

GOS2022

0.11

Generated by Doxygen 1.8.7

Thu Dec 5 2024 12:53:08

Contents

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 gos_driver_functions_t Struct Reference	5
3.1.1 Detailed Description	5
3.2 gos_gcpChannelFunctions_t Struct Reference	5
3.2.1 Detailed Description	6
3.3 gos_gcpHeaderFrame_t Struct Reference	6
3.3.1 Detailed Description	6
3.4 gos_initStruct_t Struct Reference	6
3.4.1 Detailed Description	7
3.5 gos_message_t Struct Reference	7
3.5.1 Detailed Description	7
3.6 gos_messageWaiterDesc_t Struct Reference	7
3.6.1 Detailed Description	8
3.7 gos_mutex_t Struct Reference	8
3.7.1 Detailed Description	8
3.8 gos_queue_t Struct Reference	9
3.8.1 Detailed Description	9
3.9 gos_queueDescriptor_t Struct Reference	9
3.9.1 Detailed Description	9
3.10 gos_queueElement_t Struct Reference	10
3.10.1 Detailed Description	10
3.11 gos_shellCommand_t Struct Reference	10
3.11.1 Detailed Description	10
3.12 gos_signalDescriptor_t Struct Reference	10
3.12.1 Detailed Description	11
3.13 gos_signalInvokeDescriptor Struct Reference	11
3.13.1 Detailed Description	11
3.14 gos_sysmonLut_t Struct Reference	11
3.14.1 Detailed Description	12
3.15 gos_sysmonUserMessageDescriptor_t Struct Reference	12
3.15.1 Detailed Description	12
3.16 gos_trigger_t Struct Reference	12
3.16.1 Detailed Description	13
4 File Documentation	15
4.1 gos.c File Reference	15
4.1.1 Detailed Description	16
4.1.2 Macro Definition Documentation	16
4.1.2.1 GOS_SYS_TASK_SLEEP_TIME	16
4.1.3 Typedef Documentation	16

4.1.3.1	gos_initFunc_t	16
4.1.4	Function Documentation	16
4.1.4.1	__attribute__	16
4.1.4.2	gos_Dump	17
4.1.4.3	gos_Start	18
4.1.4.4	gos_systemTask	18
4.1.5	Variable Documentation	19
4.1.5.1	dumpRequired	19
4.1.5.2	initError	19
4.1.5.3	initializers	19
4.1.5.4	systemTaskDesc	20
4.1.5.5	systemTaskId	20
4.2	gos.h File Reference	20
4.2.1	Detailed Description	21
4.2.2	Macro Definition Documentation	21
4.2.2.1	GOS_VERSION_MAJOR	21
4.2.2.2	GOS_VERSION_MINOR	22
4.2.3	Function Documentation	22
4.2.3.1	gos_Dump	22
4.3	gos_bootloader_config.h File Reference	22
4.3.1	Detailed Description	24
4.3.2	Macro Definition Documentation	24
4.3.2.1	ARM_CORTEX_M4	24
4.3.2.2	CFG_GCP_CHANNELS_MAX_NUMBER	24
4.3.2.3	CFG_IDLE_TASK_STACK_SIZE	25
4.3.2.4	CFG_MESSAGE_MAX_ADDRESSEES	25
4.3.2.5	CFG_MESSAGE_MAX_LENGTH	25
4.3.2.6	CFG_MESSAGE_MAX_NUMBER	25
4.3.2.7	CFG_MESSAGE_MAX_WAITER_IDS	25
4.3.2.8	CFG_MESSAGE_MAX_WAITERS	25
4.3.2.9	CFG_PROC_IDLE_PRIO	25
4.3.2.10	CFG_PROC_MAX_NAME_LENGTH	25
4.3.2.11	CFG_PROC_MAX_NUMBER	25
4.3.2.12	CFG_PROC_MAX_PRIO_LEVELS	26
4.3.2.13	CFG_PROC_USE_SERVICE	26
4.3.2.14	CFG_QUEUE_MAX_ELEMENTS	26
4.3.2.15	CFG_QUEUE_MAX_LENGTH	26
4.3.2.16	CFG_QUEUE_MAX_NAME_LENGTH	26
4.3.2.17	CFG_QUEUE_MAX_NUMBER	26
4.3.2.18	CFG_QUEUE_USE_NAME	26
4.3.2.19	CFG_RESET_ON_ERROR	26
4.3.2.20	CFG_RESET_ON_ERROR_DELAY_MS	26
4.3.2.21	CFG_SCHED_COOPERATIVE	27
4.3.2.22	CFG_SHELL_COMMAND_BUFFER_SIZE	27
4.3.2.23	CFG_SHELL_MAX_COMMAND_LENGTH	27
4.3.2.24	CFG_SHELL_MAX_COMMAND_NUMBER	27
4.3.2.25	CFG_SHELL_MAX_PARAMS_LENGTH	27
4.3.2.26	CFG_SHELL_USE_SERVICE	27
4.3.2.27	CFG_SIGNAL_MAX_NUMBER	27
4.3.2.28	CFG_SIGNAL_MAX_SUBSCRIBERS	27
4.3.2.29	CFG_SYSMON_GCP_CHANNEL_NUM	27
4.3.2.30	CFG_SYSMON_MAX_USER_MESSAGES	28
4.3.2.31	CFG_SYSMON_USE_SERVICE	28
4.3.2.32	CFG_SYSTEM_TASK_STACK_SIZE	28
4.3.2.33	CFG_TARGET_CPU	28
4.3.2.34	CFG_TASK_MAX_NAME_LENGTH	28
4.3.2.35	CFG_TASK_MAX_NUMBER	28
4.3.2.36	CFG_TASK_MAX_STACK_SIZE	28

4.3.2.37	CFG_TASK_MESSAGE_DAEMON_PRIO	28
4.3.2.38	CFG_TASK_MESSAGE_DAEMON_STACK	28
4.3.2.39	CFG_TASK_MIN_STACK_SIZE	29
4.3.2.40	CFG_TASK_PROC_DAEMON_PRIO	29
4.3.2.41	CFG_TASK_PROC_DAEMON_STACK	29
4.3.2.42	CFG_TASK_SHELL_DAEMON_PRIO	29
4.3.2.43	CFG_TASK_SHELL_DAEMON_STACK	29
4.3.2.44	CFG_TASK_SIGNAL_DAEMON_PRIO	29
4.3.2.45	CFG_TASK_SIGNAL_DAEMON_STACK	29
4.3.2.46	CFG_TASK_SYS_PRIO	29
4.3.2.47	CFG_TASK_SYSMON_DAEMON_PRIO	29
4.3.2.48	CFG_TASK_SYSMON_DAEMON_STACK	30
4.3.2.49	CFG_TASK_TIME_DAEMON_PRIO	30
4.3.2.50	CFG_TASK_TIME_DAEMON_STACK	30
4.3.2.51	CFG_TASK_TRACE_DAEMON_PRIO	30
4.3.2.52	CFG_TASK_TRACE_DAEMON_STACK	30
4.3.2.53	CFG_TRACE_MAX_LENGTH	30
4.3.2.54	CFG_USE_PRIO_INHERITANCE	30
4.3.2.55	GOS_CFG_OVERCONFIG	30
4.4	gos_config.h File Reference	30
4.4.1	Detailed Description	32
4.4.2	Macro Definition Documentation	32
4.4.2.1	ARM_CORTEX_M4	32
4.4.2.2	CFG_GCP_CHANNELS_MAX_NUMBER	32
4.4.2.3	CFG_IDLE_TASK_STACK_SIZE	33
4.4.2.4	CFG_MESSAGE_MAX_ADDRESSEES	33
4.4.2.5	CFG_MESSAGE_MAX_LENGTH	33
4.4.2.6	CFG_MESSAGE_MAX_NUMBER	33
4.4.2.7	CFG_MESSAGE_MAX_WAITER_IDS	33
4.4.2.8	CFG_MESSAGE_MAX_WAITERS	33
4.4.2.9	CFG_QUEUE_MAX_ELEMENTS	33
4.4.2.10	CFG_QUEUE_MAX_LENGTH	33
4.4.2.11	CFG_QUEUE_MAX_NAME_LENGTH	33
4.4.2.12	CFG_QUEUE_MAX_NUMBER	34
4.4.2.13	CFG_QUEUE_USE_NAME	34
4.4.2.14	CFG_RESET_ON_ERROR	34
4.4.2.15	CFG_RESET_ON_ERROR_DELAY_MS	34
4.4.2.16	CFG_SCHED_COOPERATIVE	34
4.4.2.17	CFG_SHELL_COMMAND_BUFFER_SIZE	34
4.4.2.18	CFG_SHELL_MAX_COMMAND_LENGTH	34
4.4.2.19	CFG_SHELL_MAX_COMMAND_NUMBER	34
4.4.2.20	CFG_SHELL_MAX_PARAMS_LENGTH	34
4.4.2.21	CFG_SHELL_USE_SERVICE	35
4.4.2.22	CFG_SIGNAL_MAX_NUMBER	35
4.4.2.23	CFG_SIGNAL_MAX_SUBSCRIBERS	35
4.4.2.24	CFG_SYSMON_GCP_CHANNEL_NUM	35
4.4.2.25	CFG_SYSMON_MAX_USER_MESSAGES	35
4.4.2.26	CFG_SYSMON_USE_SERVICE	35
4.4.2.27	CFG_SYSTEM_TASK_STACK_SIZE	35
4.4.2.28	CFG_TARGET_CPU	35
4.4.2.29	CFG_TASK_MAX_NAME_LENGTH	35
4.4.2.30	CFG_TASK_MAX_NUMBER	36
4.4.2.31	CFG_TASK_MAX_STACK_SIZE	36
4.4.2.32	CFG_TASK_MESSAGE_DAEMON_PRIO	36
4.4.2.33	CFG_TASK_MESSAGE_DAEMON_STACK	36
4.4.2.34	CFG_TASK_MIN_STACK_SIZE	36
4.4.2.35	CFG_TASK_SHELL_DAEMON_PRIO	36
4.4.2.36	CFG_TASK_SHELL_DAEMON_STACK	36

4.4.2.37	CFG_TASK_SIGNAL_DAEMON_PRIO	36
4.4.2.38	CFG_TASK_SIGNAL_DAEMON_STACK	36
4.4.2.39	CFG_TASK_SYS_PRIO	37
4.4.2.40	CFG_TASK_SYSMON_DAEMON_PRIO	37
4.4.2.41	CFG_TASK_SYSMON_DAEMON_STACK	37
4.4.2.42	CFG_TASK_TIME_DAEMON_PRIO	37
4.4.2.43	CFG_TASK_TIME_DAEMON_STACK	37
4.4.2.44	CFG_TASK_TRACE_DAEMON_PRIO	37
4.4.2.45	CFG_TASK_TRACE_DAEMON_STACK	37
4.4.2.46	CFG_TRACE_MAX_LENGTH	37
4.4.2.47	CFG_USE_PRIO_INHERITANCE	37
4.5	gos_crc_driver.c File Reference	38
4.5.1	Detailed Description	38
4.5.2	Macro Definition Documentation	39
4.5.2.1	CRC_INITIAL_VALUE	39
4.5.2.2	CRC_POLYNOMIAL_VALUE	39
4.5.3	Function Documentation	39
4.5.3.1	gos_crcDriverGetCrc	39
4.6	gos_crc_driver.h File Reference	40
4.6.1	Detailed Description	40
4.6.2	Function Documentation	41
4.6.2.1	gos_crcDriverGetCrc	41
4.7	gos_driver.c File Reference	41
4.7.1	Detailed Description	42
4.7.2	Function Documentation	42
4.7.2.1	gos_driverInit	42
4.8	gos_driver.h File Reference	43
4.8.1	Detailed Description	44
4.8.2	Function Documentation	44
4.8.2.1	gos_driverInit	44
4.9	gos_error.c File Reference	44
4.9.1	Detailed Description	46
4.9.2	Macro Definition Documentation	46
4.9.2.1	ERROR_BUFFER_SIZE	46
4.9.2.2	RESULT_STRING_ERROR	46
4.9.2.3	RESULT_STRING_SUCCESS	46
4.9.2.4	RESULT_STRING_UNKNOWN	46
4.9.2.5	SEPARATOR_LINE	46
4.9.3	Function Documentation	47
4.9.3.1	__attribute__	47
4.9.3.2	gos_errorHandler	47
4.9.3.3	gos_errorTraceInit	49
4.9.3.4	gos_traceResultToString	49
4.9.4	Variable Documentation	50
4.9.4.1	errorBuffer	50
4.10	gos_error.h File Reference	50
4.10.1	Detailed Description	52
4.10.2	Enumeration Type Documentation	52
4.10.2.1	gos_errorLevel_t	52
4.10.3	Function Documentation	52
4.10.3.1	gos_errorHandler	52
4.10.3.2	gos_errorTraceInit	54
4.10.3.3	gos_printStartupLogo	55
4.11	gos_gcp.c File Reference	55
4.11.1	Detailed Description	57
4.11.2	Macro Definition Documentation	57
4.11.2.1	GCP_PROTOCOL_VERSION_MAJOR	57
4.11.2.2	GCP_PROTOCOL_VERSION_MINOR	57

4.11.3	Enumeration Type Documentation	58
4.11.3.1	gos_gcpAck_t	58
4.11.4	Function Documentation	58
4.11.4.1	gos_gcpInit	58
4.11.4.2	gos_gcpReceiveMessage	58
4.11.4.3	gos_gcpReceiveMessageInternal	59
4.11.4.4	gos_gcpRegisterPhysicalDriver	60
4.11.4.5	gos_gcpTransmitMessage	61
4.11.4.6	gos_gcpTransmitMessageInternal	62
4.11.4.7	gos_gcpValidateData	63
4.11.4.8	gos_gcpValidateHeader	64
4.11.5	Variable Documentation	65
4.11.5.1	channelFunctions	65
4.11.5.2	gcpRxMutexes	65
4.11.5.3	gcpTxMutexes	65
4.12	gos_gcp.h File Reference	65
4.12.1	Detailed Description	66
4.12.2	Typedef Documentation	67
4.12.2.1	gos_gcpReceiveFunction_t	67
4.12.2.2	gos_gcpTransmitFunction_t	67
4.12.3	Function Documentation	67
4.12.3.1	gos_gcpInit	67
4.12.3.2	gos_gcpReceiveMessage	67
4.12.3.3	gos_gcpRegisterPhysicalDriver	68
4.12.3.4	gos_gcpTransmitMessage	69
4.13	gos_kernel.c File Reference	70
4.13.1	Detailed Description	73
4.13.2	Macro Definition Documentation	73
4.13.2.1	BINARY_PATTERN	73
4.13.2.2	CONFIG_DUMP_SEPARATOR	73
4.13.2.3	ICSR	73
4.13.2.4	MAX_CPU_DUMP_SEPARATOR	73
4.13.2.5	SHCSR	74
4.13.2.6	STACK_STATS_SEPARATOR	74
4.13.2.7	TASK_DUMP_SEPARATOR	74
4.13.2.8	TO_BINARY	74
4.13.3	Function Documentation	74
4.13.3.1	gos_idleTask	74
4.13.3.2	gos_kernelCalculateTaskCpuUsages	75
4.13.3.3	gos_kernelCheckTaskStack	76
4.13.3.4	gos_kernelDelayMs	77
4.13.3.5	gos_kernelDelayUs	78
4.13.3.6	gos_kernelDump	79
4.13.3.7	gos_kernelGetCpuUsage	80
4.13.3.8	gos_kernelGetCurrentPsp	80
4.13.3.9	gos_kernelGetMaxCpuLoad	80
4.13.3.10	gos_kernelGetSysTicks	81
4.13.3.11	gos_kernelGetTaskStateString	81
4.13.3.12	gos_kernellInit	82
4.13.3.13	gos_kernellsCallersLs	82
4.13.3.14	gos_kernelPrivilegedModeSetRequired	83
4.13.3.15	gos_kernelProcessorReset	83
4.13.3.16	gos_kernelRegisterPrivilegedHook	84
4.13.3.17	gos_kernelRegisterSwapHook	85
4.13.3.18	gos_kernelRegisterSysTickHook	85
4.13.3.19	gos_kernelReschedule	86
4.13.3.20	gos_kernelReset	86
4.13.3.21	gos_kernelSaveCurrentPsp	87

4.13.3.22 gos_kernelSelectNextTask	88
4.13.3.23 gos_kernelSetMaxCpuLoad	88
4.13.3.24 gos_kernelStart	89
4.13.4 Variable Documentation	89
4.13.4.1 atomicCntr	89
4.13.4.2 cpuUseLimit	89
4.13.4.3 currentTaskIndex	90
4.13.4.4 inLsr	90
4.13.4.5 isKernelRunning	90
4.13.4.6 kernelDumpReadySignal	90
4.13.4.7 kernelDumpSignal	90
4.13.4.8 kernelPrivilegedHookFunction	90
4.13.4.9 kernelSwapHookFunction	90
4.13.4.10 kernelSysTickHookFunction	90
4.13.4.11 monitoringTime	90
4.13.4.12 previousTick	91
4.13.4.13 primask	91
4.13.4.14 privilegedModeSetRequired	91
4.13.4.15 resetRequired	91
4.13.4.16 schedDisableCntr	91
4.13.4.17 sysTicks	91
4.13.4.18 sysTimerValue	91
4.14 gos_kernel.h File Reference	91
4.14.1 Detailed Description	97
4.14.2 Macro Definition Documentation	97
4.14.2.1 GLOBAL_STACK	97
4.14.2.2 GOS_ASM	97
4.14.2.3 GOS_ATOMIC_ENTER	97
4.14.2.4 GOS_ATOMIC_EXIT	98
4.14.2.5 GOS_CONCAT_RESULT	98
4.14.2.6 GOS_CONST	99
4.14.2.7 GOS_DEFAULT_TASK_ID	99
4.14.2.8 GOS_DISABLE_SCHED	99
4.14.2.9 GOS_ENABLE_SCHED	99
4.14.2.10 GOS_EXTERN	99
4.14.2.11 GOS_INLINE	99
4.14.2.12 GOS_INVALID_TASK_ID	100
4.14.2.13 GOS_ISR_ENTER	100
4.14.2.14 GOS_ISR_EXIT	100
4.14.2.15 GOS_NAKED	100
4.14.2.16 GOS_NOP	100
4.14.2.17 GOS_PRIV_RESERVED_3	100
4.14.2.18 GOS_PRIV_RESERVED_4	100
4.14.2.19 GOS_PRIV_RESERVED_5	101
4.14.2.20 GOS_PRIV_SIGNALING	101
4.14.2.21 GOS_PRIV_TASK_MANIPULATE	101
4.14.2.22 GOS_PRIV_TASK_PRIO_CHANGE	101
4.14.2.23 GOS_PRIV_TRACE	101
4.14.2.24 GOS_STATIC	101
4.14.2.25 GOS_STATIC_INLINE	101
4.14.2.26 GOS_TASK_IDLE_PRIO	101
4.14.2.27 GOS_TASK_MAX_BLOCK_TIME_MS	101
4.14.2.28 GOS_TASK_MAX_PRIO_LEVELS	102
4.14.2.29 GOS_UNUSED	102
4.14.2.30 MAIN_STACK	102
4.14.2.31 NULL	102
4.14.2.32 RAM_SIZE	102
4.14.2.33 RAM_START	102

