminated by NAK receiving.
- [murasaki::ki2csArbitrationLost](#) : Transmission terminated by an arbitration error of the multi-master.
- [murasaki::ki2csBussError](#) : Transmission terminated by bus error
- [murasaki::ki2csTimeOut](#) : Transmission abort by timeout.
- other value : Unhandled error. I2C device are re-initialized.

Implements [murasaki::I2CMasterStrategy](#).

13.14.3.5 `bool murasaki::I2cMaster::TransmitCompleteCallback ( void * ptr ) [virtual]`

Call back to be called notify the transfer is complete.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

#### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements [murasaki::I2CMasterStrategy](#).

13.14.3.6 `murasaki::I2cStatus murasaki::I2cMaster::TransmitThenReceive ( unsigned int addr, const uint8_t * tx_data, unsigned int tx_size, uint8_t * rx_data, unsigned int rx_size, unsigned int * tx_transferred_count, unsigned int * rx_transferred_count, WaitMilliseconds timeout_ms ) [virtual]`

Thread safe, blocking transmission and then receiving over I2C.

#### Parameters

<i>addr</i>	7bit address of the I2C device.
<i>tx_data</i>	Data array to transmit.
<i>tx_size</i>	Data counts[bytes] to transmit. Must be smaller than 65536
<i>rx_data</i>	Data array to transmit.
<i>rx_size</i>	Data counts[bytes] to transmit. Must be smaller than 65536
<i>tx_transferred_count</i>	( Currently, Just ignored) the count of the bytes transmitted during the API execution.
<i>rx_transferred_count</i>	( Currently, Just ignored) the count of the bytes received during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

**Returns**

Result of the processing

First, this member function transmit the data, and the, by repeated start function, it receives data. The transmission device address and receiving device address is same.

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code :

- [murasaki::ki2csOK](#) : All transmission and receive completed.
- [murasaki::ki2csNak](#) : Transmission or receive terminated by NAK receiving.
- [murasaki::ki2csArbitrationLost](#) : Transmission or receive terminated by an arbitration error of the multi-master.
- [murasaki::ki2csBussError](#) : Transmission or receive terminated by bus error
- [murasaki::ki2csTimeOut](#) : Transmission or receive abort by timeout.
- other value : Unhandled error. I2C device are re-initialized.

Implements [murasaki::I2CMasterStrategy](#).

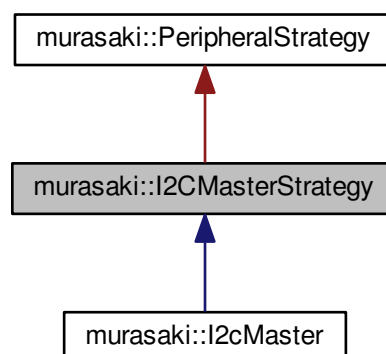
The documentation for this class was generated from the following files:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/i2cmaster.hpp](#)
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Src/i2cmaster.cpp](#)

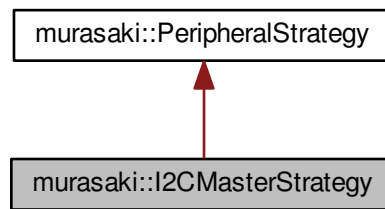
## 13.15 murasaki::I2CMasterStrategy Class Reference

```
#include <i2cmasterstrategy.hpp>
```

Inheritance diagram for murasaki::I2CMasterStrategy:



Collaboration diagram for `murasaki::I2CMasterStrategy`:



## Public Member Functions

- virtual `murasaki::I2cStatus Transmit` (unsigned int `addr`, const uint8\_t \*`tx_data`, unsigned int `tx_size`, unsigned int \*`transferred_count`=nullptr, `WaitMilliseconds` `timeout_ms`=`murasaki::kwmsIndefinitely`)=0
- virtual `murasaki::I2cStatus Receive` (unsigned int `addr`, uint8\_t \*`rx_data`, unsigned int `rx_size`, unsigned int \*`transferred_count`=nullptr, `WaitMilliseconds` `timeout_ms`=`murasaki::kwmsIndefinitely`)=0
- virtual `murasaki::I2cStatus TransmitThenReceive` (unsigned int `addr`, const uint8\_t \*`tx_data`, unsigned int `tx_size`, uint8\_t \*`rx_data`, unsigned int `rx_size`, unsigned int \*`tx_transferred_count`=nullptr, unsigned int \*`rx_transferred_count`=nullptr, `WaitMilliseconds` `timeout_ms`=`murasaki::kwmsIndefinitely`)=0
- virtual bool `TransmitCompleteCallback` (void \*`ptr`)=0
- virtual bool `ReceiveCompleteCallback` (void \*`ptr`)=0
- virtual bool `HandleError` (void \*`ptr`)=0

### 13.15.1 Detailed Description

A prototype of the I2C master peripheral.

This prototype assumes the derived class will transmit / receive data in the task context on RTOS. And these member functions should be blocking. That mean, until the transmit / receive terminates, both method doesn't return.

Two call back member functions are prepared to sync with the interrupt which tells the end of Transmit/Receive.

### 13.15.2 Member Function Documentation

13.15.2.1 `virtual bool murasaki::I2CMasterStrategy::HandleError ( void * ptr )` [pure virtual]

Handling error report of device.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a device control
------------	---

**Returns**

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in [murasaki::I2cMaster](#).

**13.15.2.2** `virtual murasaki::I2cStatus murasaki::I2CMasterStrategy::Receive ( unsigned int addr, uint8_t * rx_data, unsigned int rx_size, unsigned int * transferred_count = nullptr, WaitMilliseconds timeout_ms = murasaki::kwmsIndefinitely ) [pure virtual]`

Thread safe, blocking receiving over I2C.

**Parameters**

<i>addr</i>	7bit address of the I2C device.
<i>rx_data</i>	Data array to transmit.
<i>rx_size</i>	Data counts[bytes] to transmit.
<i>transferred_count</i>	the count of the bytes transfered during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

**Returns**

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in [murasaki::I2cMaster](#).

**13.15.2.3** `virtual bool murasaki::I2CMasterStrategy::ReceiveCompleteCallback ( void * ptr ) [pure virtual]`

Call back to be called for entire block transfer is complete.

**Parameters**

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

**Returns**

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in [murasaki::I2cMaster](#).

13.15.2.4 **virtual** **murasaki::I2cStatus** **murasaki::I2CMasterStrategy::Transmit** ( **unsigned int** *addr*, **const** **uint8\_t** \* *tx\_data*, **unsigned int** *tx\_size*, **unsigned int** \* *transferred\_count* = **nullptr**, **WaitMilliseconds** *timeout\_ms* = **murasaki::kwmsIndefinitely** ) [pure virtual]

Thread safe, blocking transmission over I2C.

#### Parameters

<i>addr</i>	7bit address of the I2C device.
<i>tx_data</i>	Data array to transmit.
<i>tx_size</i>	Data counts[bytes] to transmit.
<i>transferred_count</i>	the count of the bytes transfered during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

#### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in [murasaki::I2cMaster](#).

13.15.2.5 **virtual bool** **murasaki::I2CMasterStrategy::TransmitCompleteCallback** ( **void** \* *ptr* ) [pure virtual]

Call back to be called notify the transfer is complete.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

#### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in [murasaki::I2cMaster](#).

13.15.2.6 **virtual** **murasaki::I2cStatus** **murasaki::I2CMasterStrategy::TransmitThenReceive** ( **unsigned int** *addr*, **const** **uint8\_t** \* *tx\_data*, **unsigned int** *tx\_size*, **uint8\_t** \* *rx\_data*, **unsigned int** *rx\_size*, **unsigned int** \* *tx\_transferred\_count* = **nullptr**, **unsigned int** \* *rx\_transferred\_count* = **nullptr**, **WaitMilliseconds** *timeout\_ms* = **murasaki::kwmsIndefinitely** ) [pure virtual]

Thread safe, blocking transmission and then receiving over I2C.

## Parameters

<i>addr</i>	7bit address of the I2C device.
<i>tx_data</i>	Data array to transmit.
<i>tx_size</i>	Data counts[bytes] to transmit.
<i>rx_data</i>	Data array to transmit.
<i>rx_size</i>	Data counts[bytes] to transmit.
<i>tx_transferred_count</i>	the count of the bytes transmitted during the API execution.
<i>rx_transferred_count</i>	the count of the bytes received during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

## Returns

Result of the processing

First, this member function transmit the data, and the, by repeated start function, it receives data. The transmission device address and receiving device address is same.

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in [murasaki::I2cMaster](#).

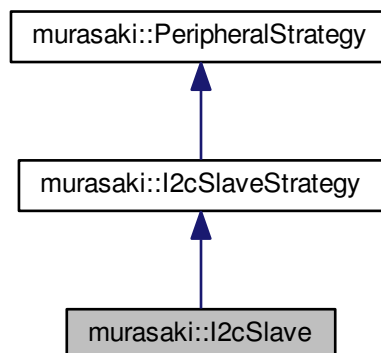
The documentation for this class was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/i2cmasterstrategy.hpp](#)

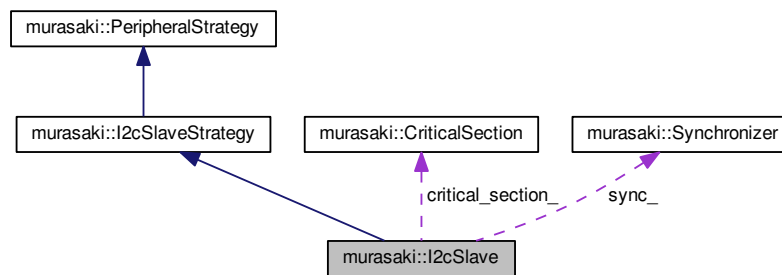
## 13.16 murasaki::I2cSlave Class Reference

```
#include <i2cslave.hpp>
```

Inheritance diagram for murasaki::I2cSlave:



Collaboration diagram for `murasaki::I2cSlave`:



## Public Member Functions

- virtual `murasaki::I2cStatus Transmit` (const uint8\_t \*tx\_data, unsigned int tx\_size, unsigned int \*transferred\_count, `WaitMilliseconds` timeout\_ms)
- virtual `murasaki::I2cStatus Receive` (uint8\_t \*rx\_data, unsigned int rx\_size, unsigned int \*transferred\_count, `WaitMilliseconds` timeout\_ms)
- virtual bool `TransmitCompleteCallback` (void \*ptr)
- virtual bool `ReceiveCompleteCallback` (void \*ptr)
- virtual bool `HandleError` (void \*ptr)

### 13.16.1 Detailed Description

The `I2cSlave` class is the wrapper of the I2C controller. To use the `I2cSlave` class, make an instance with `I2C_HandleTypeDef * type pointer`. For example, to create an instance for the I2C3 peripheral :

```
my_i2c3 = new murasaki::I2cSlave(&hi2c3);
```

Where `hi2c3` is the handle generated by CubeMX for I2C3 peripheral. To use this class, the I2C peripheral have to be configured to use the interrupt functionality without DMA. The bit rate and the peripheral address should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback. and error callback

```

void HAL_I2C_TxCpltCallback(I2C_HandleTypeDef * hi2c)
{
    if ( my_i2c3->TransmitCompleteCallback(hi2c) )
        return;
}

void HAL_I2C_ErrorCallback(I2C_HandleTypeDef * hi2c)
{
    if (my_i2c3->HandleError(hi2c) )
        return;
}
  
```



Where HAL\_I2C\_TxCpltCallback is a predefined name of the I2C interrupt handler. This is invoked by system whenever a interrupt baed I2C transmission is complete. Because the default function is weakly bound, above definition will override the default one.

Note that above callback are invoked for any I2Cn where n is 1, 2, 3... To avoid the confusion, [I2cMaster::TransmitCompleteCallback\(\)](#) method checks whether given parameter matches with its I2C\_HandleTypeDef \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process. In case of the successful match, it returns true.

As same as Tx, RX needs HAL\_I2C\_TxCpltCallback().

Once the instance and callbacks are correctly prepared, we can use the Tx/Rx member function.

The [I2cSlave::Transmit\(\)](#) member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

The [I2cSlave::Receive\(\)](#) member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

- Note : In case an time out occurs during transmit / receive, this implementation calls HAL\_I2C\_DeInit()/HAL\_I2C\_Init(). But it is unknown whether this is right thing to do. The HAL reference of the STM32F7 is not clear for this case. For example, it doesn't tell what programmer do to stop the transfer at the middle.

## 13.16.2 Member Function Documentation

### 13.16.2.1 bool murasaki::I2cSlave::HandleError ( void \* *ptr* ) [virtual]

Error handling.

#### Parameters

<i>ptr</i>	Pointer to I2C_HandleTypeDef struct.
------------	--------------------------------------

#### Returns

true: ptr matches with device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements [murasaki::I2cSlaveStrategy](#).

### 13.16.2.2 murasaki::I2cStatus murasaki::I2cSlave::Receive ( uint8\_t \* *rx\_data*, unsigned int *rx\_size*, unsigned int \* *transfered\_count*, WaitMilliseconds *timeout\_ms* ) [virtual]

Thread safe, blocking receiving over I2C.

## Parameters

<i>rx_data</i>	Data array to transmit.
<i>rx_size</i>	Data counts[bytes] to transmit. Must be smaller than 65536
<i>transferred_count</i>	( Currently, Just ignored) the count of the bytes transfered during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

## Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code :

- [murasaki::ki2csOK](#) : All Receive completed.
- [murasaki::ki2csNak](#) : Receive terminated by NAK receiving.
- [murasaki::ki2csArbitrationLost](#) : Receive terminated by an arbitration error of the multi-master.
- [murasaki::ki2csBussError](#) : Receive terminated by bus error
- [murasaki::ki2csTimeOut](#) : Receive abort by timeout.
- other value : Unhandled error. I2C device are re-initialized.

Implements [murasaki::I2cSlaveStrategy](#).

**13.16.2.3** `bool murasaki::I2cSlave::ReceiveCompleteCallback ( void * ptr ) [virtual]`

Call back to be called for entire block transfer is complete.

## Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

## Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements [murasaki::I2cSlaveStrategy](#).

**13.16.2.4** `murasaki::I2cStatus murasaki::I2cSlave::Transmit ( const uint8_t * tx_data, unsigned int tx_size, unsigned int * transferred_count, WaitMilliseconds timeout_ms ) [virtual]`

Thread safe, blocking transmission over I2C.

## Parameters

<i>tx_data</i>	Data array to transmit.
<i>tx_size</i>	Data counts[bytes] to transmit. Must be smaller than 65536
<i>transferred_count</i>	( Currently, Just ignored) the count of the bytes transfered during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

## Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code :

- [murasaki::ki2csOK](#) : All transmission completed.
- [murasaki::ki2csNak](#) : Transmission terminated by NAK receiving.
- [murasaki::ki2csArbitrationLost](#) : Transmission terminated by an arbitration error of the multi-master.
- [murasaki::ki2csBussError](#) : Transmission terminated by bus error
- [murasaki::ki2csTimeOut](#) : Transmission abort by timeout.
- other value : Unhandled error. I2C device are re-initialized.

Implements [murasaki::I2cSlaveStrategy](#).

**13.16.2.5** `bool murasaki::I2cSlave::TransmitCompleteCallback ( void * ptr ) [virtual]`

Call back to be called notify the transfer is complete.

## Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

## Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements [murasaki::I2cSlaveStrategy](#).

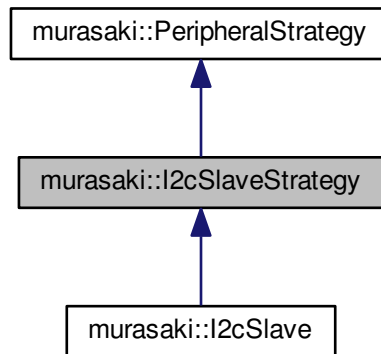
The documentation for this class was generated from the following files:

- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/i2cslave.hpp`
- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Src/i2cslave.cpp`

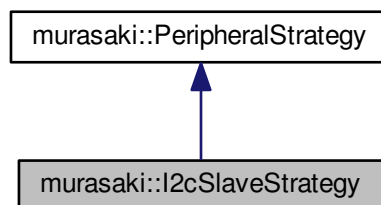
## 13.17 murasaki::I2cSlaveStrategy Class Reference

```
#include <i2cslavestrategy.hpp>
```

Inheritance diagram for murasaki::I2cSlaveStrategy:



Collaboration diagram for murasaki::I2cSlaveStrategy:



### Public Member Functions

- virtual [murasaki::I2cStatus Transmit](#) (const uint8\_t \*tx\_data, unsigned int tx\_size, unsigned int \*transferred\_↵\_count=nullptr, [murasaki::WaitMilliseconds](#) timeout\_ms=[murasaki::kwmsIndefinitely](#))=0
- virtual [murasaki::I2cStatus Receive](#) (uint8\_t \*rx\_data, unsigned int rx\_size, unsigned int \*transferred\_↵\_count=nullptr, [murasaki::WaitMilliseconds](#) timeout\_ms=[murasaki::kwmsIndefinitely](#))=0
- virtual bool [TransmitCompleteCallback](#) (void \*ptr)=0
- virtual bool [ReceiveCompleteCallback](#) (void \*ptr)=0
- virtual bool [HandleError](#) (void \*ptr)=0

### 13.17.1 Detailed Description

A prototype of the I2C slave peripheral.

This prototype assumes the derived class will transmit / receive data in the task context on RTOS. And these member functions should be blocking. That mean, until the transmit / receive terminates, both method doesn't return.

Two call back member functions are prepared to sync with the interrupt which tells the end of Transmit/Receive.

### 13.17.2 Member Function Documentation

#### 13.17.2.1 virtual bool murasaki::I2cSlaveStrategy::HandleError ( void \* *ptr* ) [pure virtual]

Handling error report of device.

##### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a device control
------------	---

##### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in [murasaki::I2cSlave](#).

#### 13.17.2.2 virtual murasaki::I2cStatus murasaki::I2cSlaveStrategy::Receive ( uint8\_t \* *rx\_data*, unsigned int *rx\_size*, unsigned int \* *transfered\_count* = nullptr, murasaki::WaitMilliseconds *timeout\_ms* = murasaki::kwmsIndefinitely ) [pure virtual]

Thread safe, blocking receiving over I2C.

##### Parameters

<i>rx_data</i>	Data array to transmit.
<i>rx_size</i>	Data counts[bytes] to transmit.
<i>transfered_count</i>	the count of the bytes transfered during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

##### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in [murasaki::I2cSlave](#).

### 13.17.2.3 `virtual bool murasaki::I2cSlaveStrategy::ReceiveCompleteCallback ( void * ptr ) [pure virtual]`

Call back to be called for entire block transfer is complete.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

#### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in [murasaki::I2cSlave](#).

### 13.17.2.4 `virtual murasaki::I2cStatus murasaki::I2cSlaveStrategy::Transmit ( const uint8_t * tx_data, unsigned int tx_size, unsigned int * transferred_count = nullptr, murasaki::WaitMilliseconds timeout_ms = murasaki::kwmsIndefinitely ) [pure virtual]`

Thread safe, blocking transmission over I2C.

#### Parameters

<i>tx_data</i>	Data array to transmit.
<i>tx_size</i>	Data counts[bytes] to transmit.
<i>transferred_count</i>	the count of the bytes transfered during the API execution.
<i>timeout_ms</i>	Time ou [mS]. By default, there is not timeout.

#### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in [murasaki::I2cSlave](#).

### 13.17.2.5 `virtual bool murasaki::I2cSlaveStrategy::TransmitCompleteCallback ( void * ptr ) [pure virtual]`

Call back to be called notify the transfer is complete.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a peripheral control
------------	---

**Returns**

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in [murasaki::I2cSlave](#).

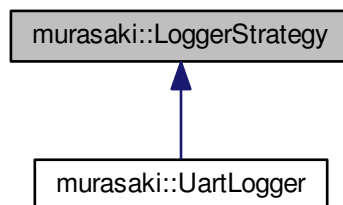
The documentation for this class was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/i2cslavestrategy.hpp](#)

## 13.18 murasaki::LoggerStrategy Class Reference

```
#include <loggerstrategy.hpp>
```

Inheritance diagram for murasaki::LoggerStrategy:

**Public Member Functions**

- virtual [~LoggerStrategy](#) ()
- virtual void [putMessage](#) (char message[ ], unsigned int size)=0
- virtual char [getCharacter](#) ()=0
- virtual void [DoPostMortem](#) (void \*debugger\_fifo)

### 13.18.1 Detailed Description

A generic class to serve a logging function. This class is designed to pass to the [murasaki::Debugger](#).

As a service class to Debug. This class's two member functions ( [putMessage\(\)](#) and [getCharacter\(\)](#) ) have to be able to run in the task context. Both member functions also have to be the blocking function.

## 13.18.2 Constructor & Destructor Documentation

13.18.2.1 `virtual murasaki::LoggerStrategy::~LoggerStrategy ( ) [inline],[virtual]`

Destructor.

Do nothing here. Declared to enforce the derived class's constructor as "virtual".

## 13.18.3 Member Function Documentation

13.18.3.1 `virtual void murasaki::LoggerStrategy::DoPostMortem ( void * debugger_fifo ) [inline],[virtual]`

Start post mortem process.

### Parameters

<i>debugger_fifo</i>	Pointer to the <a href="#">DebuggerFifo</a> class object. This is declared as void to avoid the include confusion. This member function read the data in given FIFO, and then do the auto history.
----------------------	--

By default this is not implemented. But in case user implments a method, it should call the `Debugger::SetPostMortem()` internally.

Reimplemented in [murasaki::UartLogger](#).

13.18.3.2 `virtual char murasaki::LoggerStrategy::getCharacter ( ) [pure virtual]`

Character input member function.

### Returns

A character from input is returned.

This function is considered as blocking. That mean, the function will wait for any user input forever.

Implemented in [murasaki::UartLogger](#).

13.18.3.3 `virtual void murasaki::LoggerStrategy::putMessage ( char message[], unsigned int size ) [pure virtual]`

Message output member function.

### Parameters

<i>message</i>	Non null terminated character array. This data is stored or output to the logger.
<i>size</i>	Byte length of the message parameter of the putMessage member function.

This function is considered as blcoking. That mean, it will not wayt until data is stored to the storage or output.



Implemented in [murasaki::UartLogger](#).

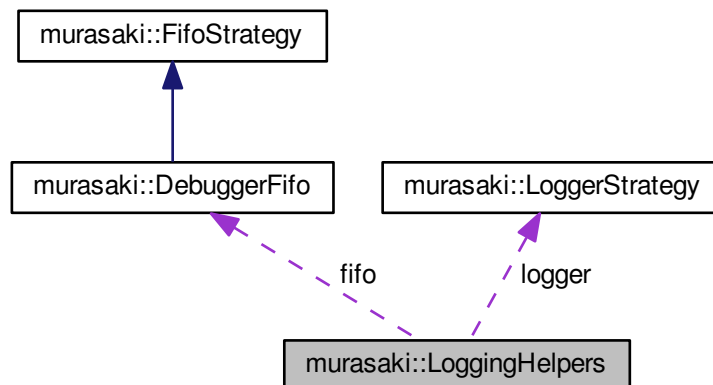
The documentation for this class was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/loggerstrategy.hpp](#)

## 13.19 murasaki::LoggingHelpers Struct Reference

```
#include <debuggerfifo.hpp>
```

Collaboration diagram for murasaki::LoggingHelpers:



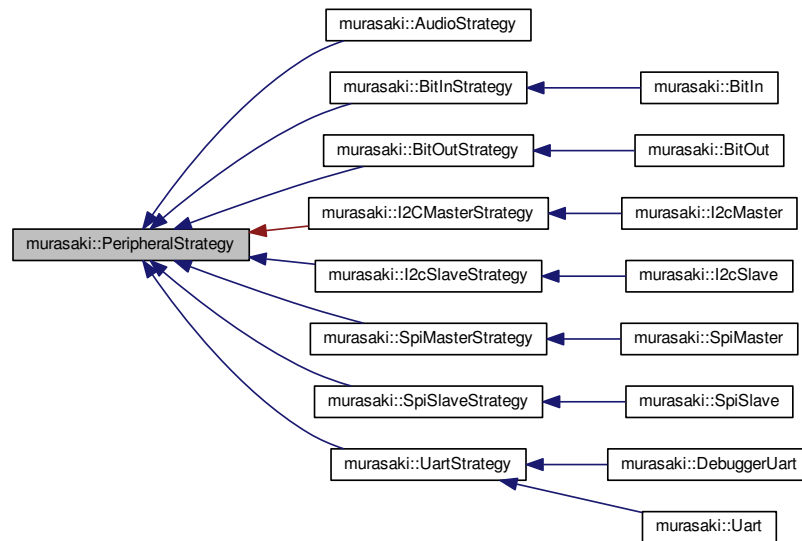
The documentation for this struct was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/debuggerfifo.hpp](#)

## 13.20 murasaki::PeripheralStrategy Class Reference

```
#include <peripheralstrategy.hpp>
```

Inheritance diagram for `murasaki::PeripheralStrategy`:



### 13.20.1 Detailed Description

This class provides the `GetPeripheralHandle()` member function as a common stub for the debugging logger. The loggers sometimes refers the raw peripheral to respond to the post mortem situation. By using class, programmer can pass the raw peripheral handler to loggers, while keep it hidden from the application.

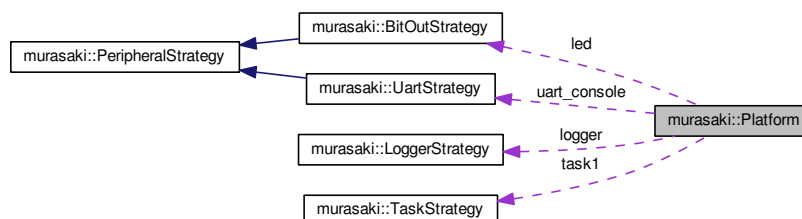
The documentation for this class was generated from the following file:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/peripheralstrategy.hpp](/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/peripheralstrategy.hpp)

## 13.21 murasaki::Platform Struct Reference

```
#include <platform_defs.hpp>
```

Collaboration diagram for `murasaki::Platform`:



### 13.21.1 Detailed Description

A collection of the peripheral / MPU control variable.

This is a custom struct. Programmer can change this struct as suitable to the hardware and software. But `debugger_` member variable have to be left untouched.

In the run time, the `debugger_` variable have to be initialized by appropriate [murasaki::Debugger](#) class instance.