4.14.3 Enumeration Type Documentation	102
4.14.3.1 gos_boolValue_t	102
4.14.3.2 gos_kernel_privilege_t	103
4.14.3.3 gos_result_t	103
4.14.3.4 gos_taskPrivilegeLevel_t	103
4.14.3.5 gos_taskState_t	103
4.14.4 Function Documentation	104
4.14.4.1 __attribute__	104
4.14.4.2 __attribute__	105
4.14.4.3 gos_kernelCalculateTaskCpuUsages	106
4.14.4.4 gos_kernelDelayMs	106
4.14.4.5 gos_kernelDelayUs	107
4.14.4.6 gos_kernelDump	108
4.14.4.7 gos_kernelGetCpuUsage	108
4.14.4.8 gos_kernelGetMaxCpuLoad	109
4.14.4.9 gos_kernelGetSysTicks	109
4.14.4.10 gos_kernellInit	110
4.14.4.11 gos_kernellsCallerlsr	111
4.14.4.12 gos_kernelPrivilegedModeSetRequired	111
4.14.4.13 gos_kernelRegisterIdleHook	111
4.14.4.14 gos_kernelRegisterPrivilegedHook	112
4.14.4.15 gos_kernelRegisterSwapHook	112
4.14.4.16 gos_kernelRegisterSysTickHook	113
4.14.4.17 gos_kernelReschedule	113
4.14.4.18 gos_kernelReset	114
4.14.4.19 gos_kernelSetMaxCpuLoad	115
4.14.4.20 gos_kernelStart	116
4.14.4.21 gos_kernelSubscribeDumpReadySignal	116
4.14.4.22 gos_taskAddPrivilege	118
4.14.4.23 gos_taskBlock	118
4.14.4.24 gos_taskDelete	119
4.14.4.25 gos_taskGetCurrentId	120
4.14.4.26 gos_taskGetData	121
4.14.4.27 gos_taskGetDataByIndex	122
4.14.4.28 gos_taskGetId	123
4.14.4.29 gos_taskGetName	123
4.14.4.30 gos_taskGetNumber	123
4.14.4.31 gos_taskGetOriginalPriority	124
4.14.4.32 gos_taskGetPriority	124
4.14.4.33 gos_taskGetPrivileges	125
4.14.4.34 gos_taskRegister	126
4.14.4.35 gos_taskRegisterTasks	127
4.14.4.36 gos_taskRemovePrivilege	128
4.14.4.37 gos_taskResume	128
4.14.4.38 gos_taskSetOriginalPriority	129
4.14.4.39 gos_taskSetPriority	130
4.14.4.40 gos_taskSetPrivileges	131
4.14.4.41 gos_taskSleep	132
4.14.4.42 gos_taskSubscribeDeleteSignal	133
4.14.4.43 gos_taskSuspend	133
4.14.4.44 gos_taskUnblock	134
4.14.4.45 gos_taskWakeUp	135
4.14.4.46 gos_taskYield	136
4.15 gos_message.c File Reference	137
4.15.1 Detailed Description	138
4.15.2 Macro Definition Documentation	139
4.15.2.1 GOS_MESSAGE_DAEMON_POLL_TIME_MS	139
4.15.3 Function Documentation	139

4.15.3.1	gos_messageDaemonTask	139
4.15.3.2	gos_messageInit	139
4.15.3.3	gos_messageRx	140
4.15.3.4	gos_messageTx	140
4.15.4	Variable Documentation	141
4.15.4.1	messageArray	141
4.15.4.2	messageDaemonTaskDesc	141
4.15.4.3	messageDaemonTaskId	141
4.15.4.4	messageMutex	142
4.15.4.5	messageWaiterArray	142
4.15.4.6	nextMessageIndex	142
4.15.4.7	nextWaiterIndex	142
4.16	gos_message.h File Reference	142
4.16.1	Detailed Description	144
4.16.2	Macro Definition Documentation	144
4.16.2.1	GOS_MESSAGE_ENDLESS_TMO	144
4.16.2.2	GOS_MESSAGE_INVALID_ID	144
4.16.3	Typedef Documentation	144
4.16.3.1	gos_messageId_t	144
4.16.3.2	gos_messageTimeout_t	144
4.16.4	Function Documentation	145
4.16.4.1	gos_messageInit	145
4.16.4.2	gos_messageRx	145
4.16.4.3	gos_messageTx	146
4.17	gos_minimal_example_app_config.h File Reference	146
4.17.1	Detailed Description	148
4.17.2	Macro Definition Documentation	148
4.17.2.1	ARM_CORTEX_M4	148
4.17.2.2	CFG_GCP_CHANNELS_MAX_NUMBER	148
4.17.2.3	CFG_IDLE_TASK_STACK_SIZE	149
4.17.2.4	CFG_MESSAGE_MAX_ADDRESSEES	149
4.17.2.5	CFG_MESSAGE_MAX_LENGTH	149
4.17.2.6	CFG_MESSAGE_MAX_NUMBER	149
4.17.2.7	CFG_MESSAGE_MAX_WAITER_IDS	149
4.17.2.8	CFG_MESSAGE_MAX_WAITERS	149
4.17.2.9	CFG_PROC_IDLE_PRIO	149
4.17.2.10	CFG_PROC_MAX_NAME_LENGTH	149
4.17.2.11	CFG_PROC_MAX_NUMBER	149
4.17.2.12	CFG_PROC_MAX_PRIO_LEVELS	150
4.17.2.13	CFG_PROC_USE_SERVICE	150
4.17.2.14	CFG_QUEUE_MAX_ELEMENTS	150
4.17.2.15	CFG_QUEUE_MAX_LENGTH	150
4.17.2.16	CFG_QUEUE_MAX_NAME_LENGTH	150
4.17.2.17	CFG_QUEUE_MAX_NUMBER	150
4.17.2.18	CFG_QUEUE_USE_NAME	150
4.17.2.19	CFG_RESET_ON_ERROR	150
4.17.2.20	CFG_RESET_ON_ERROR_DELAY_MS	150
4.17.2.21	CFG_SCHED_COOPERATIVE	151
4.17.2.22	CFG_SHELL_COMMAND_BUFFER_SIZE	151
4.17.2.23	CFG_SHELL_MAX_COMMAND_LENGTH	151
4.17.2.24	CFG_SHELL_MAX_COMMAND_NUMBER	151
4.17.2.25	CFG_SHELL_MAX_PARAMS_LENGTH	151
4.17.2.26	CFG_SHELL_USE_SERVICE	151
4.17.2.27	CFG_SIGNAL_MAX_NUMBER	151
4.17.2.28	CFG_SIGNAL_MAX_SUBSCRIBERS	151
4.17.2.29	CFG_SYSMON_GCP_CHANNEL_NUM	151
4.17.2.30	CFG_SYSMON_MAX_USER_MESSAGES	152
4.17.2.31	CFG_SYSMON_USE_SERVICE	152

4.17.2.32 CFG_SYSTEM_TASK_STACK_SIZE	152
4.17.2.33 CFG_TARGET_CPU	152
4.17.2.34 CFG_TASK_MAX_NAME_LENGTH	152
4.17.2.35 CFG_TASK_MAX_NUMBER	152
4.17.2.36 CFG_TASK_MAX_STACK_SIZE	152
4.17.2.37 CFG_TASK_MESSAGE_DAEMON_PRIO	152
4.17.2.38 CFG_TASK_MESSAGE_DAEMON_STACK	152
4.17.2.39 CFG_TASK_MIN_STACK_SIZE	153
4.17.2.40 CFG_TASK_PROC_DAEMON_PRIO	153
4.17.2.41 CFG_TASK_PROC_DAEMON_STACK	153
4.17.2.42 CFG_TASK_SHELL_DAEMON_PRIO	153
4.17.2.43 CFG_TASK_SHELL_DAEMON_STACK	153
4.17.2.44 CFG_TASK_SIGNAL_DAEMON_PRIO	153
4.17.2.45 CFG_TASK_SIGNAL_DAEMON_STACK	153
4.17.2.46 CFG_TASK_SYS_PRIO	153
4.17.2.47 CFG_TASK_SYSMON_DAEMON_PRIO	153
4.17.2.48 CFG_TASK_SYSMON_DAEMON_STACK	154
4.17.2.49 CFG_TASK_TIME_DAEMON_PRIO	154
4.17.2.50 CFG_TASK_TIME_DAEMON_STACK	154
4.17.2.51 CFG_TASK_TRACE_DAEMON_PRIO	154
4.17.2.52 CFG_TASK_TRACE_DAEMON_STACK	154
4.17.2.53 CFG_TRACE_MAX_LENGTH	154
4.17.2.54 CFG_USE_PRIO_INHERITANCE	154
4.17.2.55 GOS_CFG_OVERCONFIG	154
4.18 gos_mutex.c File Reference	154
4.18.1 Detailed Description	155
4.18.2 Macro Definition Documentation	156
4.18.2.1 GOS_MUTEX_LOCK_SLEEP_MS	156
4.18.3 Function Documentation	156
4.18.3.1 gos_mutexInit	156
4.18.3.2 gos_mutexLock	157
4.18.3.3 gos_mutexUnlock	158
4.19 gos_mutex.h File Reference	159
4.19.1 Detailed Description	160
4.19.2 Macro Definition Documentation	161
4.19.2.1 GOS_MUTEX_ENDLESS_TMO	161
4.19.2.2 GOS_MUTEX_NO_TMO	161
4.19.3 Enumeration Type Documentation	161
4.19.3.1 gos_mutexState_t	161
4.19.4 Function Documentation	161
4.19.4.1 gos_mutexInit	161
4.19.4.2 gos_mutexLock	162
4.19.4.3 gos_mutexUnlock	163
4.20 gos_port.h File Reference	164
4.20.1 Detailed Description	166
4.20.2 Macro Definition Documentation	166
4.20.2.1 gos_ported_doContextSwitch	166
4.20.2.2 gos_ported_enableFaultHandlers	167
4.20.2.3 gos_ported_handleSVC	167
4.20.2.4 gos_ported_handleSVCMain	167
4.20.2.5 gos_ported_kernelStartInit	168
4.20.2.6 gos_ported_pendSVHandler	168
4.20.2.7 gos_ported_procReset	168
4.20.2.8 gos_ported_reschedule	168
4.20.2.9 gos_ported_svcHandler	169
4.20.2.10 gos_ported_svcHandlerMain	169
4.20.2.11 gos_ported_sysTickInterrupt	169
4.21 gos_queue.c File Reference	169

4.21.1	Detailed Description	171
4.21.2	Macro Definition Documentation	171
4.21.2.1	DUMP_SEPARATOR	171
4.21.3	Function Documentation	171
4.21.3.1	gos_queueCreate	171
4.21.3.2	gos_queueDump	172
4.21.3.3	gos_queueGet	172
4.21.3.4	gos_queueGetElementNumber	173
4.21.3.5	gos_queueGetName	174
4.21.3.6	gos_queueInit	174
4.21.3.7	gos_queuePeek	175
4.21.3.8	gos_queuePut	175
4.21.3.9	gos_queueRegisterEmptyHook	176
4.21.3.10	gos_queueRegisterFullHook	176
4.21.3.11	gos_queueReset	177
4.21.4	Variable Documentation	177
4.21.4.1	queueEmptyHook	177
4.21.4.2	queueFullHook	177
4.21.4.3	queueMutex	178
4.21.4.4	queues	178
4.21.4.5	readCounters	178
4.21.4.6	writeCounters	178
4.22	gos_queue.h File Reference	178
4.22.1	Detailed Description	180
4.22.2	Macro Definition Documentation	180
4.22.2.1	GOS_DEFAULT_QUEUE_ID	180
4.22.2.2	GOS_INVALID_QUEUE_ID	181
4.22.3	Typedef Documentation	181
4.22.3.1	gos_queueByte_t	181
4.22.3.2	gos_queueEmptyHook	181
4.22.3.3	gos_queueFullHook	181
4.22.4	Function Documentation	181
4.22.4.1	gos_queueCreate	181
4.22.4.2	gos_queueDump	182
4.22.4.3	gos_queueGet	183
4.22.4.4	gos_queueGetElementNumber	183
4.22.4.5	gos_queueGetName	184
4.22.4.6	gos_queueInit	184
4.22.4.7	gos_queuePeek	185
4.22.4.8	gos_queuePut	186
4.22.4.9	gos_queueRegisterEmptyHook	186
4.22.4.10	gos_queueRegisterFullHook	187
4.22.4.11	gos_queueReset	187
4.23	gos_shell.c File Reference	188
4.23.1	Detailed Description	189
4.23.2	Macro Definition Documentation	189
4.23.2.1	GOS_SHELL_DAEMON_POLL_TIME_MS	189
4.23.2.2	GOS_SHELL_DISPLAY_TEXT	189
4.23.3	Function Documentation	190
4.23.3.1	gos_shellCommandHandler	190
4.23.3.2	gos_shellDaemonTask	190
4.23.3.3	gos_shellEchoOff	191
4.23.3.4	gos_shellEchoOn	191
4.23.3.5	gos_shellInit	191
4.23.3.6	gos_shellRegisterCommand	192
4.23.3.7	gos_shellRegisterCommands	192
4.23.3.8	gos_shellResume	194
4.23.3.9	gos_shellSuspend	195

4.23.4	Variable Documentation	195
4.23.4.1	actualCommand	195
4.23.4.2	commandBuffer	196
4.23.4.3	commandBufferIndex	196
4.23.4.4	commandParams	196
4.23.4.5	shellCommand	196
4.23.4.6	shellCommands	196
4.23.4.7	shellDaemonTaskDesc	196
4.23.4.8	shellDaemonTaskId	196
4.23.4.9	useEcho	197
4.24	gos_shell.h File Reference	197
4.24.1	Detailed Description	198
4.24.2	Typedef Documentation	199
4.24.2.1	gos_shellFunction	199
4.24.3	Function Documentation	199
4.24.3.1	gos_shellEchoOff	199
4.24.3.2	gos_shellEchoOn	199
4.24.3.3	gos_shellInit	200
4.24.3.4	gos_shellRegisterCommand	200
4.24.3.5	gos_shellRegisterCommands	201
4.24.3.6	gos_shellResume	201
4.24.3.7	gos_shellSuspend	202
4.25	gos_shell_driver.c File Reference	203
4.25.1	Detailed Description	204
4.25.2	Function Documentation	204
4.25.2.1	gos_shellDriverReceiveChar	204
4.25.2.2	gos_shellDriverTransmitString	204
4.25.3	Variable Documentation	205
4.25.3.1	formattedBuffer	205
4.26	gos_shell_driver.h File Reference	205
4.26.1	Detailed Description	206
4.26.2	Typedef Documentation	207
4.26.2.1	gos_shellDriverReceiveChar_t	207
4.26.2.2	gos_shellDriverTransmitString_t	207
4.26.3	Function Documentation	207
4.26.3.1	gos_shellDriverReceiveChar	207
4.26.3.2	gos_shellDriverTransmitString	208
4.27	gos_signal.c File Reference	208
4.27.1	Detailed Description	210
4.27.2	Function Documentation	210
4.27.2.1	gos_signalCreate	210
4.27.2.2	gos_signalDaemonTask	211
4.27.2.3	gos_signallInit	211
4.27.2.4	gos_signallInvoke	211
4.27.2.5	gos_signalSubscribe	212
4.27.3	Variable Documentation	213
4.27.3.1	signalArray	213
4.27.3.2	signalDaemonTaskDescriptor	213
4.27.3.3	signallInvokeTrigger	213
4.28	gos_signal.h File Reference	213
4.28.1	Detailed Description	215
4.28.2	Function Documentation	215
4.28.2.1	gos_signalCreate	215
4.28.2.2	gos_signallInit	216
4.28.2.3	gos_signallInvoke	216
4.28.2.4	gos_signalSubscribe	217
4.29	gos_sysmon.c File Reference	218
4.29.1	Detailed Description	221

4.29.2	Macro Definition Documentation	221
4.29.2.1	RECEIVE_BUFFER_SIZE	221
4.29.3	Typedef Documentation	221
4.29.3.1	gos_sysmonMessageHandler_t	221
4.29.4	Enumeration Type Documentation	221
4.29.4.1	gos_sysmonMessageEnum_t	221
4.29.4.2	gos_sysmonMessageId_t	222
4.29.4.3	gos_sysmonMessagePv_t	222
4.29.4.4	gos_sysmonMessageResult_t	223
4.29.4.5	gos_sysmonTaskModifyType_t	223
4.29.5	Function Documentation	223
4.29.5.1	__attribute__	223
4.29.5.2	gos_sysmonCheckMessage	225
4.29.5.3	gos_sysmonDaemonTask	226
4.29.5.4	gos_sysmonGetLutIndex	226
4.29.5.5	gos_sysmonHandleCpuUsageGet	226
4.29.5.6	gos_sysmonHandlePingRequest	227
4.29.5.7	gos_sysmonHandleResetRequest	227
4.29.5.8	gos_sysmonHandleSysRuntimeGet	228
4.29.5.9	gos_sysmonHandleSystimeSet	228
4.29.5.10	gos_sysmonHandleTaskDataGet	229
4.29.5.11	gos_sysmonHandleTaskModification	229
4.29.5.12	gos_sysmonHandleTaskVariableDataGet	230
4.29.5.13	gos_sysmonInit	230
4.29.5.14	gos_sysmonRegisterUserMessage	231
4.29.5.15	gos_sysmonSendResponse	231
4.29.6	Variable Documentation	232
4.29.6.1	cpuMessage	232
4.29.6.2	pingMessage	232
4.29.6.3	receiveBuffer	233
4.29.6.4	sysmonDaemonTaskDesc	233
4.29.6.5	sysmonLut	233
4.29.6.6	sysRuntimeGetResultMessage	233
4.29.6.7	sysTimeSetMessage	233
4.29.6.8	sysTimeSetResultMessage	233
4.29.6.9	taskDataGetMsg	233
4.29.6.10	taskDataMsg	234
4.29.6.11	taskDesc	234
4.29.6.12	taskModifyMessage	234
4.29.6.13	taskModifyResultMessage	234
4.29.6.14	taskVariableDataMsg	234
4.29.6.15	userMessages	234
4.30	gos_sysmon.h File Reference	234
4.30.1	Detailed Description	236
4.30.2	Typedef Documentation	236
4.30.2.1	gos_sysmonMessageReceivedCallback	236
4.30.3	Function Documentation	236
4.30.3.1	gos_sysmonInit	236
4.30.3.2	gos_sysmonRegisterUserMessage	237
4.31	gos_sysmon_app_config.h File Reference	237
4.31.1	Detailed Description	239
4.31.2	Macro Definition Documentation	239
4.31.2.1	ARM_CORTEX_M4	239
4.31.2.2	CFG_GCP_CHANNELS_MAX_NUMBER	239
4.31.2.3	CFG_IDLE_TASK_STACK_SIZE	239
4.31.2.4	CFG_MESSAGE_MAX_ADDRESSEES	239
4.31.2.5	CFG_MESSAGE_MAX_LENGTH	239
4.31.2.6	CFG_MESSAGE_MAX_NUMBER	239

4.31.2.7	CFG_MESSAGE_MAX_WAITER_IDS	240
4.31.2.8	CFG_MESSAGE_MAX_WAITERS	240
4.31.2.9	CFG_PROC_IDLE_PRIO	240
4.31.2.10	CFG_PROC_MAX_NAME_LENGTH	240
4.31.2.11	CFG_PROC_MAX_NUMBER	240
4.31.2.12	CFG_PROC_MAX_PRIO_LEVELS	240
4.31.2.13	CFG_PROC_USE_SERVICE	240
4.31.2.14	CFG_QUEUE_MAX_ELEMENTS	240
4.31.2.15	CFG_QUEUE_MAX_LENGTH	240
4.31.2.16	CFG_QUEUE_MAX_NAME_LENGTH	241
4.31.2.17	CFG_QUEUE_MAX_NUMBER	241
4.31.2.18	CFG_QUEUE_USE_NAME	241
4.31.2.19	CFG_RESET_ON_ERROR	241
4.31.2.20	CFG_RESET_ON_ERROR_DELAY_MS	241
4.31.2.21	CFG_SCHED_COOPERATIVE	241
4.31.2.22	CFG_SHELL_COMMAND_BUFFER_SIZE	241
4.31.2.23	CFG_SHELL_MAX_COMMAND_LENGTH	241
4.31.2.24	CFG_SHELL_MAX_COMMAND_NUMBER	241
4.31.2.25	CFG_SHELL_MAX_PARAMS_LENGTH	242
4.31.2.26	CFG_SHELL_USE_SERVICE	242
4.31.2.27	CFG_SIGNAL_MAX_NUMBER	242
4.31.2.28	CFG_SIGNAL_MAX_SUBSCRIBERS	242
4.31.2.29	CFG_SYSMON_GCP_CHANNEL_NUM	242
4.31.2.30	CFG_SYSMON_MAX_USER_MESSAGES	242
4.31.2.31	CFG_SYSMON_USE_SERVICE	242
4.31.2.32	CFG_SYSTEM_TASK_STACK_SIZE	242
4.31.2.33	CFG_TARGET_CPU	242
4.31.2.34	CFG_TASK_MAX_NAME_LENGTH	243
4.31.2.35	CFG_TASK_MAX_NUMBER	243
4.31.2.36	CFG_TASK_MAX_STACK_SIZE	243
4.31.2.37	CFG_TASK_MESSAGE_DAEMON_PRIO	243
4.31.2.38	CFG_TASK_MESSAGE_DAEMON_STACK	243
4.31.2.39	CFG_TASK_MIN_STACK_SIZE	243
4.31.2.40	CFG_TASK_PROC_DAEMON_PRIO	243
4.31.2.41	CFG_TASK_PROC_DAEMON_STACK	243
4.31.2.42	CFG_TASK_SHELL_DAEMON_PRIO	243
4.31.2.43	CFG_TASK_SHELL_DAEMON_STACK	244
4.31.2.44	CFG_TASK_SIGNAL_DAEMON_PRIO	244
4.31.2.45	CFG_TASK_SIGNAL_DAEMON_STACK	244
4.31.2.46	CFG_TASK_SYS_PRIO	244
4.31.2.47	CFG_TASK_SYSMON_DAEMON_PRIO	244
4.31.2.48	CFG_TASK_SYSMON_DAEMON_STACK	244
4.31.2.49	CFG_TASK_TIME_DAEMON_PRIO	244
4.31.2.50	CFG_TASK_TIME_DAEMON_STACK	244
4.31.2.51	CFG_TASK_TRACE_DAEMON_PRIO	244
4.31.2.52	CFG_TASK_TRACE_DAEMON_STACK	245
4.31.2.53	CFG_TRACE_MAX_LENGTH	245
4.31.2.54	CFG_USE_PRIO_INHERITANCE	245
4.31.2.55	GOS_CFG_OVERCONFIG	245
4.32	gos_sysmon_driver.c File Reference	245
4.32.1	Detailed Description	246
4.32.2	Function Documentation	246
4.32.2.1	gos_sysmonDriverReceive	246
4.32.2.2	gos_sysmonDriverTransmit	247
4.33	gos_sysmon_driver.h File Reference	247
4.33.1	Detailed Description	249
4.33.2	Typedef Documentation	249
4.33.2.1	gos_sysmonDriverReceive_t	249

4.33.2.2 gos_sysmonDriverTransmit_t	249
4.33.3 Function Documentation	249
4.33.3.1 gos_sysmonDriverReceive	249
4.33.3.2 gos_sysmonDriverTransmit	250
4.34 gos_task.c File Reference	250
4.34.1 Detailed Description	252
4.34.2 Function Documentation	253
4.34.2.1 gos_idleTask	253
4.34.2.2 gos_taskAddPrivilege	254
4.34.2.3 gos_taskBlock	255
4.34.2.4 gos_taskCheckDescriptor	255
4.34.2.5 gos_taskDelete	257
4.34.2.6 gos_taskGetCurrentId	258
4.34.2.7 gos_taskGetData	259
4.34.2.8 gos_taskGetDataByIndex	260
4.34.2.9 gos_taskGetId	261
4.34.2.10 gos_taskGetName	261
4.34.2.11 gos_taskGetNumber	262
4.34.2.12 gos_taskGetOriginalPriority	262
4.34.2.13 gos_taskGetPriority	263
4.34.2.14 gos_taskGetPrivileges	264
4.34.2.15 gos_taskRegister	265
4.34.2.16 gos_taskRegisterTasks	266
4.34.2.17 gos_taskRemovePrivilege	267
4.34.2.18 gos_taskResume	267
4.34.2.19 gos_taskSetOriginalPriority	268
4.34.2.20 gos_taskSetPriority	269
4.34.2.21 gos_taskSetPrivileges	270
4.34.2.22 gos_taskSleep	271
4.34.2.23 gos_taskSuspend	272
4.34.2.24 gos_taskUnblock	273
4.34.2.25 gos_taskWakeUp	274
4.34.2.26 gos_taskYield	275
4.34.3 Variable Documentation	275
4.34.3.1 kernelIdleHookFunction	275
4.34.3.2 kernelTaskDeleteSignal	275
4.34.3.3 taskDescriptors	276
4.35 gos_time.c File Reference	276
4.35.1 Detailed Description	277
4.35.2 Macro Definition Documentation	278
4.35.2.1 TIME_DEFAULT_DAY	278
4.35.2.2 TIME_DEFAULT_MONTH	278
4.35.2.3 TIME_DEFAULT_YEAR	278
4.35.2.4 TIME_SLEEP_TIME_MS	278
4.35.3 Function Documentation	278
4.35.3.1 gos_runTimeAddMicroseconds	278
4.35.3.2 gos_runTimeAddMilliseconds	279
4.35.3.3 gos_runTimeAddSeconds	280
4.35.3.4 gos_runTimeGet	280
4.35.3.5 gos_timeAddMilliseconds	280
4.35.3.6 gos_timeAddSeconds	282
4.35.3.7 gos_timeCompare	282
4.35.3.8 gos_timeDaemonTask	283
4.35.3.9 gos_timeGet	283
4.35.3.10 gos_timeIncreaseSystemTime	284
4.35.3.11 gos_timeInit	285
4.35.3.12 gos_timeSet	285
4.35.4 Variable Documentation	286

4.35.4.1	dayLookupTable	286
4.35.4.2	systemRunTime	286
4.35.4.3	systemTime	286
4.35.4.4	timeDaemonTaskDesc	286
4.35.4.5	timeDaemonTaskId	287
4.35.4.6	timeSignalId	287
4.36	gos_time.h File Reference	287
4.36.1	Detailed Description	289
4.36.2	Enumeration Type Documentation	289
4.36.2.1	gos_timeComprareResult_t	289
4.36.2.2	gos_timeElapsedSenderId_t	289
4.36.2.3	gos_timeMonthEnum_t	290
4.36.3	Function Documentation	290
4.36.3.1	__attribute__	290
4.36.3.2	gos_runTimeAddMicroseconds	290
4.36.3.3	gos_runTimeAddMilliseconds	291
4.36.3.4	gos_runTimeAddSeconds	292
4.36.3.5	gos_runTimeGet	292
4.36.3.6	gos_timeAddMilliseconds	292
4.36.3.7	gos_timeAddSeconds	294
4.36.3.8	gos_timeCompare	294
4.36.3.9	gos_timeGet	295
4.36.3.10	gos_timeIncreaseSystemTime	295
4.36.3.11	gos_timeInit	297
4.36.3.12	gos_timeSet	298
4.37	gos_timer_driver.h File Reference	298
4.37.1	Detailed Description	299
4.37.2	Typedef Documentation	300
4.37.2.1	gos_timerDriverSysTimerGetVal_t	300
4.37.3	Function Documentation	300
4.37.3.1	gos_timerDriverSysTimerGet	300
4.38	gos_trace.c File Reference	301
4.38.1	Detailed Description	302
4.38.2	Macro Definition Documentation	302
4.38.2.1	GOS_TRACE_MUTEX_TMO_MS	302
4.38.2.2	GOS_TRACE_QUEUE_TMO_MS	302
4.38.2.3	GOS_TRACE_TIMESTAMP_FORMAT	302
4.38.2.4	GOS_TRACE_TIMESTAMP_LENGTH	302
4.38.3	Function Documentation	303
4.38.3.1	gos_traceDaemonTask	303
4.38.3.2	gos_tracelInit	303
4.38.3.3	gos_traceTrace	303
4.38.3.4	gos_traceTraceFormatted	304
4.38.3.5	gos_traceTraceFormattedUnsafe	305
4.38.4	Variable Documentation	306
4.38.4.1	formattedBuffer	306
4.38.4.2	timeStampBuffer	306
4.38.4.3	traceDaemonTaskDesc	306
4.38.4.4	traceLine	307
4.38.4.5	traceMutex	307
4.38.4.6	traceQueue	307
4.39	gos_trace.h File Reference	307
4.39.1	Detailed Description	309
4.39.2	Function Documentation	310
4.39.2.1	gos_tracelInit	310
4.39.2.2	gos_traceTrace	310
4.39.2.3	gos_traceTraceFormatted	311
4.39.2.4	gos_traceTraceFormattedUnsafe	312

4.40 gos_trace_driver.c File Reference	313
4.40.1 Detailed Description	314
4.40.2 Function Documentation	315
4.40.2.1 gos_traceDriverTransmitString	315
4.40.2.2 gos_traceDriverTransmitString_Unsafe	316
4.41 gos_trace_driver.h File Reference	317
4.41.1 Detailed Description	319
4.41.2 Typedef Documentation	319
4.41.2.1 gos_traceDriverTransmitString_t	319
4.41.2.2 gos_traceDriverTransmitString_Unsafe_t	319
4.41.3 Function Documentation	319
4.41.3.1 gos_traceDriverTransmitString	319
4.41.3.2 gos_traceDriverTransmitString_Unsafe	320
4.42 gos_trigger.c File Reference	321
4.42.1 Detailed Description	322
4.42.2 Function Documentation	323
4.42.2.1 gos_triggerDecrement	323
4.42.2.2 gos_triggerIncrement	323
4.42.2.3 gos_triggerInit	324
4.42.2.4 gos_triggerReset	325
4.42.2.5 gos_triggerWait	325
4.43 gos_trigger.h File Reference	326
4.43.1 Detailed Description	328
4.43.2 Macro Definition Documentation	328
4.43.2.1 GOS_TRIGGER_ENDLESS_TMO	328
4.43.2.2 GOS_TRIGGER_NO_TMO	328
4.43.3 Function Documentation	328
4.43.3.1 gos_triggerDecrement	328
4.43.3.2 gos_triggerIncrement	329
4.43.3.3 gos_triggerInit	330
4.43.3.4 gos_triggerReset	330
4.43.3.5 gos_triggerWait	331
Index	333

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

gos_driver_functions_t	5
gos_gcpChannelFunctions_t	5
gos_gcpHeaderFrame_t	6
gos_initStruct_t	6
gos_message_t	7
gos_messageWaiterDesc_t	7
gos_mutex_t	8
gos_queue_t	9
gos_queueDescriptor_t	9
gos_queueElement_t	10
gos_shellCommand_t	10
gos_signalDescriptor_t	10
gos_signallInvokeDescriptor	11
gos_sysmonLut_t	11
gos_sysmonUserMessageDescriptor_t	12
gos_trigger_t	12

Chapter 2

File Index

2.1 File List

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

gos.c	GOS source	15
gos.h	GOS header	20
gos_bootloader_config.h	GOS bootloader configuration header	22
gos_config.h	GOS configuration header	30
gos_crc_driver.c	GOS Cyclic Redundancy Check driver source	38
gos_crc_driver.h	GOS Cyclic Redundancy Check driver header	40
gos_driver.c	GOS driver source	41
gos_driver.h	GOS driver header	43
gos_error.c	GOS error handler service source	44
gos_error.h	GOS error handler header	50
gos_gcp.c	GOS General Communication Protocol handler service source	55
gos_gcp.h	GOS General Communication Protocol header	65
gos_kernel.c	GOS kernel source	70
gos_kernel.h	GOS kernel header	91
gos_message.c	GOS message service source	137
gos_message.h	GOS message service header	142
gos_minimal_example_app_config.h	GOS minimal example application configuration header	146
gos_mutex.c	GOS mutex service source	154
gos_mutex.h	GOS mutex service header	159

gos_port.h	GOS port header	164
gos_queue.c	GOS queue service source	169
gos_queue.h	GOS queue service header	178
gos_shell.c	GOS shell service source	188
gos_shell.h	GOS shell service header	197
gos_shell_driver.c	GOS SHELL driver source	203
gos_shell_driver.h	GOS SHELL driver header	205
gos_signal.c	GOS signal service source	208
gos_signal.h	GOS signal service header	213
gos_sysmon.c	GOS system monitoring service source	218
gos_sysmon.h	GOS system monitoring service header	234
gos_sysmon_app_config.h	GOS system monitoring application configuration header	237
gos_sysmon_driver.c	GOS SYSMON driver source	245
gos_sysmon_driver.h	GOS SYSMON driver header	247
gos_task.c	GOS task source	250
gos_time.c	GOS time service source	276
gos_time.h	GOS time service header	287
gos_timer_driver.c	??
gos_timer_driver.h	GOS timer driver header	298
gos_trace.c	GOS trace service source	301
gos_trace.h	GOS trace service header	307
gos_trace_driver.c	GOS trace driver source	313
gos_trace_driver.h	GOS trace driver header	317
gos_trigger.c	GOS trigger service source	321
gos_trigger.h	GOS trigger service header	326

Chapter 3

Data Structure Documentation

3.1 gos_driver_functions_t Struct Reference

```
#include <gos_driver.h>
```

Data Fields

- [gos_shellDriverReceiveChar_t shellDriverReceiveChar](#)
Shell character receive function.
- [gos_shellDriverTransmitString_t shellDriverTransmitString](#)
Shell string transmit function.
- [gos_traceDriverTransmitString_t traceDriverTransmitString](#)
Log string transmit function.
- [gos_traceDriverTransmitString_Unsafe_t traceDriverTransmitStringUnsafe](#)
Log unsafe string transmit function.
- [gos_timerDriverSysTimerGetVal_t timerDriverSysTimerGetValue](#)
System timer get function.
- [gos_sysmonDriverTransmit_t sysmonDriverTransmit](#)
Sysmon transmit function.
- [gos_sysmonDriverReceive_t sysmonDriverReceive](#)
Sysmon receive function.

3.1.1 Detailed Description

Driver functions type.

Definition at line 67 of file gos_driver.h.

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

- [gos_driver.h](#)

3.2 gos_gcpChannelFunctions_t Struct Reference

Data Fields

- [gos_gcpTransmitFunction_t gcpTransmitFunction](#)

- [gos_gcpReceiveFunction_t gcpReceiveFunction](#)
GCP receive function.

3.2.1 Detailed Description

GCP channel functions type.

Definition at line 113 of file [gos_gcp.c](#).

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

- [gos_gcp.c](#)

3.3 gos_gcpHeaderFrame_t Struct Reference

Data Fields

- [u8_t protocolMajor](#)
Protocol version major.
- [u8_t protocolMinor](#)
Protocol version minor.
- [u8_t ackType](#)
Acknowledge type.
- [u8_t dummy](#)
Dummy byte (padding).
- [u16_t messageId](#)
Message ID.
- [u16_t dataSize](#)
Data size.
- [u32_t dataCrc](#)
Data CRC.
- [u32_t headerCrc](#)
Header CRC.

3.3.1 Detailed Description

GCP frame header type.

Definition at line 98 of file [gos_gcp.c](#).

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

- [gos_gcp.c](#)

3.4 gos_initStruct_t Struct Reference

Data Fields

- [char_t initDesc \[32\]](#)
Initialization descriptor text.
- [gos_initFunc_t initFunc](#)
Initializer function.

3.4.1 Detailed Description

Initializer structure type.

Definition at line 87 of file gos.c.

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

- [gos.c](#)

3.5 gos_message_t Struct Reference

```
#include <gos_message.h>
```

Data Fields

- [gos_messageId_t messageId](#)
Message ID.
- [gos_messageSize_t messageSize](#)
Message size.
- [u8_t messageBytes \[CFG_MESSAGE_MAX_LENGTH\]](#)
Message bytes.

3.5.1 Detailed Description

Message type.

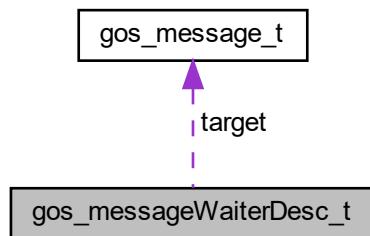
Definition at line 125 of file gos_message.h.

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

- [gos_message.h](#)

3.6 gos_messageWaiterDesc_t Struct Reference

Collaboration diagram for gos_messageWaiterDesc_t:



Data Fields

- [gos_tid_t waiterTaskId](#)
Waiter task ID.
- [gos_messageTimeout_t waitTmo](#)
Wait timeout value.
- [gos_messageTimeout_t waitTmoCounter](#)
Wait timeout counter.
- [gos_messageId_t messageIdArray \[CFG_MESSAGE_MAX_WAITER_IDS\]](#)
Message ID array.
- [gos_message_t * target](#)
Target buffer.
- [bool_t waiterServed](#)
Waiter served flag.

3.6.1 Detailed Description

Message waiter descriptor type.

Definition at line 86 of file `gos_message.c`.

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

- [gos_message.c](#)

3.7 gos_mutex_t Struct Reference

```
#include <gos_mutex.h>
```

Data Fields

- [gos_mutexState_t mutexState](#)
Mutex state.
- [gos_tid_t owner](#)
Mutex owner task.

3.7.1 Detailed Description

Mutex type.

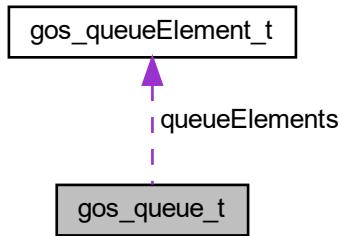
Definition at line 88 of file `gos_mutex.h`.

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

- [gos_mutex.h](#)

3.8 gos_queue_t Struct Reference

Collaboration diagram for gos_queue_t:



Data Fields

- [gos_queueld_t queueld](#)
Queue ID.
- [gos_queueElement_t queueElements \[CFG_QUEUE_MAX_ELEMENTS\]](#)
Queue element array.
- [gos_queueIndex_t actualElementNumber](#)
Actual number of queue elements.

3.8.1 Detailed Description

Queue type.

Definition at line 101 of file [gos_queue.c](#).

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

- [gos_queue.c](#)

3.9 gos_queueDescriptor_t Struct Reference

```
#include <gos_queue.h>
```

Data Fields

- [gos_queueld_t queueld](#)
Queue ID.

3.9.1 Detailed Description

Public queue descriptor type.

Definition at line 121 of file [gos_queue.h](#).

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

- [gos_queue.h](#)

3.10 gos_queueElement_t Struct Reference

Data Fields

- [gos_queueByte_t queueElementBytes \[CFG_QUEUE_MAX_LENGTH\]](#)
Queue element bytes.
- [gos_queueLength_t elementLength](#)
Queue element length.

3.10.1 Detailed Description

Queue element type.

Definition at line 92 of file [gos_queue.c](#).

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

- [gos_queue.c](#)

3.11 gos_shellCommand_t Struct Reference

```
#include <gos_shell.h>
```

Data Fields

- [char_t command \[CFG_SHELL_MAX_COMMAND_LENGTH\]](#)
Command name.
- [gos_shellFunction commandHandler](#)
Command handler function.
- [gos_taskPrivilegeLevel_t commandHandlerPrivileges](#)
Command handler privileges.

3.11.1 Detailed Description

Shell command type.

Definition at line 75 of file [gos_shell.h](#).

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

- [gos_shell.h](#)

3.12 gos_signalDescriptor_t Struct Reference

Data Fields

- [bool_t inUse](#)
Flag to indicate whether the signal is in use.

- [gos_signalHandler_t](#) `handlers` [`CFG_SIGNAL_MAX_SUBSCRIBERS`]
Signal handler array.
- [gos_taskPrivilegeLevel_t](#) `handlerPriviliges` [`CFG_SIGNAL_MAX_SUBSCRIBERS`]
Signal handler privileges array.
- [bool_t](#) `invokeRequired`
Invoke required flag.
- [gos_signalSenderId_t](#) `senderId`
Sender ID.

3.12.1 Detailed Description

Signal descriptor type.

Definition at line 74 of file `gos_signal.c`.

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

- [gos_signal.c](#)

3.13 gos_signalInvokeDescriptor Struct Reference

Data Fields

- [gos_signalId_t](#) `signalId`
Signal ID.
- [gos_signalSenderId_t](#) `senderId`
Sender ID.

3.13.1 Detailed Description

Signal invoke descriptor type.

Definition at line 86 of file `gos_signal.c`.

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

- [gos_signal.c](#)

3.14 gos_sysmonLut_t Struct Reference

Data Fields

- [gos_sysmonMessageId_t](#) `messageId`
Message ID.
- [gos_sysmonMessagePv_t](#) `messagePv`
Message PV.
- [void_t *](#) `pMessagePayload`
Payload pointer.
- [u16_t](#) `payloadSize`
Payload size.
- [gos_sysmonMessageHandler_t](#) `pHandler`
Handler function pointer.

3.14.1 Detailed Description

Look-up table entry structure.

Definition at line 297 of file gos_sysmon.c.

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

- [gos_sysmon.c](#)

3.15 gos_sysmonUserMessageDescriptor_t Struct Reference

```
#include <gos_sysmon.h>
```

Data Fields

- [u16_t messageID](#)
Message ID.
- [u16_t protocolVersion](#)
Message PV.
- [void_t * payload](#)
Pointer to payload target.
- [u32_t payloadSize](#)
Size of payload.
- [gos_sysmonMessageReceivedCallback callback](#)
Callback function pointer.

3.15.1 Detailed Description

User sysmon message descriptor.

Definition at line 70 of file gos_sysmon.h.

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

- [gos_sysmon.h](#)

3.16 gos_trigger_t Struct Reference

```
#include <gos_trigger.h>
```

Data Fields

- [u32_t valueCounter](#)
Value counter.
- [u32_t desiredValue](#)
Desired value.
- [gos_tid_t waiterTaskId](#)
Owner task ID.

3.16.1 Detailed Description

Trigger descriptor type.

Definition at line 85 of file gos_trigger.h.