See [murasaki::platform](#)

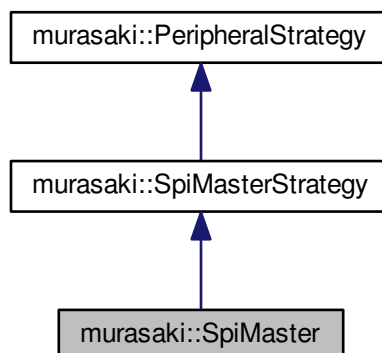
The documentation for this struct was generated from the following file:

- `/home/takemasa/workspace_st/nucleo-f446-sample/Inc/platform_defs.hpp`

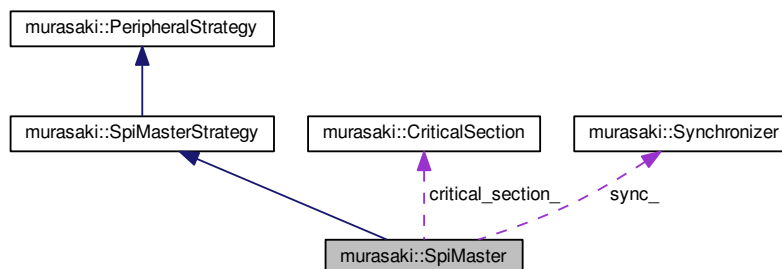
## 13.22 murasaki::SpiMaster Class Reference

```
#include <spimaster.hpp>
```

Inheritance diagram for `murasaki::SpiMaster`:



Collaboration diagram for `murasaki::SpiMaster`:



## Public Member Functions

- [SpiMaster](#) ([SPI\\_HandleTypeDef](#) \*spi\_handle)
- virtual [SpiStatus TransmitAndReceive](#) ([murasaki::SpiSlaveAdapterStrategy](#) \*spi\_spec, const [uint8\\_t](#) \*tx\_data, [uint8\\_t](#) \*rx\_data, unsigned int size, [murasaki::WaitMilliseconds](#) timeout\_ms=[murasaki::kwmsIndefinitely](#))
- virtual bool [TransmitAndReceiveCompleteCallback](#) (void \*ptr)
- virtual bool [HandleError](#) (void \*ptr)

### 13.22.1 Detailed Description

The [SpiMaster](#) class is the wrapper of the SPI controller. To use the [SpiMaster](#) class, make an instance with [SPI\\_HandleTypeDef](#) \* type pointer. For example, to create an instance for the SPI3 peripheral :

```
my_spi3 = new murasaki::SpiMaster(&hspi3);
```

Where hspi3 is the handle generated by CubeMX for SPI3 peripheral. To use this class, the SPI peripheral have to be configured to use the interrupt and DMA. The bitrate should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
{
    my_spi3->TransmitAndReceiveCompleteCallback(hspi);
}
```

Where HAL\_SPI\_TxRxCpltCallback is a predefined name of the SPI interrupt handler. This is invoked by system whenever a interrupt baed SPI transmission is complete. Becuase the default function is weakly bound, above definition will overwrite the default one.

Note that above callback is invoked for any SPIn where n is 1, 2, 3... To avoid the confusion, SpiMaster::TransferCompleteCallback() method chckes whether given parameter matches with its [SPI\\_HandleTypeDef](#) \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

Once the instance and callbacks are correctly prepared, we can use the Transfer member function.

The [SpiMaster::TransmitAndReceive\(\)](#) member function is a blocking function. A programmer can specify the time-out by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

Note : The behavior of when the timeout happen is not tested. Actually, it should not happen because DMA is taken in SPI transmission. Murasaki stpos internal DMA, interrupt and SPI processing internally then, return.

Other error will cause the re-initializing of the SPI master. Murasaki doesn't support any of CRC detection, TI frame mode or Multi-master SPI.

### 13.22.2 Constructor & Destructor Documentation

#### 13.22.2.1 murasaki::SpiMaster::SpiMaster ( SPI\_HandleTypeDef \* spi\_handle )

Constructor.

## Parameters

<i>spi_handle</i>	Handle to the SPI peripheral. This have to be configured to use DMA by CubeMX.
-------------------	--

## 13.22.3 Member Function Documentation

13.22.3.1 bool murasaki::SpiMaster::HandleError ( void \* *ptr* ) [virtual]

Error handling.

## Parameters

<i>ptr</i>	Pointer to l2c_HandleTypeDef struct.
------------	--------------------------------------

## Returns

true: ptr matches with device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements [murasaki::SpiMasterStrategy](#).

13.22.3.2 SpiStatus murasaki::SpiMaster::TransmitAndReceive ( murasaki::SpiSlaveAdapterStrategy \* *spi\_spec*, const uint8\_t \* *tx\_data*, uint8\_t \* *rx\_data*, unsigned int *size*, murasaki::WaitMilliseconds *timeout\_ms* = murasaki::kwmsIndefinitely ) [virtual]

Data transfer to/from SPI slave.

## Parameters

<i>spi_spec</i>	A pointer to the AbstractSpiSpecification to specify the slave device.
<i>tx_data</i>	Data to be transmitted
<i>rx_data</i>	Data buffer to receive data
<i>size</i>	Transfer data size [byte] for each way.
<i>timeout_ms</i>	Timeout limit [mS]

## Returns

true if transfer complete, false if timeout

Transfer the data to/from SPI slave specified by parameter *spi\_spec*.

This member function re-initialize the SPI peripheral based on the clock information from the *spi\_spec*. And then, assert the chips elect through the *spi\_spec* during the data transfer.

Following are the return codes:

- [murasaki::kspisOK](#) : The transfer complete without error.
- [murasaki::kspisModeCRC](#) : CRC error was detected.
- [murasaki::kspisOverflow](#) : SPI overflow or underflow was detected.
- [murasaki::kspisFrameError](#) Frame error in TI mode.
- [murasaki::kspisDMA](#) : Some DMA error was detected in HAL. SPI re-initialized.
- [murasaki::kspisErrorFlag](#) : Unhandled flags. SPI re-initialized.
- [murasaki::ki2csTimeOut](#) : Timeout detected. DMA stopped.
- Other : Unhandled error . SPI re-initialized.

Implements [murasaki::SpiMasterStrategy](#).

**13.22.3.3** `bool murasaki::SpiMaster::TransmitAndReceiveCompleteCallback ( void * ptr )` `[virtual]`

Callback to notify the end of transfer.

#### Parameters

<i>ptr</i>	Pointer to the control object.
------------	--------------------------------

#### Returns

true if no error.

Implements [murasaki::SpiMasterStrategy](#).

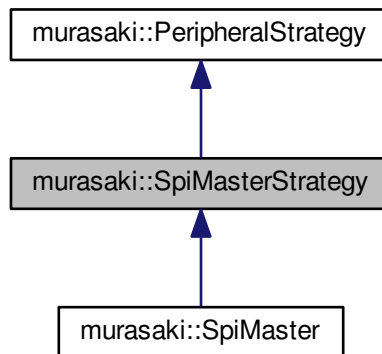
The documentation for this class was generated from the following files:

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/[spimaster.hpp](#)
- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Src/spimaster.cpp

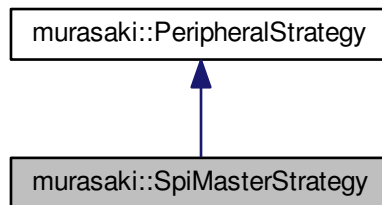
## 13.23 murasaki::SpiMasterStrategy Class Reference

```
#include <spimasterstrategy.hpp>
```

Inheritance diagram for murasaki::SpiMasterStrategy:



Collaboration diagram for murasaki::SpiMasterStrategy:



## Public Member Functions

- virtual [SpiStatus TransmitAndReceive](#) ([murasaki::SpiSlaveAdapterStrategy](#) \*spi\_spec, const uint8\_t \*tx←\_data, uint8\_t \*rx\_data, unsigned int size, [murasaki::WaitMilliseconds](#) timeout\_ms=[murasaki::kwms←Indefinitely](#))=0
- virtual bool [TransmitAndReceiveCompleteCallback](#) (void \*ptr)=0
- virtual bool [HandleError](#) (void \*ptr)=0

### 13.23.1 Detailed Description

This class provides a thread safe, blocking SPI transfer.

### 13.23.2 Member Function Documentation

13.23.2.1 virtual bool `murasaki::SpiMasterStrategy::HandleError ( void * ptr )` [pure virtual]

Handling error report of device.

## Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a device control
------------	---

## Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in [murasaki::SpiMaster](#).

**13.23.2.2** `virtual SpiStatus murasaki::SpiMasterStrategy::TransmitAndReceive ( murasaki::SpiSlaveAdapterStrategy * spi_spec, const uint8_t * tx_data, uint8_t * rx_data, unsigned int size, murasaki::WaitMilliseconds timeout_ms = murasaki::kwmsIndefinitely ) [pure virtual]`

Thread safe, blocking SPI transfer.

## Parameters

<i>spi_spec</i>	Pointer to the SPI slave adapter which has clock configuraiton and chip select handling.
<i>tx_data</i>	Data to be transmitted
<i>rx_data</i>	Data buffer to receive data
<i>size</i>	Transfer data size [byte] for each way. Must be smaller than 65536
<i>timeout_ms</i>	Timeout limit [mS]

## Returns

true if transfer complete, false if timeout

Implemented in [murasaki::SpiMaster](#).

**13.23.2.3** `virtual bool murasaki::SpiMasterStrategy::TransmitAndReceiveCompleteCallback ( void * ptr ) [pure virtual]`

Callback to notify the end of transfer.

## Parameters

<i>ptr</i>	Pointer to the control object.
------------	--------------------------------

## Returns

true if no error.

Implemented in [murasaki::SpiMaster](#).

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

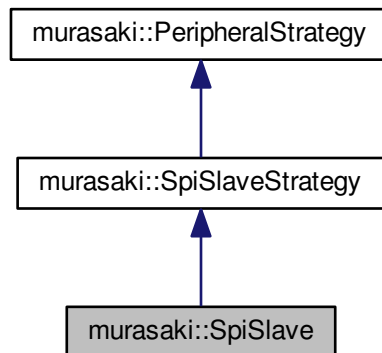


- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/spimasterstrategy.hpp](/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spimasterstrategy.hpp)

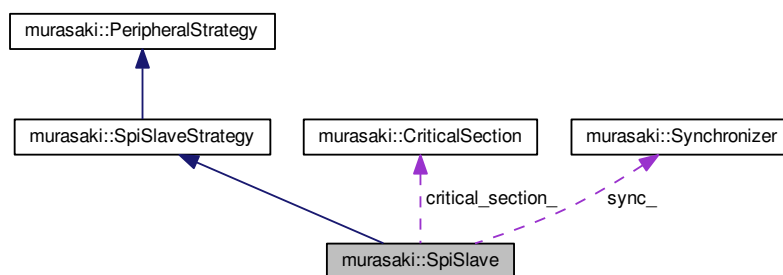
## 13.24 murasaki::SpiSlave Class Reference

```
#include <spislave.hpp>
```

Inheritance diagram for murasaki::SpiSlave:



Collaboration diagram for murasaki::SpiSlave:



### Public Member Functions

- [SpiSlave](#) (`SPI_HandleTypeDef *spi_handle`)
- virtual [SpiStatus TransmitAndReceive](#) (`const uint8_t *tx_data`, `uint8_t *rx_data`, `unsigned int size`, `unsigned int *transferred_count`, [murasaki::WaitMilliseconds](#) `timeout_ms=murasaki::kwmsIndefinitely`)
- virtual bool [TransmitAndReceiveCompleteCallback](#) (`void *ptr`)
- virtual bool [HandleError](#) (`void *ptr`)

### 13.24.1 Detailed Description

The [SpiSlave](#) class is the wrapper of the SPI controller. To use the [SpiSlave](#) class, make an instance with `SPI_HandleTypeDef * type pointer`. For example, to create an instance for the SPI3 peripheral :

```
my_spi3 = new murasaki::SpiSlave(&hspi3);
```

Where `hspi3` is the handle generated by CubeMX for SPI3 peripheral. To use this class, the SPI peripheral have to be configured to use the interrupt and DMA. Also the bitrate, CPOL and CPHA should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
{
    my_spi3->TransmitAndReceiveCompleteCallback(hspi);
}
```

Where `HAL_SPI_TxRxCpltCallback` is a predefined name of the SPI interrupt handler. This is invoked by system whenever a interrupt baed SPI transmission is complete. Because the default function is weakly bound, above definition will override the default one.

Note that above callback is invoked for any SPIn where n is 1, 2, 3... To avoid the confusion, `SpiSlave::TransferCompleteCallback()` method checks whether given parameter matches with its `SPI_HandleTypeDef * pointer` ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

Once the instance and callback are correctly prepared, we can use the `Transfer` member function.

The [SpiSlave::TransmitAndReceive\(\)](#) member function is a blocking function. A programmer can specify the timeout by `timeout_ms` parameter. By default, this parameter is set by `kwmsIndefinitely` which specifies never time out.

This methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

Other error will cause the re-initializing of the SPI slave. Murasaki doesn't support any of CRC detection, TI frame mode or Multi-master SPI.

### 13.24.2 Constructor & Destructor Documentation

#### 13.24.2.1 murasaki::SpiSlave::SpiSlave ( SPI\_HandleTypeDef \* spi\_handle )

Constructor.

##### Parameters

<code>spi_handle</code>	Handle to the SPI peripheral. This have to be configured to use DMA by CubeMX.
-------------------------	--

### 13.24.3 Member Function Documentation

### 13.24.3.1 `bool murasaki::SpiSlave::HandleError ( void * ptr ) [virtual]`

Error handling.

#### Parameters

<i>ptr</i>	Pointer to I2C_HandleTypeDef struct.
------------	--------------------------------------

#### Returns

true: ptr matches with device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements [murasaki::SpiSlaveStrategy](#).

### 13.24.3.2 `SpiStatus murasaki::SpiSlave::TransmitAndReceive ( const uint8_t * tx_data, uint8_t * rx_data, unsigned int size, unsigned int * transferred_count, murasaki::WaitMilliseconds timeout_ms = murasaki::kwmsIndefinitely ) [virtual]`

Data transfer to/from SPI slave.

#### Parameters

<i>tx_data</i>	Data to be transmitted
<i>rx_data</i>	Data buffer to receive data
<i>size</i>	Transfer data size [byte] for each way.
<i>transferred_count</i>	( Currently, Just ignored) The transfered number of bytes during API.
<i>timeout_ms</i>	Timeout limit [mS]

#### Returns

true if transfer complete, false if timeout

Transfer the data to/from SPI slave specified by parameter `spi_spec`.

This member function re-initialize the SPI peripheral based on the clock information from the `spi_spec`. And then, assert the chips select through the `spi_spec` during the data transfer.

Following are the return codes:

- [murasaki::kspisOK](#) : The transfer complete without error.
- [murasaki::kspisModeCRC](#) : CRC error was detected.
- [murasaki::kspisOverflow](#) : SPI overflow or underflow was detected.
- [murasaki::kspisFrameError](#) Frame error in TI mode.
- [murasaki::kspisDMA](#) : Some DMA error was detected in HAL. SPI re-initialized.

- [murasaki::kspisErrorFlag](#) : Unhandled flags. SPI re-initialized.
- [murasaki::ki2csTimeOut](#) : Timeout detected. DMA stopped.
- Other : Unhandled error . SPI re-initialized.

Implements [murasaki::SpiSlaveStrategy](#).

13.24.3.3 `bool murasaki::SpiSlave::TransmitAndReceiveCompleteCallback ( void * ptr )` `[virtual]`

Callback to notify the end of transfer.

#### Parameters

<i>ptr</i>	Pointer to the control object.
------------	--------------------------------

#### Returns

true if no error.

Implements [murasaki::SpiSlaveStrategy](#).

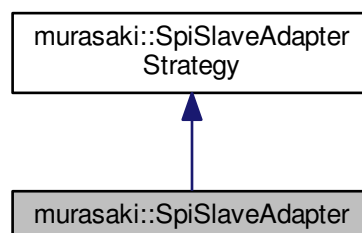
The documentation for this class was generated from the following files:

- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislave.hpp`
- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Src/spislave.cpp`

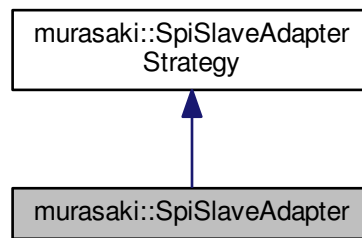
## 13.25 murasaki::SpiSlaveAdapter Class Reference

```
#include <spislaveadapter.hpp>
```

Inheritance diagram for `murasaki::SpiSlaveAdapter`:



Collaboration diagram for murasaki::SpiSlaveAdapter:



## Public Member Functions

- [SpiSlaveAdapter](#) ([murasaki::SpiClockPolarity](#) pol, [murasaki::SpiClockPhase](#) pha, ::GPIO\_TypeDef \*port, uint16\_t pin)
- [SpiSlaveAdapter](#) (unsigned int pol, unsigned int pha, ::GPIO\_TypeDef \*const port, uint16\_t pin)
- virtual void [AssertCs](#) ()
- virtual void [DeassertCs](#) ()

### 13.25.1 Detailed Description

This class describes how this slave is. The description is clock POL and PHA for the specific slave device.

In addition to the clock polarity, the instances of this class work as a surrogate of the chip select control.

The instances will be passed to the [SpiMaster](#) class.

### 13.25.2 Constructor & Destructor Documentation

**13.25.2.1** `murasaki::SpiSlaveAdapter::SpiSlaveAdapter ( murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha, ::GPIO_TypeDef * port, uint16_t pin )`

Constructor.

#### Parameters

<i>pol</i>	Polarity setting
<i>pha</i>	Phase setting
<i>port</i>	GPIO port of the chip select
<i>pin</i>	GPIO pin of the chip select

The port and pin parameters are passed to the `HAL_GPIO_WritePin()`. The port and pin have to be configured by CubeMX correctly.

13.25.2.2 `murasaki::SpiSlaveAdapter::SpiSlaveAdapter ( unsigned int pol, unsigned int pha, ::GPIO_TypeDef *const port, uint16_t pin )`

Constructor.

#### Parameters

<i>pol</i>	Polarity setting
<i>pha</i>	Phase setting
<i>port</i>	GPIO port of the chip select
<i>pin</i>	GPIO pin of the chip select

The port and pin parameters are passed to the `HAL_GPIO_WritePin()`. The port and pin have to be configured by CubeMX correctly.

### 13.25.3 Member Function Documentation

13.25.3.1 `void murasaki::SpiSlaveAdapter::AssertCs ( ) [virtual]`

Chip select assertion.

This member function asset the output line to select the slave chip.

Reimplemented from [murasaki::SpiSlaveAdapterStrategy](#).

13.25.3.2 `void murasaki::SpiSlaveAdapter::DeassertCs ( ) [virtual]`

Chip select deassertoin.

This member function deasset the output line to de-select the slave chip.

Reimplemented from [murasaki::SpiSlaveAdapterStrategy](#).

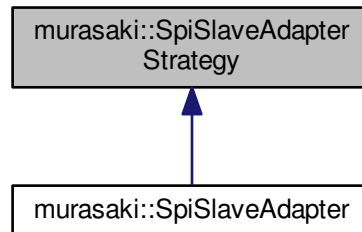
The documentation for this class was generated from the following files:

- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/spislaveadapter.hpp`
- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Src/spislaveadapter.cpp`

## 13.26 murasaki::SpiSlaveAdapterStrategy Class Reference

```
#include <spislaveadapterstrategy.hpp>
```

Inheritance diagram for murasaki::SpiSlaveAdapterStrategy:



### Public Member Functions

- [SpiSlaveAdapterStrategy](#) ([murasaki::SpiClockPolarity](#) pol, [murasaki::SpiClockPhase](#) pha)
- [SpiSlaveAdapterStrategy](#) (unsigned int pol, unsigned int pha)
- virtual void [AssertCs](#) ()
- virtual void [DeassertCs](#) ()
- [murasaki::SpiClockPhase](#) [GetCpha](#) ()
- [murasaki::SpiClockPolarity](#) [GetCpol](#) ()

### 13.26.1 Detailed Description

A prototype of the SPI slave device adapter.

The adapter adds the following SPI attributes :

- CPOL
- CPHA
- Chip select control for slave.

Because SPI slave has different setting device by device, this adapter should be passed to the each transactions.

[AssetCs\(\)](#) and [DeassertCs\(\)](#) have to be overridden to control the chip select output. These member functions will be called from the [AbstractSpiMaster](#).

### 13.26.2 Constructor & Destructor Documentation

13.26.2.1 [murasaki::SpiSlaveAdapterStrategy::SpiSlaveAdapterStrategy](#) ( [murasaki::SpiClockPolarity](#) pol, [murasaki::SpiClockPhase](#) pha )

Constructor.

## Parameters

<i>pol</i>	Polarity setting
<i>pha</i>	Phase setting

13.26.2.2 `murasaki::SpiSlaveAdapterStrategy::SpiSlaveAdapterStrategy ( unsigned int pol, unsigned int pha )`

Constructor.

## Parameters

<i>pol</i>	Polarity setting
<i>pha</i>	Phase setting

### 13.26.3 Member Function Documentation

13.26.3.1 `void murasaki::SpiSlaveAdapterStrategy::AssertCs ( ) [virtual]`

Chip select assertion.

This member function asset the output line to select the slave chip.

This have to be overridden.

Reimplemented in [murasaki::SpiSlaveAdapter](#).

13.26.3.2 `void murasaki::SpiSlaveAdapterStrategy::DeassertCs ( ) [virtual]`

Chip select deassertoin.

This member function deasset the output line to de-select the slave chip.

This have to be overridden.

Reimplemented in [murasaki::SpiSlaveAdapter](#).

13.26.3.3 `murasaki::SpiClockPhase murasaki::SpiSlaveAdapterStrategy::GetCpha ( )`

Getter of the CPHA.

## Returns

CPHA setting



#### 13.26.3.4 murasaki::SpiClockPolarity murasaki::SpiSlaveAdapterStrategy::GetCpol ( )

Getter of the CPOL.

Returns

CPOL setting

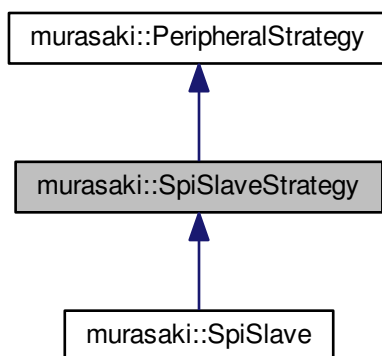
The documentation for this class was generated from the following files:

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/spislaveadapterstrategy.hpp
- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Src/spislaveadapterstrategy.cpp

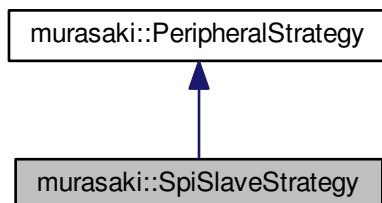
## 13.27 murasaki::SpiSlaveStrategy Class Reference

```
#include <spislavestrategy.hpp>
```

Inheritance diagram for murasaki::SpiSlaveStrategy:



Collaboration diagram for murasaki::SpiSlaveStrategy:



## Public Member Functions

- virtual [SpiStatus TransmitAndReceive](#) (const uint8\_t \*tx\_data, uint8\_t \*rx\_data, unsigned int size, unsigned int \*transferred\_count=nullptr, [murasaki::WaitMilliseconds](#) timeout\_ms=[murasaki::kwmsIndefinitely](#))=0
- virtual bool [TransmitAndReceiveCompleteCallback](#) (void \*ptr)=0
- virtual bool [HandleError](#) (void \*ptr)=0

### 13.27.1 Detailed Description

This class provides a thread safe, blocking SPI transfer.

### 13.27.2 Member Function Documentation

13.27.2.1 virtual bool [murasaki::SpiSlaveStrategy::HandleError](#) ( void \* *ptr* ) [pure virtual]

Handling error report of device.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a device control
------------	---

#### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in [murasaki::SpiSlave](#).

13.27.2.2 virtual [SpiStatus](#) [murasaki::SpiSlaveStrategy::TransmitAndReceive](#) ( const uint8\_t \* *tx\_data*, uint8\_t \* *rx\_data*, unsigned int *size*, unsigned int \* *transferred\_count* = nullptr, [murasaki::WaitMilliseconds](#) *timeout\_ms* = [murasaki::kwmsIndefinitely](#) ) [pure virtual]

Thread safe, blocking SPI transfer.

#### Parameters

<i>tx_data</i>	Data to be transmitted
<i>rx_data</i>	Data buffer to receive data
<i>size</i>	Transfer data size [byte] for each way. Must be smaller than 65536
<i>transferred_count</i>	The transferred number of bytes during API.
<i>timeout_ms</i>	Timeout limit [mS]

#### Returns

true if transfer complete, false if timeout

Implemented in [murasaki::SpiSlave](#).

**13.27.2.3** `virtual bool murasaki::SpiSlaveStrategy::TransmitAndReceiveCompleteCallback ( void * ptr )` [pure virtual]

Callback to notify the end of transfer.

#### Parameters

<i>ptr</i>	Pointer to the control object.
------------	--------------------------------

#### Returns

true if no error.

Implemented in [murasaki::SpiSlave](#).

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

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/spislavestrategy.hpp](#)

## 13.28 murasaki::Synchronizer Class Reference

```
#include <synchronizer.hpp>
```

### Public Member Functions

- bool [Wait](#) ([WaitMilliseconds](#) timeout\_ms=[kwmsIndefinitely](#))
- void [Release](#) ()

#### 13.28.1 Detailed Description

Synchronization mean, task waits for a interrupt by calling `InterruptSynchronizer::WaitForInterruptFromTask()` and during the wait, task yields the cpu to other task. So, CPU can do other job during a task is waiting for interrupt. Interrupt will allow task run again by `InterruptSynchronizer::ReleasetaskFromISR()` member function.

#### 13.28.2 Member Function Documentation

**13.28.2.1** `void murasaki::Synchronizer::Release ( )`

Release the task.

Release the task waiting. This member function must be called from both task and the interrupt context.

**13.28.2.2** `bool murasaki::Synchronizer::Wait ( WaitMilliseconds timeout_ms = kwmsIndefinitely )`

Let the task wait for an interrupt.

**Parameters**

<i>timeout_ms</i>	Timeout by millisecond. The default value let the task wait for interrupt forever.
-------------------	--

**Returns**

True if interrupt came before timeout. False if timeout happen.

This member function have to be called from the task context. Otherwise, the behavior is not predictable.

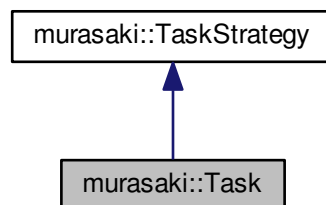
The documentation for this class was generated from the following files:

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/[synchronizer.hpp](#)
- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Src/synchronizer.cpp

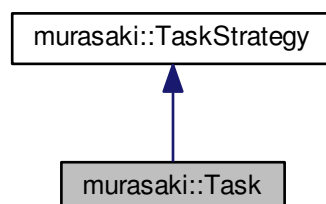
## 13.29 murasaki::Task Class Reference

```
#include <task.hpp>
```

Inheritance diagram for murasaki::Task:



Collaboration diagram for murasaki::Task:



## Public Member Functions

- [Task](#) (const char \*task\_name, unsigned short stack\_depth, UBaseType\_t task\_priority, const void \*task\_↔parameter, void(\*task\_body\_func)(const void \*))

## Protected Member Functions

- virtual void [TaskBody](#) (const void \*ptr)

## Additional Inherited Members

### 13.29.1 Detailed Description

This is handy class to encapsulate the task creation without inheriting. A task can be created easy like :

```
// For demonstration of FreeRTOS task.
murasaki::platform.task1 = new murasaki::Task(
    "Master",
    256,
    (( configMAX_PRIORITIES > 1) ? 1 : 0),
    nullptr,
    &TaskBodyFunction
);
```

Then, task you can call [Start\(\)](#) member function to run.

```
murasaki::platform.task1->Start();
```

### 13.29.2 Constructor & Destructor Documentation

**13.29.2.1** murasaki::Task ( const char \* task\_name, unsigned short stack\_depth, UBaseType\_t task\_priority, const void \* task\_parameter, void(\*) (const void \*) task\_body\_func )

Ease to use task class.

#### Parameters

<i>task_name</i>	A name of task. This is relevant to the FreeRTOS's API manner.
<i>stack_depth</i>	<a href="#">Task</a> stack size by byte.
<i>task_priority</i>	The task priority. Max priority is defined by configMAX_PRIORITIES in FreeRTOSConfig.h
<i>task_parameter</i>	A pointer to the parameter passed to task.
<i>task_body_func</i>	A pointer to the task body function.

Create an task object. Given parameters are stored internally. And then passed to the FreeRTOS API when task is started by [Start\(\)](#) member function.

A task parameter can be passed to task through the task\_parameter. This pointer is simply passed to the task body function without modification.

### 13.29.3 Member Function Documentation

13.29.3.1 void `murasaki::Task::TaskBody` ( const void \* *ptr* ) [protected], [virtual]

`Task` member function.

#### Parameters

<i>ptr</i>	The task_parameter parameter of the constructor is passed to this parameter.
------------	--

This member function runs as task. In this function, the function passed thorough task\_body\_func parameter is invoked as actual task body.

Implements `murasaki::TaskStrategy`.

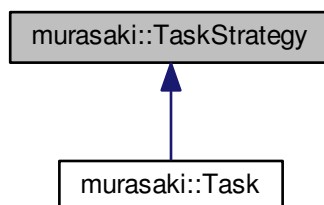
The documentation for this class was generated from the following files:

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/task.hpp
- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Src/task.cpp

## 13.30 murasaki::TaskStrategy Class Reference

```
#include <taskstrategy.hpp>
```

Inheritance diagram for `murasaki::TaskStrategy`:



### Public Member Functions

- `TaskStrategy` (const char \*task\_name, unsigned short stack\_depth, UBaseType\_t task\_priority, const void \*task\_parameter)
- void `Start` ()
- const char \* `GetName` ()

### Protected Member Functions

- virtual void `TaskBody` (const void \*ptr)=0

## Static Protected Member Functions

- static void [Launch](#) (void \*ptr)

### 13.30.1 Detailed Description

Encapsulate a FreeRTOS task.

The constructor just stores given parameter internally. And then, these parameter is passed to a task when [Start\(\)](#) member function is called. Actual task creation is done inside [Start\(\)](#).

The destructor deletes the task. Releasing task from all the resources ( ex: semaphore ) before deleting, is the responsibility of the programmer.

Base on the description at [http://idken.net/posts/2017-02-01-freertos\\_task\\_cpp/](http://idken.net/posts/2017-02-01-freertos_task_cpp/)

### 13.30.2 Constructor & Destructor Documentation

13.30.2.1 `murasaki::TaskStrategy::TaskStrategy ( const char * task_name, unsigned short stack_depth, UBaseType_t task_priority, const void * task_parameter )`

Constructor. [Task](#) entity is not created here.

#### Parameters

<i>task_name</i>	Name of task. Will be passed to task when started.
<i>stack_depth</i>	[Byte]
<i>task_priority</i>	Priority of the task. from 1 to up to configMAX_PRIORITIES -1. The high number is the high priority.
<i>task_parameter</i>	Optional parameter to the task.

### 13.30.3 Member Function Documentation

13.30.3.1 `const char * murasaki::TaskStrategy::GetName ( )`

Get a name of task.

#### Returns

A name of task.

13.30.3.2 `void murasaki::TaskStrategy::Launch ( void * ptr )` `[static], [protected]`

Internal use only. Create a task from [TaskBody\(\)](#)

## Parameters

<i>ptr</i>	passing "this" pointer.
------------	-------------------------

13.30.3.3 void `murasaki::TaskStrategy::Start ( )`

Create a task and run it.

A task is created with given parameter to the constructors and then run.

13.30.3.4 virtual void `murasaki::TaskStrategy::TaskBody ( const void * ptr )` `[protected]`, `[pure virtual]`

Actual task entity. Must be overridden by programmer.

## Parameters

<i>ptr</i>	Optional parameter to the task body. This ptr is copied from the <code>task_parameter</code> of the Constructor.
------------	--

The task body is called only once as task entity. Programmer have to override this member function with his/her own [TaskBody\(\)](#).