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

- [gos_trigger.h](#)

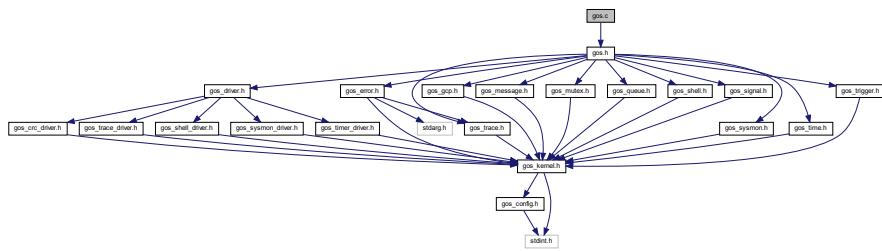
Chapter 4

File Documentation

4.1 gos.c File Reference

GOS source.

```
#include <gos.h>
Include dependency graph for gos.c:
```



Data Structures

- struct [gos_initStruct_t](#)

Macros

- #define [GOS_SYS_TASK_SLEEP_TIME](#) (100u)

Typedefs

- typedef [gos_result_t](#)(* [gos_initFunc_t](#))([void_t](#))

Functions

- [GOS_STATIC void_t gos_systemTask \(\[void_t\]\(#\)\)](#)
Initializes the system.
- [GOS_STATIC gos_result_t gos_Start \(\[void_t\]\(#\)\)](#)
Starts the OS.
- [int main \(\[void_t\]\(#\)\)](#)
- [void_t gos_Dump \(\[void_t\]\(#\)\)](#)

- *GOS dump.*
- `__attribute__ ((weak))`
Platform driver initializer. Used for the platform-specific driver initializations.

Variables

- `GOS_EXTERN gos_signallId_t kernelDumpReadySignal`
- `GOS_STATIC bool_t initError`
- `GOS_STATIC bool_t dumpRequired`
- `GOS_STATIC gos_initStruct_t initializers []`
- `GOS_STATIC gos_tid_t systemTaskId`
- `GOS_STATIC gos_taskDescriptor_t systemTaskDesc`

4.1.1 Detailed Description

GOS source.

Author

Ahmed Gazar

Date

2023-07-12

Version

1.9

For a more detailed description of this service, please refer to [gos.h](#)

Definition in file [gos.c](#).

4.1.2 Macro Definition Documentation

4.1.2.1 `#define GOS_SYS_TASK_SLEEP_TIME (100u)`

System task sleep time.

Definition at line 74 of file gos.c.

4.1.3 Typedef Documentation

4.1.3.1 `typedef gos_result_t(* gos_initFunc_t)(void_t)`

Initializer function type.

Definition at line 82 of file gos.c.

4.1.4 Function Documentation

4.1.4.1 `__attribute__ ((weak))`

Platform driver initializer. Used for the platform-specific driver initializations.

User application initializer. Used for the application-related initializations.

This function is weak and therefore should be over-defined by the user. It prints a warning message to the log output in case it is not over-defined.

Returns

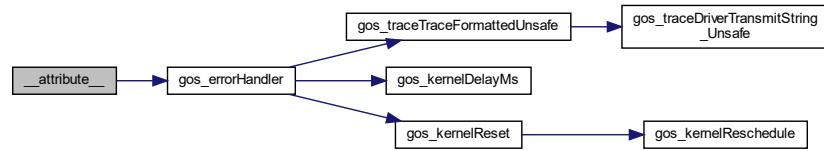
-

Return values

<code>GOS_ERROR</code>	: -
------------------------	-----

Definition at line 238 of file gos.c.

Here is the call graph for this function:



4.1.4.2 void_t gos_Dump(void_t)

GOS dump.

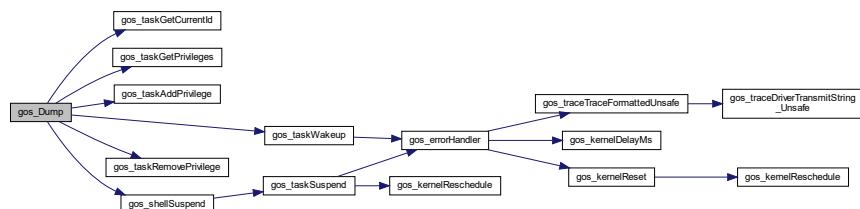
Sets the dump required flag and wakes up the system task that calls the individual dump functions.

Returns

-

Definition at line 205 of file gos.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4.3 GOS_STATIC gos_result_t gos_Start (void_t)

Starts the OS.

Checks whether the initializer function has set the error flag to GOS_FALSE, and if so, it starts the kernel (and thus the scheduling of tasks).

Returns

Result of OS starting.

Return values

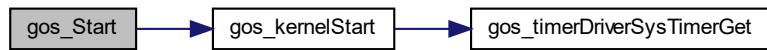
<code>GOS_ERROR</code>	: OS not started due to initialization error or kernel start error.
------------------------	---

Remarks

This function should only return with error. If the initialization is successful, the function is not expected to return.

Definition at line 271 of file gos.c.

Here is the call graph for this function:



4.1.4.4 GOS_STATIC void_t gos_systemTask (void_t)

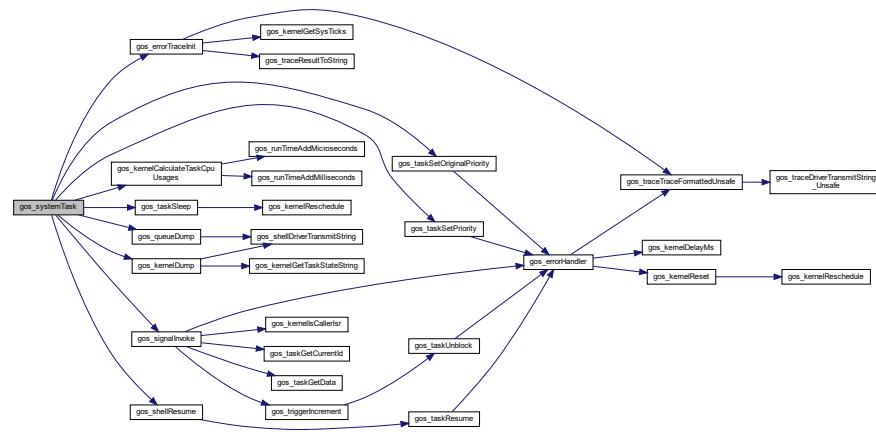
Initializes the system.

Calls the OS service initializer functions and the user application initializer, and deletes itself.

Returns

Definition at line 300 of file gos.c.

Here is the call graph for this function:



4.1.5 Variable Documentation

4.1.5.1 GOS_STATIC bool_t dumpRequired

Dump required flag.

Definition at line 109 of file gos.c.

4.1.5.2 GOS_STATIC bool_t initError

Initialization error flag.

Definition at line 104 of file gos.c.

4.1.5.3 GOS_STATIC gos_initStruct_t initializers[]

Initial value:

```
=  
{  
    {"Queue service initialization" , gos_queueInit},  
    {"Trace service initialization" , gos_traceInit},  
    {"Signal service initialization" , gos_signalInit},  
    {"Time service initialization" , gos_timeInit},  
  
    {"Message service initialization" , gos_messageInit},  
    {"GCP service initialization" , gos_gcplinit},  
  
    {"User application initialization", gos_userApplicationInit}  
}
```

Initializer function and description lookup table.

Definition at line 114 of file qos.c.

4.1.5.4 GOS_STATIC gos_taskDescriptor_t systemTaskDesc

Initial value:

```
=
{
    .taskFunction      = gos_systemTask,
    .taskName         = "gos_system_task",
    .taskPriority     = 0u,
    .taskStackSize    = CFG_SYSTEM_TASK_STACK_SIZE,
    .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL
}
```

Initializer task descriptor.

Definition at line 145 of file gos.c.

4.1.5.5 GOS_STATIC gos_tid_t systemTaskId

Initializer task ID.

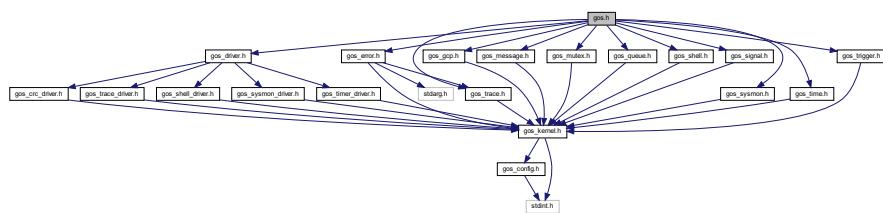
Definition at line 134 of file gos.c.

4.2 gos.h File Reference

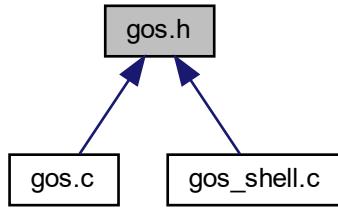
GOS header.

```
#include <gos_driver.h>
#include <gos_error.h>
#include <gos_gcp.h>
#include <gos_message.h>
#include <gos_mutex.h>
#include <gos_queue.h>
#include <gos_shell.h>
#include <gos_signal.h>
#include <gos_sysmon.h>
#include <gos_time.h>
#include <gos_trace.h>
#include <gos_trigger.h>
```

Include dependency graph for gos.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define GOS_VERSION_MAJOR (0)`
- `#define GOS_VERSION_MINOR (11)`

Functions

- `void_t gos_Dump (void_t)`
GOS dump.

4.2.1 Detailed Description

GOS header.

Author

Ahmed Gazar

Date

2024-07-16

Version

1.12

This header is a wrapper for the inclusion of all OS services and drivers for GOS2022 v0.10

Definition in file [gos.h](#).

4.2.2 Macro Definition Documentation

4.2.2.1 `#define GOS_VERSION_MAJOR (0)`

OS major version.

Definition at line 90 of file gos.h.

4.2.2.2 #define GOS_VERSION_MINOR (11)

OS minor version.

Definition at line 95 of file gos.h.

4.2.3 Function Documentation

4.2.3.1 void_t gos_Dump(void_t)

GOS dump.

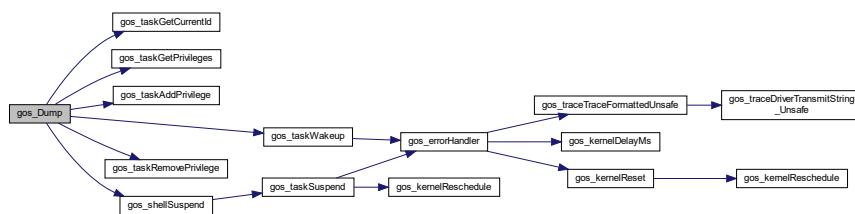
Sets the dump required flag and wakes up the system task that calls the individual dump functions.

Returns

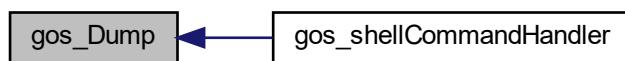
-

Definition at line 205 of file gos.c.

Here is the call graph for this function:



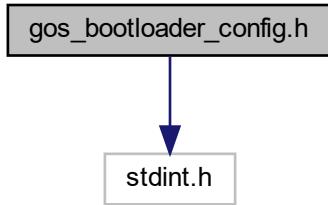
Here is the caller graph for this function:



4.3 gos_bootloader_config.h File Reference

GOS bootloader configuration header.

```
#include <stdint.h>
Include dependency graph for gos_bootloader_config.h:
```



Macros

- #define GOS_CFG_OVERCONFIG
- #define ARM_CORTEX_M4 (1)
- #define CFG_TARGET_CPU (ARM_CORTEX_M4)
- #define CFG_SCHED_COOPERATIVE (0)
- #define CFG_USE_PRIO_INHERITANCE (1)
- #define CFG_TASK_MAX_NAME_LENGTH (32)
- #define CFG_TASK_MAX_NUMBER (18)
- #define CFG_TASK_MIN_STACK_SIZE (0x200)
- #define CFG_TASK_MAX_STACK_SIZE (0x4000)
- #define CFG_IDLE_TASK_STACK_SIZE (0x400)
- #define CFG_SYSTEM_TASK_STACK_SIZE (0x400)
- #define CFG_TASK_SIGNAL_DAEMON_STACK (0x300)
- #define CFG_TASK_PROC_DAEMON_STACK (0x300)
- #define CFG_TASK_TIME_DAEMON_STACK (0x300)
- #define CFG_TASK_MESSAGE_DAEMON_STACK (0x300)
- #define CFG_TASK_SHELL_DAEMON_STACK (0x300)
- #define CFG_TASK_TRACE_DAEMON_STACK (0x400)
- #define CFG_TASK_SYSMON_DAEMON_STACK (0x400)
- #define CFG_TASK_TRACE_DAEMON_PRIO (193)
- #define CFG_TASK_MESSAGE_DAEMON_PRIO (198)
- #define CFG_TASK_SIGNAL_DAEMON_PRIO (197)
- #define CFG_TASK_PROC_DAEMON_PRIO (194)
- #define CFG_TASK_SHELL_DAEMON_PRIO (192)
- #define CFG_TASK_TIME_DAEMON_PRIO (196)
- #define CFG_TASK_SYS_PRIO (195)
- #define CFG_TASK_SYSMON_DAEMON_PRIO (191)
- #define CFG_PROC_USE_SERVICE (0)
- #define CFG_PROC_MAX_PRIO_LEVELS (UINT8_MAX)
- #define CFG_PROC_IDLE_PRIO (CFG_PROC_MAX_PRIO_LEVELS)
- #define CFG_PROC_MAX_NAME_LENGTH (24)
- #define CFG_PROC_MAX_NUMBER (4)
- #define CFG_QUEUE_MAX_NUMBER (1)
- #define CFG_QUEUE_MAX_ELEMENTS (40)
- #define CFG_QUEUE_MAX_LENGTH (200)
- #define CFG_QUEUE_USE_NAME (1)

- #define CFG_QUEUE_MAX_NAME_LENGTH (24)
- #define CFG_SIGNAL_MAX_NUMBER (3)
- #define CFG_SIGNAL_MAX_SUBSCRIBERS (6)
- #define CFG_MESSAGE_MAX_NUMBER (4)
- #define CFG_MESSAGE_MAX_LENGTH (80)
- #define CFG_MESSAGE_MAX_WAITERS (10)
- #define CFG_MESSAGE_MAX_WAITER_IDS (8)
- #define CFG_MESSAGE_MAX_ADDRESSEES (8)
- #define CFG_SHELL_USE_SERVICE (1)
- #define CFG_SHELL_MAX_COMMAND_NUMBER (16)
- #define CFG_SHELL_MAX_COMMAND_LENGTH (20)
- #define CFG_SHELL_MAX_PARAMS_LENGTH (128)
- #define CFG_SHELL_COMMAND_BUFFER_SIZE (200)
- #define CFG_GCP_CHANNELS_MAX_NUMBER (2)
- #define CFG_TRACE_MAX_LENGTH (200)
- #define CFG_SYSMON_USE_SERVICE (0)
- #define CFG_SYSMON_GCP_CHANNEL_NUM (0)
- #define CFG_SYSMON_MAX_USER_MESSAGES (6)
- #define CFG_RESET_ON_ERROR (1)
- #define CFG_RESET_ON_ERROR_DELAY_MS (2000)

4.3.1 Detailed Description

GOS bootloader configuration header.

Author

Ahmed Gazar

Date

2023-09-26

Version

1.0

This header contains the kernel and service configurations of the operating system.

Definition in file [gos_bootloader_config.h](#).

4.3.2 Macro Definition Documentation

4.3.2.1 #define ARM_CORTEX_M4 (1)

ARM Cortex-M4.

Definition at line 49 of file [gos_bootloader_config.h](#).

4.3.2.2 #define CFG_GCP_CHANNELS_MAX_NUMBER (2)

GCP maximum number of channels.

Definition at line 279 of file [gos_bootloader_config.h](#).

4.3.2.3 #define CFG_IDLE_TASK_STACK_SIZE (0x400)

Idle task stack size.

Definition at line 94 of file gos_bootloader_config.h.

4.3.2.4 #define CFG_MESSAGE_MAX_ADDRESSEES (8)

Maximum number of message addressees.

Definition at line 247 of file gos_bootloader_config.h.

4.3.2.5 #define CFG_MESSAGE_MAX_LENGTH (80)

Maximum length of a message in bytes.

Definition at line 235 of file gos_bootloader_config.h.

4.3.2.6 #define CFG_MESSAGE_MAX_NUMBER (4)

Maximum number of messages handled at once.

Definition at line 231 of file gos_bootloader_config.h.

4.3.2.7 #define CFG_MESSAGE_MAX_WAITER_IDS (8)

Maximum number of message IDs a task can wait for (includes the terminating 0).

Definition at line 243 of file gos_bootloader_config.h.

4.3.2.8 #define CFG_MESSAGE_MAX_WAITERS (10)

Maximum number of message waiters.

Definition at line 239 of file gos_bootloader_config.h.

4.3.2.9 #define CFG_PROC_IDLE_PRIO (CFG_PROC_MAX_PRIO_LEVELS)

Idle process priority.

Definition at line 178 of file gos_bootloader_config.h.

4.3.2.10 #define CFG_PROC_MAX_NAME_LENGTH (24)

Maximum process name length.

Definition at line 182 of file gos_bootloader_config.h.

4.3.2.11 #define CFG_PROC_MAX_NUMBER (4)

Maximum number of processes.

Definition at line 186 of file gos_bootloader_config.h.

4.3.2.12 #define CFG_PROC_MAX_PRIO_LEVELS (UINT8_MAX)

Maximum process priority levels.

Definition at line 174 of file gos_bootloader_config.h.

4.3.2.13 #define CFG_PROC_USE_SERVICE (0)

Process service use flag.

Definition at line 170 of file gos_bootloader_config.h.

4.3.2.14 #define CFG_QUEUE_MAX_ELEMENTS (40)

Maximum number of queue elements.

Definition at line 198 of file gos_bootloader_config.h.

4.3.2.15 #define CFG_QUEUE_MAX_LENGTH (200)

Maximum queue length.

Definition at line 202 of file gos_bootloader_config.h.

4.3.2.16 #define CFG_QUEUE_MAX_NAME_LENGTH (24)

Maximum queue name length.

Definition at line 210 of file gos_bootloader_config.h.

4.3.2.17 #define CFG_QUEUE_MAX_NUMBER (1)

Maximum number of queues.

Definition at line 194 of file gos_bootloader_config.h.

4.3.2.18 #define CFG_QUEUE_USE_NAME (1)

Queue use name flag.

Definition at line 206 of file gos_bootloader_config.h.

4.3.2.19 #define CFG_RESET_ON_ERROR (1)

Flag to indicate if the system should reset on error.

Definition at line 313 of file gos_bootloader_config.h.

4.3.2.20 #define CFG_RESET_ON_ERROR_DELAY_MS (2000)

Delay time before system reset.

Definition at line 317 of file gos_bootloader_config.h.

4.3.2.21 `#define CFG_SCHED_COOPERATIVE (0)`

Cooperative scheduling flag.

Definition at line 62 of file gos_bootloader_config.h.

4.3.2.22 `#define CFG_SHELL_COMMAND_BUFFER_SIZE (200)`

Command buffer size.

Definition at line 271 of file gos_bootloader_config.h.

4.3.2.23 `#define CFG_SHELL_MAX_COMMAND_LENGTH (20)`

Maximum command length.

Definition at line 263 of file gos_bootloader_config.h.

4.3.2.24 `#define CFG_SHELL_MAX_COMMAND_NUMBER (16)`

Maximum number of shell commands.

Definition at line 259 of file gos_bootloader_config.h.

4.3.2.25 `#define CFG_SHELL_MAX_PARAMS_LENGTH (128)`

Maximum parameters length.

Definition at line 267 of file gos_bootloader_config.h.

4.3.2.26 `#define CFG_SHELL_USE_SERVICE (1)`

Shell service use flag.

Definition at line 255 of file gos_bootloader_config.h.

4.3.2.27 `#define CFG_SIGNAL_MAX_NUMBER (3)`

Maximum number of signals.

Definition at line 219 of file gos_bootloader_config.h.

4.3.2.28 `#define CFG_SIGNAL_MAX_SUBSCRIBERS (6)`

Maximum number of signal subscribers.

Definition at line 223 of file gos_bootloader_config.h.

4.3.2.29 `#define CFG_SYSMON_GCP_CHANNEL_NUM (0)`

Define sysmon GCP channel number.

Definition at line 300 of file gos_bootloader_config.h.

4.3.2.30 #define CFG_SYSMON_MAX_USER_MESSAGES (6)

Maximum number of user messages.

Definition at line 305 of file gos_bootloader_config.h.

4.3.2.31 #define CFG_SYSMON_USE_SERVICE (0)

Sysmon use service flag.

Definition at line 295 of file gos_bootloader_config.h.

4.3.2.32 #define CFG_SYSTEM_TASK_STACK_SIZE (0x400)

System task stack size.

Definition at line 98 of file gos_bootloader_config.h.

4.3.2.33 #define CFG_TARGET_CPU (ARM_CORTEX_M4)

Target CPU.