From this member function, class members are able to access.

Implemented in [murasaki::Task](#).

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

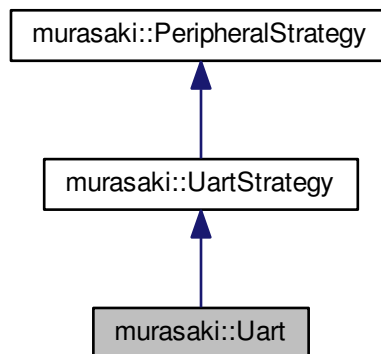
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/taskstrategy.hpp](#)
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Src/taskstrategy.cpp](#)

13.31 `murasaki::Uart` Class Reference

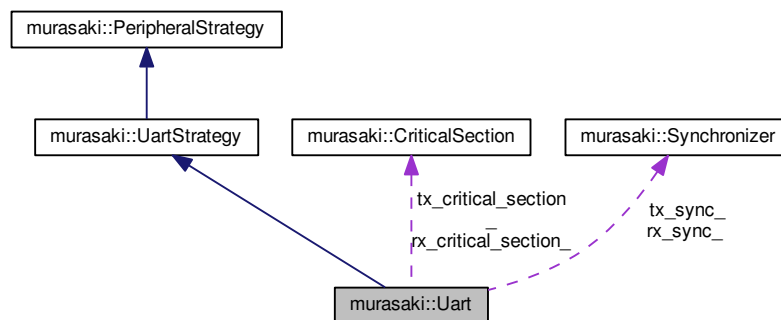
```
#include <uart.hpp>
```



Inheritance diagram for murasaki::Uart:



Collaboration diagram for murasaki::Uart:



## Public Member Functions

- [Uart](#) (UART\_HandleTypeDef \*uart)
- virtual void [SetHardwareFlowControl](#) (UartHardwareFlowControl control)
- virtual void [SetSpeed](#) (unsigned int baud\_rate)
- virtual [murasaki::UartStatus Transmit](#) (const uint8\_t \*data, unsigned int size, [WaitMilliseconds](#) timeout\_ms)
- virtual [murasaki::UartStatus Receive](#) (uint8\_t \*data, unsigned int count, unsigned int \*transferred\_count, [UartTimeout](#) uart\_timeout, [WaitMilliseconds](#) timeout\_ms)
- virtual bool [TransmitCompleteCallback](#) (void \*const ptr)
- virtual bool [ReceiveCompleteCallback](#) (void \*const ptr)
- virtual bool [HandleError](#) (void \*const ptr)

### 13.31.1 Detailed Description

The [Uart](#) class is the wrapper of the UART controller. To use the [Uart](#) class, make an instance with `UART_HandleTypeDef * type pointer`. For example, to create an instance for the UART3 peripheral :

```
my_uart3 = new murasaki::Uart (&huart3);
```

Where `huart3` is the handle generated by CubeMX for UART3 peripheral. To use this class, the UART peripheral have to be configured to use the DMA functionality. The baud rate, length and flow control should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)
{
    my_uart3->TransmitCompleteCallback (huart);
}
```

Where `HAL_UART_TxCpltCallback` is a predefined name of the UART interrupt handler. This is invoked by system whenever a DMA baed UART transmission is complete. Becuase the default function is weakly bound, above definition will overwrite the default one.

Note that above callback is invoked for any UARTn where n is 1, 2, 3... To avoid the confusion, [Uart::TransmitCompleteCallback\(\)](#) method chckes whether given parameter matches with its `UART_HandleTypeDef * pointer` ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

As same as Tx, RX needs [HAL\\_UART\\_TxCpltCallback\(\)](#).

Once the instance and callbacks are correctly prepared, we can use the Tx/Rx member function.

The [Uart::Transmit\(\)](#) member function is a blocking function. A programmer can specify the timeout by `timeout_ms` parameter. By default, this parameter is set by `kwmsIndefinitely` which specifes never time out.

The [Uart::Receive\(\)](#) member function is a blocking function. A programmer can specify the timeout by `timeout_ms` parameter. By default, this parameter is set by `kwmsIndefinitely` which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

### 13.31.2 Constructor & Destructor Documentation

#### 13.31.2.1 murasaki::Uart ( UART\_HandleTypeDef \* uart )

Constructor.

##### Parameters

<i>uart</i>	Pointer to a UART control struct. This device have to be configured to use DMA and interrupt for both Tx and Rx.
-------------	--

Store the given `uart` pointer into the internal variable. This pointer is passed to the STM32Cube HAL UART functions when needed.

### 13.31.3 Member Function Documentation

#### 13.31.3.1 bool murasaki::Uart::HandleError ( void \*const *ptr* ) [virtual]

Error handling.

##### Parameters

<i>ptr</i>	Pointer to UART_HandleTypeDef struct.
------------	---------------------------------------

##### Returns

true: ptr matches with UART device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements [murasaki::UartStrategy](#).

#### 13.31.3.2 murasaki::UartStatus murasaki::Uart::Receive ( uint8\_t \* *data*, unsigned int *count*, unsigned int \* *transferred\_count*, UARTTimeout *uart\_timeout*, WaitMilliseconds *timeout\_ms* ) [virtual]

Receive raw data through an UART by blocking mode.

##### Parameters

<i>data</i>	Data buffer to place the received data..
<i>count</i>	The count of the data ( byte ) to be transferred. Must be smaller than 65536
<i>transferred_count</i>	( Currently, Just ignored) Number of bytes transferred. The nullPtr means no need to return value.
<i>uart_timeout</i>	Specify <a href="#">murasaki::kutIdleTimeout</a> , if idle line timeout is needed.
<i>timeout_ms</i>	Time out limit by milliseconds.

##### Returns

True if all data transferred completely. False if time out happen.

Receive to given data buffer through an UART device.

The receiving mode is blocking. That means, function returns when specified number of data has been received, except timeout. Passing [murasaki::kwmsIndefinitely](#) to the parameter *timeout\_ms* orders not to return until complete receiving. Other value of *timeout\_ms* parameter specifies the time out by millisecond. If time out happen, function returns false. If not happen, it returns true.

This function is exclusive. Internally this function is guarded by mutex. Then this function is thread safe. This function is forbidden to call from ISR.

The return values are:

- [murasaki::kursOK](#) : Transmit complete.
- [murasaki::kursTimeOut](#) : Time out occur.
- [murasaki::kursOverrun](#) : Next char was written to TX register. This is fatal problem in HAL. Periperal is re-initialized internally.
- [murasaki::kursDMA](#) : This is fatal problem in HAL. Peripheral is re-initialized internally.
- other : This is fatal problem in HAL. Peripheral is re-initialized internally.

Implements [murasaki::UartStrategy](#).

**13.31.3.3** `bool murasaki::Uart::ReceiveCompleteCallback ( void *const ptr )` [virtual]

Call back for entire block transfer completion.

#### Parameters

<i>ptr</i>	Pointer to UART_HandleTypeDef struct.
------------	---------------------------------------

#### Returns

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based receiving. The context have to be interrupt.

This member function checks whether the given ptr parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from [HAL\\_UART\\_RxCpltCallback\(\)](#). See STM32F7 HAL manual for detail

Implements [murasaki::UartStrategy](#).

**13.31.3.4** `void murasaki::Uart::SetHardwareFlowControl ( UartHardwareFlowControl control )` [virtual]

Set the behavior of the hardware flow control.

#### Parameters

<i>control</i>	The control mode.
----------------	-------------------

Before calling this method, all transmission and recevie activites have to be finished. This is responsibility of the programmer.

Note this method is NOT re-etnrant. In other word, this member function can be called from both task and interrupt context.

Reimplemented from [murasaki::UartStrategy](#).

13.31.3.5 void murasaki::Uart::SetSpeed ( unsigned int *baud\_rate* ) [virtual]

Set the BAUD rate.

#### Parameters

<i>baud_rate</i>	BAUD rate ( 110, 300,... 57600,... )
------------------	--------------------------------------

Before calling this method, all transmission and receive activities have to be finished. This is responsibility of the programmer.

Note this method is NOT re-entrant. In other words, this member function can be called from both task and interrupt context.

Reimplemented from [murasaki::UartStrategy](#).

13.31.3.6 murasaki::UartStatus murasaki::Uart::Transmit ( const uint8\_t\* *data*, unsigned int *size*, WaitMilliseconds *timeout\_ms* ) [virtual]

Transmit raw data through an UART by blocking mode.

#### Parameters

<i>data</i>	Data buffer to be transmitted.
<i>size</i>	The count of the data ( byte ) to be transferred. Must be smaller than 65536
<i>timeout_ms</i>	Time out limit by milliseconds.

#### Returns

True if all data transferred completely. False if time out happens.

Transmit given data buffer through an UART device.

The transmission mode is blocking. That means, function returns when all data has been transmitted, except timeout. Passing [murasaki::kwmsIndefinitely](#) to the parameter *timeout\_ms* orders not to return until complete transmission. Other value of *timeout\_ms* parameter specifies the time out by millisecond. If time out happens, function returns false. If not happens, it returns true.

This function is exclusive. Internally the function is guarded by mutex. Then this function is thread safe. This function is forbidden to call from ISR.

Implements [murasaki::UartStrategy](#).

13.31.3.7 bool murasaki::Uart::TransmitCompleteCallback ( void\*const *ptr* ) [virtual]

Call back for entire block transfer completion.

#### Parameters

<i>ptr</i>	Pointer to UART_HandleTypeDef struct.
------------	---------------------------------------

**Returns**

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based transmission. The context have to be interrupt.

This member function checks whether the given ptr parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from [HAL\\_UART\\_TxCpltCallback\(\)](#). See STM32F7 HAL manual for detail

The return values are:

- [murasaki::kursOK](#) : Received complete.
- [murasaki::kursTimeOut](#) : Time out occur.
- [murasaki::kursFrame](#) : Receive error by wrong word size configuration.
- [murasaki::kursParity](#) : Parity error.
- [murasaki::kursNoise](#) : Error by noise.
- [murasaki::kursDMA](#) : This is fatal problem in HAL. Peripheral is re-initialized internally.
- other : This is fatal problem in HAL. Peripheral is re-initialized internally.

Implements [murasaki::UartStrategy](#).

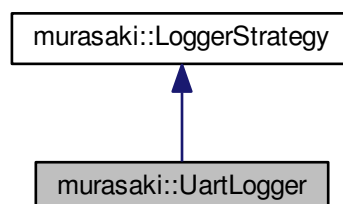
The documentation for this class was generated from the following files:

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/uart.hpp](#)
- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Src/uart.cpp](#)

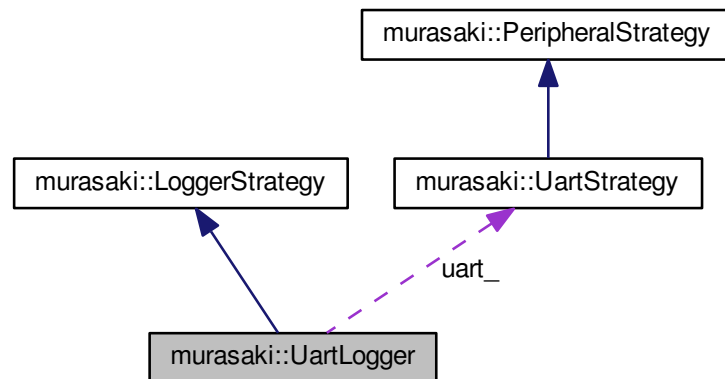
## 13.32 murasaki::UartLogger Class Reference

```
#include <uartlogger.hpp>
```

Inheritance diagram for murasaki::UartLogger:



Collaboration diagram for murasaki::UartLogger:



## Public Member Functions

- [UartLogger](#) ([UartStrategy](#) \*uart)
- virtual void [putMessage](#) (char message[ ], unsigned int size)
- virtual char [getCharacter](#) ()
- virtual void [DoPostMortem](#) (void \*debugger\_fifo)

### 13.32.1 Detailed Description

This is a standard logging class through the UART port. The instance of this class can be passed to the [murasaki::Debugger](#) constructor.

See [Application Specific Platform](#) as usage example.

### 13.32.2 Constructor & Destructor Documentation

#### 13.32.2.1 murasaki::UartLogger::UartLogger ( [UartStrategy](#) \* *uart* )

Constructor.

Parameters

<i>uart</i>	Pointer to the uart object.
-------------	-----------------------------

### 13.32.3 Member Function Documentation

**13.32.3.1** void `murasaki::UartLogger::DoPostMortem ( void * debugger_fifo )` [virtual]

Start post mortem process.

#### Parameters

<i>debugger_fifo</i>	Pointer to the <a href="#">DebuggerFifo</a> class object. The data inside this FIFO will be sent to UART This member function read the data in given FIFO, and then do the auto history.
----------------------	--

This function call the [DebuggerFifo::SetPostMortem\(\)](#) internally. Then, output the data inside FIFO through the given UART.

Once all the data is output, this function wait for a receive data. Once data received, this function rewind the FIFO and then, start to transmit the data again.

Reimplemented from [murasaki::LoggerStrategy](#).

**13.32.3.2** char `murasaki::UartLogger::getCharacter ( )` [virtual]

Character input member function.

#### Returns

A character from input is returned.

This function is considered as blocking. That mean, the function will wait for any user input forever.

Implements [murasaki::LoggerStrategy](#).

**13.32.3.3** void `murasaki::UartLogger::putMessage ( char message[], unsigned int size )` [virtual]

Message output member function.

#### Parameters

<i>message</i>	Non null terminated character array. This data is stored or output to the logger.
<i>size</i>	Size of the message[bytes]. Must be smaller than 65536

Implements [murasaki::LoggerStrategy](#).

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

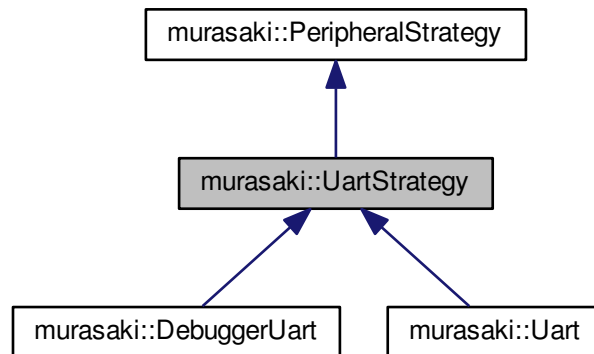
- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Inc/uartlogger.hpp`
- `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/Src/uartlogger.cpp`

## 13.33 `murasaki::UartStrategy` Class Reference

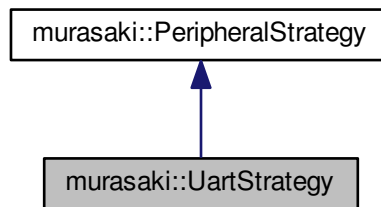
```
#include <uartstrategy.hpp>
```



Inheritance diagram for murasaki::UartStrategy:



Collaboration diagram for murasaki::UartStrategy:



## Public Member Functions

- virtual void [SetHardwareFlowControl](#) ([UartHardwareFlowControl](#) control)
- virtual void [SetSpeed](#) (unsigned int speed)
- virtual [murasaki::UartStatus Transmit](#) (const uint8\_t \*data, unsigned int size, [WaitMilliseconds](#) timeout\_↔ ms=[murasaki::kwmsIndefinitely](#))=0
- virtual [murasaki::UartStatus Receive](#) (uint8\_t \*data, unsigned int size, unsigned int \*transferred\_count=nullptr, [UartTimeout](#) uart\_timeout=[murasaki::kutNoldleTimeout](#), [WaitMilliseconds](#) timeout\_ms=[murasaki::kwms↔ Indefinitely](#))=0
- virtual bool [TransmitCompleteCallback](#) (void \*ptr)=0
- virtual bool [ReceiveCompleteCallback](#) (void \*ptr)=0
- virtual bool [HandleError](#) (void \*ptr)=0

### 13.33.1 Detailed Description

A prototype of the UART device. This abstract class shows the usage of the UART peripheral.

This prototype assumes the derived class will transmit / receive data in the task context on RTOS. And both methods should be blocking. That means, until the transmit / receive terminates, both methods don't return.

Two callback methods are prepared to sync with the interrupt which tells the end of Transmit/Receive.

### 13.33.2 Member Function Documentation

#### 13.33.2.1 `virtual bool murasaki::UartStrategy::HandleError ( void * ptr ) [pure virtual]`

Handling error report of device.

##### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a device control
------------	---

##### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match. A member function to detect error.

The error handling is dependent on the implementation.

Implemented in [murasaki::Uart](#), and [murasaki::DebuggerUart](#).

#### 13.33.2.2 `virtual murasaki::UartStatus murasaki::UartStrategy::Receive ( uint8_t * data, unsigned int size, unsigned int * transfered_count = nullptr, UartTimeout uart_timeout = murasaki::kUartIdleTimeout, WaitMilliseconds timeout_ms = murasaki::kWaitIndefinitely ) [pure virtual]`

buffer receive over the UART. Blocking

##### Parameters

<i>data</i>	Pointer to the buffer to save the received data.
<i>size</i>	Number of the data to be received.
<i>transfered_count</i>	Number of bytes transferred. The nullptr means no need to return value.
<i>uart_timeout</i>	Specify <a href="#">murasaki::kUartIdleTimeout</a> , if idle line timeout is needed.
<i>timeout_ms</i>	Time out by milli Second.

##### Returns

Status of the IO processing

Implemented in [murasaki::Uart](#), and [murasaki::DebuggerUart](#).

13.33.2.3 `virtual bool murasaki::UartStrategy::ReceiveCompleteCallback ( void * ptr )` `[pure virtual]`

Call back to be called for entire block transfer is complete.

#### Parameters

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a UART device control
------------	--

#### Returns

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in [murasaki::Uart](#), and [murasaki::DebuggerUart](#).

13.33.2.4 `virtual void murasaki::UartStrategy::SetHardwareFlowControl ( UartHardwareFlowControl control )`  
`[inline], [virtual]`

Set the behavior of the hardware flow control.

#### Parameters

<i>control</i>	The control mode.
----------------	-------------------

Reimplemented in [murasaki::DebuggerUart](#), and [murasaki::Uart](#).

13.33.2.5 `virtual void murasaki::UartStrategy::SetSpeed ( unsigned int speed )` `[inline], [virtual]`

the baud rate

#### Parameters

<i>speed</i>	BAUD rate ( 110, 300, ... 9600,... )
--------------	--------------------------------------

Reimplemented in [murasaki::DebuggerUart](#), and [murasaki::Uart](#).

13.33.2.6 `virtual murasaki::UartStatus murasaki::UartStrategy::Transmit ( const uint8_t * data, unsigned int size,  
WaitMilliseconds timeout_ms = murasaki::kwmsIndefinitely )` `[pure virtual]`

buffer transmission over the UART. Blocking

**Parameters**

<i>data</i>	Pointer to the buffer to be sent.
<i>size</i>	Number of the data to be sent.
<i>timeout_ms</i>	Time out by mili Second.

**Returns**

Status of the IO processing

Implemented in [murasaki::DebuggerUart](#), and [murasaki::Uart](#).

13.33.2.7 `virtual bool murasaki::UartStrategy::TransmitCompleteCallback ( void * ptr ) [pure virtual]`

Call back to be called notify the transfer is complete.

**Parameters**

<i>ptr</i>	Pointer for generic use. Usually, points a struct of a UART device control
------------	--

**Returns**

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in [murasaki::Uart](#), and [murasaki::DebuggerUart](#).

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

- [/home/takemasa/workspace\\_st/nucleo-f446-sample/murasaki/Inc/uartstrategy.hpp](#)

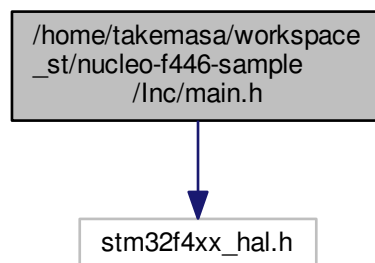
## Chapter 14

# File Documentation

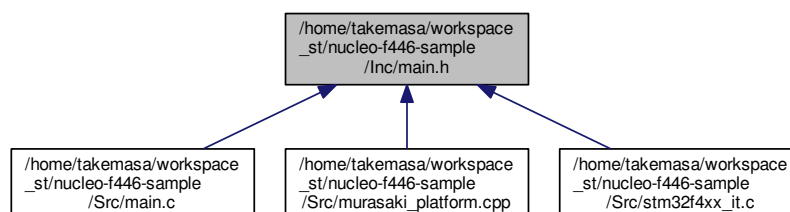
### 14.1 /home/takemasa/workspace\_st/nucleo-f446-sample/Inc/main.h File Reference

```
#include "stm32f4xx_hal.h"
```

Include dependency graph for main.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [Error\\_Handler](#) (void)

### 14.1.1 Detailed Description

#### Attention

© Copyright (c) 2019 STMicroelectronics. All rights reserved.

This software component is licensed by ST under Ultimate Liberty license SLA0044, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: [www.st.com/SLA0044](http://www.st.com/SLA0044)

### 14.1.2 Function Documentation

#### 14.1.2.1 void Error\_Handler ( void )

This function is executed in case of error occurrence.

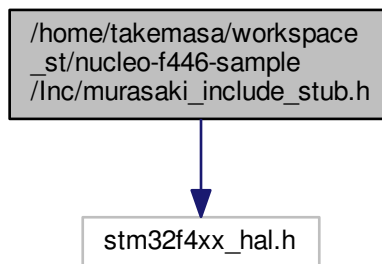
#### Return values

<i>None</i>	
-------------	--

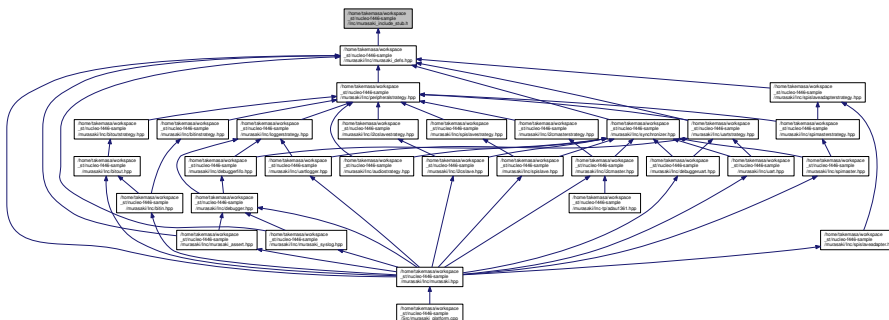
## 14.2 /home/takemasa/workspace\_st/nucleo-f446-sample/lnc/murasaki\_include\_stub.h File Reference

```
#include <stm32f4xx_hal.h>
```

Include dependency graph for murasaki\_include\_stub.h:



This graph shows which files directly or indirectly include this file:



### 14.2.1 Detailed Description

The CubeMX add the STM32 microprocessor product name as pre-defined macro when a file is compiled. For example, following is the macro definition for STM32F446 processor at the compiler command line.

```
-DSTM32F446xx
```

On the other hand, this is not enough to determine the appropriate include file inside [murasaki\\_defs.hpp](#). As a result, there are difficulties to include the appropriate file.

One of the naive approach is to enumerate all possible pre-defined macro to determine the filename as following.

```

#elif defined (STM32F405xx) || defined (STM32F415xx) || defined (STM32F407xx) || defined (STM32F417xx) || *
    defined (STM32F427xx) || defined (STM32F437xx) || defined (STM32F429xx) || defined (STM32F439xx) || *
    defined (STM32F401xC) || defined (STM32F401xE) || defined (STM32F410Tx) || defined (STM32F410Cx) || *
    defined (STM32F410Rx) || defined (STM32F411xE) || defined (STM32F446xx) || defined (STM32F469xx) || *
    defined (STM32F479xx) || defined (STM32F412Cx) || defined (STM32F412Rx) || defined (STM32F412Vx) || *
    defined (STM32F412Zx) || defined (STM32F413xx) || defined (STM32F423xx)
#include "stm32f4xx_hal.h"
  
```

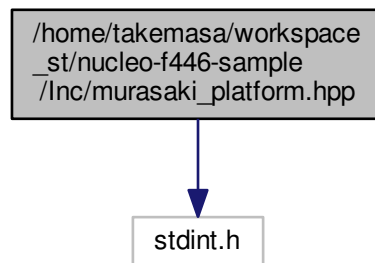
This is easy to understand. But boring to maintain.

This stub is alternate way. [murasaki\\_defs.hpp](#) is including this file ([murasaki\\_include\\_stub.h](#)). And this stub file include the appropriate HAL header file. This stub file is generated by `murasaki/install` script. Thus, user doesn't need to maintain this file.

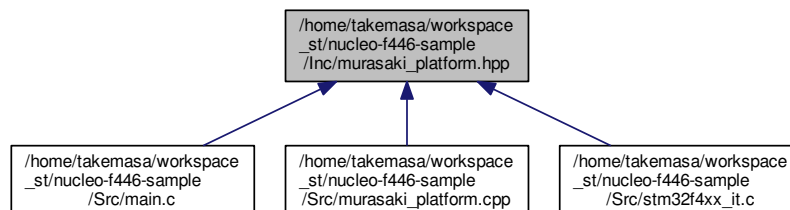
### 14.3 /home/takemasa/workspace\_st/nucleo-f446-sample/Inc/murasaki\_platform.hpp File Reference

```
#include <stdint.h>
```

Include dependency graph for `murasaki_platform.hpp`:



This graph shows which files directly or indirectly include this file:



### Functions

- void [InitPlatform](#) ()
- void [ExecPlatform](#) ()
- void [CustomAssertFailed](#) (uint8\_t \*file, uint32\_t line)
- void [CustomDefaultHandler](#) ()



### 14.3.1 Detailed Description

**Date**

2017/11/12

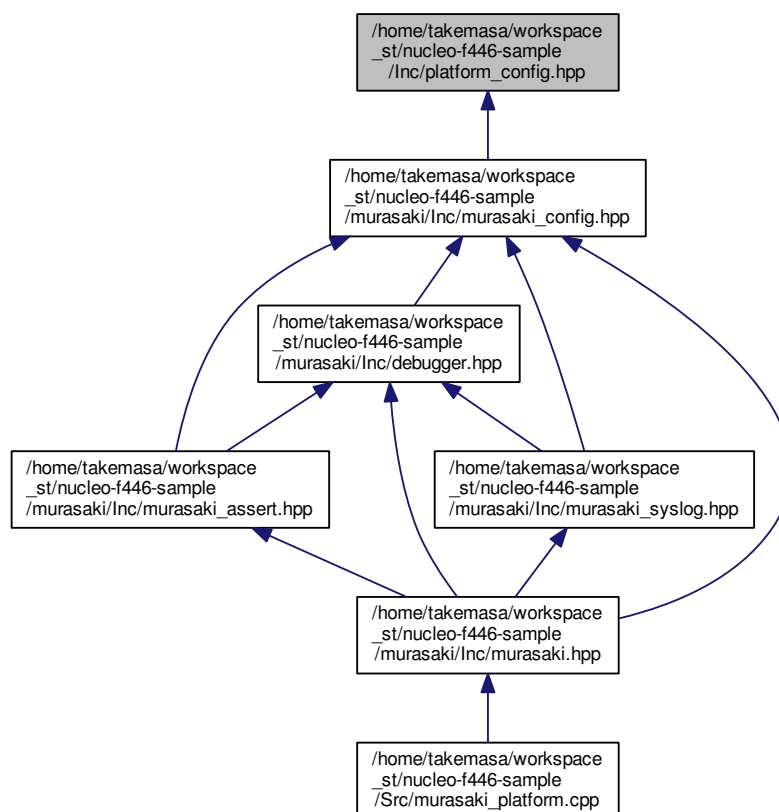
**Author**

Seiichi "Suikan" Horie

The resources below are implemented in the [murasaki\\_platform.cpp](#) and serve as glue to the [main.c](#).

## 14.4 /home/takemasa/workspace\_st/nucleo-f446-sample/Inc/platform\_config.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define MURASAKI_CONFIG_NOSYSLOG false`

### 14.4.1 Detailed Description

Date

2018/01/07

Author

Seiichi "Suikan" Horie

If you want to override the macro definition inside [platform\\_config.hpp](#), add your definition here.

### 14.4.2 Macro Definition Documentation

#### 14.4.2.1 `#define MURASAKI_CONFIG_NOSYSLOG false`

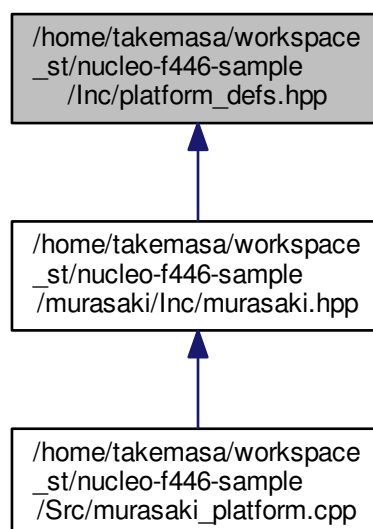
Surpress [MURASAKI\\_SYSLOG](#) macro.

Set this macro to true, to discard the [MURASAKI\\_SYSLOG](#). Set this macro false, to use the syslog.

To override the definition here, define same macro inside [platform\\_config.hpp](#).

## 14.5 [/home/takemasa/workspace\\_st/nucleo-f446-sample/Inc/platform\\_defs.hpp](#) File Reference

This graph shows which files directly or indirectly include this file:



## Classes

- struct [murasaki::Platform](#)

## Namespaces

- [murasaki](#)

## Variables

- Platform [murasaki::platform](#)

### 14.5.1 Detailed Description

#### Date

2018/01/16

#### Author

Seiichi "Suikan" Horie

This file contains user defined struct [murasaki::Platform](#).

This file will be included by [murasaki.hpp](#).

## 14.6 /home/takemasa/workspace\_st/nucleo-f446-sample/Inc/stm32f4xx\_hal\_conf.h File Reference