Definition at line 54 of file gos_bootloader_config.h.

4.3.2.34 #define CFG_TASK_MAX_NAME_LENGTH (32)

Maximum task name length.

Definition at line 74 of file gos_bootloader_config.h.

4.3.2.35 #define CFG_TASK_MAX_NUMBER (18)

Maximum number of tasks.

Definition at line 78 of file gos_bootloader_config.h.

4.3.2.36 #define CFG_TASK_MAX_STACK_SIZE (0x4000)

Maximum task stack size.

Definition at line 90 of file gos_bootloader_config.h.

4.3.2.37 #define CFG_TASK_MESSAGE_DAEMON_PRIO (198)

Message daemon task priority.

Definition at line 138 of file gos_bootloader_config.h.

4.3.2.38 #define CFG_TASK_MESSAGE_DAEMON_STACK (0x300)

Message daemon task stack size.

Definition at line 114 of file gos_bootloader_config.h.

4.3.2.39 #define CFG_TASK_MIN_STACK_SIZE (0x200)

Minimum task stack size.

Definition at line 86 of file gos_bootloader_config.h.

4.3.2.40 #define CFG_TASK_PROC_DAEMON_PRIO (194)

Process daemon task priority.

Definition at line 146 of file gos_bootloader_config.h.

4.3.2.41 #define CFG_TASK_PROC_DAEMON_STACK (0x300)

Process daemon task stack size.

Definition at line 106 of file gos_bootloader_config.h.

4.3.2.42 #define CFG_TASK_SHELL_DAEMON_PRIO (192)

Shell daemon task priority.

Definition at line 150 of file gos_bootloader_config.h.

4.3.2.43 #define CFG_TASK_SHELL_DAEMON_STACK (0x300)

Shell daemon task stack size.

Definition at line 118 of file gos_bootloader_config.h.

4.3.2.44 #define CFG_TASK_SIGNAL_DAEMON_PRIO (197)

Signal daemon task priority.

Definition at line 142 of file gos_bootloader_config.h.

4.3.2.45 #define CFG_TASK_SIGNAL_DAEMON_STACK (0x300)

Signal daemon task stack size.

Definition at line 102 of file gos_bootloader_config.h.

4.3.2.46 #define CFG_TASK_SYS_PRIO (195)

System task priority.

Definition at line 158 of file gos_bootloader_config.h.

4.3.2.47 #define CFG_TASK_SYSMON_DAEMON_PRIO (191)

Sysmon daemon task priority.

Definition at line 162 of file gos_bootloader_config.h.

4.3.2.48 `#define CFG_TASK_SYSMON_DAEMON_STACK (0x400)`

Sysmon daemon task stack size.

Definition at line 126 of file gos_bootloader_config.h.

4.3.2.49 `#define CFG_TASK_TIME_DAEMON_PRIO (196)`

Time daemon task priority.

Definition at line 154 of file gos_bootloader_config.h.

4.3.2.50 `#define CFG_TASK_TIME_DAEMON_STACK (0x300)`

Time daemon task stack size.

Definition at line 110 of file gos_bootloader_config.h.

4.3.2.51 `#define CFG_TASK_TRACE_DAEMON_PRIO (193)`

Trace daemon task priority.

Definition at line 134 of file gos_bootloader_config.h.

4.3.2.52 `#define CFG_TASK_TRACE_DAEMON_STACK (0x400)`

Log daemon task stack size.

Definition at line 122 of file gos_bootloader_config.h.

4.3.2.53 `#define CFG_TRACE_MAX_LENGTH (200)`

Trace maximum (line) length.

Definition at line 287 of file gos_bootloader_config.h.

4.3.2.54 `#define CFG_USE_PRIO_INHERITANCE (1)`

Priority inheritance flag for lock.

Definition at line 66 of file gos_bootloader_config.h.

4.3.2.55 `#define GOS_CFG_OVERCONFIG`

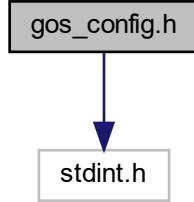
Overconfiguration macro.

Definition at line 42 of file gos_bootloader_config.h.

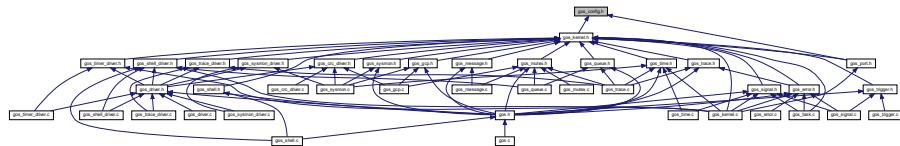
4.4 gos_config.h File Reference

GOS configuration header.

```
#include <stdint.h>
Include dependency graph for gos_config.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- #define ARM_CORTEX_M4 (1)
- #define CFG_TARGET_CPU (ARM_CORTEX_M4)
- #define CFG_SCHED_COOPERATIVE (0)
- #define CFG_USE_PRIO_INHERITANCE (1)
- #define CFG_TASK_MAX_NAME_LENGTH (32)
- #define CFG_TASK_MAX_NUMBER (48)
- #define CFG_TASK_MIN_STACK_SIZE (0x200)
- #define CFG_TASK_MAX_STACK_SIZE (0x4000)
- #define CFG_IDLE_TASK_STACK_SIZE (0x400)
- #define CFG_SYSTEM_TASK_STACK_SIZE (0x400)
- #define CFG_TASK_SIGNAL_DAEMON_STACK (0x400)
- #define CFG_TASK_TIME_DAEMON_STACK (0x300)
- #define CFG_TASK_MESSAGE_DAEMON_STACK (0x300)
- #define CFG_TASK_SHELL_DAEMON_STACK (0x300)
- #define CFG_TASK_TRACE_DAEMON_STACK (0x400)
- #define CFG_TASK_SYSMON_DAEMON_STACK (0x800)
- #define CFG_TASK_TRACE_DAEMON_PRIO (193)
- #define CFG_TASK_MESSAGE_DAEMON_PRIO (198)
- #define CFG_TASK_SIGNAL_DAEMON_PRIO (197)
- #define CFG_TASK_SHELL_DAEMON_PRIO (192)
- #define CFG_TASK_TIME_DAEMON_PRIO (196)
- #define CFG_TASK_SYS_PRIO (195)
- #define CFG_TASK_SYSMON_DAEMON_PRIO (191)
- #define CFG_QUEUE_MAX_NUMBER (4)
- #define CFG_QUEUE_MAX_ELEMENTS (40)
- #define CFG_QUEUE_MAX_LENGTH (200)

- #define CFG_QUEUE_USE_NAME (1)
- #define CFG_QUEUE_MAX_NAME_LENGTH (24)
- #define CFG_SIGNAL_MAX_NUMBER (6)
- #define CFG_SIGNAL_MAX_SUBSCRIBERS (6)
- #define CFG_MESSAGE_MAX_NUMBER (8)
- #define CFG_MESSAGE_MAX_LENGTH (80)
- #define CFG_MESSAGE_MAX_WAITERS (10)
- #define CFG_MESSAGE_MAX_WAITER_IDS (8)
- #define CFG_MESSAGE_MAX_ADDRESSEES (8)
- #define CFG_SHELL_USE_SERVICE (1)
- #define CFG_SHELL_MAX_COMMAND_NUMBER (16)
- #define CFG_SHELL_MAX_COMMAND_LENGTH (20)
- #define CFG_SHELL_MAX_PARAMS_LENGTH (128)
- #define CFG_SHELL_COMMAND_BUFFER_SIZE (200)
- #define CFG_GCP_CHANNELS_MAX_NUMBER (4)
- #define CFG_TRACE_MAX_LENGTH (200)
- #define CFG_SYSMON_USE_SERVICE (1)
- #define CFG_SYSMON_GCP_CHANNEL_NUM (0)
- #define CFG_SYSMON_MAX_USER_MESSAGES (6)
- #define CFG_RESET_ON_ERROR (1)
- #define CFG_RESET_ON_ERROR_DELAY_MS (3000)

4.4.1 Detailed Description

GOS configuration header.

Author

Ahmed Gazar

Date

2024-04-24

Version

1.10

This header contains the kernel and service configurations of the operating system.

Definition in file [gos_config.h](#).

4.4.2 Macro Definition Documentation

4.4.2.1 #define ARM_CORTEX_M4 (1)

ARM Cortex-M4

Definition at line 74 of file [gos_config.h](#).

4.4.2.2 #define CFG_GCP_CHANNELS_MAX_NUMBER (4)

GCP maximum number of channels.

Definition at line 272 of file [gos_config.h](#).

4.4.2.3 #define CFG_IDLE_TASK_STACK_SIZE (0x400)

Idle task stack size.

Definition at line 119 of file gos_config.h.

4.4.2.4 #define CFG_MESSAGE_MAX_ADDRESSEES (8)

Maximum number of message addressees.

Definition at line 240 of file gos_config.h.

4.4.2.5 #define CFG_MESSAGE_MAX_LENGTH (80)

Maximum length of a message in bytes.

Definition at line 228 of file gos_config.h.

4.4.2.6 #define CFG_MESSAGE_MAX_NUMBER (8)

Maximum number of messages handled at once.

Definition at line 224 of file gos_config.h.

4.4.2.7 #define CFG_MESSAGE_MAX_WAITER_IDS (8)

Maximum number of message IDs a task can wait for (includes the terminating 0).

Definition at line 236 of file gos_config.h.

4.4.2.8 #define CFG_MESSAGE_MAX_WAITERS (10)

Maximum number of message waiters.

Definition at line 232 of file gos_config.h.

4.4.2.9 #define CFG_QUEUE_MAX_ELEMENTS (40)

Maximum number of queue elements.

Definition at line 191 of file gos_config.h.

4.4.2.10 #define CFG_QUEUE_MAX_LENGTH (200)

Maximum queue length.

Definition at line 195 of file gos_config.h.

4.4.2.11 #define CFG_QUEUE_MAX_NAME_LENGTH (24)

Maximum queue name length.

Definition at line 203 of file gos_config.h.

4.4.2.12 #define CFG_QUEUE_MAX_NUMBER (4)

Maximum number of queues.

Definition at line 187 of file gos_config.h.

4.4.2.13 #define CFG_QUEUE_USE_NAME (1)

Queue use name flag.

Definition at line 199 of file gos_config.h.

4.4.2.14 #define CFG_RESET_ON_ERROR (1)

Flag to indicate if the system should reset on error.

Definition at line 306 of file gos_config.h.

4.4.2.15 #define CFG_RESET_ON_ERROR_DELAY_MS (3000)

Delay time before system reset.

Definition at line 310 of file gos_config.h.

4.4.2.16 #define CFG_SCHED_COOPERATIVE (0)

Cooperative scheduling flag.

Definition at line 87 of file gos_config.h.

4.4.2.17 #define CFG_SHELL_COMMAND_BUFFER_SIZE (200)

Command buffer size.

Definition at line 264 of file gos_config.h.

4.4.2.18 #define CFG_SHELL_MAX_COMMAND_LENGTH (20)

Maximum command length.

Definition at line 256 of file gos_config.h.

4.4.2.19 #define CFG_SHELL_MAX_COMMAND_NUMBER (16)

Maximum number of shell commands.

Definition at line 252 of file gos_config.h.

4.4.2.20 #define CFG_SHELL_MAX_PARAMS_LENGTH (128)

Maximum parameters length.

Definition at line 260 of file gos_config.h.

4.4.2.21 #define CFG_SHELL_USE_SERVICE (1)

Shell service use flag.

Definition at line 248 of file gos_config.h.

4.4.2.22 #define CFG_SIGNAL_MAX_NUMBER (6)

Maximum number of signals.

Definition at line 212 of file gos_config.h.

4.4.2.23 #define CFG_SIGNAL_MAX_SUBSCRIBERS (6)

Maximum number of signal subscribers.

Definition at line 216 of file gos_config.h.

4.4.2.24 #define CFG_SYSMON_GCP_CHANNEL_NUM (0)

Define sysmon GCP channel number.

Definition at line 293 of file gos_config.h.

4.4.2.25 #define CFG_SYSMON_MAX_USER_MESSAGES (6)

Maximum number of user messages.

Definition at line 298 of file gos_config.h.

4.4.2.26 #define CFG_SYSMON_USE_SERVICE (1)

Sysmon use service flag.

Definition at line 288 of file gos_config.h.

4.4.2.27 #define CFG_SYSTEM_TASK_STACK_SIZE (0x400)

System task stack size.

Definition at line 123 of file gos_config.h.

4.4.2.28 #define CFG_TARGET_CPU (ARM_CORTEX_M4)

Target CPU.

Definition at line 79 of file gos_config.h.

4.4.2.29 #define CFG_TASK_MAX_NAME_LENGTH (32)

Maximum task name length.

Definition at line 99 of file gos_config.h.

4.4.2.30 #define CFG_TASK_MAX_NUMBER (48)

Maximum number of tasks.

Definition at line 103 of file gos_config.h.

4.4.2.31 #define CFG_TASK_MAX_STACK_SIZE (0x4000)

Maximum task stack size.

Definition at line 115 of file gos_config.h.

4.4.2.32 #define CFG_TASK_MESSAGE_DAEMON_PRIO (198)

Message daemon task priority.

Definition at line 159 of file gos_config.h.

4.4.2.33 #define CFG_TASK_MESSAGE_DAEMON_STACK (0x300)

Message daemon task stack size.

Definition at line 135 of file gos_config.h.

4.4.2.34 #define CFG_TASK_MIN_STACK_SIZE (0x200)

Minimum task stack size.

Definition at line 111 of file gos_config.h.

4.4.2.35 #define CFG_TASK_SHELL_DAEMON_PRIO (192)

Shell daemon task priority.

Definition at line 167 of file gos_config.h.

4.4.2.36 #define CFG_TASK_SHELL_DAEMON_STACK (0x300)

Shell daemon task stack size.

Definition at line 139 of file gos_config.h.

4.4.2.37 #define CFG_TASK_SIGNAL_DAEMON_PRIO (197)

Signal daemon task priority.

Definition at line 163 of file gos_config.h.

4.4.2.38 #define CFG_TASK_SIGNAL_DAEMON_STACK (0x400)

Signal daemon task stack size.

Definition at line 127 of file gos_config.h.

4.4.2.39 #define CFG_TASK_SYS_PRIO (195)

System task priority.

Definition at line 175 of file gos_config.h.

4.4.2.40 #define CFG_TASK_SYSMON_DAEMON_PRIO (191)

Sysmon daemon task priority.

Definition at line 179 of file gos_config.h.

4.4.2.41 #define CFG_TASK_SYSMON_DAEMON_STACK (0x800)

Sysmon daemon task stack size.

Definition at line 147 of file gos_config.h.

4.4.2.42 #define CFG_TASK_TIME_DAEMON_PRIO (196)

Time daemon task priority.

Definition at line 171 of file gos_config.h.

4.4.2.43 #define CFG_TASK_TIME_DAEMON_STACK (0x300)

Time daemon task stack size.

Definition at line 131 of file gos_config.h.

4.4.2.44 #define CFG_TASK_TRACE_DAEMON_PRIO (193)

Trace daemon task priority.

Definition at line 155 of file gos_config.h.

4.4.2.45 #define CFG_TASK_TRACE_DAEMON_STACK (0x400)

Log daemon task stack size.

Definition at line 143 of file gos_config.h.

4.4.2.46 #define CFG_TRACE_MAX_LENGTH (200)

Trace maximum (line) length.

Definition at line 280 of file gos_config.h.

4.4.2.47 #define CFG_USE_PRIO_INHERITANCE (1)

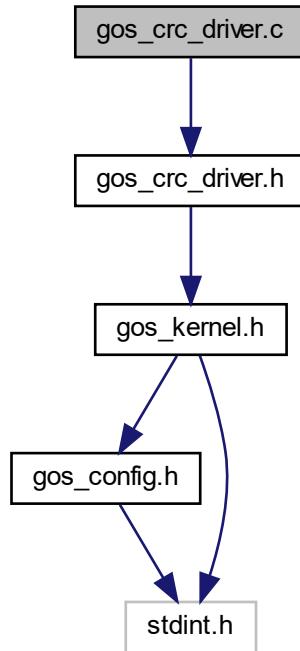
Priority inheritance flag for lock.

Definition at line 91 of file gos_config.h.

4.5 gos_crc_driver.c File Reference

GOS Cyclic Redundancy Check driver source.

```
#include "gos_crc_driver.h"  
Include dependency graph for gos_crc_driver.c:
```



Macros

- #define **CRC_INITIAL_VALUE** (0xFFFFFFFF)
- #define **CRC_POLYNOMIAL_VALUE** (0xEDB88320)

Functions

- **u32_t gos_crcDriverGetCrc (u8_t *pData, u32_t dataSize)**
Calculates the 32-bit CRC value of the given data buffer.

4.5.1 Detailed Description

GOS Cyclic Redundancy Check driver source.

Author

Ahmed Gazar

Date

2022-12-10

Version

1.0

For a more detailed description of this driver, please refer to [gos_crc_driver.h](#)

Definition in file [gos_crc_driver.c](#).

4.5.2 Macro Definition Documentation

4.5.2.1 #define CRC_INITIAL_VALUE (0xFFFFFFFF)

CRC initializer value.

Definition at line 60 of file [gos_crc_driver.c](#).

4.5.2.2 #define CRC_POLYNOMIAL_VALUE (0xEDB88320)

CRC polynomial value.

Definition at line 65 of file [gos_crc_driver.c](#).

4.5.3 Function Documentation

4.5.3.1 u32_t gos_crcDriverGetCrc (u8_t * pData, u32_t dataSize)

Calculates the 32-bit CRC value of the given data buffer.

Calculates the 32-bit CRC value of the given data buffer.

Parameters

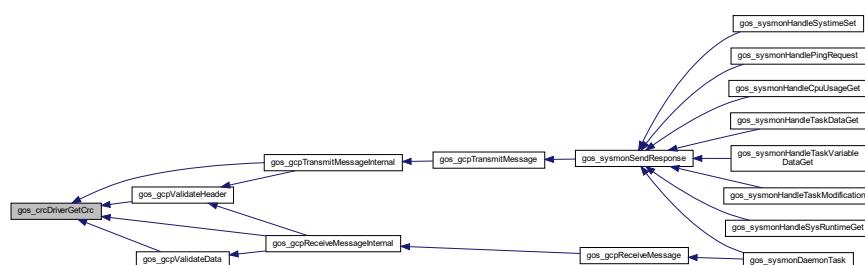
<i>pData</i>	: Pointer to the data buffer.
<i>dataSize</i>	: Size of the data buffer in bytes.

Returns

32-bit CRC value.

Definition at line 70 of file [gos_crc_driver.c](#).

Here is the caller graph for this function:

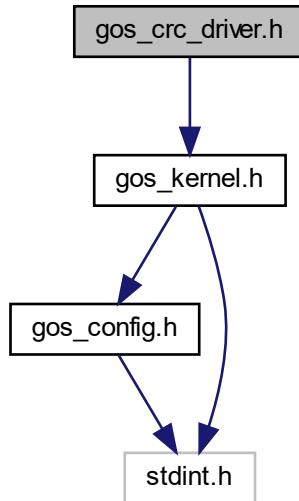


4.6 gos_crc_driver.h File Reference

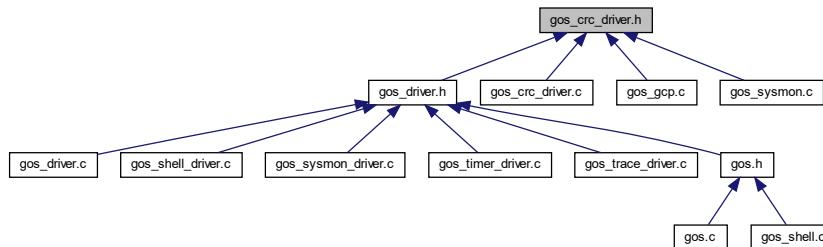
GOS Cyclic Redundancy Check driver header.

```
#include "gos_kernel.h"
```

Include dependency graph for gos_crc_driver.h:



This graph shows which files directly or indirectly include this file:



Functions

- `u32_t gos_crcDriverGetCrc (u8_t *pData, u32_t dataSize)`

Calculates the 32-bit CRC value of the given data buffer.

4.6.1 Detailed Description

GOS Cyclic Redundancy Check driver header.

Author

Ahmed Gazar

Date

2022-12-10

Version

1.0

This driver provides a simple 32-bit CRC calculator algorithm.

Definition in file [gos_crc_driver.h](#).

4.6.2 Function Documentation

4.6.2.1 u32_t gos_crcDriverGetCrc (u8_t * pData, u32_t dataSize)

Calculates the 32-bit CRC value of the given data buffer.

Calculates the 32-bit CRC value of the given data buffer.

Parameters

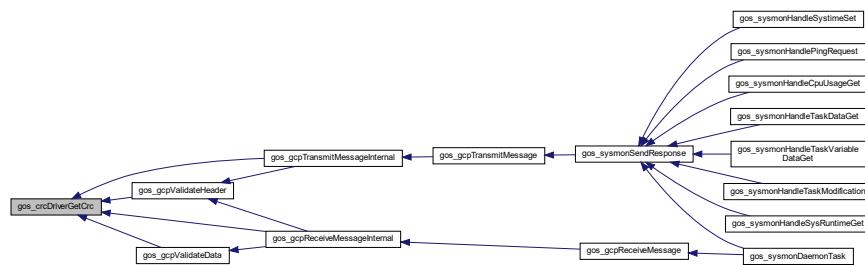
<i>pData</i>	: Pointer to the data buffer.
<i>dataSize</i>	: Size of the data buffer in bytes.

Returns

32-bit CRC value.

Definition at line 70 of file gos_crc_driver.c.

Here is the caller graph for this function:

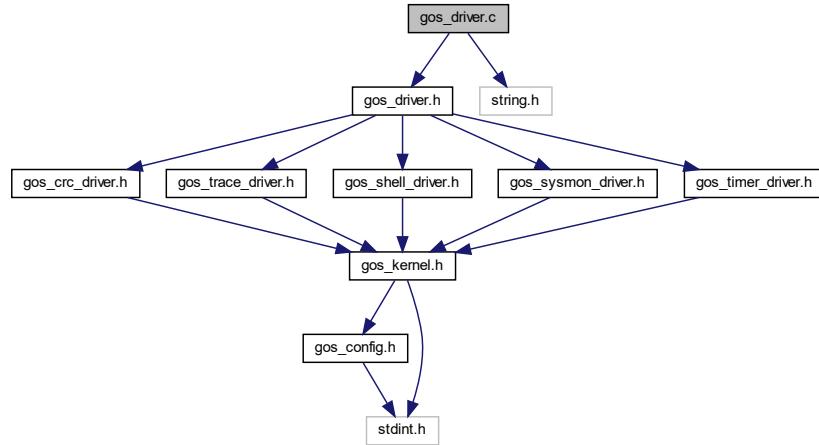


4.7 gos_driver.c File Reference

GOS driver source.

```
#include "gos_driver.h"
#include <string.h>
```

Include dependency graph for gos_driver.c:



Functions

- [gos_result_t gos_driverInit \(gos_driver_functions_t *pDriverFunctions\)](#)
Initializes the kernel drivers.

Variables

- [gos_driver_functions_t driverFunctions = { NULL, NULL, NULL, NULL, NULL, NULL, NULL }](#)

4.7.1 Detailed Description

GOS driver source.

Author

Ahmed Gazar

Date

2022-12-11

Version

1.0

For a more detailed description of this driver, please refer to [gos_driver.h](#)

Definition in file [gos_driver.c](#).

4.7.2 Function Documentation

4.7.2.1 [gos_result_t gos_driverInit \(gos_driver_functions_t * pDriverFunctions \)](#)

Initializes the kernel drivers.

Copies the function pointers into the internal structure.

Parameters

<i>pDriverFunctions</i>	: Pointer to the function pointer structure.
-------------------------	--

Returns

Result of initialization.

Return values

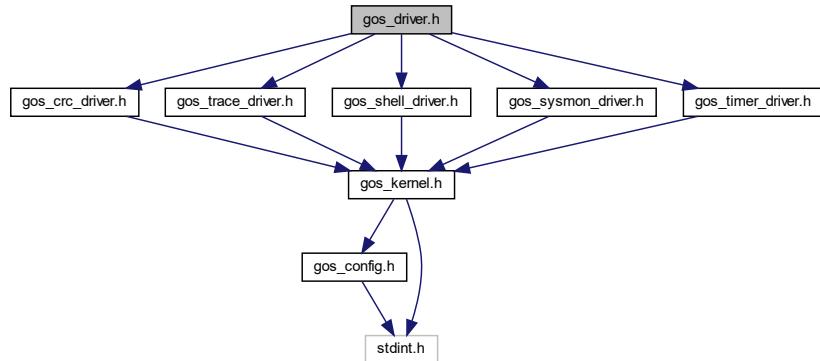
<i>GOS_SUCCESS</i>	: Initialization successful.
<i>GOS_ERROR</i>	: NULL pointer parameter.

Definition at line 63 of file gos_driver.c.

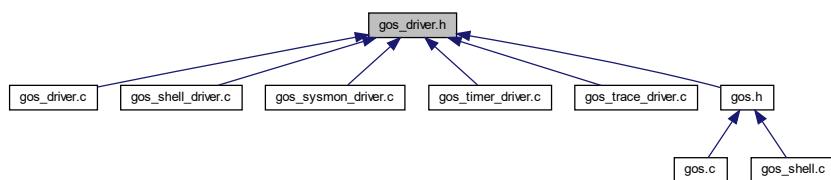
4.8 gos_driver.h File Reference

GOS driver header.

```
#include <gos_crc_driver.h>
#include <gos_trace_driver.h>
#include <gos_shell_driver.h>
#include <gos_sysmon_driver.h>
#include <gos_timer_driver.h>
Include dependency graph for gos_driver.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [gos_driver_functions_t](#)

Functions

- [gos_result_t gos_driverInit \(gos_driver_functions_t *pDriverFunctions\)](#)
Initializes the kernel drivers.

4.8.1 Detailed Description

GOS driver header.

Author

Ahmed Gazar

Date

2023-07-25

Version

1.2

This header is used for the inclusion of all driver skeletons.

Definition in file [gos_driver.h](#).

4.8.2 Function Documentation

4.8.2.1 [gos_result_t gos_driverInit \(gos_driver_functions_t * pDriverFunctions \)](#)

Initializes the kernel drivers.

Copies the function pointers into the internal structure.

Parameters

<i>pDriverFunctions</i>	: Pointer to the function pointer structure.
-------------------------	--

Returns

Result of initialization.

Return values

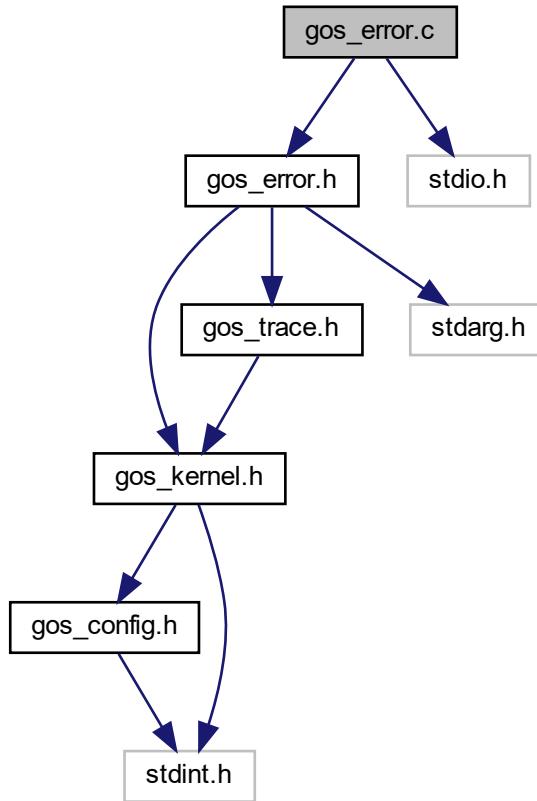
<i>GOS_SUCCESS</i>	: Initialization successful.
<i>GOS_ERROR</i>	: NULL pointer parameter.

Definition at line 63 of file [gos_driver.c](#).

4.9 [gos_error.c](#) File Reference

GOS error handler service source.

```
#include <gos_error.h>
#include <stdio.h>
Include dependency graph for gos_error.c:
```



Macros

- #define SEPARATOR_LINE "+-----+\\r\\n"
- #define ERROR_BUFFER_SIZE (80u)
- #define RESULT_STRING_SUCCESS " OK "
- #define RESULT_STRING_ERROR " ERROR "
- #define RESULT_STRING_UNKNOWN "UNKNOWN"

Functions

- [GOS_STATIC char_t * gos_traceResultToString \(gos_result_t *pResult\)](#)
Converts the result enumerator to a string.
- [__attribute__ \(\(weak\)\)](#)
Platform driver initializer. Used for the platform-specific driver initializations.
- [void_t gos_errorHandler \(gos_errorLevel_t errorLevel, const char_t *function, u32_t line, const char_t *errorMessage,...\)](#)
Handles the given error.
- [gos_result_t gos_errorTraceInit \(const char_t *initDescription, gos_result_t initResult\)](#)

Traces an initialization message.

Variables

- `GOS_STATIC char_t errorBuffer [ERROR_BUFFER_SIZE]`

4.9.1 Detailed Description

GOS error handler service source.

Author

Ahmed Gazar

Date

2024-04-24

Version

2.4

For a more detailed description of this service, please refer to [gos_error.h](#)

Definition in file [gos_error.c](#).

4.9.2 Macro Definition Documentation

4.9.2.1 #define ERROR_BUFFER_SIZE (80u)

Error buffer size.

Definition at line 74 of file [gos_error.c](#).

4.9.2.2 #define RESULT_STRING_ERROR " ERROR "

Error result string.

Definition at line 84 of file [gos_error.c](#).

4.9.2.3 #define RESULT_STRING_SUCCESS " OK "

Success result string.

Definition at line 79 of file [gos_error.c](#).

4.9.2.4 #define RESULT_STRING_UNKNOWN "UNKNOWN"

Unknown result string.

Definition at line 89 of file [gos_error.c](#).

4.9.2.5 #define SEPARATOR_LINE "+-----+\r\n"

Separator line.

Definition at line 69 of file [gos_error.c](#).

4.9.3 Function Documentation

4.9.3.1 __attribute__ (weak)

Platform driver initializer. Used for the platform-specific driver initializations.

User application initializer. Used for the application-related initializations.

This function is weak and therefore should be over-defined by the user. It prints a warning message to the log output in case it is not over-defined.

Returns

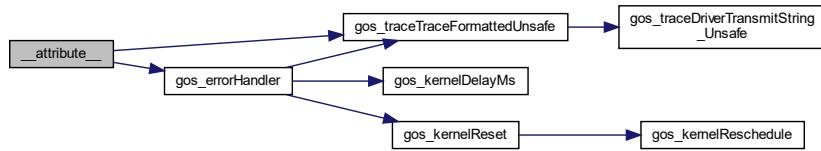
-

Return values

GOS_ERROR	:	-
-----------	---	---

Definition at line 107 of file gos_error.c.

Here is the call graph for this function:



4.9.3.2 void_t gos_errorHandler (gos_errorLevel_t errorLevel, const char_t * function, u32_t line, const char_t * errorMessage, ...)

Handles the given error.

Prints the formatted error message on the trace output, and based on the error level, it returns or stays in an infinite loop and disables scheduling.

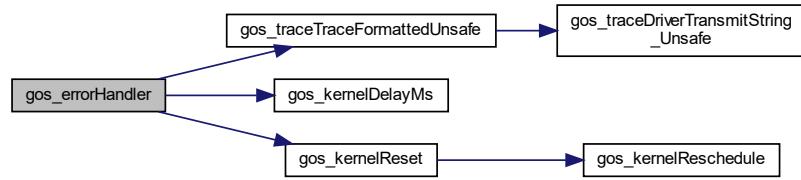
Parameters

<i>errorLevel</i>	: Level of error (OS/user, warning/fatal).
<i>function</i>	: Function name.
<i>line</i>	: Line number.
<i>errorMessage</i>	: Error message.
...	: Formatter variable arguments.

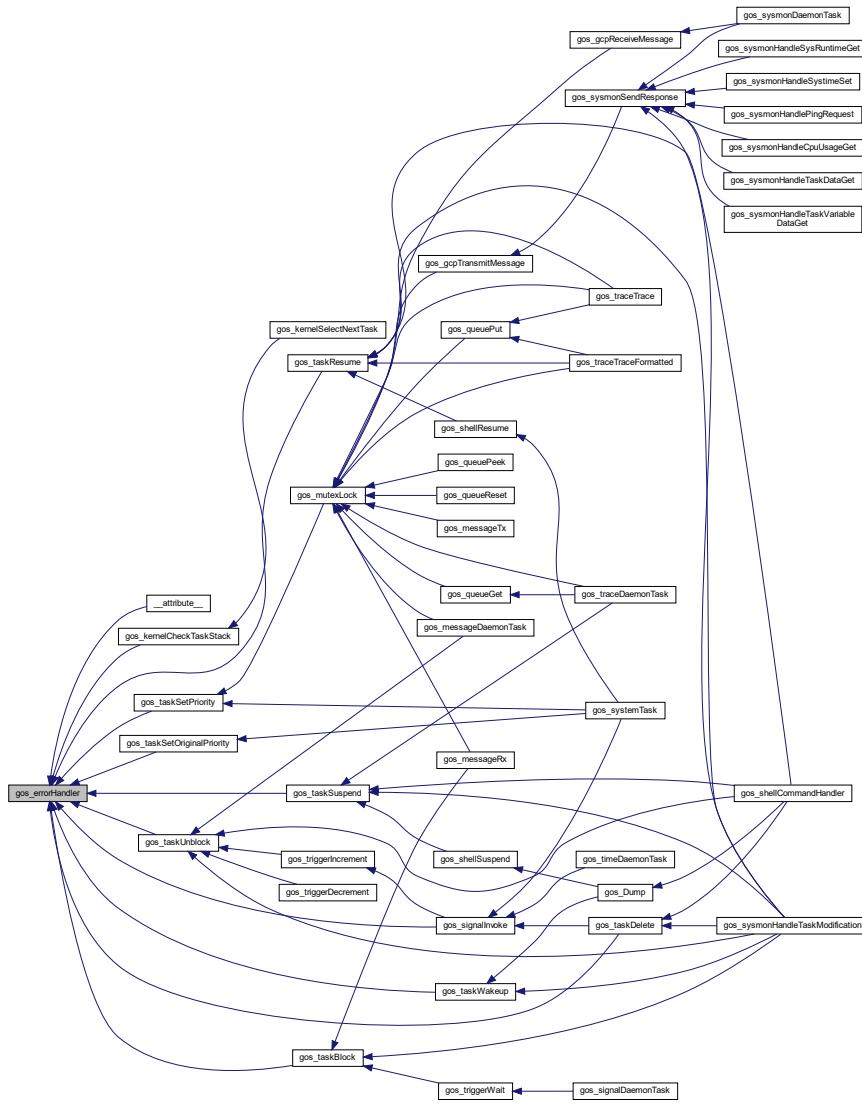
Returns

Definition at line 128 of file gos_error.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.3 gos_result_t gos_errorTraceInit(const char_t * initDescription, gos_result_t initResult)

Traces an initialization message.

Writes the formatted initialization message on the trace output.

Parameters

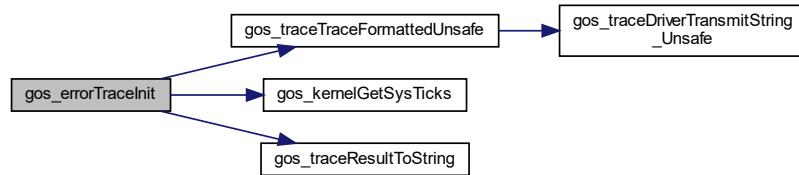
<i>initDescription</i>	: Message to describe the initialization step.
<i>initResult</i>	: Result of initialization.

Returns

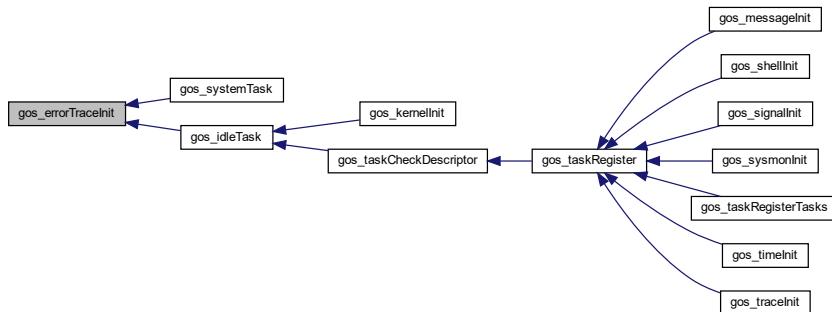
Result of initialization.

Definition at line 236 of file gos_error.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.4 GOS_STATIC char_t * gos_traceResultToString(gos_result_t * pResult)

Converts the result enumerator to a string.

Returns the formatted string version of the result enumerator.

Parameters

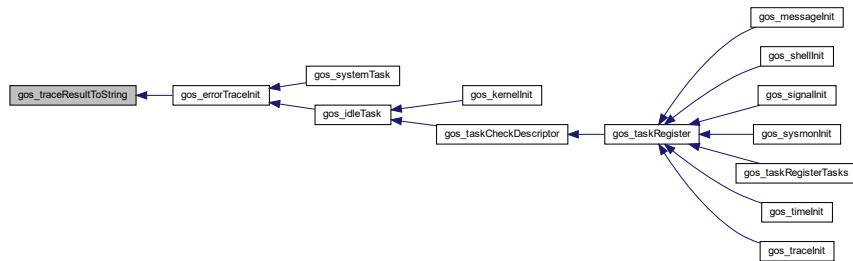
<i>pResult</i>	: Pointer to the result variable.
----------------	-----------------------------------

Returns

Formatted string.

Definition at line 260 of file gos_error.c.

Here is the caller graph for this function:



4.9.4 Variable Documentation

4.9.4.1 GOS_STATIC char_t errorBuffer[ERROR_BUFFER_SIZE]

Buffer for error message formatting.

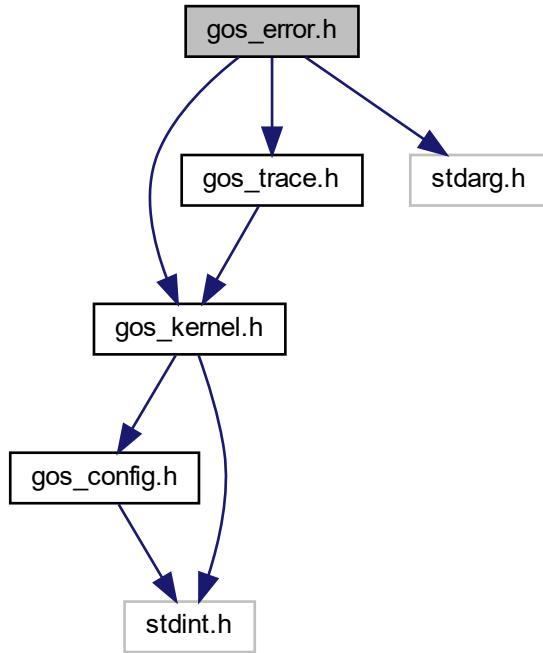
Definition at line 97 of file gos_error.c.

4.10 gos_error.h File Reference

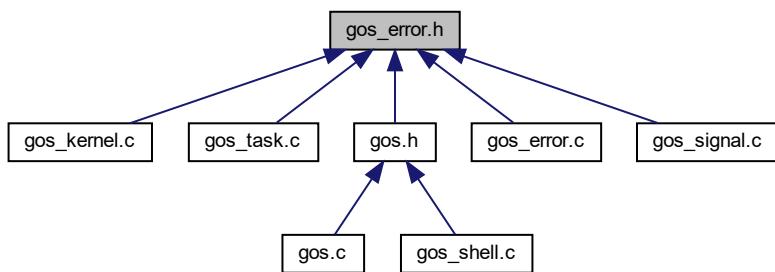
GOS error handler header.

```
#include <gos_kernel.h>
#include <gos_trace.h>
#include <stdarg.h>
```

Include dependency graph for gos_error.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum `gos_errorLevel_t` { `GOS_ERROR_LEVEL_OS_FATAL` = 0b110100, `GOS_ERROR_LEVEL_OS_WARNING` = 0b101110, `GOS_ERROR_LEVEL_USER_FATAL` = 0b011010, `GOS_ERROR_LEVEL_USER_WARNING` = 0b111010 }

Functions

- [`void_t gos_printStartupLogo \(void_t\)`](#)
Prints the startup logo on the trace output.
- [`void_t gos_errorHandler \(gos_errorLevel_t errorLevel, const char_t *function, u32_t line, const char_t *errorMessage, ...\)`](#)
Handles the given error.
- [`gos_result_t gos_errorTraceInit \(const char_t *initDescription, gos_result_t initResult\)`](#)
Traces an initialization message.

4.10.1 Detailed Description

GOS error handler header.

Author

Ahmed Gazar

Date

2022-12-20

Version

2.0

This service is used for tracing error and initialization messages on the log output.