```
#include "stm32f4xx_hal_rcc.h"
#include "stm32f4xx_hal_exti.h"
#include "stm32f4xx_hal_gpio.h"
#include "stm32f4xx_hal_dma.h"
#include "stm32f4xx_hal_cortex.h"
#include "stm32f4xx_hal_flash.h"
#include "stm32f4xx_hal_i2c.h"
#include "stm32f4xx_hal_pwr.h"
#include "stm32f4xx_hal_spi.h"
#include "stm32f4xx_hal_tim.h"
#include "stm32f4xx_hal_uart.h"
```

Include dependency graph for stm32f4xx\_hal\_conf.h:



## Macros

- #define [HSE\\_VALUE](#) ((uint32\_t)8000000U)
- #define [HSE\\_STARTUP\\_TIMEOUT](#) ((uint32\_t)100U)
- #define [HSI\\_VALUE](#) ((uint32\_t)16000000U)
- #define [LSI\\_VALUE](#) ((uint32\_t)32000U)
- #define [LSE\\_VALUE](#) ((uint32\_t)32768U)
- #define [LSE\\_STARTUP\\_TIMEOUT](#) ((uint32\_t)5000U)
- #define [EXTERNAL\\_CLOCK\\_VALUE](#) ((uint32\_t)12288000U)
- #define [VDD\\_VALUE](#) ((uint32\_t)3300U)
- #define [TICK\\_INT\\_PRIORITY](#) ((uint32\_t)0U)
- #define [PHY\\_BCR](#) ((uint16\_t)0x0000U)
- #define [PHY\\_BSR](#) ((uint16\_t)0x0001U)
- #define [PHY\\_RESET](#) ((uint16\_t)0x8000U)
- #define [PHY\\_LOOPBACK](#) ((uint16\_t)0x4000U)
- #define [PHY\\_FULLDUPLEX\\_100M](#) ((uint16\_t)0x2100U)
- #define [PHY\\_HALFDUPLEX\\_100M](#) ((uint16\_t)0x2000U)
- #define [PHY\\_FULLDUPLEX\\_10M](#) ((uint16\_t)0x0100U)
- #define [PHY\\_HALFDUPLEX\\_10M](#) ((uint16\_t)0x0000U)
- #define [PHY\\_AUTONEGOTIATION](#) ((uint16\_t)0x1000U)
- #define [PHY\\_RESTART\\_AUTONEGOTIATION](#) ((uint16\_t)0x0200U)
- #define [PHY\\_POWERDOWN](#) ((uint16\_t)0x0800U)
- #define [PHY\\_ISOLATE](#) ((uint16\_t)0x0400U)
- #define [PHY\\_AUTONEGO\\_COMPLETE](#) ((uint16\_t)0x0020U)
- #define [PHY\\_LINKED\\_STATUS](#) ((uint16\_t)0x0004U)
- #define [PHY\\_JABBER\\_DETECTION](#) ((uint16\_t)0x0002U)
- #define [PHY\\_SR](#) ((uint16\_t)0x10U)
- #define [PHY\\_SPEED\\_STATUS](#) ((uint16\_t)0x0002U)
- #define [PHY\\_DUPLEX\\_STATUS](#) ((uint16\_t)0x0004U)
- #define [assert\\_param](#)(expr) ((expr) ? (void)0U : assert\_failed((uint8\_t \*)\_\_FILE\_\_, \_\_LINE\_\_))

## Functions

- void [assert\\_failed](#) (uint8\_t \*file, uint32\_t line)

### 14.6.1 Detailed Description

#### Attention

© COPYRIGHT(c) 2019 STMicroelectronics

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 14.6.2 Macro Definition Documentation

14.6.2.1 `#define assert_param( expr ) ((expr) ? (void)0U : assert_failed((uint8_t *)__FILE__, __LINE__))`

Include module's header file.

The `assert_param` macro is used for function's parameters check.

### Parameters

<i>expr</i>	If <i>expr</i> is false, it calls <code>assert_failed</code> function which reports the name of the source file and the source line number of the call that failed. If <i>expr</i> is true, it returns no value.
-------------	--

### Return values

<i>None</i>	
-------------	--

14.6.2.2 `#define EXTERNAL_CLOCK_VALUE ((uint32_t)12288000U)`

External clock source for I2S peripheral This value is used by the I2S HAL module to compute the I2S clock source frequency, this source is inserted directly through I2S\_CKIN pad.

Value of the External audio frequency in Hz

14.6.2.3 `#define HSE_STARTUP_TIMEOUT ((uint32_t)100U)`

Time out for HSE start up, in ms

14.6.2.4 `#define HSE_VALUE ((uint32_t)8000000U)`

Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).

Value of the External oscillator in Hz

14.6.2.5 `#define HSI_VALUE ((uint32_t)16000000U)`

Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).

Value of the Internal oscillator in Hz

14.6.2.6 `#define LSE_STARTUP_TIMEOUT ((uint32_t)5000U)`

Time out for LSE start up, in ms

14.6.2.7 `#define LSE_VALUE ((uint32_t)32768U)`

External Low Speed oscillator (LSE) value.

< Value of the Internal Low Speed oscillator in Hz The real value may vary depending on the variations in voltage and temperature. Value of the External Low Speed oscillator in Hz

14.6.2.8 `#define LSI_VALUE ((uint32_t)32000U)`

Internal Low Speed oscillator (LSI) value.

LSI Typical Value in Hz

14.6.2.9 `#define PHY_AUTONEGO_COMPLETE ((uint16_t)0x0020U)`

Auto-Negotiation process completed

14.6.2.10 `#define PHY_AUTONEGOTIATION ((uint16_t)0x1000U)`

Enable auto-negotiation function

14.6.2.11 `#define PHY_BCR ((uint16_t)0x0000U)`

Transceiver Basic Control Register

14.6.2.12 `#define PHY_BSR ((uint16_t)0x0001U)`

Transceiver Basic Status Register

14.6.2.13 `#define PHY_DUPLEX_STATUS ((uint16_t)0x0004U)`

PHY Duplex mask

14.6.2.14 `#define PHY_FULLDUPLEX_100M ((uint16_t)0x2100U)`

Set the full-duplex mode at 100 Mb/s

14.6.2.15 `#define PHY_FULLDUPLEX_10M ((uint16_t)0x0100U)`

Set the full-duplex mode at 10 Mb/s

14.6.2.16 `#define PHY_HALFDUPLEX_100M ((uint16_t)0x2000U)`

Set the half-duplex mode at 100 Mb/s

14.6.2.17 `#define PHY_HALFDUPLEX_10M ((uint16_t)0x0000U)`

Set the half-duplex mode at 10 Mb/s

14.6.2.18 `#define PHY_ISOLATE ((uint16_t)0x0400U)`

Isolate PHY from MII

14.6.2.19 `#define PHY_JABBER_DETECTION ((uint16_t)0x0002U)`

Jabber condition detected

14.6.2.20 `#define PHY_LINKED_STATUS ((uint16_t)0x0004U)`

Valid link established

14.6.2.21 `#define PHY_LOOPBACK ((uint16_t)0x4000U)`

Select loop-back mode

14.6.2.22 `#define PHY_POWERDOWN ((uint16_t)0x0800U)`

Select the power down mode

14.6.2.23 `#define PHY_RESET ((uint16_t)0x8000U)`

PHY Reset

14.6.2.24 `#define PHY_RESTART_AUTONEGOTIATION ((uint16_t)0x0200U)`

Restart auto-negotiation function

14.6.2.25 `#define PHY_SPEED_STATUS ((uint16_t)0x0002U)`

PHY Speed mask

14.6.2.26 `#define PHY_SR ((uint16_t)0x10U)`

PHY status register Offset

14.6.2.27 `#define TICK_INT_PRIORITY ((uint32_t)0U)`

tick interrupt priority

14.6.2.28 `#define VDD_VALUE ((uint32_t)3300U)`

This is the HAL system configuration section.

Value of VDD in mv

### 14.6.3 Function Documentation

14.6.3.1 `void assert_failed ( uint8_t * file, uint32_t line )`

Reports the name of the source file and the source line number where the `assert_param` error has occurred.

#### Parameters

<i>file</i>	pointer to the source file name
<i>line</i>	<code>assert_param</code> error line source number

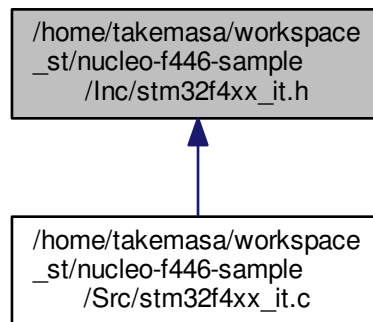
#### Return values

<i>None</i>	
-------------	--



## 14.7 /home/takemasa/workspace\_st/nucleo-f446-sample/Inc/stm32f4xx\_it.h File Reference

This graph shows which files directly or indirectly include this file:



### 14.7.1 Detailed Description

Attention

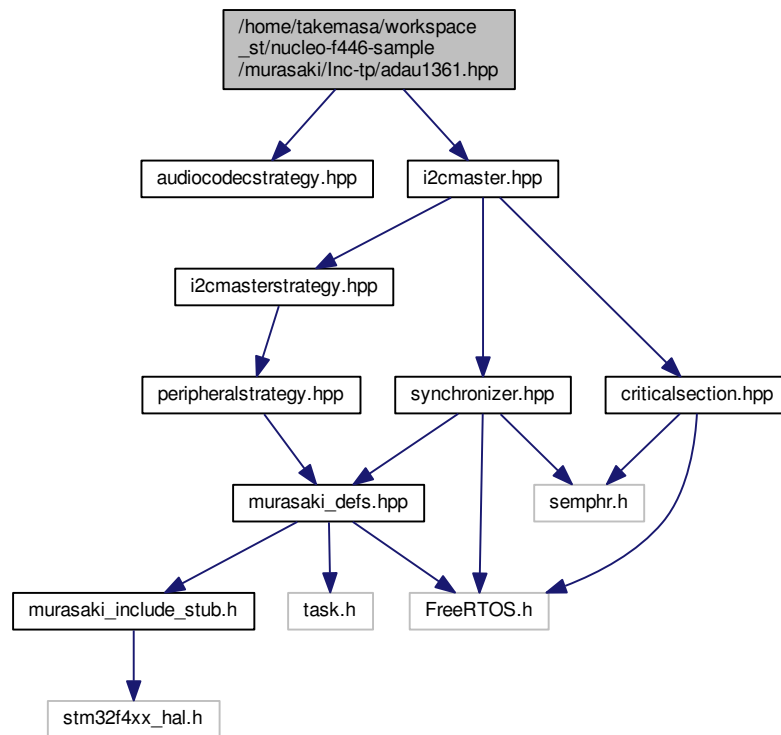
© Copyright (c) 2019 STMicroelectronics. All rights reserved.

This software component is licensed by ST under Ultimate Liberty license SLA0044, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: [www.st.com/SLA0044](http://www.st.com/SLA0044)

## 14.8 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc-tp/adau1361.hpp File Reference

```
#include <audiocodecstrategy.hpp>
#include "i2cmaster.hpp"
```

Include dependency graph for adau1361.hpp:



## Classes

- class [murasaki::Adu1361](#)

## Namespaces

- [murasaki](#)

### 14.8.1 Detailed Description

#### Date

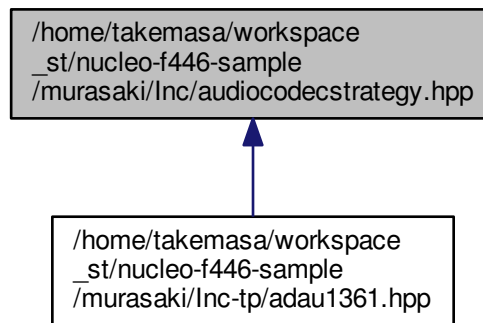
2018/05/11

#### Author

: Seiichi "Suikan" Horie

## 14.9 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/audiocodecstrategy.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- class [murasaki::AudioCodecStrategy](#)

### Namespaces

- [murasaki](#)

### 14.9.1 Detailed Description

#### Date

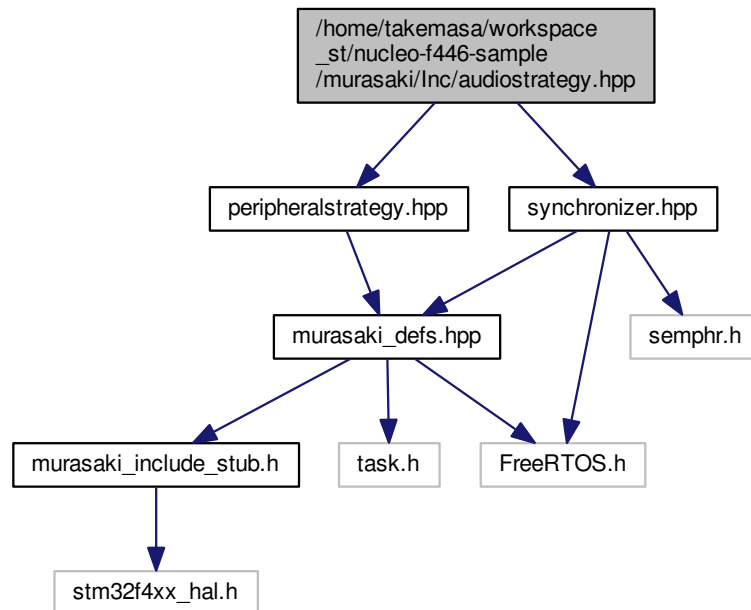
2018/05/11

#### Author

: Seiichi "Suikan" Horie

## 14.10 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/audiostrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
#include <synchronizer.hpp>
Include dependency graph for audiostrategy.hpp:
```



### Classes

- class [murasaki::AudioStrategy](#)

### Namespaces

- [murasaki](#)

#### 14.10.1 Detailed Description

##### Date

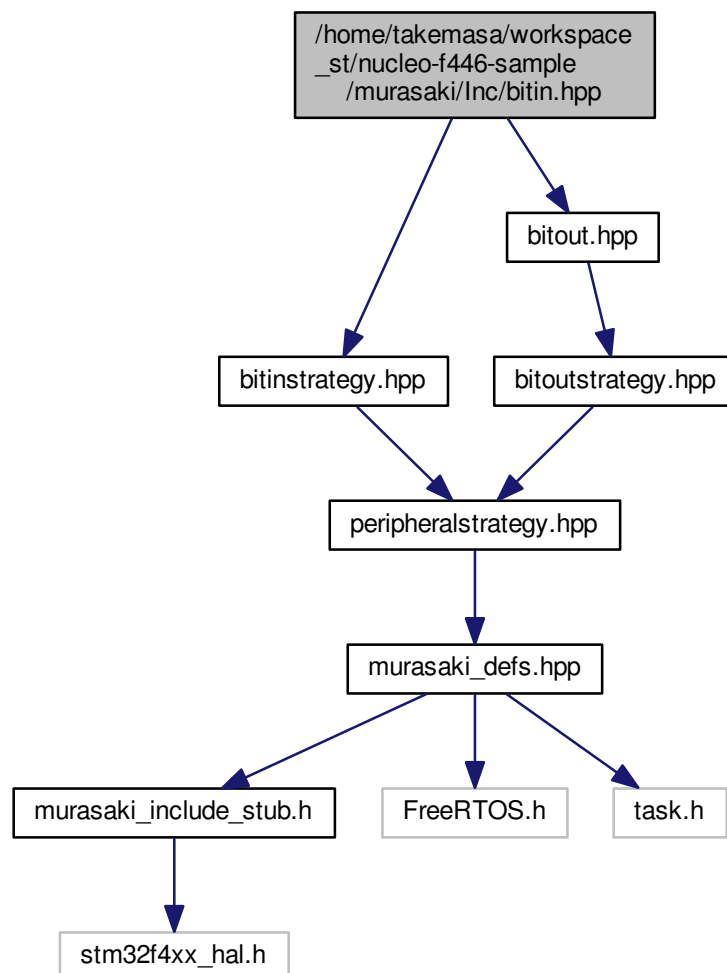
2019/03/02

##### Author

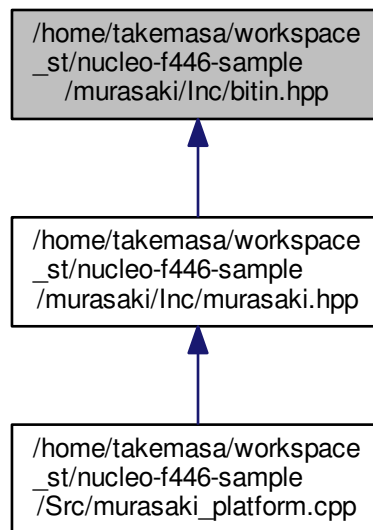
Seiichi "Suikan" Horie

## 14.11 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/bitin.hpp File Reference

```
#include <bitinstrategy.hpp>
#include "bitout.hpp"
Include dependency graph for bitin.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::BitIn](#)

## Namespaces

- [murasaki](#)

### 14.11.1 Detailed Description

#### Date

2018/05/07

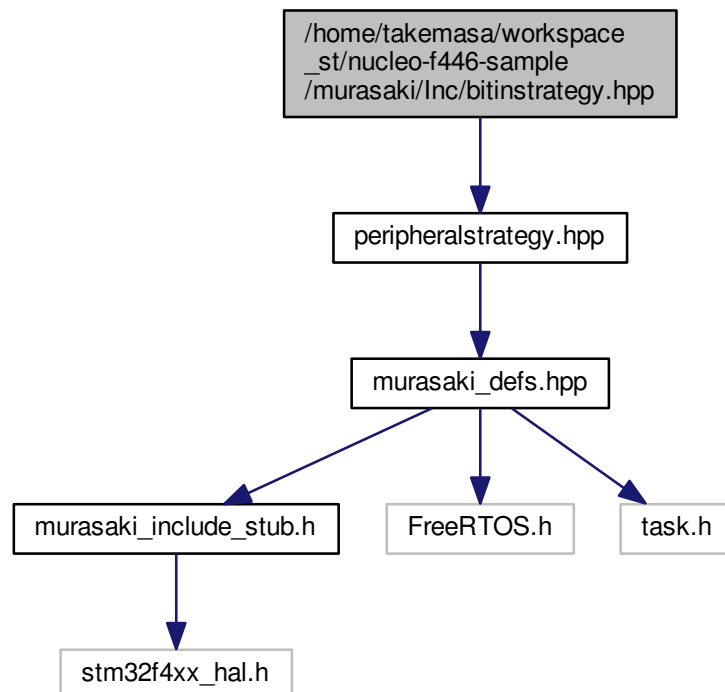
#### Author

Seiichi "Suikan" Horie

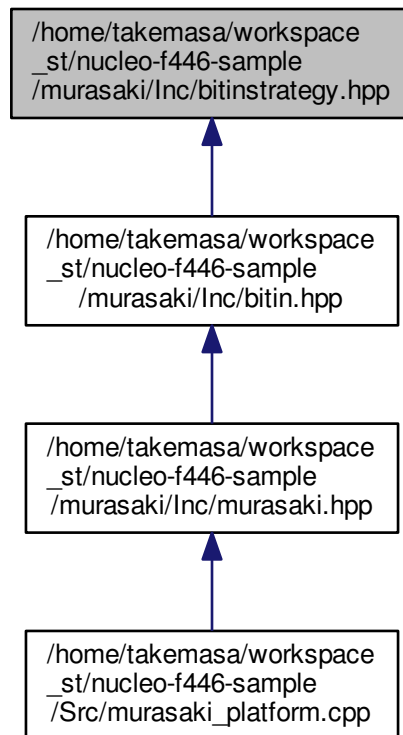
## 14.12 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/bitinstrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
```

Include dependency graph for bitinstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::BitInStrategy](#)

## Namespaces

- [murasaki](#)

### 14.12.1 Detailed Description

#### Date

2018/05/07

#### Author

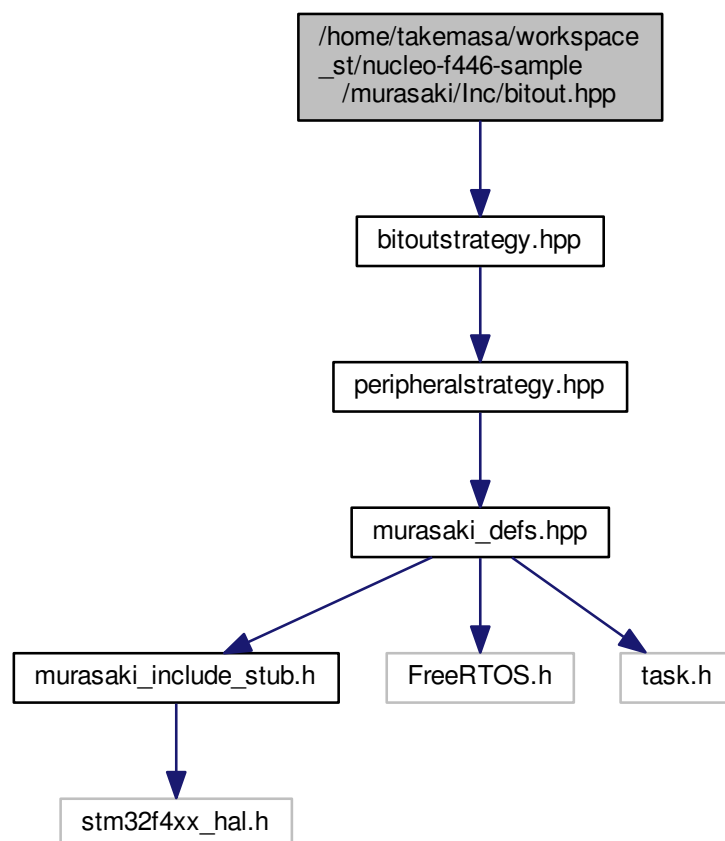
Seiichi "Suikan" Horie



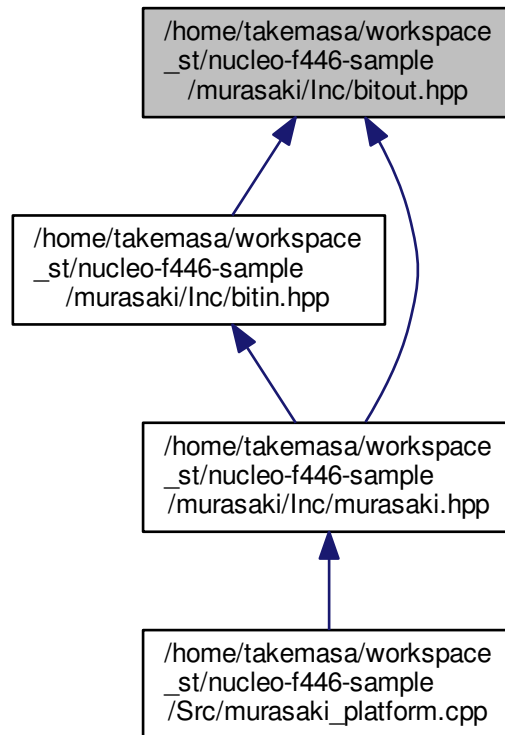
## 14.13 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/bitout.hpp File Reference

```
#include <bitoutstrategy.hpp>
```

Include dependency graph for bitout.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [murasaki::GPIO\\_type](#)
- class [murasaki::BitOut](#)

## Namespaces

- [murasaki](#)

### 14.13.1 Detailed Description

#### Date

2018/05/07

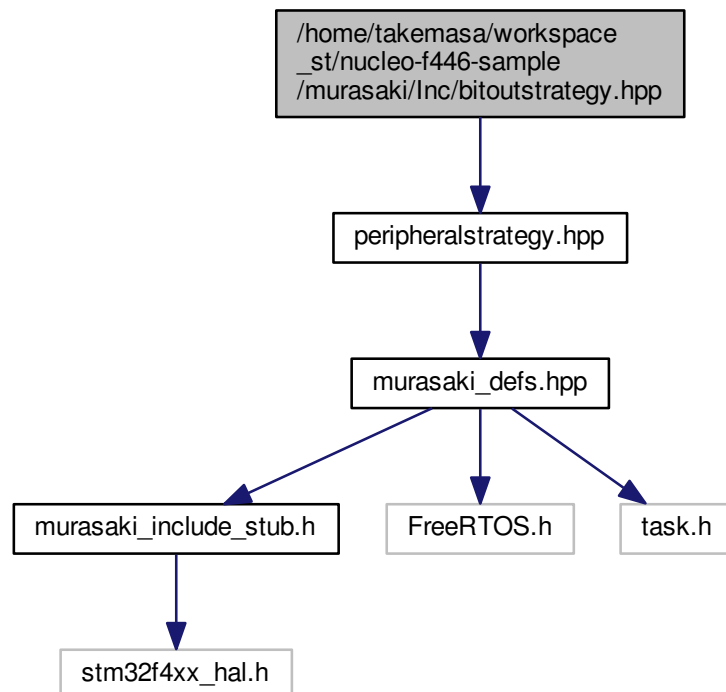
#### Author

Seiichi "Suikan" Horie

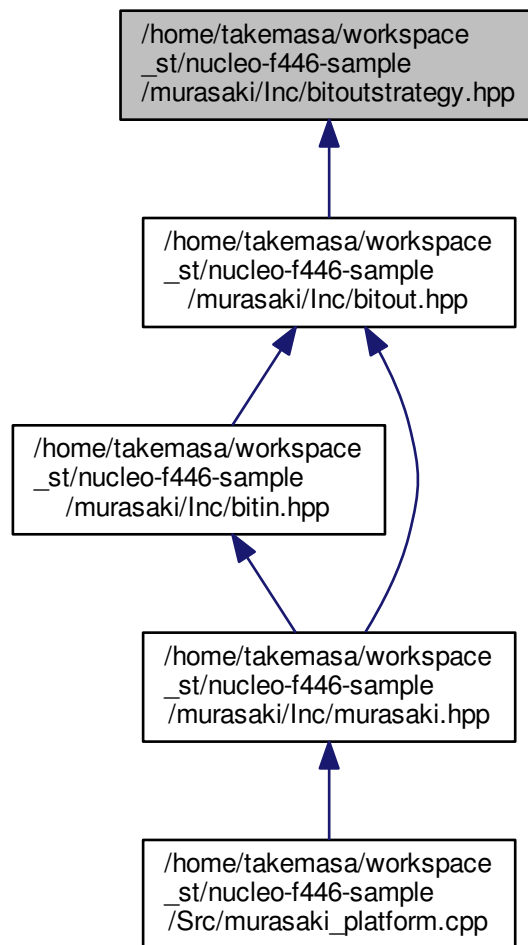
## 14.14 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/bitoutstrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
```

Include dependency graph for bitoutstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::BitOutStrategy](#)

## Namespaces

- [murasaki](#)

### 14.14.1 Detailed Description

#### Date

2018/05/07

#### Author

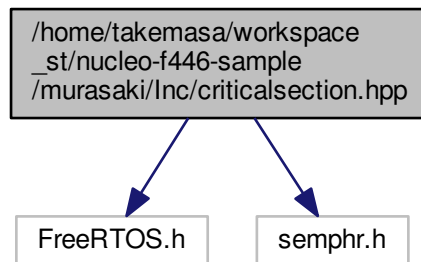
Seiichi "Suikan" Horie

## 14.15 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/inc/criticalsection.hpp File Reference

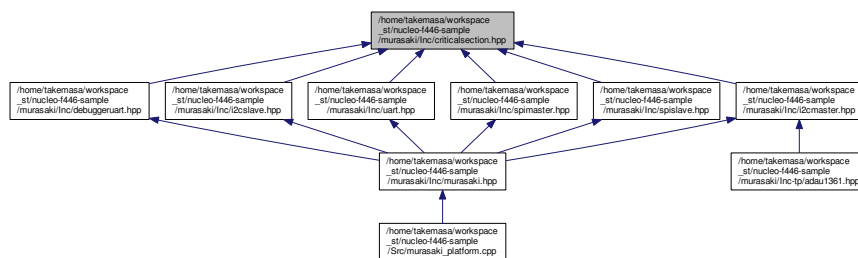
```
#include <FreeRTOS.h>
```