Definition in file [gos_error.h](#).

4.10.2 Enumeration Type Documentation

4.10.2.1 enum gos_errorLevel_t

Error level enumerator.

Enumerator

`GOS_ERROR_LEVEL_OS_FATAL` OS-level fatal error causing the system to stop.

`GOS_ERROR_LEVEL_OS_WARNING` OS-level warning message will be logged, but system will not be stopped.

`GOS_ERROR_LEVEL_USER_FATAL` User-level fatal error causing system stop.

`GOS_ERROR_LEVEL_USER_WARNING` User-level warning.

Definition at line 66 of file gos_error.h.

4.10.3 Function Documentation

4.10.3.1 `void_t gos_errorHandler (gos_errorLevel_t errorLevel, const char_t * function, u32_t line, const char_t * errorMessage, ...)`

Handles the given error.

Prints the formatted error message on the trace output, and based on the error level, it returns or stays in an infinite loop and disables scheduling.

Parameters

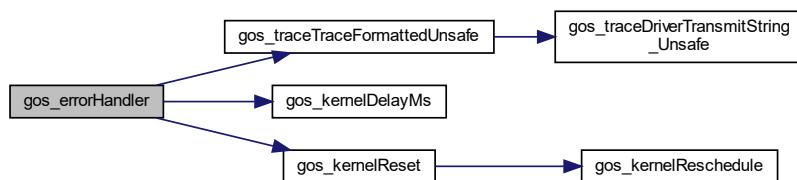
<i>errorLevel</i>	: Level of error (OS/user, warning/fatal).
<i>function</i>	: Function name.
<i>line</i>	: Line number.
<i>errorMessage</i>	: Error message.
...	: Formatter variable arguments.

Returns

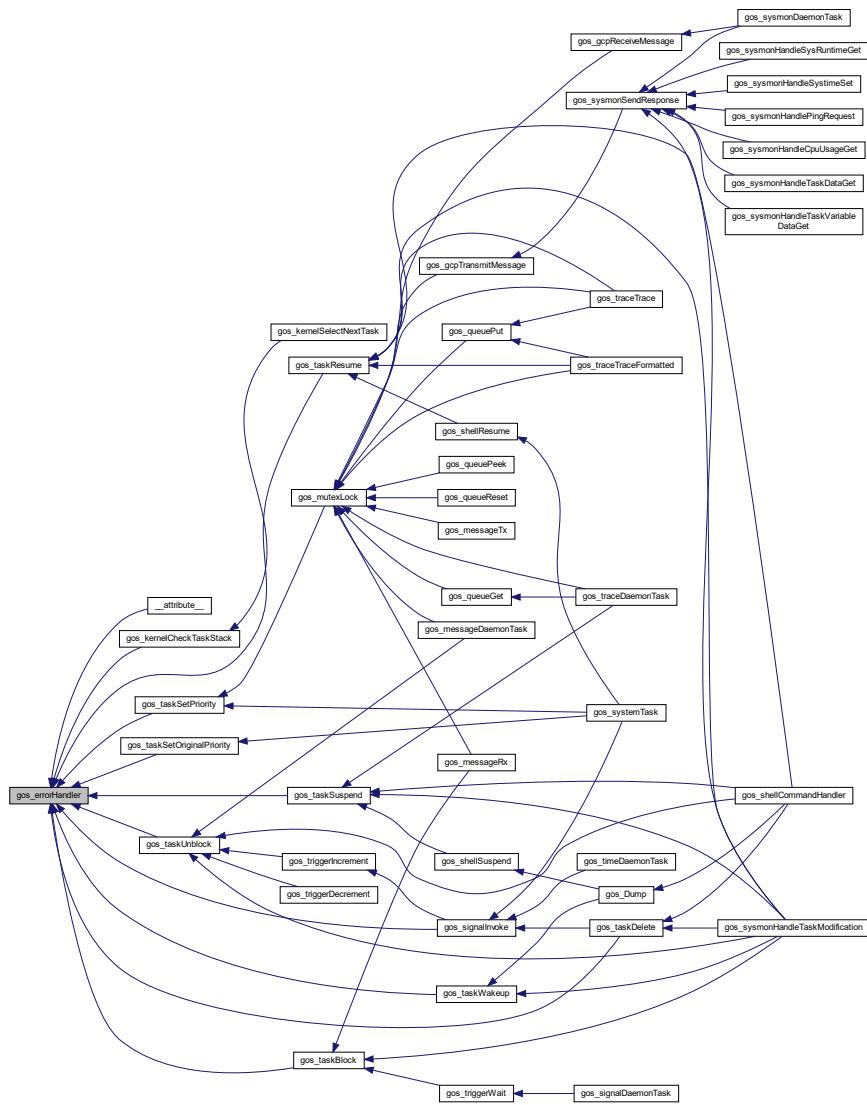
-

Definition at line 128 of file gos_error.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.2 gos_result_t gos_errorTraceInit(const char_t * initDescription, gos_result_t initResult)

Traces an initialization message.

Writes the formatted initialization message on the trace output.

Parameters

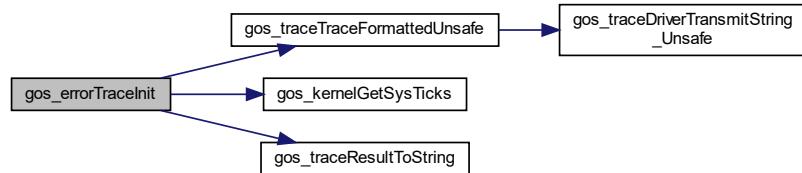
<i>initDescription</i>	: Message to describe the initialization step.
<i>initResult</i>	: Result of initialization.

Returns

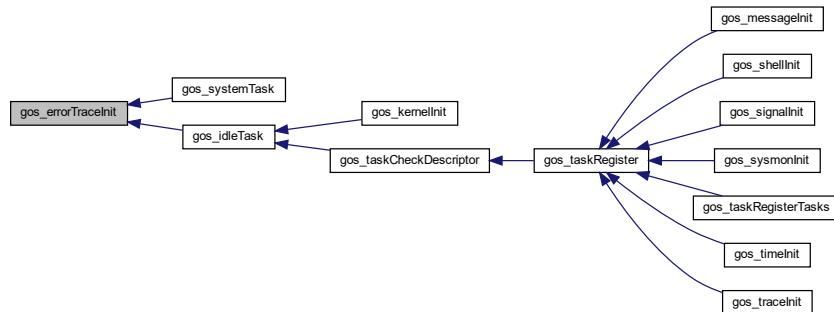
Result of initialization.

Definition at line 236 of file gos_error.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.3 void_t gos_printStartupLogo(void_t)

Prints the startup logo on the trace output.

Prints the GOS logo (similar to the ones used in the file headers) on the log output.

Returns

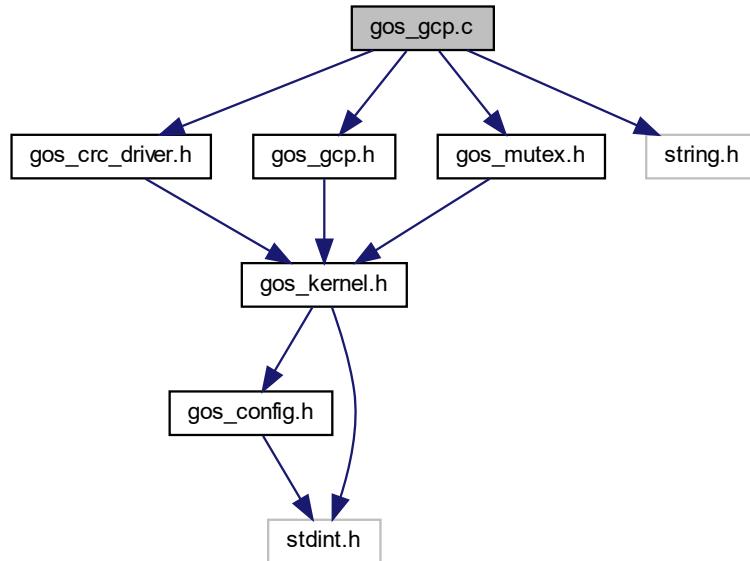
-

4.11 gos_gcp.c File Reference

GOS General Communication Protocol handler service source.

```
#include <gos_crc_driver.h>
#include <gos_gcp.h>
#include <gos_mutex.h>
#include <string.h>
```

Include dependency graph for gos_gcp.c:



Data Structures

- struct [gos_gcpHeaderFrame_t](#)
- struct [gos_gcpChannelFunctions_t](#)

Macros

- #define [GCP_PROTOCOL_VERSION_MAJOR](#) (2)
- #define [GCP_PROTOCOL_VERSION_MINOR](#) (0)

Enumerations

- enum [gos_gcpAck_t](#) {
 [GCP_ACK_REQ](#) = 0, [GCP_ACK_OK](#) = 1, [GCP_ACK_CRC_ERROR](#) = 2, [GCP_ACK_RESEND](#) = 3,
 [GCP_ACK_SIZE_ERROR](#) = 4, [GCP_ACK_PV_ERROR](#) = 5, [GCP_ACK_INVALID](#) = 6
 }

Functions

- [`GOS_STATIC GOS_INLINE gos_result_t gos_gcpTransmitMessageInternal \(gos_gcpChannelNumber_t channel, u16_t messageId, void_t *pMessagePayload, u16_t payloadSize\)`](#)
Internal transmitter function (re-entrant).
- [`GOS_STATIC GOS_INLINE gos_result_t gos_gcpReceiveMessageInternal \(gos_gcpChannelNumber_t channel, u16_t *pMessageId, void_t *pPayloadTarget, u16_t targetSize\)`](#)
Internal receiver function (re-entrant).
- [`GOS_STATIC gos_result_t gos_gcpValidateHeader \(gos_gcpHeaderFrame_t *pHeader, gos_gcpAck_t *pAck\)`](#)
Validates the given GCP header.

- `GOS_STATIC gos_result_t gos_gcpValidateData (gos_gcpHeaderFrame_t *pHeader, void_t *pData, gos_gcpAck_t *pAck)`
Validates the given GCP payload/data.
- `gos_result_t gos_gcpInit (void_t)`
Initializes the GCP service.
- `gos_result_t gos_gcpRegisterPhysicalDriver (gos_gcpChannelNumber_t channelNumber, gos_gcpTransmitFunction_t transmitFunction, gos_gcpReceiveFunction_t receiveFunction)`
Registers the physical-layer transmit and receive driver functions.
- `gos_result_t gos_gcpTransmitMessage (gos_gcpChannelNumber_t channel, u16_t messageID, void_t *pMessagePayload, u16_t payloadSize)`
Transmits the given message via the GCP protocol.
- `gos_result_t gos_gcpReceiveMessage (gos_gcpChannelNumber_t channel, u16_t *pMessageID, void_t *pPayloadTarget, u16_t targetSize)`
Receives the given message via the GCP protocol.

Variables

- `GOS_STATIC gos_gcpChannelFunctions_t channelFunctions [CFG_GCP_CHANNELS_MAX_NUMBER]`
- `GOS_STATIC gos_mutex_t gcpRxMutexes [CFG_GCP_CHANNELS_MAX_NUMBER]`
- `GOS_STATIC gos_mutex_t gcpTxMutexes [CFG_GCP_CHANNELS_MAX_NUMBER]`

4.11.1 Detailed Description

GOS General Communication Protocol handler service source.

Author

Ahmed Gazar

Date

2024-07-18

Version

3.0

For a more detailed description of this service, please refer to [gos_gcp.h](#)

Definition in file [gos_gcp.c](#).

4.11.2 Macro Definition Documentation

4.11.2.1 #define GCP_PROTOCOL_VERSION_MAJOR (2)

GCP protocol version high byte.

Definition at line 71 of file [gos_gcp.c](#).

4.11.2.2 #define GCP_PROTOCOL_VERSION_MINOR (0)

GCP protocol version low byte.

Definition at line 76 of file [gos_gcp.c](#).

4.11.3 Enumeration Type Documentation

4.11.3.1 enum gos_gcpAck_t

GCP acknowledge type.

Enumerator

GCP_ACK_REQ Request.
GCP_ACK_OK OK.
GCP_ACK_CRC_ERROR CRC error.
GCP_ACK_RESEND Re-send request.
GCP_ACK_SIZE_ERROR Size error.
GCP_ACK_PV_ERROR Protocol version error.
GCP_ACK_INVALID Invalid message.

Definition at line 84 of file gos_gcp.c.

4.11.4 Function Documentation

4.11.4.1 gos_result_t gos_gcplinit(void_t)

Initializes the GCP service.

Creates the GCP lock.

Returns

Result of initialization.

Return values

GOS_SUCCESS	: GCP service initialized successfully.
GOS_ERROR	: Lock creation error.

Definition at line 168 of file gos_gcp.c.

Here is the call graph for this function:



4.11.4.2 gos_result_t gos_gcpReceiveMessage(gos_gcpChannelNumber_t channel, u16_t * pMessageId, void_t * pPayloadTarget, u16_t targetSize)

Receives the given message via the GCP protocol.

Calls the internal receiver function.

Parameters

<i>channel</i>	: GCP channel.
<i>pMessageId</i>	: Pointer to a variable to store the message ID.
<i>pPayloadTarget</i>	: Pointer to the payload target buffer.
<i>targetSize</i>	: Size of the target buffer (in bytes).

Returns

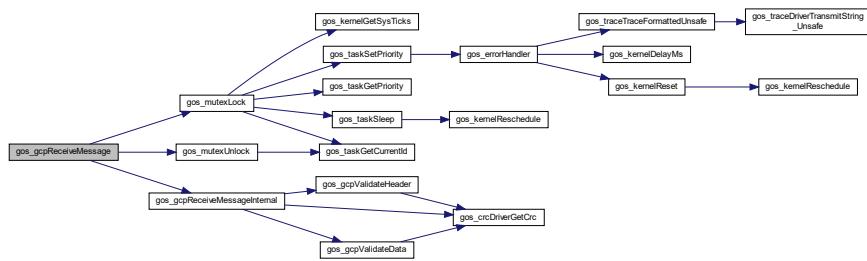
Result of message reception.

Return values

<i>GOS_SUCCESS</i>	: Message received successfully.
<i>GOS_ERROR</i>	: An error occurred during reception or validation.

Definition at line 263 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.4.3 GOS_STATIC GOS_INLINE gos_result_t gos_gcpReceiveMessageInternal (gos_gcpChannelNumber_t channel, u16_t * pMessageId, void_t * pPayloadTarget, u16_t targetSize)

Internal receiver function (re-entrant).

Receives a message over GCP (header, payload, and then transmits a response header).

Parameters

<i>channel</i>	: GCP channel number.
----------------	-----------------------

<i>messageId</i>	: Pointer to a variable to store the message ID.
<i>pMessage</i> ↪ <i>Payload</i>	: Pointer to a buffer to store the payload.
<i>payloadSize</i>	: Size of the payload buffer (in bytes).

Returns

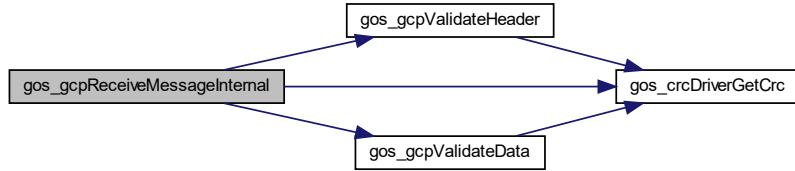
Result of message reception.

Return values

<i>GOS_SUCCESS</i>	: Reception successful.
<i>GOS_ERROR</i>	: One of the function parameters are invalid or request header validation failed or payload validation failed.

Definition at line 381 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.4.4 `gos_result_t gos_gcpRegisterPhysicalDriver (gos_gcpChannelNumber_t channel, gos_gcpTransmitFunction_t transmitFunction, gos_gcpReceiveFunction_t receiveFunction)`

Registers the physical-layer transmit and receive driver functions.

Registers the physical-layer transmit and receive driver functions.

Parameters

<i>channel</i>	: GCP channel.
<i>transmitFunction</i>	: Transmit function to register.
<i>receiveFunction</i>	: Receive function to register.

Returns

Result of physical driver registration.

Return values

<i>GOS_SUCCESS</i>	: Physical driver registration successful.
<i>GOS_ERROR</i>	: Transmit or receive function is NULL.

Definition at line 200 of file gos_gcp.c.

Here is the caller graph for this function:



4.11.4.5 *gos_result_t gos_gcpTransmitMessage(gos_gcpChannelNumber_t channel, u16_t messageId, void_t * pMessagePayload, u16_t payloadSize)*

Transmits the given message via the GCP protocol.

Calls the internal message transmitter function.

Parameters

<i>channel</i>	: GCP channel.
<i>messageId</i>	: Message ID.
<i>pMessagePayload</i>	: Pointer to the message payload.
<i>payloadSize</i>	: Size of the payload (in bytes).

Returns

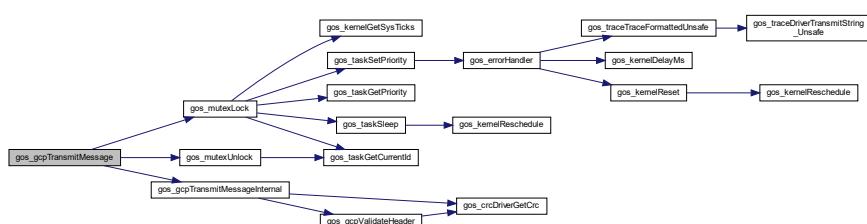
Result of message transmission.

Return values

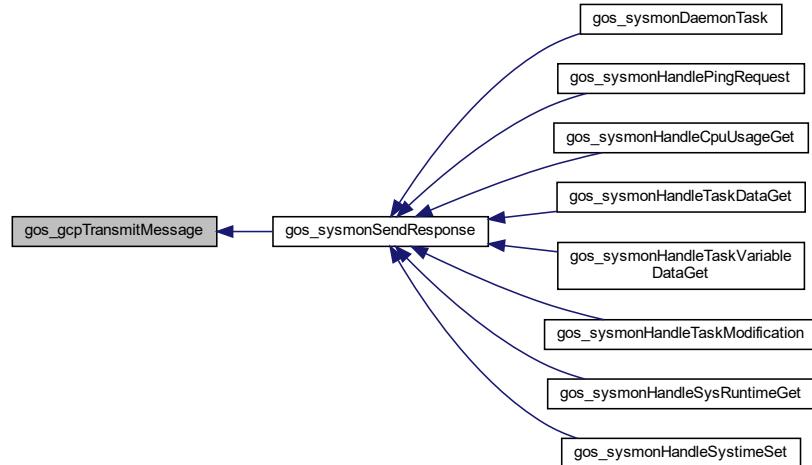
<i>GOS_SUCCESS</i>	: Message transmitted successfully.
<i>GOS_ERROR</i>	: An error occurred during transmission or validation.

Definition at line 231 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.4.6 GOS_STATIC GOS_INLINE gos_result_t gos_gcpTransmitMessageInternal (gos_gcpChannelNumber_t channel, u16_t messageId, void_t * pMessagePayload, u16_t payloadSize)

Internal transmitter function (re-entrant).

Transmits a message over GCP (header request, payload, and then receives a response header).

Parameters

<i>channel</i>	: GCP channel number.
<i>messageId</i>	: ID of the message.
<i>pMessagePayload</i>	: Pointer to the payload buffer.
<i>payloadSize</i>	: Size of the payload (number of bytes).

Returns

Result of message transmission.

Return values

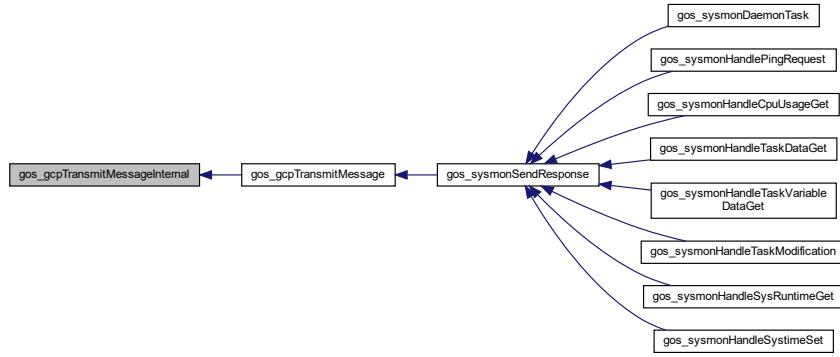
GOS_SUCCESS	: Transmission successful.
GOS_ERROR	: One of the function parameters are invalid or there was a transmission or reception error.

Definition at line 308 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.4.7 GOS_STATIC gos_result_t gos_gcpValidateData(gos_gcpHeaderFrame_t * pHeader, void_t * pData, gos_gcpAck_t * pAck)

Validates the given GCP payload/data.