```
#include <semphr.h>
```

Include dependency graph for criticalsection.hpp:



This graph shows which files directly or indirectly include this file:



### Classes

- class [murasaki::CriticalSection](#)

### Namespaces

- [murasaki](#)

#### 14.15.1 Detailed Description

##### Date

2018/01/27

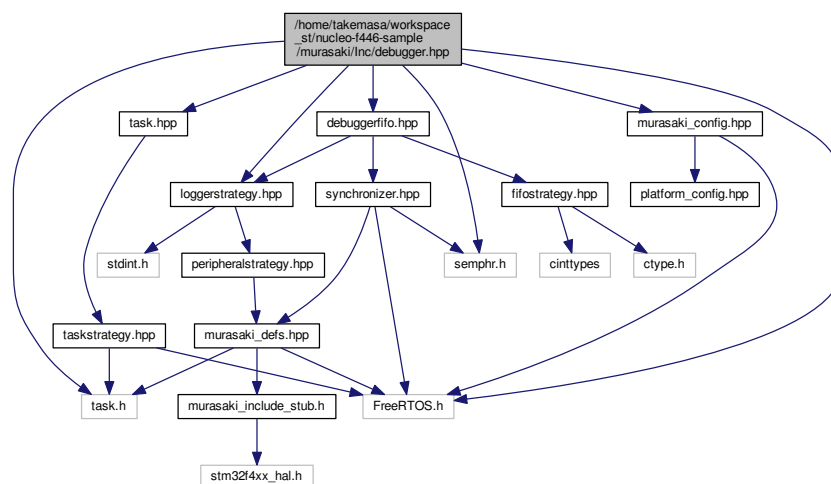
##### Author

Seiichi "Suikan" Horie

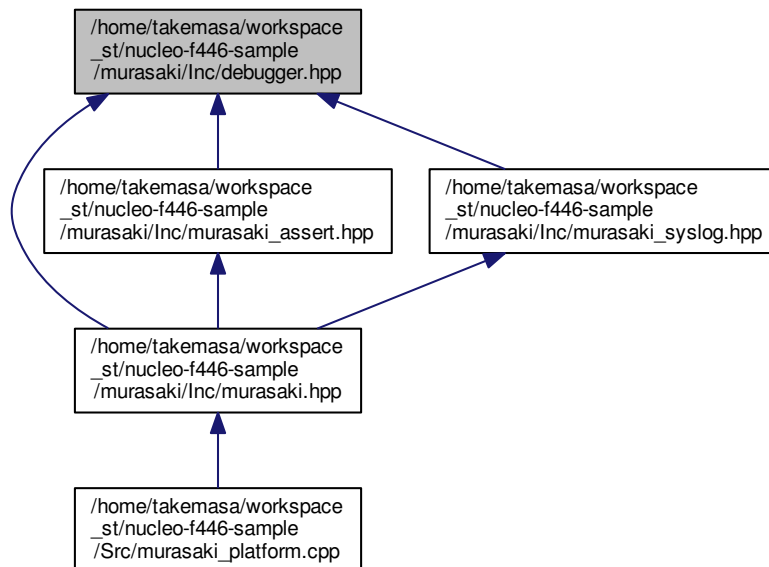
## 14.16 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/inc/debugger.hpp File Reference

```
#include <FreeRTOS.h>
#include <loggerstrategy.hpp>
#include <task.h>
#include <semphr.h>
#include "murasaki_config.hpp"
#include "debuggerfifo.hpp"
#include "task.hpp"
```

Include dependency graph for debugger.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::Debugger](#)

## Namespaces

- [murasaki](#)

## Variables

- Debugger \* [murasaki::debugger](#)

### 14.16.1 Detailed Description

#### Date

2018/01/03

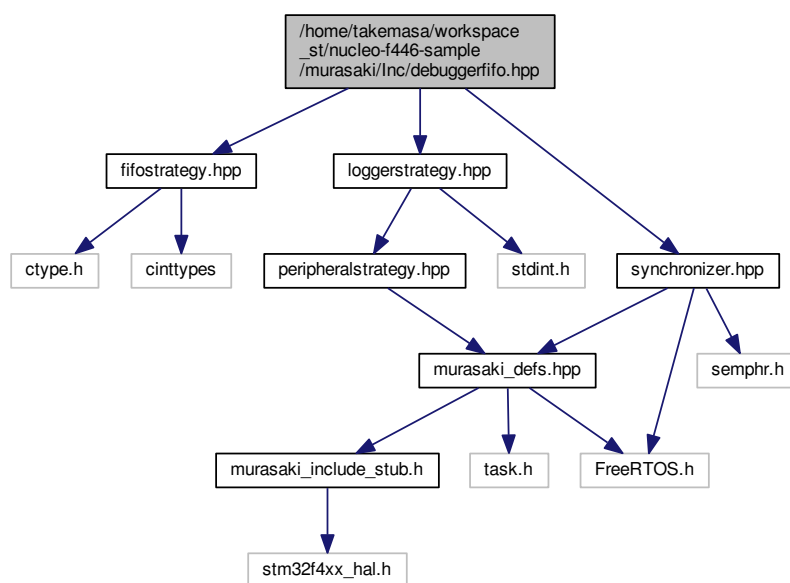
#### Author

Seiichi "Suikan" Horie

This class serves printf function for both task context and ISR context.

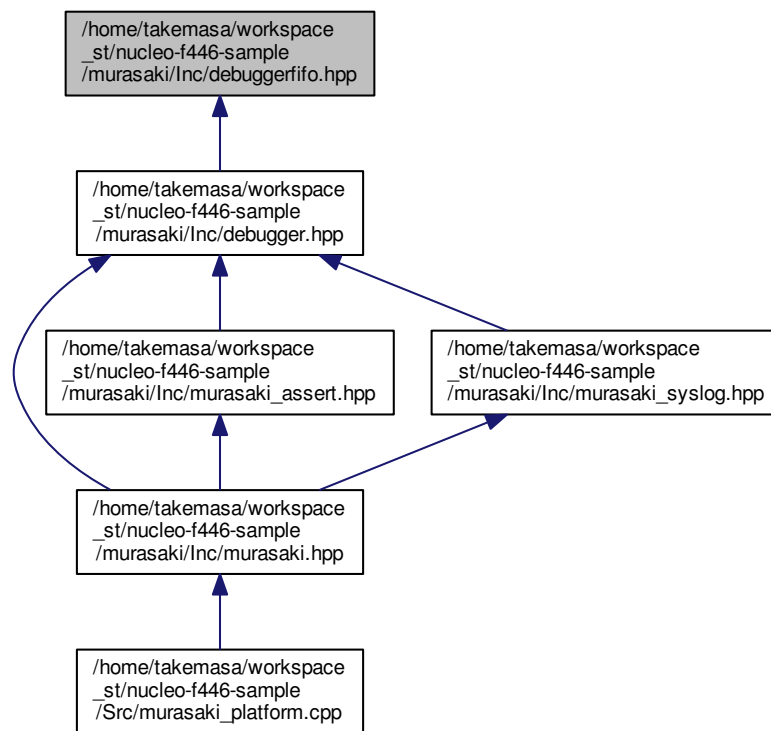
## 14.17 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/inc/debuggerfifo.hpp File Reference

```
#include <fifostrategy.hpp>
#include <loggerstrategy.hpp>
#include "synchronizer.hpp"
Include dependency graph for debuggerfifo.hpp:
```





This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::DebuggerFifo](#)
- struct [murasaki::LoggingHelpers](#)

## Namespaces

- [murasaki](#)

### 14.17.1 Detailed Description

#### Date

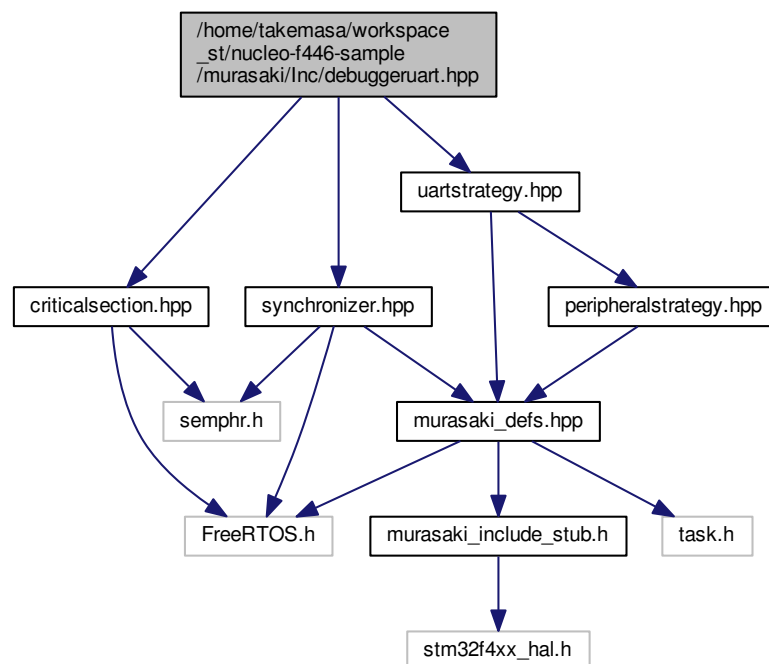
2018/03/01

#### Author

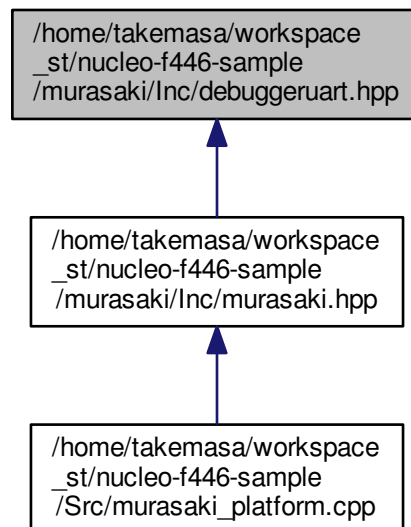
Seiichi "Suikan" Horie

## 14.18 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/debuggeruart.hpp File Reference

```
#include <synchronizer.hpp>
#include <uartstrategy.hpp>
#include "criticalsection.hpp"
Include dependency graph for debuggeruart.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::DebuggerUart](#)

## Namespaces

- [murasaki](#)

### 14.18.1 Detailed Description

#### Date

2018/09/23

#### Author

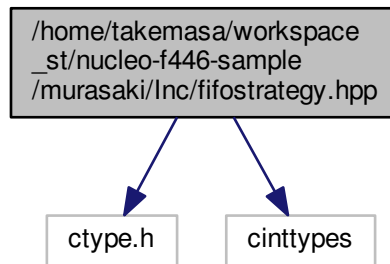
Seiichi "Suikan" Horie

## 14.19 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/fifostrategy.hpp File Reference

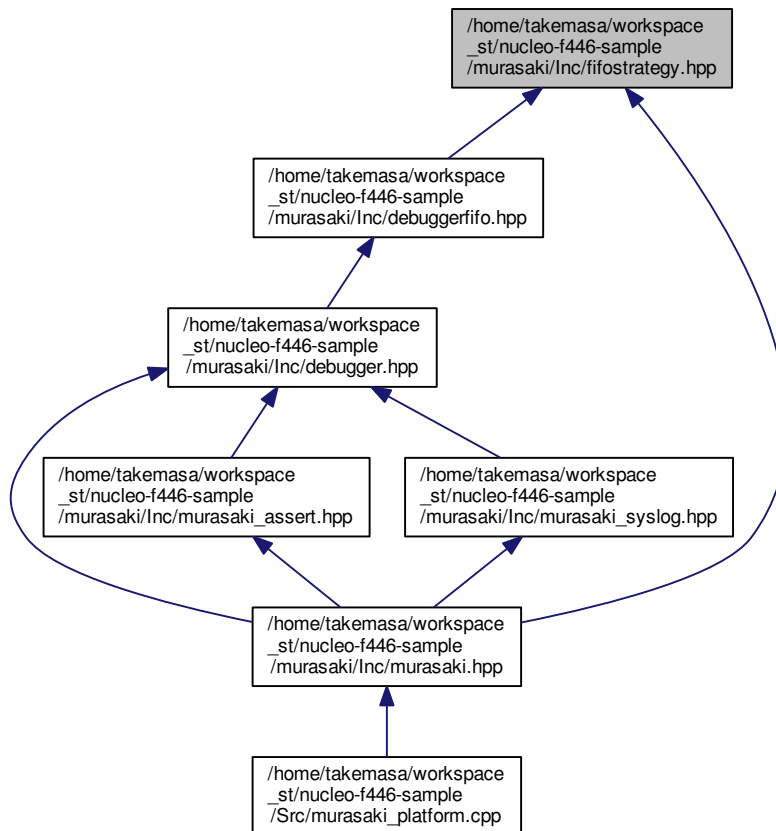
```
#include <ctype.h>
```

```
#include <cinttypes>
```

Include dependency graph for fifostrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::FifoStrategy](#)

## Namespaces

- [murasaki](#)

### 14.19.1 Detailed Description

#### Date

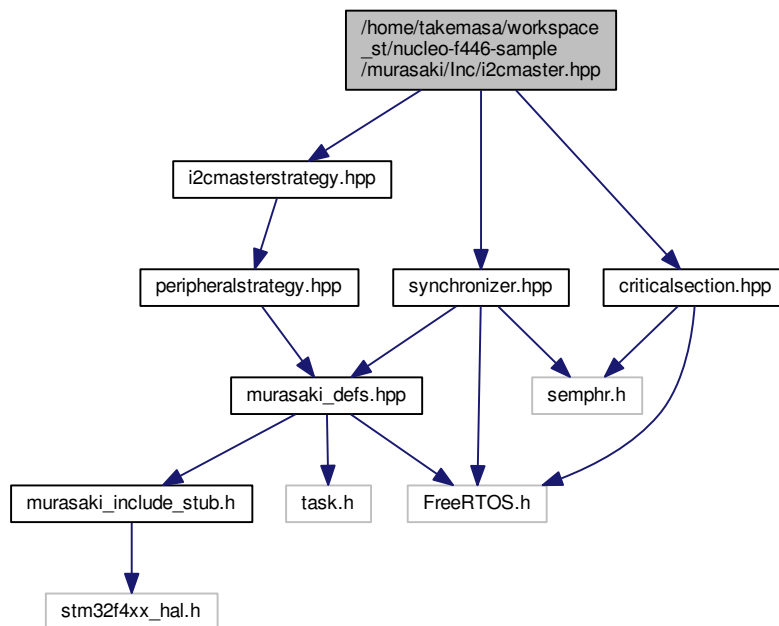
2018/02/26

#### Author

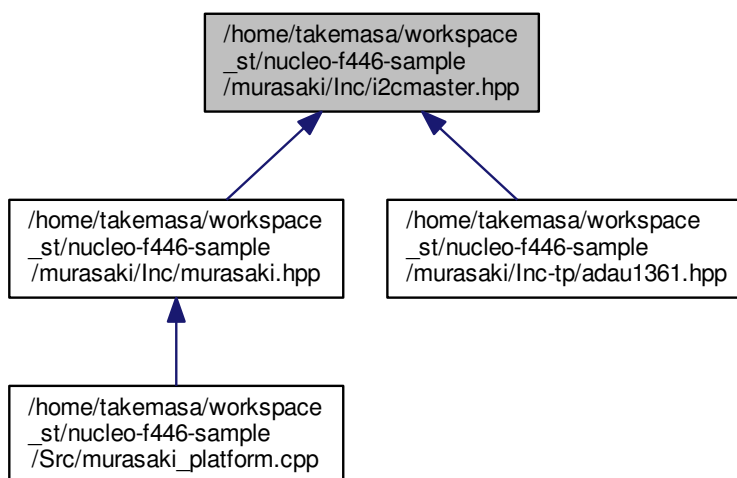
Seiichi "Suikan" Horie

## 14.20 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/i2cmaster.hpp File Reference

```
#include <i2cmasterstrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
Include dependency graph for i2cmaster.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::I2cMaster](#)

## Namespaces

- [murasaki](#)

### 14.20.1 Detailed Description

#### Date

2018/02/12

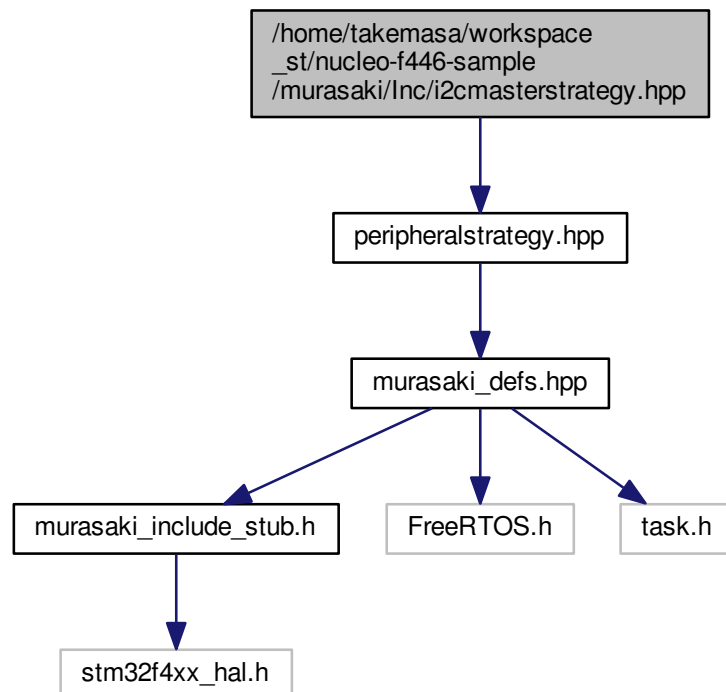
#### Author

: Seiichi "Suikan" Horie

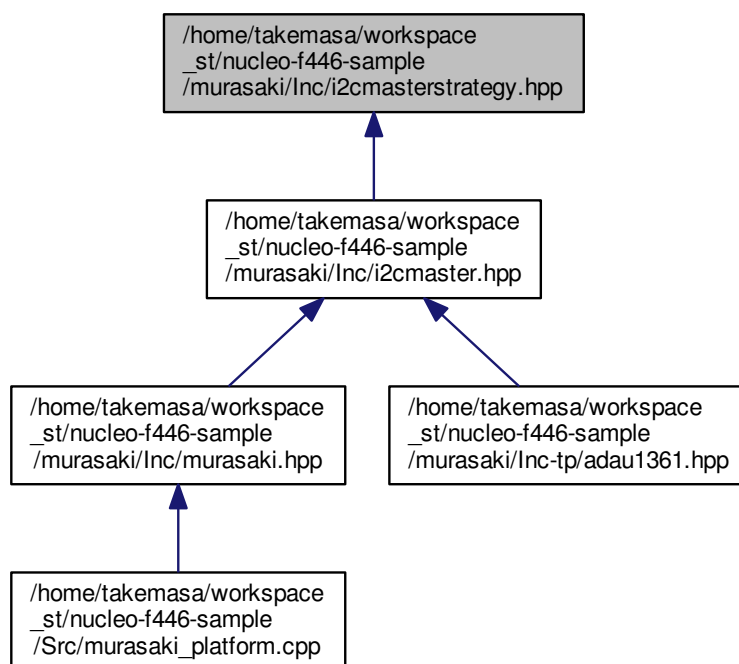
## 14.21 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/i2cmasterstrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
```

Include dependency graph for i2cmasterstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::I2CMasterStrategy](#)

## Namespaces

- [murasaki](#)

### 14.21.1 Detailed Description

#### Date

2018/02/11

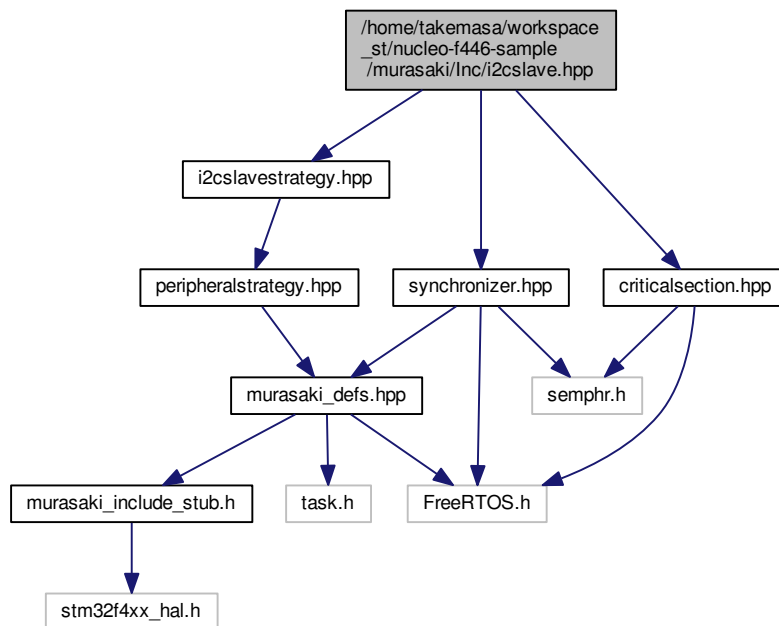
#### Author

: Seiichi "Suikan" Horie

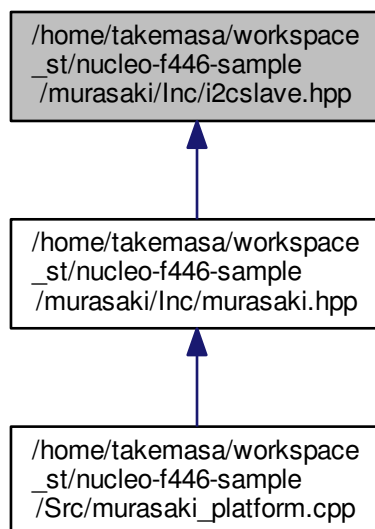


## 14.22 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/i2cslave.hpp File Reference

```
#include <i2cslavestrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
Include dependency graph for i2cslave.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::I2cSlave](#)

## Namespaces

- [murasaki](#)

### 14.22.1 Detailed Description

#### Date

2018/10/07

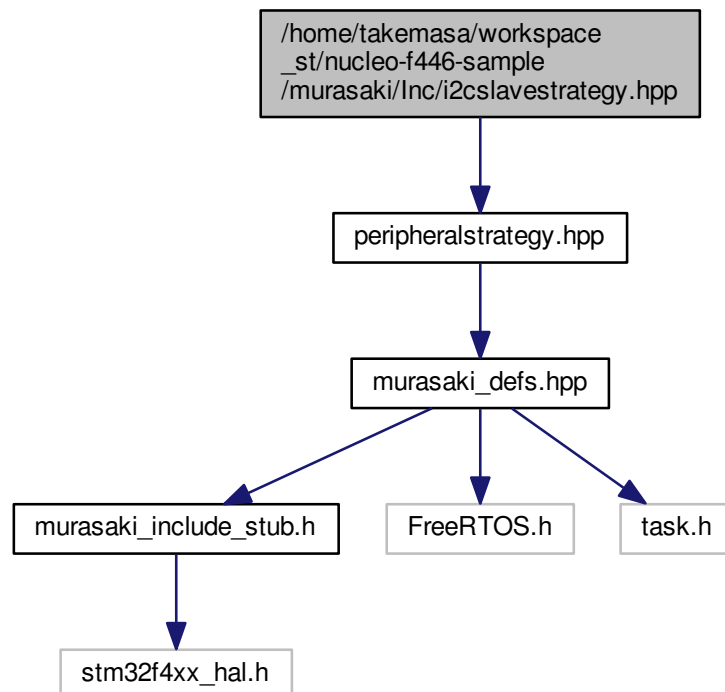
#### Author

Seiichi "Suikan" Horie

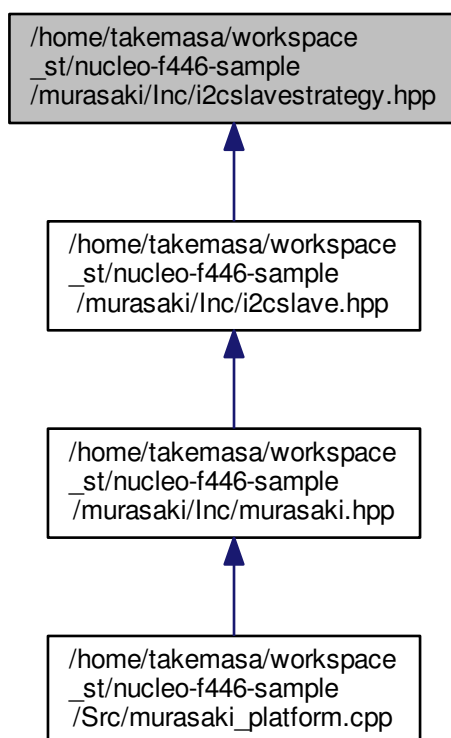
## 14.23 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/i2cslavestrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
```

Include dependency graph for i2cslavestrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::I2cSlaveStrategy](#)

## Namespaces

- [murasaki](#)

### 14.23.1 Detailed Description

#### Date

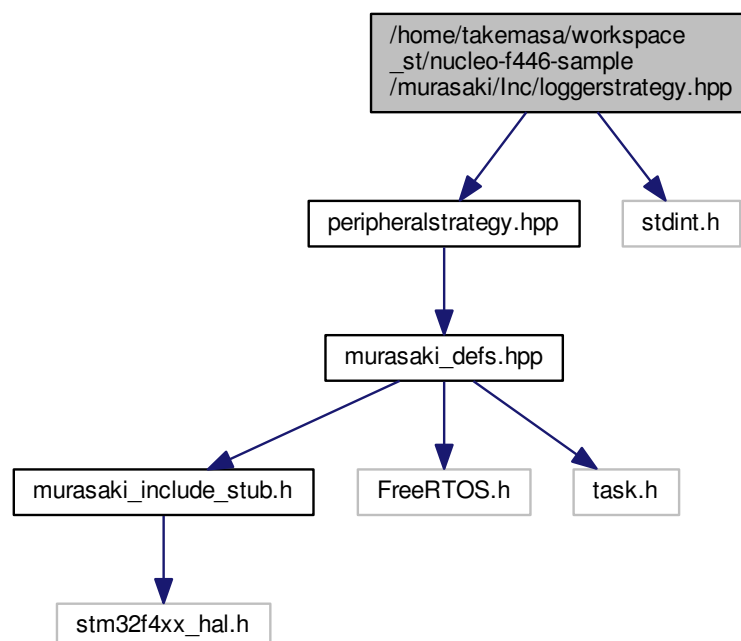
2018/10/07

#### Author

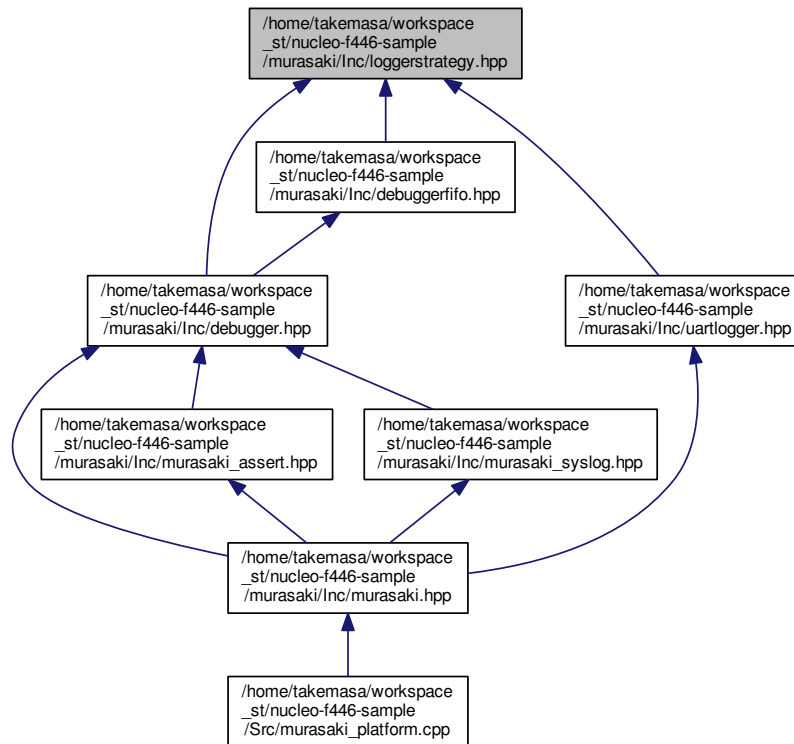
Seiichi "Suikan" Horie

## 14.24 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/loggerstrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
#include <stdint.h>
Include dependency graph for loggerstrategy.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::LoggerStrategy](#)

## Namespaces

- [murasaki](#)

### 14.24.1 Detailed Description

#### Date

2018/01/20

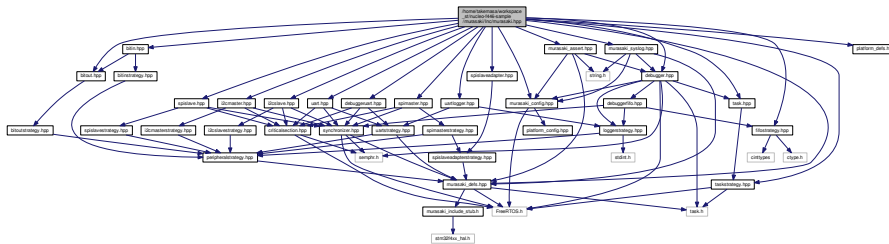
#### Author

: Seiichi "Suikan" Horie

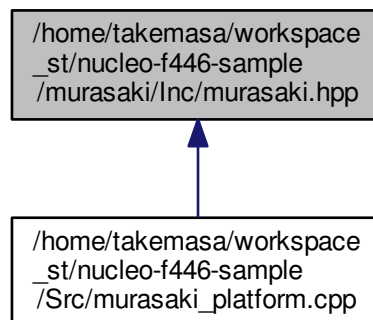
## 14.25 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/inc/murasaki.hpp File Reference

```
#include <debugger.hpp>
#include <fifostrategy.hpp>
#include <taskstrategy.hpp>
#include "murasaki_config.hpp"
#include "murasaki_defs.hpp"
#include "task.hpp"
#include "uart.hpp"
#include "debuggeruart.hpp"
#include "spimaster.hpp"
#include "spislave.hpp"
#include "spislaveadapter.hpp"
#include "i2cmaster.hpp"
#include "i2cslave.hpp"
#include "bitin.hpp"
#include "bitout.hpp"
#include "uartlogger.hpp"
#include "murasaki_assert.hpp"
#include "murasaki_syslog.hpp"
#include "platform_defs.hpp"
```

Include dependency graph for murasaki.hpp:



This graph shows which files directly or indirectly include this file:



### 14.25.1 Detailed Description

**Date**

2018/01/21

**Author**

Seiichi "Suikan" Horie

Application can include only this file. Other essential header files are automatically included from this file.

## 14.26 `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/inc/murasaki_0_` `intro.hpp` File Reference

### 14.26.1 Detailed Description

**Date**

2018/02/01

**Author**

Seiichi "Suikan" Horie

## 14.27 `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/inc/murasaki_1_` `env.hpp` File Reference

### 14.27.1 Detailed Description

**Date**

2018/02/01

**Author**

Seiichi "Suikan" Horie

## 14.28 `/home/takemasa/workspace_st/nucleo-f446-sample/murasaki/inc/murasaki_2_` `ug.hpp` File Reference

### 14.28.1 Detailed Description

**Date**

2018/02/01

**Author**

Seiichi "Suikan" Horie



## 14.29 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/murasaki\_3\_pg.hpp File Reference

### 14.29.1 Detailed Description

#### Date

May 25, 2018

#### Author

Seiichi "Suikan" Horie

## 14.30 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/murasaki\_4\_mod.hpp File Reference

### 14.30.1 Detailed Description

#### Date

May 25, 2018

#### Author

Seiichi "Suikan" Horie

## 14.31 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/murasaki\_5\_spg.hpp File Reference

## 14.32 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/murasaki\_assert.hpp File Reference

```
#include <debugger.hpp>
#include "murasaki_config.hpp"
#include "murasaki_defs.hpp"
#include <string.h>
```



### 14.32.1 Detailed Description

#### Date

2018/01/31

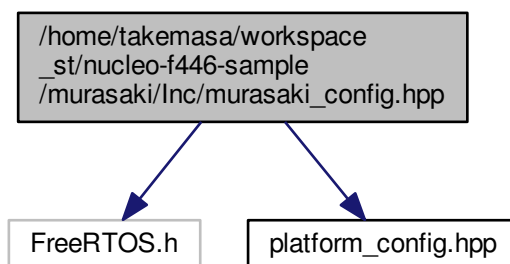
#### Author

Seiichi "Suikan" Horie

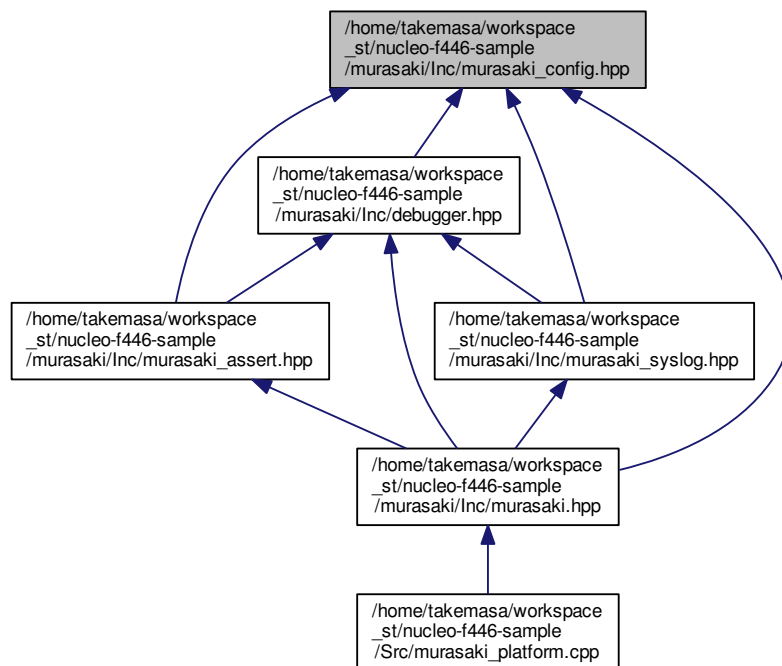
## 14.33 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/murasaki\_config.hpp File Reference

```
#include <FreeRTOS.h>
#include <platform_config.hpp>
```

Include dependency graph for murasaki\_config.hpp:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define PLATFORM_CONFIG_DEBUG_LINE_SIZE 256`
- `#define PLATFORM_CONFIG_DEBUG_BUFFER_SIZE 4096`
- `#define PLATFORM_CONFIG_DEBUG_SERIAL_TIMEOUT (murasaki::kwmsIndefinitely)`
- `#define PLATFORM_CONFIG_DEBUG_TASK_STACK_SIZE 256`
- `#define PLATFORM_CONFIG_DEBUG_TASK_PRIORITY (( configMAX_PRIORITIES-1 > 0 ) ? configMAX_PRIORITIES-1 : 0)`
- `#define MURASAKI_CONFIG_NODEBUG false`

### 14.33.1 Detailed Description

#### Date

2018/01/03

#### Author

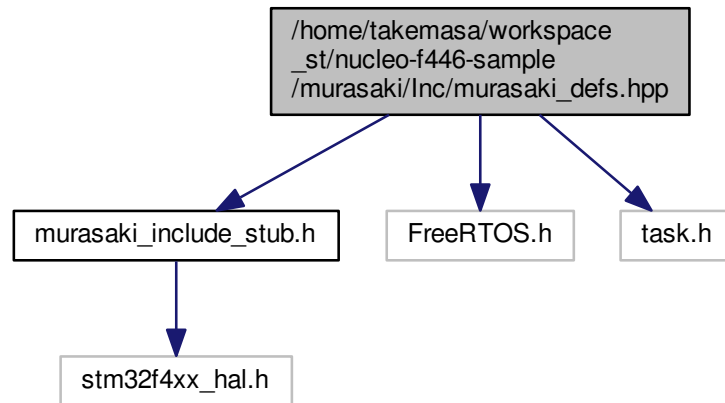
Seiichi "Suikan" Horie

To override the configuration, define the same name macro inside application\_config.hpp

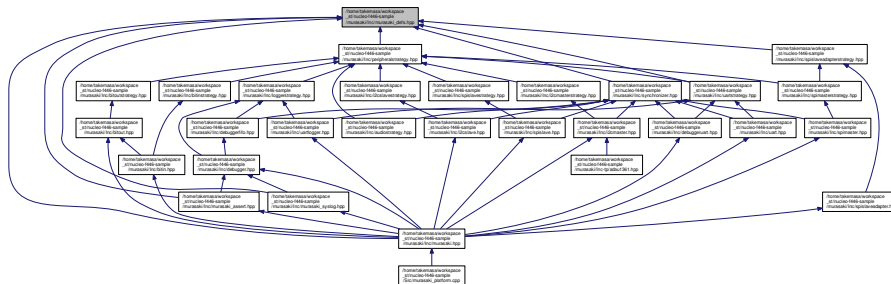
## 14.34 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/murasaki\_defs.hpp File Reference

```
#include "murasaki_include_stub.h"  
#include <FreeRTOS.h>  
#include <task.h>
```

Include dependency graph for murasaki\_defs.hpp:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [murasaki](#)

### 14.34.1 Detailed Description

Date

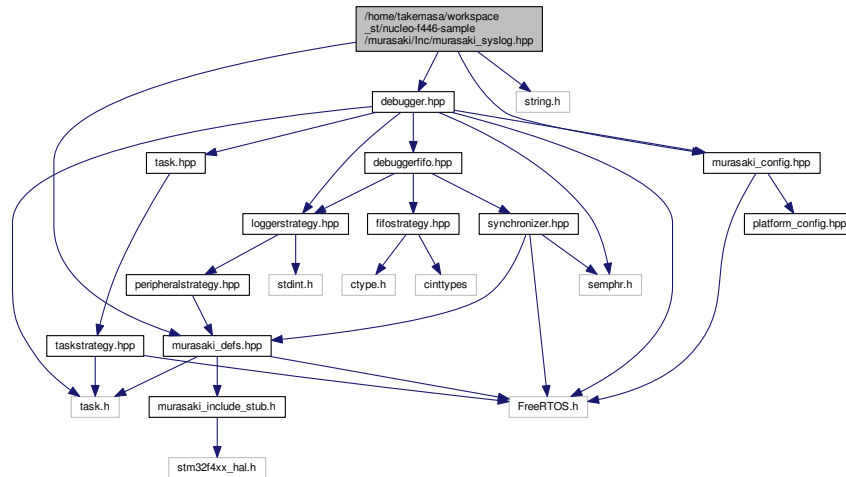
2017/11/05

Author

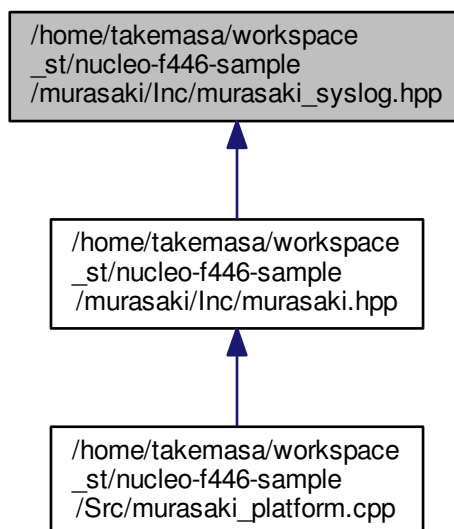
Seiichi "Suikan" Horie

## 14.35 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/murasaki\_ syslog.hpp File Reference

```
#include <debugger.hpp>
#include "murasaki_config.hpp"
#include "murasaki_defs.hpp"
#include "string.h"
Include dependency graph for murasaki_syslog.hpp:
```



This graph shows which files directly or indirectly include this file:



## Namespaces

- [murasaki](#)

## Macros

- #define [MURASAKI\\_SYSLOG](#)(FACILITY, SEVERITY, FORMAT, ...)

## Functions

- void [murasaki::SetSyslogSererityThreshold](#) ([murasaki::SyslogSeverity](#) severity)
- void [murasaki::SetSyslogFacilityMask](#) (uint32\_t mask)
- void [murasaki::AddSyslogFacilityToMask](#) ([murasaki::SyslogFacility](#) facility)
- void [murasaki::RemoveSyslogFacilityFromMask](#) ([murasaki::SyslogFacility](#) facility)
- bool [murasaki::AllowedSyslogOut](#) ([murasaki::SyslogFacility](#) facility, [murasaki::SyslogSeverity](#) severity)

### 14.35.1 Detailed Description

#### Date

2018/09/01

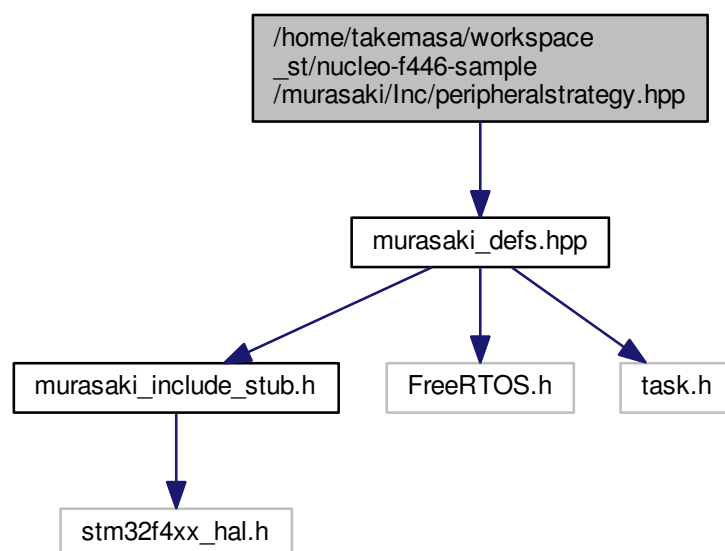
#### Author

Seiichi "Suikan" Horie

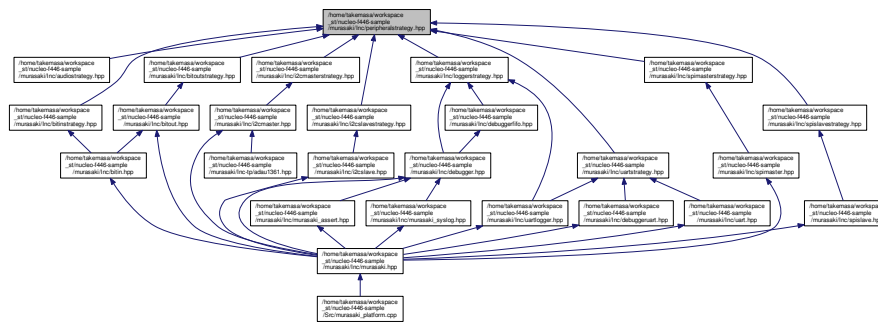
## 14.36 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/peripheralstrategy.hpp File Reference

```
#include "murasaki_defs.hpp"
```

Include dependency graph for peripheralstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::PeripheralStrategy](#)

## Namespaces

- [murasaki](#)

### 14.36.1 Detailed Description

#### Date

2018/04/26

#### Author

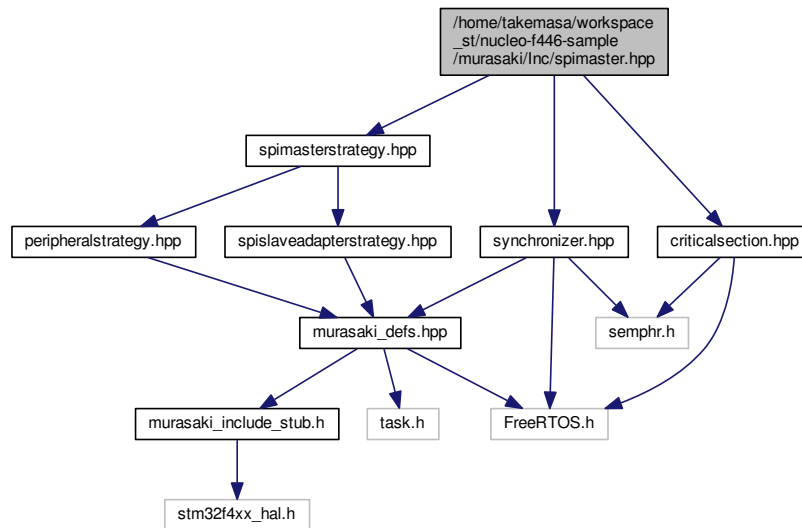
: Seiichi "Suikan" Horie

## 14.37 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/inc/spimaster.hpp File Reference

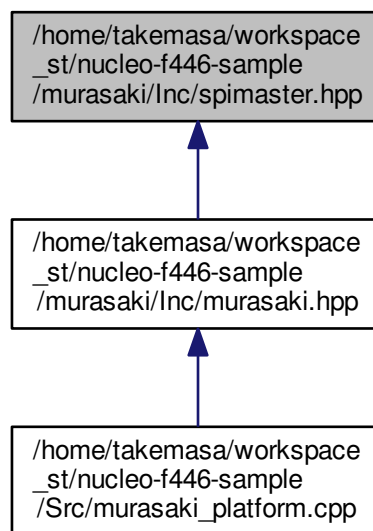
```
#include <spimasterstrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
```



Include dependency graph for spimaster.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `murasaki::SpiMaster`

## Namespaces

- [murasaki](#)

### 14.37.1 Detailed Description

#### Date

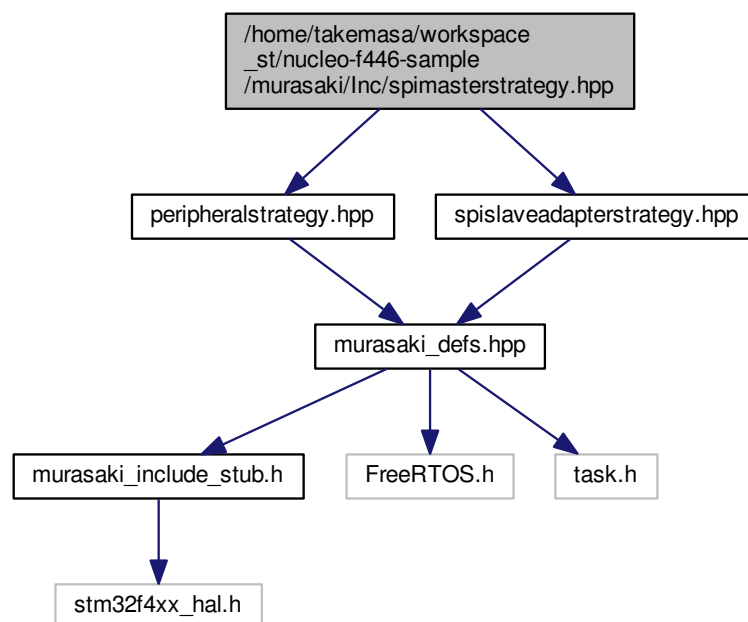
2018/02/14

#### Author

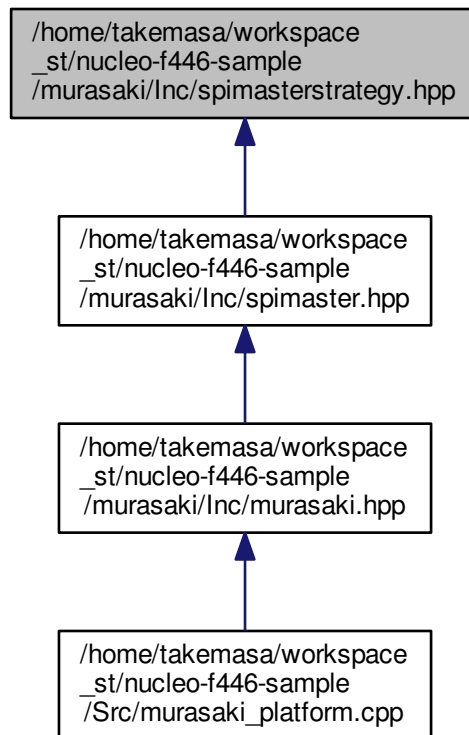
Seiichi "Suikan" Horie

## 14.38 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/spimasterstrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
#include <spislaveadapterstrategy.hpp>
Include dependency graph for spimasterstrategy.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::SpiMasterStrategy](#)

## Namespaces

- [murasaki](#)

### 14.38.1 Detailed Description

#### Date

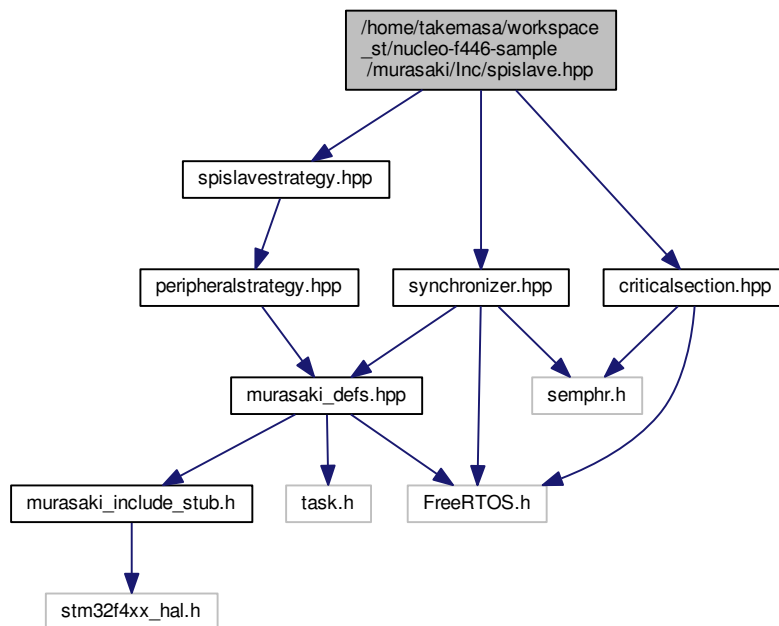
2018/02/11

#### Author

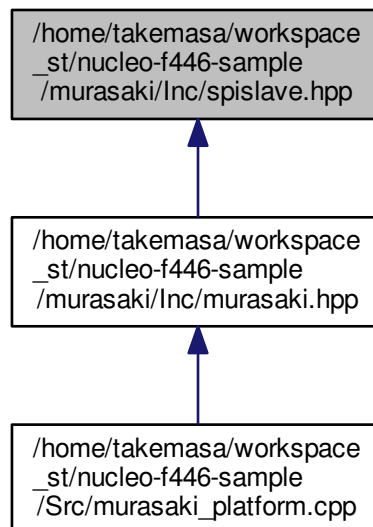
: Seiichi "Suikan" Horie

## 14.39 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/spislave.hpp File Reference

```
#include <spislavestrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
Include dependency graph for spislave.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::SpiSlave](#)

## Namespaces

- [murasaki](#)

### 14.39.1 Detailed Description

#### Date

2018/02/14

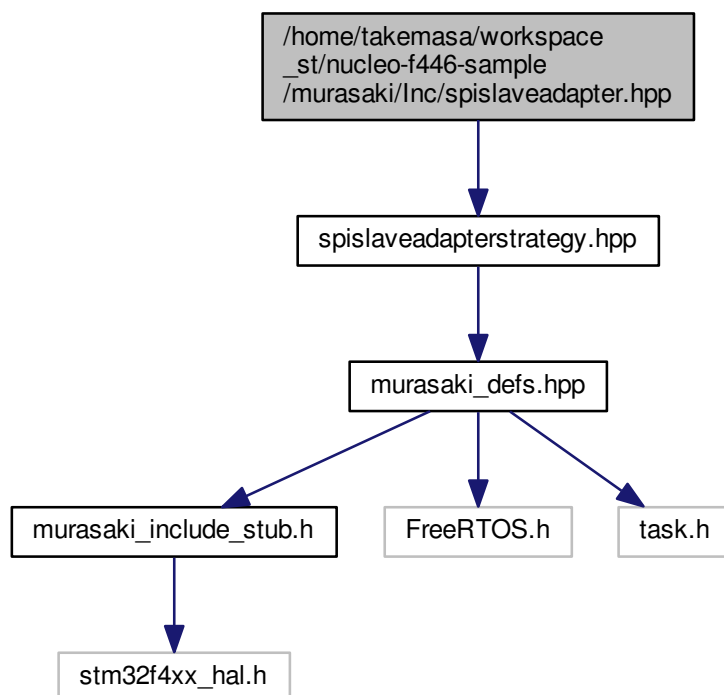
#### Author

Seiichi "Suikan" Horie

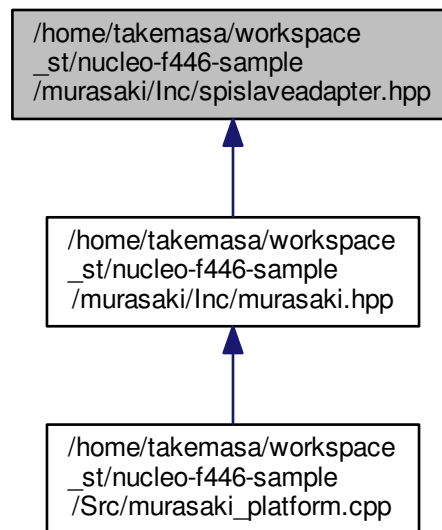
## 14.40 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/spislaveadapter.hpp File Reference

```
#include <spislaveadapterstrategy.hpp>
```

Include dependency graph for spislaveadapter.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::SpiSlaveAdapter](#)

## Namespaces

- [murasaki](#)

### 14.40.1 Detailed Description

#### Date

2018/02/17

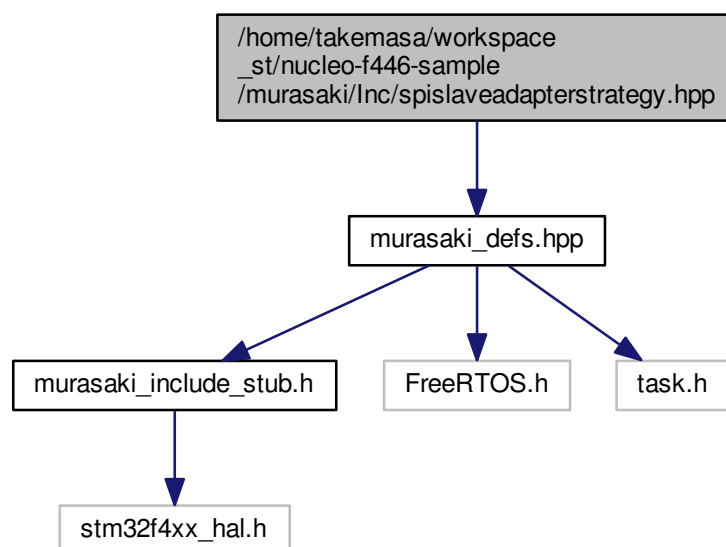
#### Author

Seiichi "Suikan" Horie

#### 14.41 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/spislaveadapterstrategy.hpp File Reference

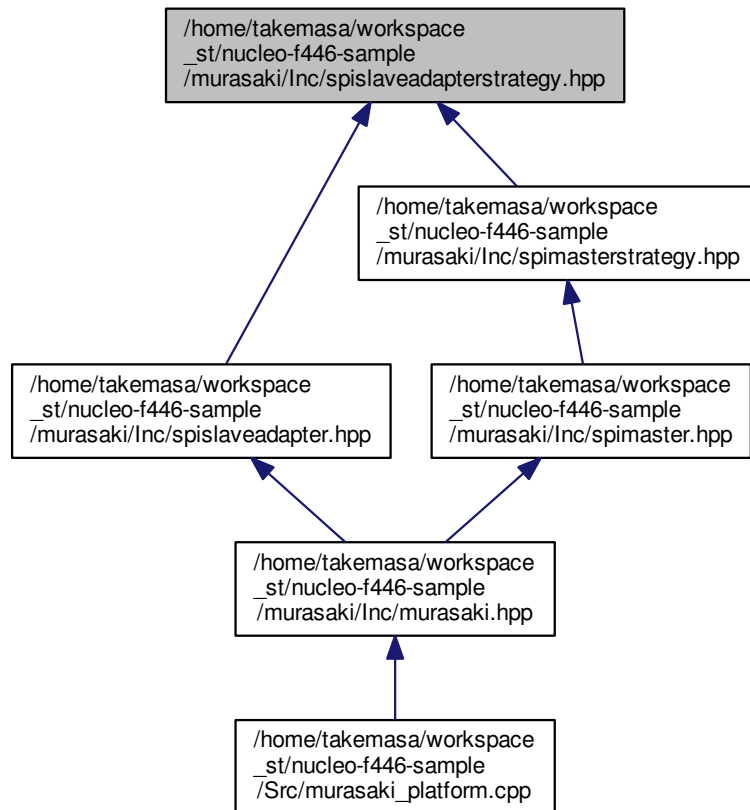
```
#include "murasaki_defs.hpp"
```

Include dependency graph for spislaveadapterstrategy.hpp:





This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::SpiSlaveAdapterStrategy](#)

## Namespaces

- [murasaki](#)

### 14.41.1 Detailed Description

#### Date

2018/02/11

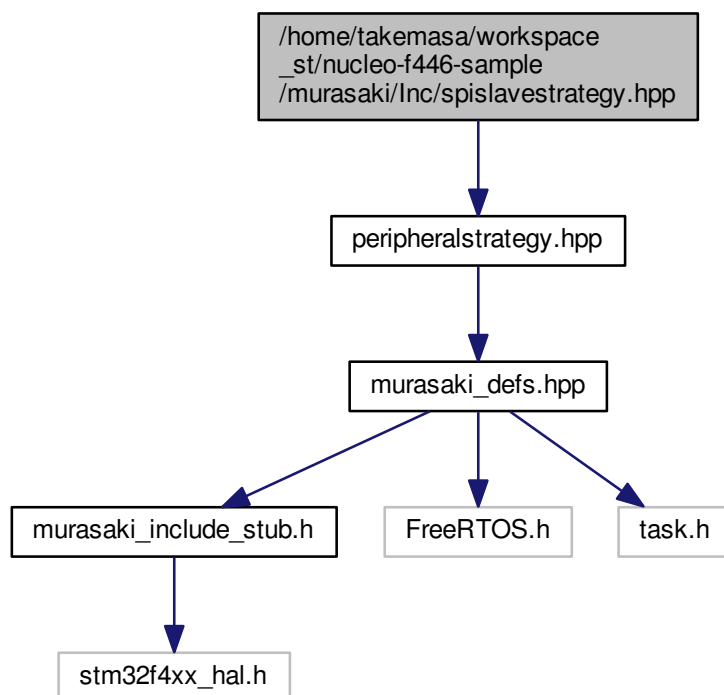
#### Author

: Seiichi "Suikan" Horie

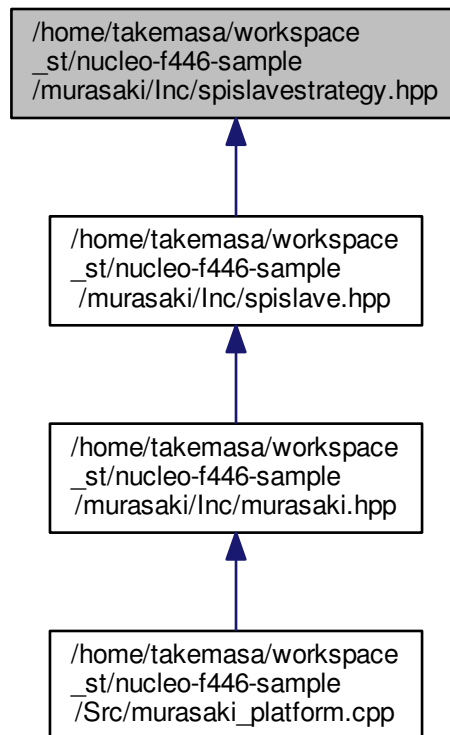
## 14.42 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/spislavestrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
```

Include dependency graph for spislavestrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::SpiSlaveStrategy](#)

## Namespaces

- [murasaki](#)

### 14.42.1 Detailed Description

#### Date

2018/02/11

#### Author

: Seiichi "Suikan" Horie



## Namespaces

- [murasaki](#)

### 14.43.1 Detailed Description

#### Date

2018/01/26

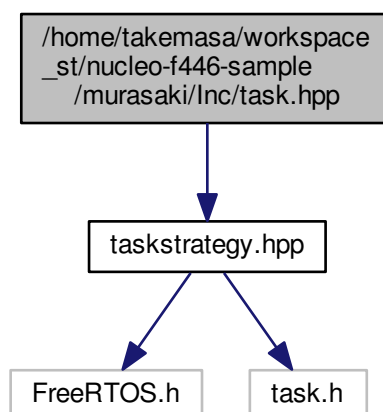
#### Author

Seiichi "Suikan" Horie

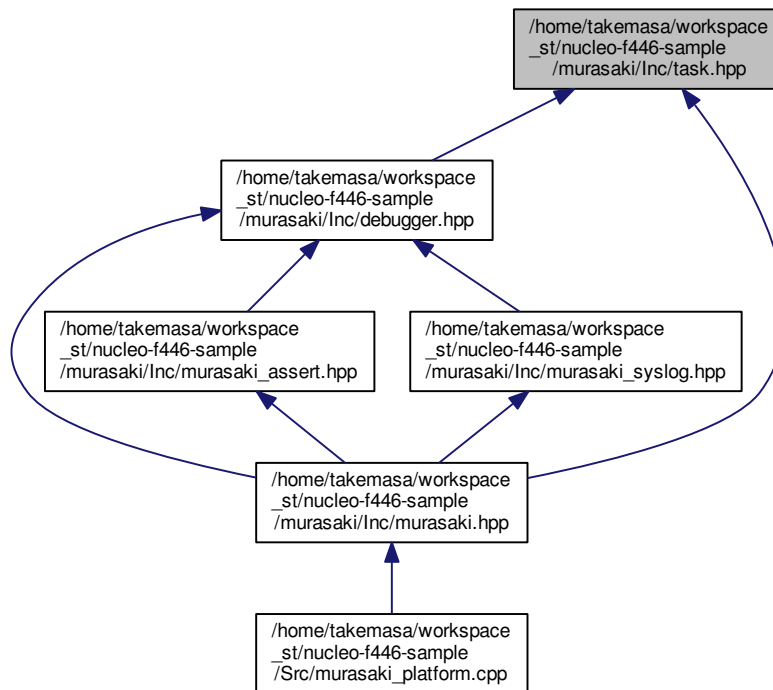
## 14.44 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/task.hpp File Reference

```
#include <taskstrategy.hpp>
```

Include dependency graph for task.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::Task](#)

## Namespaces

- [murasaki](#)

### 14.44.1 Detailed Description

#### Date

2019/02/03

#### Author

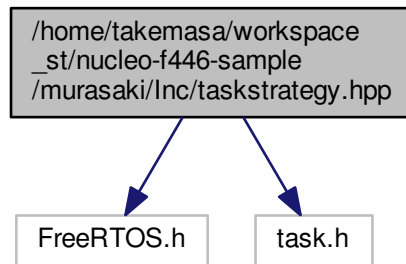
Seiichi "Suikan" Horie

## 14.45 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/lnc/taskstrategy.hpp File Reference

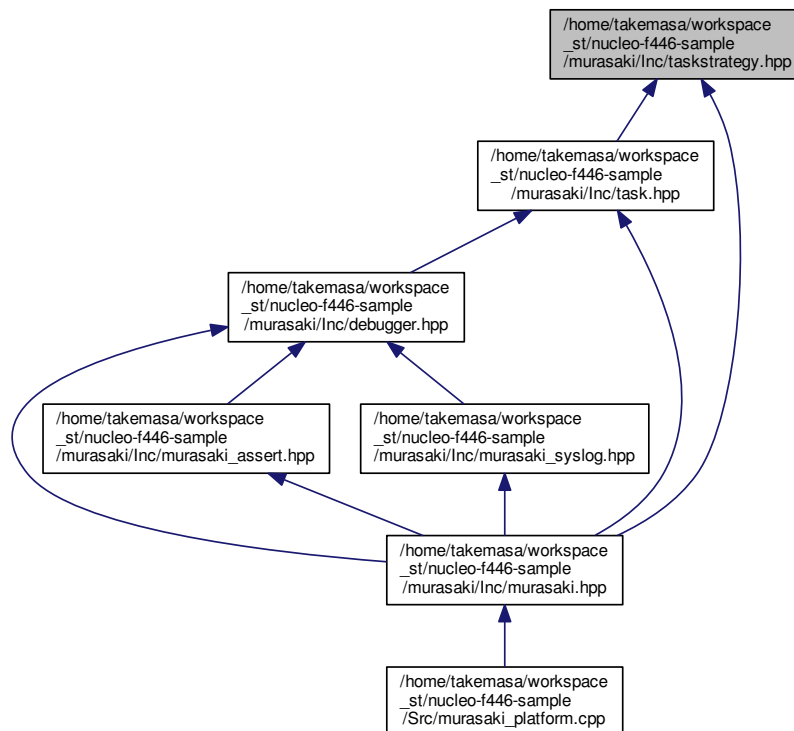
```
#include <FreeRTOS.h>
```

```
#include <task.h>
```

Include dependency graph for taskstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::TaskStrategy](#)

## Namespaces

- [murasaki](#)

### 14.45.1 Detailed Description

#### Date

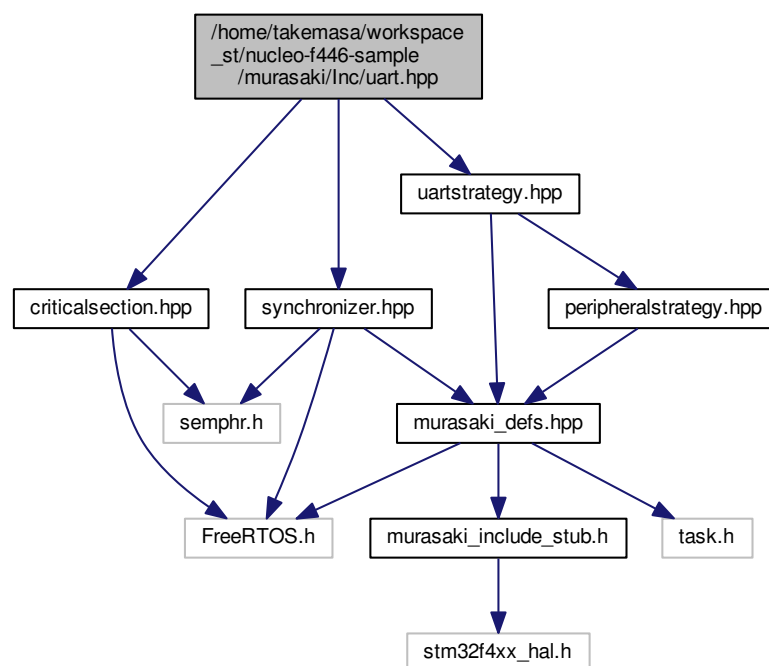
2018/02/20

#### Author

: Seiichi "Suikan" Horie

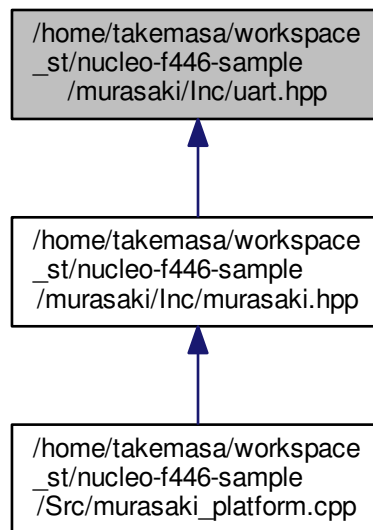
### 14.46 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/uart.hpp File Reference

```
#include <synchronizer.hpp>
#include <uartstrategy.hpp>
#include "criticalsection.hpp"
Include dependency graph for uart.hpp:
```





This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::Uart](#)

## Namespaces

- [murasaki](#)

### 14.46.1 Detailed Description

#### Date

2017/11/05

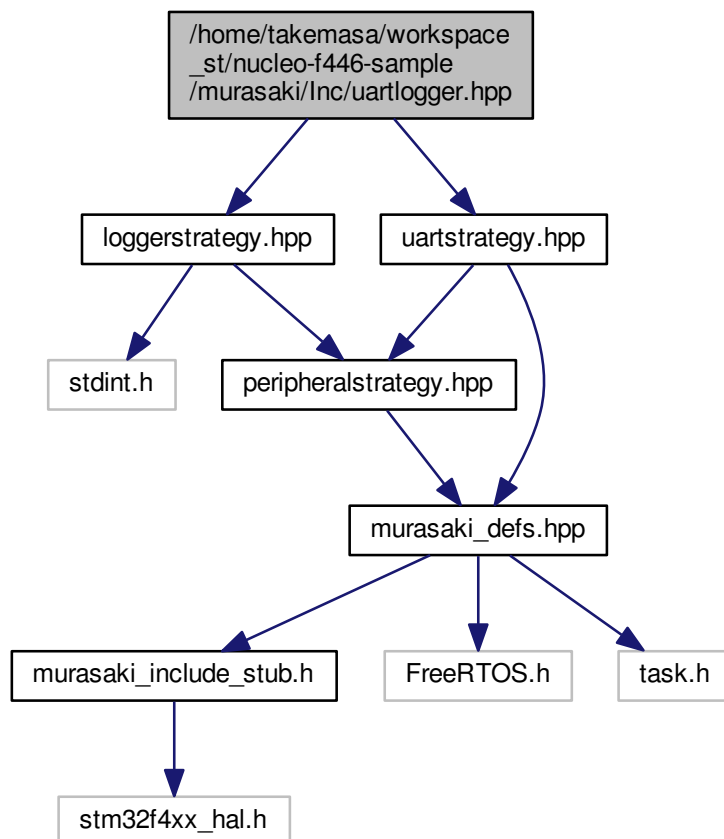
#### Author

Seiichi "Suikan" Horie

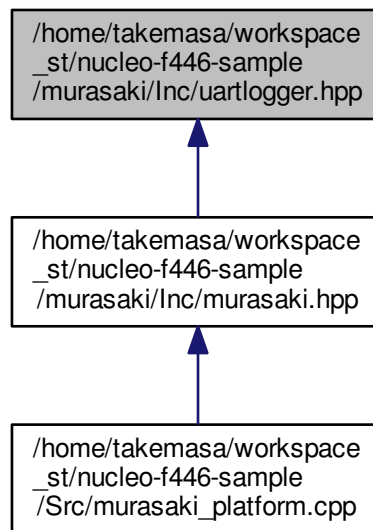
## 14.47 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/uartlogger.hpp File Reference

```
#include <loggerstrategy.hpp>
#include <uartstrategy.hpp>
```

Include dependency graph for uartlogger.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::UartLogger](#)

## Namespaces

- [murasaki](#)

### 14.47.1 Detailed Description

#### Date

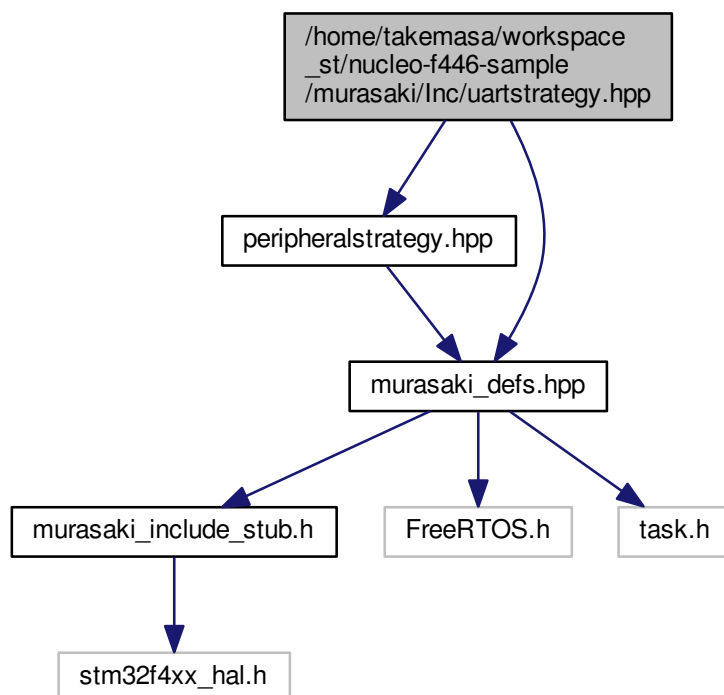
2018/01/20

#### Author

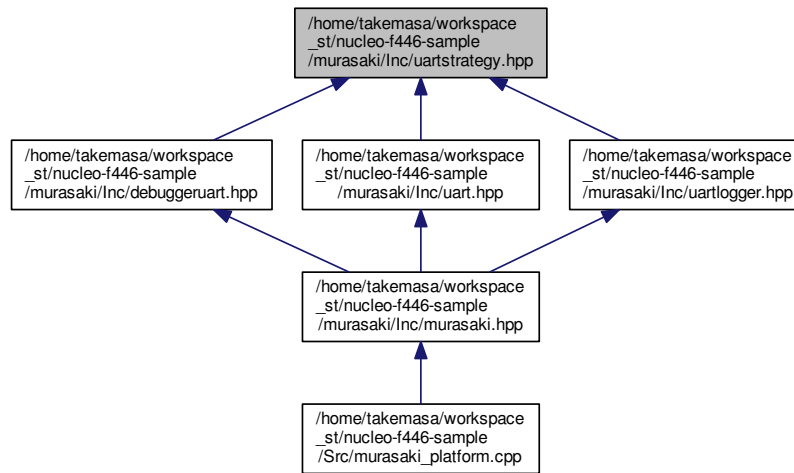
: Seiichi "Suikan" Horie

## 14.48 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Inc/uartstrategy.hpp File Reference

```
#include <peripheralstrategy.hpp>
#include "murasaki_defs.hpp"
Include dependency graph for uartstrategy.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [murasaki::UartStrategy](#)

## Namespaces

- [murasaki](#)

### 14.48.1 Detailed Description

#### Date

2017/11/04

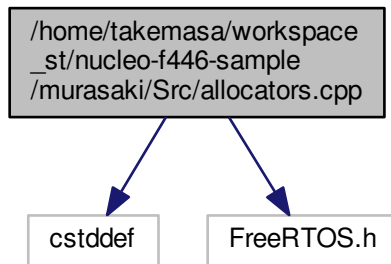
#### Author

: Seiichi "Suikan" Horie

## 14.49 /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/Src/allocators.cpp File Reference

```
#include <cstdint>
#include <FreeRTOS.h>
```

Include dependency graph for allocators.cpp:



## Functions

- void \* `operator new` (std::size\_t size)
- void \* `operator new[]` (std::size\_t size)
- void `operator delete` (void \*ptr)
- void `operator delete[]` (void \*ptr)

### 14.49.1 Detailed Description

#### Date

2018/05/02

#### Author

Seiichi "Suikan" Horie

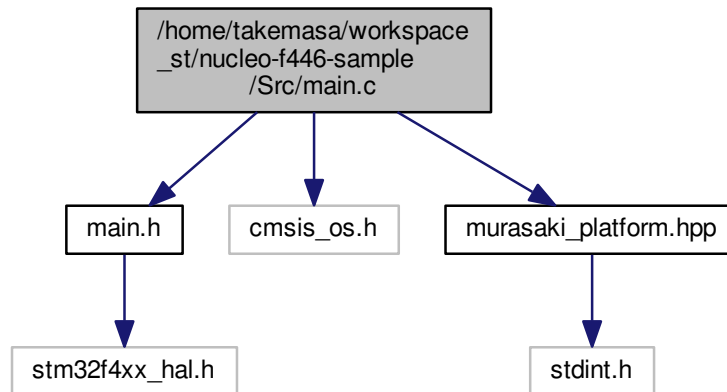
These definitions allows to used the FreeRTOS's heap instead of the system heap.

The system heap by the standard library doesn't check the limit of the heap cerefly. As a result, it is not clear how to detect the over committing memory.

FreeRTOS hepa is considered safer than system heap. Then, the new and the delete operators are overloaded to use the `pvPortMalloc()`.

## 14.50 /home/takemasa/workspace\_st/nucleo-f446-sample/Src/main.c File Reference

```
#include "main.h"  
#include "cmsis_os.h"  
#include "murasaki_platform.hpp"  
Include dependency graph for main.c:
```



### Functions

- void [SystemClock\\_Config](#) (void)
- void [StartDefaultTask](#) (void const \*argument)
- int [main](#) (void)
- void [HAL\\_TIM\\_PeriodElapsedCallback](#) (TIM\_HandleTypeDef \*htim)
- void [Error\\_Handler](#) (void)
- void [assert\\_failed](#) (uint8\_t \*file, uint32\_t line)

### Variables

- DMA\_HandleTypeDef [hdma\\_usart2\\_rx](#)

#### 14.50.1 Detailed Description

##### Attention

© Copyright (c) 2019 STMicroelectronics. All rights reserved.

This software component is licensed by ST under Ultimate Liberty license SLA0044, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: [www.st.com/SLA0044](http://www.st.com/SLA0044)

## 14.50.2 Function Documentation

### 14.50.2.1 void assert\_failed ( uint8\_t \* file, uint32\_t line )

Reports the name of the source file and the source line number where the assert\_param error has occurred.

#### Parameters

<i>file</i>	pointer to the source file name
<i>line</i>	assert_param error line source number

#### Return values

<i>None</i>	
-------------	--

### 14.50.2.2 void Error\_Handler ( void )

This function is executed in case of error occurrence.

#### Return values

<i>None</i>	
-------------	--

### 14.50.2.3 void HAL\_TIM\_PeriodElapsedCallback ( TIM\_HandleTypeDef \* htim )

Period elapsed callback in non blocking mode.

#### Note

This function is called when TIM14 interrupt took place, inside HAL\_TIM\_IRQHandler(). It makes a direct call to HAL\_IncTick() to increment a global variable "uwTick" used as application time base.

#### Parameters

<i>htim</i>	: TIM handle
-------------	--------------

#### Return values

<i>None</i>	
-------------	--

### 14.50.2.4 int main ( void )

The application entry point.



## Return values

<i>int</i>	
------------	--

14.50.2.5 void StartDefaultTask ( void const \* *argument* )

Function implementing the defaultTask thread.

## Parameters

<i>argument</i>	Not used
-----------------	----------

## Return values

<i>None</i>	
-------------	--

## 14.50.2.6 void SystemClock\_Config ( void )

System Clock Configuration.

## Return values

<i>None</i>	
-------------	--

Configure the main internal regulator output voltage

Initializes the CPU, AHB and APB busses clocks

Activate the Over-Drive mode

Initializes the CPU, AHB and APB busses clocks

## 14.50.3 Variable Documentation

## 14.50.3.1 DMA\_HandleTypeDef hdma\_usart2\_rx

File Name : stm32f4xx\_hal\_msp.c Description : This file provides code for the MSP Initialization and de-Initialization codes.

## Attention

© Copyright (c) 2019 STMicroelectronics. All rights reserved.

This software component is licensed by ST under Ultimate Liberty license SLA0044, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: [www.st.com/SLA0044](http://www.st.com/SLA0044)



## Parameters

<i>hi2c</i>	
-------------	--

This is called from inside of HAL when an I2C receive done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default RX interrupt call back.

In this call back, the uart device handle have to be passed to the [murasaki::Uart::ReceiveCompleteCallback\(\)](#) function.

14.51.2.2 void HAL\_I2C\_SlaveRxCpltCallback ( I2C\_HandleTypeDef \* *hi2c* )

Essential to sync up with I2C.

## Parameters

<i>hi2c</i>	
-------------	--

This is called from inside of HAL when an I2C receive done interrupt is accepted.

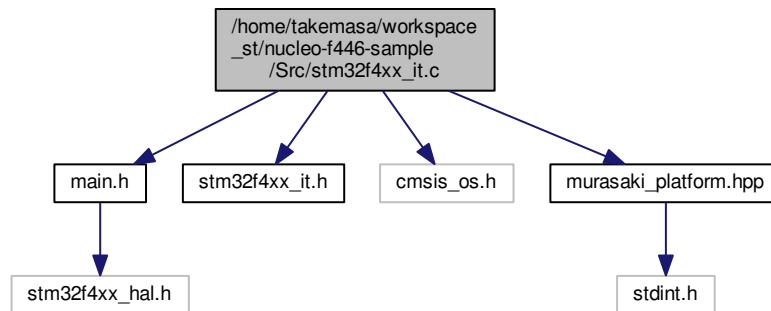
STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default RX interrupt call back.

In this call back, the I2C slave device handle have to be passed to the [murasaki::I2cSlave::ReceiveCompleteCallback\(\)](#) function.

## 14.52 /home/takemasa/workspace\_st/nucleo-f446-sample/Src/stm32f4xx\_it.c File Reference

```
#include "main.h"
#include "stm32f4xx_it.h"
#include "cmsis_os.h"
#include "murasaki_platform.hpp"
```

Include dependency graph for stm32f4xx\_it.c:



## Variables

- DMA\_HandleTypeDef [hdma\\_usart2\\_rx](#)

### 14.52.1 Detailed Description

#### Attention

© Copyright (c) 2019 STMicroelectronics. All rights reserved.

This software component is licensed by ST under Ultimate Liberty license SLA0044, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: [www.st.com/SLA0044](http://www.st.com/SLA0044)

### 14.52.2 Variable Documentation

#### 14.52.2.1 DMA\_HandleTypeDef hdma\_usart2\_rx

File Name : `stm32f4xx_hal_msp.c` Description : This file provides code for the MSP Initialization and de-Initialization codes.

#### Attention

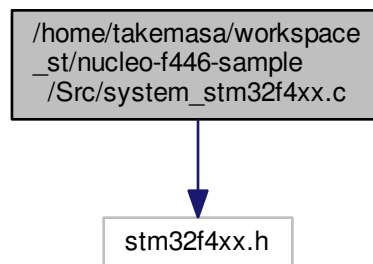
© Copyright (c) 2019 STMicroelectronics. All rights reserved.

This software component is licensed by ST under Ultimate Liberty license SLA0044, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: [www.st.com/SLA0044](http://www.st.com/SLA0044)

## 14.53 /home/takemasa/workspace\_st/nucleo-f446-sample/Src/system\_stm32f4xx.c File Reference

```
#include "stm32f4xx.h"
```

Include dependency graph for system\_stm32f4xx.c:



### Macros

- #define `HSE_VALUE` ((uint32\_t)25000000)
- #define `HSI_VALUE` ((uint32\_t)16000000)
- #define `VECT_TAB_OFFSET` 0x00

### Functions

- void `SystemInit` (void)
- void `SystemCoreClockUpdate` (void)

### 14.53.1 Detailed Description

#### Author

MCD Application Team This file provides two functions and one global variable to be called from user application:

- **SystemInit()**: This function is called at startup just after reset and before branch to main program. This call is made inside the "startup\_stm32f4xx.s" file.
- **SystemCoreClock** variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
- **SystemCoreClockUpdate()**: Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.

#### Attention

© COPYRIGHT 2017 STMicroelectronics

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Index

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Inc/main.h, 175

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Inc/murasaki\_include\_stub.h, 177

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Inc/murasaki\_platform.hpp, 178

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Inc/platform\_config.hpp, 179

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Inc/platform\_defs.hpp, 180

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Inc/stm32f4xx\_hal\_conf.h, 181

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Inc/stm32f4xx\_it.h, 187

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Src/main.c, 249

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Src/murasaki\_platform.cpp, 252

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Src/stm32f4xx\_it.c, 253

/home/takemasa/workspace\_st/nucleo-f446-sample/↔  
  Src/system\_stm32f4xx.c, 255

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc-tp/adau1361.hpp, 187

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/audiocodestrategy.hpp, 189

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/audiostrategy.hpp, 190

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/bitin.hpp, 191

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/bitinstrategy.hpp, 193

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/bitout.hpp, 195

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/bitoutstrategy.hpp, 197

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/criticalsection.hpp, 199

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/debugger.hpp, 200

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/debuggerfifo.hpp, 202

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/debuggeruart.hpp, 204

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/fifostrategy.hpp, 206

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/i2cmaster.hpp, 207

  Inc/i2cmasterstrategy.hpp, 209

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/i2cslave.hpp, 211

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/i2cslavestrategy.hpp, 213

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/loggerstrategy.hpp, 215

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki.hpp, 217

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_0\_intro.hpp, 218

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_1\_env.hpp, 218

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_2\_ug.hpp, 218

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_3\_pg.hpp, 219

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_4\_mod.hpp, 219

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_5\_spg.hpp, 219

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_assert.hpp, 219

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_config.hpp, 221

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_defs.hpp, 223

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/murasaki\_syslog.hpp, 224

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/peripheralstrategy.hpp, 225

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/spimaster.hpp, 226

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/spimasterstrategy.hpp, 228

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/spislave.hpp, 230

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/spislaveadapter.hpp, 232

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/spislaveadapterstrategy.hpp, 234

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/spislavestrategy.hpp, 236

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/synchronizer.hpp, 238

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/task.hpp, 239

/home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/↔  
  Inc/taskstrategy.hpp, 241

- /home/takemasa/workspace\_st/nucleo-f446-sample/murasaki/←Application Specific Platform, 66
  - Inc/uart.hpp, 242
  - Inc/uartlogger.hpp, 244
  - Inc/uartstrategy.hpp, 246
  - Src/allocators.cpp, 247
- ~LoggerStrategy
  - murasaki::LoggerStrategy, 138
- Abstract Classes, 72
- Adau1361
  - murasaki::Adau1361, 90
- AddSyslogFacilityToMask
  - murasaki, 86
- AllowedSyslogOut
  - murasaki, 86
- Application Specific Platform, 65
  - CustomAssertFailed, 66
  - CustomDefaultHandler, 66
  - debugger, 71
  - ExecPlatform, 67
  - HAL\_GPIO\_EXTI\_Callback, 67
  - HAL\_I2C\_ErrorCallback, 67
  - HAL\_I2C\_MasterTxCpltCallback, 68
  - HAL\_I2C\_SlaveTxCpltCallback, 68
  - HAL\_SPI\_ErrorCallback, 68
  - HAL\_SPI\_TxRxCpltCallback, 69
  - HAL\_UART\_ErrorCallback, 69
  - HAL\_UART\_RxCpltCallback, 69
  - HAL\_UART\_TxCpltCallback, 70
  - InitPlatform, 70
- assert\_failed
  - main.c, 250
  - stm32f4xx\_hal\_conf.h, 186
- assert\_param
  - stm32f4xx\_hal\_conf.h, 183
- AssertCs
  - murasaki::SpiSlaveAdapter, 152
  - murasaki::SpiSlaveAdapterStrategy, 154
- AudioCodecStrategy
  - murasaki::AudioCodecStrategy, 94
- AudioStrategy
  - murasaki::AudioStrategy, 97
- AutoRePrint
  - murasaki::Debugger, 108
- BitIn
  - murasaki::BitIn, 100
- BitOut
  - murasaki::BitOut, 104
- CMSIS, 75
- configure\_board
  - murasaki::Adau1361, 90
- configure\_pll
  - murasaki::Adau1361, 90
- CustomAssertFailed
  - Application Specific Platform, 66
  - CustomDefaultHandler
- DeassertCs
  - murasaki::SpiSlaveAdapter, 152
  - murasaki::SpiSlaveAdapterStrategy, 154
- Debugger
  - murasaki::Debugger, 108
- debugger
  - Application Specific Platform, 71
- DebuggerFifo
  - murasaki::DebuggerFifo, 111
- DebuggerUart
  - murasaki::DebuggerUart, 114
- Definitions and Configuration, 59
  - I2cStatus, 60
  - kfaAll, 62
  - kfaI2cMaster, 62
  - kfaI2cSlave, 62
  - kfaI2s, 62
  - kfaKernel, 62
  - kfaLog, 62
  - kfaNone, 62
  - kfaSai, 62
  - kfaSerial, 62
  - kfaSpiMaster, 62
  - kfaSpiSlave, 62
  - kfaUser0, 62
  - kfaUser1, 62
  - kfaUser2, 62
  - kfaUser3, 62
  - kfaUser4, 62
  - kfaUser5, 62
  - kfaUser6, 62
  - kfaUser7, 62
  - ki2csArbitrationLost, 60
  - ki2csBussError, 60
  - ki2csDMA, 60
  - ki2csNak, 60
  - ki2csOK, 60
  - ki2csOverrun, 60
  - ki2csTimeOut, 60
  - ki2csUnknown, 60
  - kseAlert, 62
  - kseCritical, 62
  - kseDebug, 62
  - kseEmergency, 62
  - kseError, 62
  - kseInfomational, 62
  - kseNotice, 62
  - kseWarning, 62
  - ksphLatchThenShift, 61
  - ksphShiftThenLatch, 61
  - kspisAbort, 61
  - kspisDMA, 61
  - kspisErrorFlag, 61
  - kspisFrameError, 61
  - kspisModeCRC, 61



- kspisModeFault, [61](#)
- kspisOK, [61](#)
- kspisOverflow, [61](#)
- kspisTimeOut, [61](#)
- kspisUnknown, [61](#)
- kspoFallThenRise, [61](#)
- kspoRiseThenFall, [61](#)
- kuhfcCts, [63](#)
- kuhfcCtsRts, [63](#)
- kuhfcNone, [63](#)
- kuhfcRts, [63](#)
- kursDMA, [63](#)
- kursFrame, [63](#)
- kursNoise, [63](#)
- kursOK, [63](#)
- kursOverrun, [63](#)
- kursParity, [63](#)
- kursTimeOut, [63](#)
- kutIdleTimeout, [63](#)
- kutNoldleTimeout, [63](#)
- kwmsIndefinitely, [64](#)
- kwmsPolling, [64](#)
- MURASAKI\_CONFIG\_NODEBUG, [59](#)
- PLATFORM\_CONFIG\_DEBUG\_BUFFER\_SIZE, [59](#)
- PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE, [59](#)
- PLATFORM\_CONFIG\_DEBUG\_SERIAL\_TIME←OUT, [59](#)
- PLATFORM\_CONFIG\_DEBUG\_TASK\_PRIORI←TY, [60](#)
- PLATFORM\_CONFIG\_DEBUG\_TASK\_STACK←\_SIZE, [60](#)
- SpiClockPhase, [60](#)
- SpiClockPolarity, [61](#)
- SpiStatus, [61](#)
- SyslogFacility, [61](#)
- SyslogSeverity, [62](#)
- UartHardwareFlowControl, [62](#)
- UartStatus, [63](#)
- UartTimeout, [63](#)
- WaitMilliseconds, [63](#)
- DoPostMortem
  - murasaki::LoggerStrategy, [138](#)
  - murasaki::UartLogger, [169](#)
- EXTERNAL\_CLOCK\_VALUE
  - stm32f4xx\_hal\_conf.h, [183](#)
- Enter
  - murasaki::CriticalSection, [107](#)
- Error\_Handler
  - main.c, [250](#)
  - main.h, [176](#)
- ExecPlatform
  - Application Specific Platform, [67](#)
- facility\_mask\_
  - murasaki::Debugger, [110](#)
- FifoStrategy
  - murasaki::FifoStrategy, [118](#)
- Get
  - murasaki::BitIn, [101](#)
  - murasaki::BitInStrategy, [102](#)
  - murasaki::BitOut, [104](#)
  - murasaki::BitOutStrategy, [106](#)
  - murasaki::DebuggerFifo, [111](#)
  - murasaki::FifoStrategy, [118](#)
- getCharacter
  - murasaki::LoggerStrategy, [138](#)
  - murasaki::UartLogger, [170](#)
- GetCpha
  - murasaki::SpiSlaveAdapterStrategy, [154](#)
- GetCpol
  - murasaki::SpiSlaveAdapterStrategy, [154](#)
- GetName
  - murasaki::TaskStrategy, [161](#)
- GetPeripheralHandle
  - murasaki::BitIn, [101](#)
  - murasaki::BitOut, [104](#)
- GetchFromTask
  - murasaki::Debugger, [109](#)
- HAL\_GPIO\_EXTI\_Callback
  - Application Specific Platform, [67](#)
- HAL\_I2C\_ErrorCallback
  - Application Specific Platform, [67](#)
- HAL\_I2C\_MasterRxCpltCallback
  - murasaki\_platform.cpp, [252](#)
- HAL\_I2C\_MasterTxCpltCallback
  - Application Specific Platform, [68](#)
- HAL\_I2C\_SlaveRxCpltCallback
  - murasaki\_platform.cpp, [253](#)
- HAL\_I2C\_SlaveTxCpltCallback
  - Application Specific Platform, [68](#)
- HAL\_SPI\_ErrorCallback
  - Application Specific Platform, [68](#)
- HAL\_SPI\_TxRxCpltCallback
  - Application Specific Platform, [69](#)
- HAL\_TIM\_PeriodElapsedCallback
  - main.c, [250](#)
- HAL\_UART\_ErrorCallback
  - Application Specific Platform, [69](#)
- HAL\_UART\_RxCpltCallback
  - Application Specific Platform, [69](#)
- HAL\_UART\_TxCpltCallback
  - Application Specific Platform, [70](#)
- HSE\_STARTUP\_TIMEOUT
  - stm32f4xx\_hal\_conf.h, [183](#)
- HSE\_VALUE
  - STM32F4xx\_System\_Private\_Includes, [77](#)
  - stm32f4xx\_hal\_conf.h, [183](#)
- HSI\_VALUE
  - STM32F4xx\_System\_Private\_Includes, [77](#)
  - stm32f4xx\_hal\_conf.h, [183](#)
- HandleError
  - murasaki::DebuggerUart, [114](#)
  - murasaki::I2CMasterStrategy, [126](#)
  - murasaki::I2cMaster, [122](#)
  - murasaki::I2cSlave, [131](#)

- murasaki::I2cSlaveStrategy, [135](#)
  - murasaki::SpiMaster, [143](#)
  - murasaki::SpiMasterStrategy, [145](#)
  - murasaki::SpiSlave, [148](#)
  - murasaki::SpiSlaveStrategy, [156](#)
  - murasaki::Uart, [165](#)
  - murasaki::UartStrategy, [172](#)
- hdma\_usart2\_rx
  - main.c, [251](#)
  - stm32f4xx\_it.c, [254](#)
- Helper classes, [73](#)
  - operator delete, [73](#)
  - operator delete[], [73](#)
  - operator new, [74](#)
  - operator new[], [74](#)
- I2cMaster
  - murasaki::I2cMaster, [121](#)
- I2cStatus
  - Definitions and Configuration, [60](#)
- InitPlatform
  - Application Specific Platform, [70](#)
- kfaAll
  - Definitions and Configuration, [62](#)
- kfaI2cMaster
  - Definitions and Configuration, [62](#)
- kfaI2cSlave
  - Definitions and Configuration, [62](#)
- kfaI2s
  - Definitions and Configuration, [62](#)
- kfaKernel
  - Definitions and Configuration, [62](#)
- kfaLog
  - Definitions and Configuration, [62](#)
- kfaNone
  - Definitions and Configuration, [62](#)
- kfaSai
  - Definitions and Configuration, [62](#)
- kfaSerial
  - Definitions and Configuration, [62](#)
- kfaSpiMaster
  - Definitions and Configuration, [62](#)
- kfaSpiSlave
  - Definitions and Configuration, [62](#)
- kfaUser0
  - Definitions and Configuration, [62](#)
- kfaUser1
  - Definitions and Configuration, [62](#)
- kfaUser2
  - Definitions and Configuration, [62](#)
- kfaUser3
  - Definitions and Configuration, [62](#)
- kfaUser4
  - Definitions and Configuration, [62](#)
- kfaUser5
  - Definitions and Configuration, [62](#)
- kfaUser6
  - Definitions and Configuration, [62](#)
- kfaUser7
  - Definitions and Configuration, [62](#)
- ki2csArbitrationLost
  - Definitions and Configuration, [60](#)
- ki2csBussError
  - Definitions and Configuration, [60](#)
- ki2csDMA
  - Definitions and Configuration, [60](#)
- ki2csNak
  - Definitions and Configuration, [60](#)
- ki2csOK
  - Definitions and Configuration, [60](#)
- ki2csOverrun
  - Definitions and Configuration, [60](#)
- ki2csTimeOut
  - Definitions and Configuration, [60](#)
- ki2csUnknown
  - Definitions and Configuration, [60](#)
- kseAlert
  - Definitions and Configuration, [62](#)
- kseCritical
  - Definitions and Configuration, [62](#)
- kseDebug
  - Definitions and Configuration, [62](#)
- kseEmergency
  - Definitions and Configuration, [62](#)
- kseError
  - Definitions and Configuration, [62](#)
- kseInfomational
  - Definitions and Configuration, [62](#)
- kseNotice
  - Definitions and Configuration, [62](#)
- kseWarning
  - Definitions and Configuration, [62](#)
- ksphLatchThenShift
  - Definitions and Configuration, [61](#)
- ksphShiftThenLatch
  - Definitions and Configuration, [61](#)
- kspisAbort
  - Definitions and Configuration, [61](#)
- kspisDMA
  - Definitions and Configuration, [61](#)
- kspisErrorFlag
  - Definitions and Configuration, [61](#)
- kspisFrameError
  - Definitions and Configuration, [61](#)
- kspisModeCRC
  - Definitions and Configuration, [61](#)
- kspisModeFault
  - Definitions and Configuration, [61](#)
- kspisOK
  - Definitions and Configuration, [61](#)
- kspisOverflow
  - Definitions and Configuration, [61](#)
- kspisTimeOut
  - Definitions and Configuration, [61](#)
- kspisUnknown
  - Definitions and Configuration, [61](#)

- kspoFallThenRise
  - Definitions and Configuration, [61](#)
- kspoRiseThenFall
  - Definitions and Configuration, [61](#)
- kuhfcCts
  - Definitions and Configuration, [63](#)
- kuhfcCtsRts
  - Definitions and Configuration, [63](#)
- kuhfcNone
  - Definitions and Configuration, [63](#)
- kuhfcRts
  - Definitions and Configuration, [63](#)
- kursDMA
  - Definitions and Configuration, [63](#)
- kursFrame
  - Definitions and Configuration, [63](#)
- kursNoise
  - Definitions and Configuration, [63](#)
- kursOK
  - Definitions and Configuration, [63](#)
- kursOverrun
  - Definitions and Configuration, [63](#)
- kursParity
  - Definitions and Configuration, [63](#)
- kursTimeOut
  - Definitions and Configuration, [63](#)
- kutIdleTimeout
  - Definitions and Configuration, [63](#)
- kutNIdleTimeout
  - Definitions and Configuration, [63](#)
- kwmsIndefinitely
  - Definitions and Configuration, [64](#)
- kwmsPolling
  - Definitions and Configuration, [64](#)
  
- LSE\_STARTUP\_TIMEOUT
  - stm32f4xx\_hal\_conf.h, [184](#)
- LSE\_VALUE
  - stm32f4xx\_hal\_conf.h, [184](#)
- LSI\_VALUE
  - stm32f4xx\_hal\_conf.h, [184](#)
- Launch
  - murasaki::TaskStrategy, [161](#)
- Leave
  - murasaki::CriticalSection, [107](#)
- line\_
  - murasaki::Debugger, [110](#)
  
- MURASAKI\_ASSERT
  - Murasaki Class Collection, [54](#)
- MURASAKI\_CONFIG\_NODEBUG
  - Definitions and Configuration, [59](#)
- MURASAKI\_CONFIG\_NOSYSLOG
  - platform\_config.hpp, [180](#)
- MURASAKI\_PRINT\_ERROR
  - Murasaki Class Collection, [55](#)
- MURASAKI\_SYSLOG
  - Murasaki Class Collection, [55](#)
- main
  - main.c, [250](#)
  - main.c
    - assert\_failed, [250](#)
    - Error\_Handler, [250](#)
    - HAL\_TIM\_PeriodElapsedCallback, [250](#)
    - hdma\_usart2\_rx, [251](#)
    - main, [250](#)
    - StartDefaultTask, [251](#)
    - SystemClock\_Config, [251](#)
  - main.h
    - Error\_Handler, [176](#)
- murasaki, [85](#)
  - AddSyslogFacilityToMask, [86](#)
  - AllowedSyslogOut, [86](#)
  - platform, [87](#)
  - RemoveSyslogFacilityFromMask, [87](#)
  - SetSyslogFacilityMask, [87](#)
  - SetSyslogSererityThreshold, [87](#)
- Murasaki Class Collection, [53](#)
  - MURASAKI\_ASSERT, [54](#)
  - MURASAKI\_PRINT\_ERROR, [55](#)
  - MURASAKI\_SYSLOG, [55](#)
- murasaki::Adau1361, [89](#)
  - Adau1361, [90](#)
  - configure\_board, [90](#)
  - configure\_pll, [90](#)
  - send\_command, [91](#)
  - send\_command\_table, [91](#)
  - set\_aux\_input\_gain, [91](#)
  - set\_hp\_output\_gain, [92](#)
  - set\_line\_input\_gain, [92](#)
  - set\_line\_output\_gain, [92](#)
  - start, [93](#)
  - wait\_pll\_lock, [93](#)
- murasaki::AudioCodecStrategy, [93](#)
  - AudioCodecStrategy, [94](#)
  - set\_aux\_input\_gain, [94](#)
  - set\_hp\_output\_gain, [94](#)
  - set\_line\_input\_gain, [95](#)
  - set\_line\_output\_gain, [95](#)
  - set\_mic\_input\_gain, [95](#)
  - start, [95](#)
- murasaki::AudioStrategy, [96](#)
  - AudioStrategy, [97](#)
  - TransmitAndReceive, [98](#)
- murasaki::BitIn, [99](#)
  - BitIn, [100](#)
  - Get, [101](#)
  - GetPeripheralHandle, [101](#)
- murasaki::BitInStrategy, [101](#)
  - Get, [102](#)
- murasaki::BitOut, [103](#)
  - BitOut, [104](#)
  - Get, [104](#)
  - GetPeripheralHandle, [104](#)
  - Set, [104](#)
- murasaki::BitOutStrategy, [105](#)
  - Get, [106](#)

- Set, 106
- murasaki::CriticalSection, 106
  - Enter, 107
  - Leave, 107
- murasaki::Debugger, 107
  - AutoRePrint, 108
  - Debugger, 108
  - facility\_mask\_, 110
  - GetchFromTask, 109
  - line\_, 110
  - Printf, 109
  - RePrint, 109
  - severity\_, 110
- murasaki::DebuggerFifo, 110
  - DebuggerFifo, 111
  - Get, 111
  - SetPostMortem, 112
- murasaki::DebuggerUart, 112
  - DebuggerUart, 114
  - HandleError, 114
  - Receive, 115
  - ReceiveCompleteCallback, 115
  - SetHardwareFlowControl, 116
  - SetSpeed, 116
  - Transmit, 116
  - TransmitCompleteCallback, 117
- murasaki::FifoStrategy, 117
  - FifoStrategy, 118
  - Get, 118
  - Put, 119
- murasaki::GPIO\_type, 119
- murasaki::I2CMasterStrategy, 125
  - HandleError, 126
  - Receive, 127
  - ReceiveCompleteCallback, 127
  - Transmit, 127
  - TransmitCompleteCallback, 128
  - TransmitThenReceive, 128
- murasaki::I2cMaster, 120
  - HandleError, 122
  - I2cMaster, 121
  - Receive, 122
  - ReceiveCompleteCallback, 123
  - Transmit, 123
  - TransmitCompleteCallback, 124
  - TransmitThenReceive, 124
- murasaki::I2cSlave, 129
  - HandleError, 131
  - Receive, 131
  - ReceiveCompleteCallback, 132
  - Transmit, 132
  - TransmitCompleteCallback, 133
- murasaki::I2cSlaveStrategy, 134
  - HandleError, 135
  - Receive, 135
  - ReceiveCompleteCallback, 135
  - Transmit, 136
  - TransmitCompleteCallback, 136
- murasaki::LoggerStrategy, 137
  - ~LoggerStrategy, 138
  - DoPostMortem, 138
  - getCharacter, 138
  - putMessage, 138
- murasaki::LoggingHelpers, 139
- murasaki::PeripheralStrategy, 139
- murasaki::Platform, 140
- murasaki::SpiMaster, 141
  - HandleError, 143
  - SpiMaster, 142
  - TransmitAndReceive, 143
  - TransmitAndReceiveCompleteCallback, 144
- murasaki::SpiMasterStrategy, 144
  - HandleError, 145
  - TransmitAndReceive, 146
  - TransmitAndReceiveCompleteCallback, 146
- murasaki::SpiSlave, 147
  - HandleError, 148
  - SpiSlave, 148
  - TransmitAndReceive, 149
  - TransmitAndReceiveCompleteCallback, 150
- murasaki::SpiSlaveAdapter, 150
  - AssertCs, 152
  - DeassertCs, 152
  - SpiSlaveAdapter, 151
- murasaki::SpiSlaveAdapterStrategy, 153
  - AssertCs, 154
  - DeassertCs, 154
  - GetCpha, 154
  - GetCpol, 154
  - SpiSlaveAdapterStrategy, 153, 154
- murasaki::SpiSlaveStrategy, 155
  - HandleError, 156
  - TransmitAndReceive, 156
  - TransmitAndReceiveCompleteCallback, 157
- murasaki::Synchronizer, 157
  - Release, 157
  - Wait, 157
- murasaki::Task, 158
  - Task, 159
  - TaskBody, 160
- murasaki::TaskStrategy, 160
  - GetName, 161
  - Launch, 161
  - Start, 162
  - TaskBody, 162
  - TaskStrategy, 161
- murasaki::Uart, 162
  - HandleError, 165
  - Receive, 165
  - ReceiveCompleteCallback, 166
  - SetHardwareFlowControl, 166
  - SetSpeed, 166
  - Transmit, 167
  - TransmitCompleteCallback, 167
  - Uart, 164
- murasaki::UartLogger, 168

- DoPostMortem, [169](#)
- getCharacter, [170](#)
- putMessage, [170](#)
- UartLogger, [169](#)
- murasaki::UartStrategy, [170](#)
  - HandleError, [172](#)
  - Receive, [172](#)
  - ReceiveCompleteCallback, [172](#)
  - SetHardwareFlowControl, [173](#)
  - SetSpeed, [173](#)
  - Transmit, [173](#)
  - TransmitCompleteCallback, [174](#)
- murasaki\_platform.cpp
  - HAL\_I2C\_MasterRxCpltCallback, [252](#)
  - HAL\_I2C\_SlaveRxCpltCallback, [253](#)
- operator delete
  - Helper classes, [73](#)
- operator delete[]
  - Helper classes, [73](#)
- operator new
  - Helper classes, [74](#)
- operator new[]
  - Helper classes, [74](#)
- PHY\_AUTONEGO\_COMPLETE
  - stm32f4xx\_hal\_conf.h, [184](#)
- PHY\_AUTONEGOTIATION
  - stm32f4xx\_hal\_conf.h, [184](#)
- PHY\_BCR
  - stm32f4xx\_hal\_conf.h, [184](#)
- PHY\_BSR
  - stm32f4xx\_hal\_conf.h, [184](#)
- PHY\_DUPLEX\_STATUS
  - stm32f4xx\_hal\_conf.h, [184](#)
- PHY\_FULLDUPLEX\_100M
  - stm32f4xx\_hal\_conf.h, [184](#)
- PHY\_FULLDUPLEX\_10M
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_HALFDUPLEX\_100M
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_HALFDUPLEX\_10M
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_ISOLATE
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_JABBER\_DETECTION
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_LINKED\_STATUS
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_LOOPBACK
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_POWERDOWN
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_RESET
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_RESTART\_AUTONEGOTIATION
  - stm32f4xx\_hal\_conf.h, [185](#)
- PHY\_SPEED\_STATUS
  - stm32f4xx\_hal\_conf.h, [186](#)
- PHY\_SR
  - stm32f4xx\_hal\_conf.h, [186](#)
- PLATFORM\_CONFIG\_DEBUG\_BUFFER\_SIZE
  - Definitions and Configuration, [59](#)
- PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE
  - Definitions and Configuration, [59](#)
- PLATFORM\_CONFIG\_DEBUG\_SERIAL\_TIMEOUT
  - Definitions and Configuration, [59](#)
- PLATFORM\_CONFIG\_DEBUG\_TASK\_PRIORITY
  - Definitions and Configuration, [60](#)
- PLATFORM\_CONFIG\_DEBUG\_TASK\_STACK\_SIZE
  - Definitions and Configuration, [60](#)
- platform
  - murasaki, [87](#)
- platform\_config.hpp
  - MURASAKI\_CONFIG\_NOSYSLOG, [180](#)
- Printf
  - murasaki::Debugger, [109](#)
- Put
  - murasaki::FifoStrategy, [119](#)
- putMessage
  - murasaki::LoggerStrategy, [138](#)
  - murasaki::UartLogger, [170](#)
- RePrint
  - murasaki::Debugger, [109](#)
- Receive
  - murasaki::DebuggerUart, [115](#)
  - murasaki::I2CMasterStrategy, [127](#)
  - murasaki::I2cMaster, [122](#)
  - murasaki::I2cSlave, [131](#)
  - murasaki::I2cSlaveStrategy, [135](#)
  - murasaki::Uart, [165](#)
  - murasaki::UartStrategy, [172](#)
- ReceiveCompleteCallback
  - murasaki::DebuggerUart, [115](#)
  - murasaki::I2CMasterStrategy, [127](#)
  - murasaki::I2cMaster, [123](#)
  - murasaki::I2cSlave, [132](#)
  - murasaki::I2cSlaveStrategy, [135](#)
  - murasaki::Uart, [166](#)
  - murasaki::UartStrategy, [172](#)
- Release
  - murasaki::Synchronizer, [157](#)
- RemoveSyslogFacilityFromMask
  - murasaki, [87](#)
- STM32F4xx\_System\_Private\_Defines, [79](#)
  - VECT\_TAB\_OFFSET, [79](#)
- STM32F4xx\_System\_Private\_FunctionPrototypes, [82](#)
- STM32F4xx\_System\_Private\_Functions, [83](#)
  - SystemCoreClockUpdate, [83](#)
  - SystemInit, [84](#)
- STM32F4xx\_System\_Private\_Includes, [77](#)
  - HSE\_VALUE, [77](#)
  - HSI\_VALUE, [77](#)
- STM32F4xx\_System\_Private\_Macros, [80](#)
- STM32F4xx\_System\_Private\_TypesDefinitions, [78](#)
- STM32F4xx\_System\_Private\_Variables, [81](#)

- send\_command
  - murasaki::Adau1361, [91](#)
- send\_command\_table
  - murasaki::Adau1361, [91](#)
- Set
  - murasaki::BitOut, [104](#)
  - murasaki::BitOutStrategy, [106](#)
- set\_aux\_input\_gain
  - murasaki::Adau1361, [91](#)
  - murasaki::AudioCodecStrategy, [94](#)
- set\_hp\_output\_gain
  - murasaki::Adau1361, [92](#)
  - murasaki::AudioCodecStrategy, [94](#)
- set\_line\_input\_gain
  - murasaki::Adau1361, [92](#)
  - murasaki::AudioCodecStrategy, [95](#)
- set\_line\_output\_gain
  - murasaki::Adau1361, [92](#)
  - murasaki::AudioCodecStrategy, [95](#)
- set\_mic\_input\_gain
  - murasaki::AudioCodecStrategy, [95](#)
- SetHardwareFlowControl
  - murasaki::DebuggerUart, [116](#)
  - murasaki::Uart, [166](#)
  - murasaki::UartStrategy, [173](#)
- SetPostMortem
  - murasaki::DebuggerFifo, [112](#)
- SetSpeed
  - murasaki::DebuggerUart, [116](#)
  - murasaki::Uart, [166](#)
  - murasaki::UartStrategy, [173](#)
- SetSyslogFacilityMask
  - murasaki, [87](#)
- SetSyslogSererityThreshold
  - murasaki, [87](#)
- severity\_
  - murasaki::Debugger, [110](#)
- SpiClockPhase
  - Definitions and Configuration, [60](#)
- SpiClockPolarity
  - Definitions and Configuration, [61](#)
- SpiMaster
  - murasaki::SpiMaster, [142](#)
- SpiSlave
  - murasaki::SpiSlave, [148](#)
- SpiSlaveAdapter
  - murasaki::SpiSlaveAdapter, [151](#)
- SpiSlaveAdapterStrategy
  - murasaki::SpiSlaveAdapterStrategy, [153](#), [154](#)
- SpiStatus
  - Definitions and Configuration, [61](#)
- Start
  - murasaki::TaskStrategy, [162](#)
- start
  - murasaki::Adau1361, [93](#)
  - murasaki::AudioCodecStrategy, [95](#)
- StartDefaultTask
  - main.c, [251](#)
- stm32f4xx\_hal\_conf.h
  - assert\_failed, [186](#)
  - assert\_param, [183](#)
  - EXTERNAL\_CLOCK\_VALUE, [183](#)
  - HSE\_STARTUP\_TIMEOUT, [183](#)
  - HSE\_VALUE, [183](#)
  - HSI\_VALUE, [183](#)
  - LSE\_STARTUP\_TIMEOUT, [184](#)
  - LSE\_VALUE, [184](#)
  - LSI\_VALUE, [184](#)
  - PHY\_AUTONEGO\_COMPLETE, [184](#)
  - PHY\_AUTONEGOTIATION, [184](#)
  - PHY\_BCR, [184](#)
  - PHY\_BSR, [184](#)
  - PHY\_DUPLEX\_STATUS, [184](#)
  - PHY\_FULLDUPLEX\_100M, [184](#)
  - PHY\_FULLDUPLEX\_10M, [185](#)
  - PHY\_HALFDUPLEX\_100M, [185](#)
  - PHY\_HALFDUPLEX\_10M, [185](#)
  - PHY\_ISOLATE, [185](#)
  - PHY\_JABBER\_DETECTION, [185](#)
  - PHY\_LINKED\_STATUS, [185](#)
  - PHY\_LOOPBACK, [185](#)
  - PHY\_POWERDOWN, [185](#)
  - PHY\_RESET, [185](#)
  - PHY\_RESTART\_AUTONEGOTIATION, [185](#)
  - PHY\_SPEED\_STATUS, [186](#)
  - PHY\_SR, [186](#)
  - TICK\_INT\_PRIORITY, [186](#)
  - VDD\_VALUE, [186](#)
- stm32f4xx\_it.c
  - hdma\_usart2\_rx, [254](#)
- Stm32f4xx\_system, [76](#)
- Synchronization and Exclusive access, [57](#)
- SyslogFacility
  - Definitions and Configuration, [61](#)
- SyslogSeverity
  - Definitions and Configuration, [62](#)
- SystemClock\_Config
  - main.c, [251](#)
- SystemCoreClockUpdate
  - STM32F4xx\_System\_Private\_Functions, [83](#)
- SystemInit
  - STM32F4xx\_System\_Private\_Functions, [84](#)
- TICK\_INT\_PRIORITY
  - stm32f4xx\_hal\_conf.h, [186](#)
- Task
  - murasaki::Task, [159](#)
- TaskBody
  - murasaki::Task, [160](#)
  - murasaki::TaskStrategy, [162](#)
- TaskStrategy
  - murasaki::TaskStrategy, [161](#)
- Third party classes, [58](#)
- Transmit
  - murasaki::DebuggerUart, [116](#)
  - murasaki::I2CMasterStrategy, [127](#)
  - murasaki::I2cMaster, [123](#)

- [murasaki::I2cSlave](#), [132](#)
  - [murasaki::I2cSlaveStrategy](#), [136](#)
  - [murasaki::Uart](#), [167](#)
  - [murasaki::UartStrategy](#), [173](#)
- TransmitAndReceive
  - [murasaki::AudioStrategy](#), [98](#)
  - [murasaki::SpiMaster](#), [143](#)
  - [murasaki::SpiMasterStrategy](#), [146](#)
  - [murasaki::SpiSlave](#), [149](#)
  - [murasaki::SpiSlaveStrategy](#), [156](#)
- TransmitAndReceiveCompleteCallback
  - [murasaki::SpiMaster](#), [144](#)
  - [murasaki::SpiMasterStrategy](#), [146](#)
  - [murasaki::SpiSlave](#), [150](#)
  - [murasaki::SpiSlaveStrategy](#), [157](#)
- TransmitCompleteCallback
  - [murasaki::DebuggerUart](#), [117](#)
  - [murasaki::I2CMasterStrategy](#), [128](#)
  - [murasaki::I2cMaster](#), [124](#)
  - [murasaki::I2cSlave](#), [133](#)
  - [murasaki::I2cSlaveStrategy](#), [136](#)
  - [murasaki::Uart](#), [167](#)
  - [murasaki::UartStrategy](#), [174](#)
- TransmitThenReceive
  - [murasaki::I2CMasterStrategy](#), [128](#)
  - [murasaki::I2cMaster](#), [124](#)
- Uart
  - [murasaki::Uart](#), [164](#)
- UartHardwareFlowControl
  - Definitions and Configuration, [62](#)
- UartLogger
  - [murasaki::UartLogger](#), [169](#)
- UartStatus
  - Definitions and Configuration, [63](#)
- UartTimeout
  - Definitions and Configuration, [63](#)
- VDD\_VALUE
  - [stm32f4xx\\_hal\\_conf.h](#), [186](#)
- VECT\_TAB\_OFFSET
  - [STM32F4xx\\_System\\_Private\\_Defines](#), [79](#)
- Wait
  - [murasaki::Synchronizer](#), [157](#)
- wait\_pll\_lock
  - [murasaki::Adau1361](#), [93](#)
- WaitMilliseconds
  - Definitions and Configuration, [63](#)