Checks the CRC of the given payload buffer based on the GCP header passed as a parameter. Returns the acknowledge code in the variable passed as a pointer in case the validation fails.

Parameters

<i>pHeader</i>	: Pointer to the GCP header containing the payload data.
<i>pData</i>	: Pointer to the data buffer to validate.
<i>pAck</i>	: Pointer to an acknowledge variable to store the result in.

Returns

Result of validation.

Return values

<i>GOS_SUCCESS</i>	: Validation successful.
<i>GOS_ERROR</i>	: CRC error or NULL pointer parameter.

Definition at line 516 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.4.8 GOS_STATIC gos_result_t gos_gcpValidateHeader (gos_gcpHeaderFrame_t * pHeader, gos_gcpAck_t * pAck)

Validates the given GCP header.

Checks the CRC and the protocol version of the header. Returns the acknowledge code in the variable passed as a pointer in case the validation fails.

Parameters

<i>pHeader</i>	: Pointer to the GCP header to validate.
<i>pAck</i>	: Pointer to an acknowledge variable to store the result in.

Returns

Result of validation.

Return values

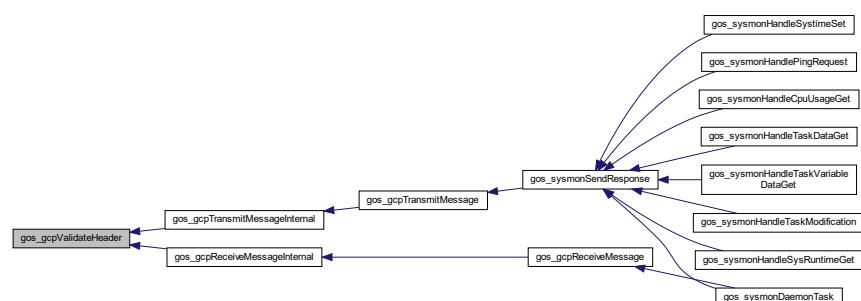
<i>GOS_SUCCESS</i>	: Validation successful.
<i>GOS_ERROR</i>	: CRC or PV error or NULL pointer parameter.

Definition at line 461 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.5 Variable Documentation

4.11.5.1 **GOS_STATIC gos_gcpChannelFunctions_t channelFunctions[CFG_GCP_CHANNELS_MAX_NUMBERS]**

Channel functions.

Definition at line 125 of file gos_gcp.c.

4.11.5.2 **GOS_STATIC gos_mutex_t gcpRxMutexes[CFG_GCP_CHANNELS_MAX_NUMBER]**

GCP RX mutex array.

Definition at line 130 of file gos_gcp.c.

4.11.5.3 **GOS_STATIC gos_mutex_t gcpTxMutexes[CFG_GCP_CHANNELS_MAX_NUMBER]**

GCP TX mutex array.

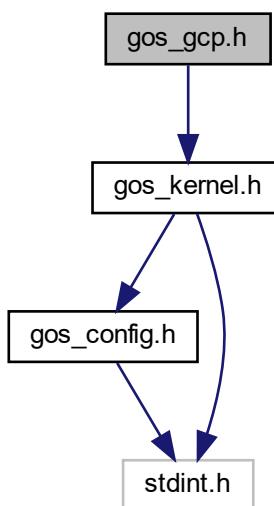
Definition at line 135 of file gos_gcp.c.

4.12 gos_gcp.h File Reference

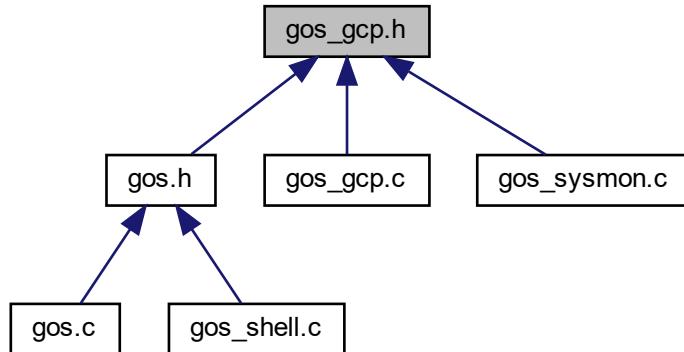
GOS General Communication Protocol header.

```
#include <gos_kernel.h>
```

Include dependency graph for gos_gcp.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- `typedef u8_t gos_gcpChannelNumber_t`
GCP channel number.
- `typedef gos_result_t(* gos_gcpTransmitFunction_t)(u8_t *, u16_t)`
- `typedef gos_result_t(* gos_gcpReceiveFunction_t)(u8_t *, u16_t)`

Functions

- `gos_result_t gos_gcpInit (void_t)`
Initializes the GCP service.
- `gos_result_t gos_gcpRegisterPhysicalDriver (gos_gcpChannelNumber_t channel, gos_gcpTransmitFunction_t transmitFunction, gos_gcpReceiveFunction_t receiveFunction)`
Registers the physical-layer transmit and receive driver functions.
- `gos_result_t gos_gcpTransmitMessage (gos_gcpChannelNumber_t channel, u16_t messageId, void_t *pMessagePayload, u16_t payloadSize)`
Transmits the given message via the GCP protocol.
- `gos_result_t gos_gcpReceiveMessage (gos_gcpChannelNumber_t channel, u16_t *pMessageId, void_t *pPayloadTarget, u16_t targetSize)`
Receives the given message via the GCP protocol.

4.12.1 Detailed Description

GOS General Communication Protocol header.

Author

Ahmed Gazar

Date

2024-07-18

Version

3.0

This service implements the GCP frame and message layers.

Definition in file [gos_gcp.h](#).

4.12.2 Typedef Documentation

4.12.2.1 `typedef gos_result_t(* gos_gcpReceiveFunction_t)(u8_t *, u16_t)`

GCP physical layer receive function type.

Definition at line 75 of file gos_gcp.h.

4.12.2.2 `typedef gos_result_t(* gos_gcpTransmitFunction_t)(u8_t *, u16_t)`

GCP physical layer transmit function type.

Definition at line 70 of file gos_gcp.h.

4.12.3 Function Documentation

4.12.3.1 `gos_result_t gos_gcplinit(void_t)`

Initializes the GCP service.

Creates the GCP lock.

Returns

Result of initialization.

Return values

<code>GOS_SUCCESS</code>	: GCP service initialized successfully.
<code>GOS_ERROR</code>	: Lock creation error.

Definition at line 168 of file gos_gcp.c.

Here is the call graph for this function:



4.12.3.2 `gos_result_t gos_gcpReceiveMessage(gos_gcpChannelNumber_t channel, u16_t * pMessageId, void_t * pPayloadTarget, u16_t targetSize)`

Receives the given message via the GCP protocol.

Calls the internal receiver function.

Parameters

<i>channel</i>	: GCP channel.
<i>pMessageId</i>	: Pointer to a variable to store the message ID.
<i>pPayloadTarget</i>	: Pointer to the payload target buffer.
<i>targetSize</i>	: Size of the target buffer (in bytes).

Returns

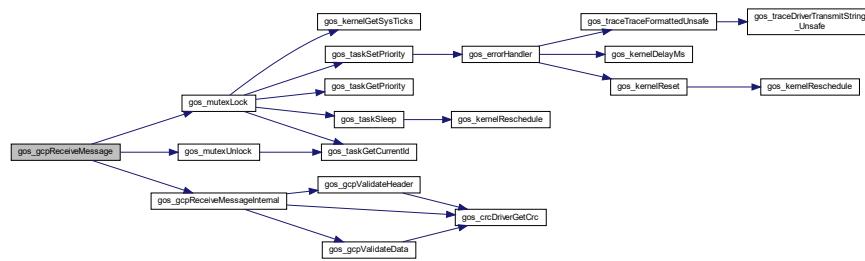
Result of message reception.

Return values

<i>GOS_SUCCESS</i>	: Message received successfully.
<i>GOS_ERROR</i>	: An error occurred during reception or validation.

Definition at line 263 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.3.3 `gos_result_t gos_gcpRegisterPhysicalDriver (gos_gcpChannelNumber_t channel, gos_gcpTransmitFunction_t transmitFunction, gos_gcpReceiveFunction_t receiveFunction)`

Registers the physical-layer transmit and receive driver functions.

Registers the physical-layer transmit and receive driver functions.

Parameters

<i>channel</i>	: GCP channel.
----------------	----------------

<i>transmitFunction</i>	: Transmit function to register.
<i>receiveFunction</i>	: Receive function to register.

Returns

Result of physical driver registration.

Return values

<i>GOS_SUCCESS</i>	: Physical driver registration successful.
<i>GOS_ERROR</i>	: Transmit or receive function is NULL.

Definition at line 200 of file gos_gcp.c.

Here is the caller graph for this function:



4.12.3.4 `gos_result_t gos_gcpTransmitMessage(gos_gcpChannelNumber_t channel, u16_t messageId, void_t * pMessagePayload, u16_t payloadSize)`

Transmits the given message via the GCP protocol.

Calls the internal message transmitter function.

Parameters

<i>channel</i>	: GCP channel.
<i>messageId</i>	: Message ID.
<i>pMessagePayload</i>	: Pointer to the message payload.
<i>payloadSize</i>	: Size of the payload (in bytes).

Returns

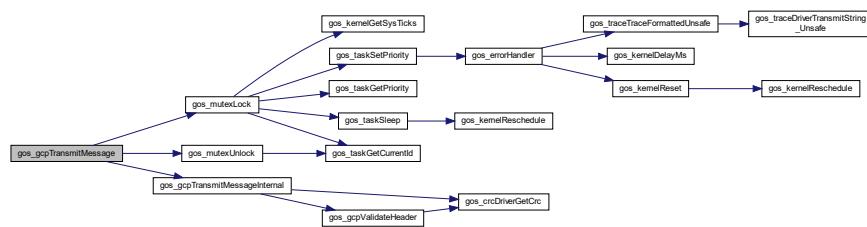
Result of message transmission.

Return values

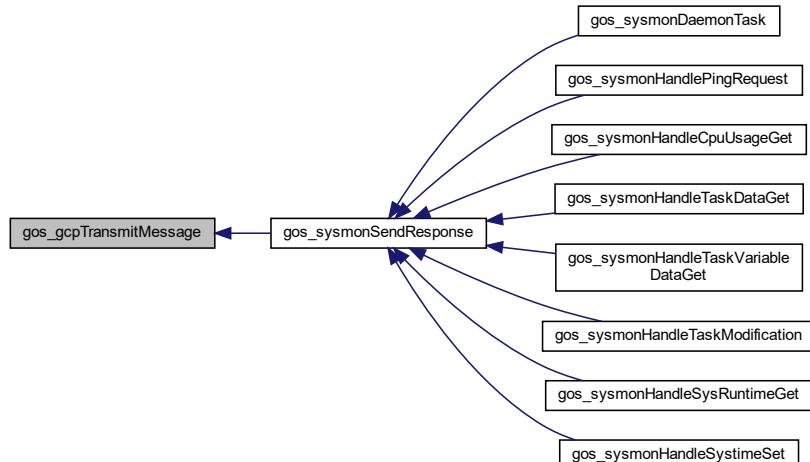
<i>GOS_SUCCESS</i>	: Message transmitted successfully.
<i>GOS_ERROR</i>	: An error occurred during transmission or validation.

Definition at line 231 of file gos_gcp.c.

Here is the call graph for this function:



Here is the caller graph for this function:



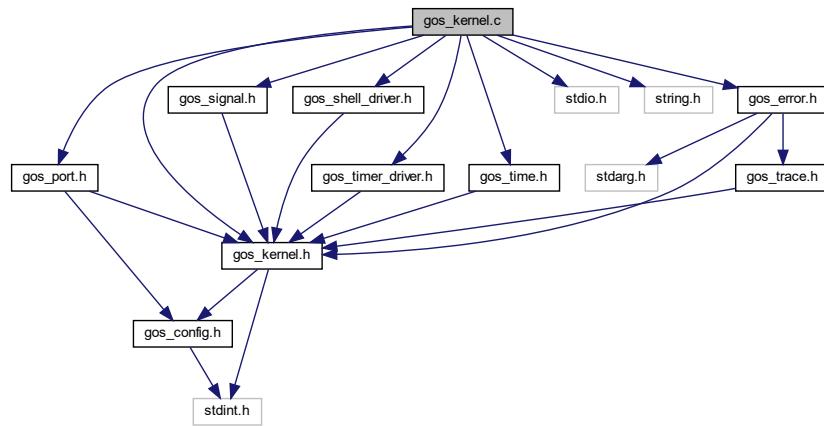
4.13 gos_kernel.c File Reference

GOS kernel source.

```

#include <gos_error.h>
#include <gos_kernel.h>
#include <gos_port.h>
#include <gos_signal.h>
#include <gos_shell_driver.h>
#include <gos_timer_driver.h>
#include <stdio.h>
#include <string.h>
#include <gos_time.h>
  
```

Include dependency graph for gos_kernel.c:



Macros

- ```
• #define SHCSR *(volatile u32_t*) 0xE000ED24u)
• #define ICSR *(volatile u32_t*) 0xE000ED04u)
• #define TASK_DUMP_SEPARATOR "+-----+-----+-----+
+\\r\\n"
• #define MAX_CPU_DUMP_SEPARATOR "+-----+-----+-----+\\r\\n"
• #define STACK_STATS_SEPARATOR "+-----+-----+-----+-----+\\r\\n"
• #define CONFIG_DUMP_SEPARATOR "+-----+-----+\\r\\n"
• #define BINARY_PATTERN "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s"TRACE_FORMAT_RESET
• #define TO_BINARY(word)
```

## Functions

- **GOS\_STATIC void\_t gos\_kernelCheckTaskStack (void\_t)**  
*Checks the stack use of the current task.*
  - **GOS\_STATIC u32\_t gos\_kernelGetCurrentPsp (void\_t)**  
*Returns the current PSP.*
  - **GOS\_STATIC void\_t gos\_kernelSaveCurrentPsp (u32\_t psp)**  
*Saves the current PSP.*
  - **GOS\_STATIC void\_t gos\_kernelSelectNextTask (void\_t)**  
*Selects the next task for execution.*
  - **GOS\_STATIC char\_t \* gos\_kernelGetTaskStateString (gos\_taskState\_t taskState)**  
*Translates the task state to a string.*
  - **GOS\_STATIC void\_t gos\_kernelProcessorReset (void\_t)**  
*Kernel processor reset.*
  - **GOS\_EXTERN void\_t gos\_idleTask (void\_t)**  
*Kernel idle task.*
  - **gos\_result\_t gos\_kernellInit (void\_t)**  
*This function initializes the kernel.*
  - **gos\_result\_t gos\_kernelStart (void\_t)**  
*Starts the kernel.*

- `gos_result_t gos_kernelRegisterSwapHook (gos_taskSwapHook_t swapHookFunction)`  
*Registers a task swap hook function.*
- `gos_result_t gos_kernelRegisterSysTickHook (gos_sysTickHook_t sysTickHookFunction)`  
*Registers a system tick hook function.*
- `gos_result_t gos_kernelRegisterPrivilegedHook (gos_privilegedHook_t privilegedHookFunction)`  
*Registers a privileged hook function.*
- `gos_result_t gos_kernelSubscribeDumpReadySignal (gos_signalHandler_t dumpReadySignalHandler)`
- `void_t gos_ported_sysTickInterrupt (void_t)`
- `u32_t gos_kernelGetSysTicks (void_t)`  
*Returns the system ticks.*
- `u16_t gos_kernelGetCpuUsage (void_t)`  
*Returns the CPU usage.*
- `void_t gos_kernelReset (void_t)`  
*Resets the microcontroller.*
- `void_t gos_kernelPrivilegedModeSetRequired (void_t)`  
*Requests a privileged mode setting.*
- `GOS_INLINE void_t gos_kernelDelayUs (u16_t microseconds)`  
*Blocking delay in microsecond range.*
- `GOS_INLINE void_t gos_kernelDelayMs (u16_t milliseconds)`  
*Blocking delay in millisecond range.*
- `GOS_INLINE void_t gos_kernelCalculateTaskCpuUsages (bool_t isResetRequired)`  
*Calculates the CPU usage for the tasks.*
- `void_t gos_kernelDump (void_t)`  
*Kernel dump.*
- `gos_result_t gos_kernelSetMaxCpuLoad (u16_t maxCpuLoad)`  
*Sets the maximum (global) CPU load. Sets the value of the global CPU load above which the scheduler will start running the idle task until the CPU load falls below the limit value.*
- `gos_result_t gos_kernelGetMaxCpuLoad (u16_t *maxCpuLoad)`  
*Gets the maximum (global) CPU load. Returns the value of the maximum CPU load (limit).*
- `bool_t gos_kernellsCallerIsr (void_t)`  
*Returns if the current task is ISR. Returns if the inlsr flag is greater than zero.*
- `void_t gos_ported_svcHandler (void_t)`
- `void_t gos_ported_svcHandlerMain (u32_t *sp)`
- `void_t gos_ported_pendSVHandler (void_t)`
- `GOS_INLINE void_t gos_kernelReschedule (gos_kernel_privilege_t privilege)`  
*Reschedules the kernel.*
- `void_t NMI_Handler (void_t)`
- `void_t HardFault_Handler (void_t)`
- `void_t MemManage_Handler (void_t)`
- `void_t BusFault_Handler (void_t)`
- `void_t UsageFault_Handler (void_t)`

## Variables

- `gos_signallId_t kernelDumpSignal`
- `gos_signallId_t kernelDumpReadySignal`
- `u8_t schedDisableCntr = 0u`
- `u8_t inlsr = 0u`
- `u8_t atomicCntr = 0u`
- `u32_t primask = 0u`
- `u32_t currentTaskIndex = 0u`
- `u16_t cpuUseLimit = 10000`

- `GOS_STATIC u32_t sysTicks = 0u`
- `GOS_STATIC u16_t sysTimerValue = 0u`
- `GOS_STATIC gos_runtime_t monitoringTime = {0}`
- `GOS_STATIC gos_taskSwapHook_t kernelSwapHookFunction = NULL`
- `GOS_STATIC gos_sysTickHook_t kernelSysTickHookFunction = NULL`
- `GOS_STATIC gos_privilegedHook_t kernelPrivilegedHookFunction = NULL`
- `GOS_STATIC bool_t resetRequired = GOS_FALSE`
- `GOS_STATIC bool_t privilegedModeSetRequired = GOS_FALSE`
- `GOS_STATIC u32_t previousTick = 0u`
- `GOS_STATIC bool_t isKernelRunning = GOS_FALSE`
- `GOS_EXTERN gos_taskDescriptor_t taskDescriptors [CFG_TASK_MAX_NUMBER]`

#### 4.13.1 Detailed Description

GOS kernel source.

Author

Ahmed Gazar

Date

2024-04-24

Version

1.20

For a more detailed description of this module, please refer to [gos\\_kernel.h](#)

Definition in file [gos\\_kernel.c](#).

#### 4.13.2 Macro Definition Documentation

##### 4.13.2.1 #define BINARY\_PATTERN "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s"TRACE\_FORMAT\_RESET

Pattern for binary format printing.

Definition at line 145 of file [gos\\_kernel.c](#).

##### 4.13.2.2 #define CONFIG\_DUMP\_SEPARATOR "+-----+\r\n"

Config dump separator line.

Definition at line 140 of file [gos\\_kernel.c](#).

##### 4.13.2.3 #define ICSR \*(volatile u32\_t\*)0xE000ED04u )

Interrupt Control and State Register.

Definition at line 120 of file [gos\\_kernel.c](#).

##### 4.13.2.4 #define MAX\_CPU\_DUMP\_SEPARATOR "+-----+\r\n"

Max CPU loads dump separator line.

Definition at line 130 of file [gos\\_kernel.c](#).

#### 4.13.2.5 #define SHCSR \*( volatile u32\_t\* ) 0xE000ED24u )

System Handler control and state register.

Definition at line 115 of file gos\_kernel.c.

#### 4.13.2.6 #define STACK\_STATS\_SEPARATOR "-----+\r\n"

Stack statistics separator line.

Definition at line 135 of file gos\_kernel.c.

#### 4.13.2.7 #define TASK\_DUMP\_SEPARATOR "-----+\r\n"

Task dump separator line.

Definition at line 125 of file gos\_kernel.c.

#### 4.13.2.8 #define TO\_BINARY( word )

**Value:**

```
(word & 0x8000 ? TRACE_FG_GREEN_START"1" :
 TRACE_FG_RED_START"0"), \
(word & 0x4000 ? TRACE_FG_GREEN_START"1" :
 TRACE_FG_RED_START"0"), \
(word & 0x2000 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x1000 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0800 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0400 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0200 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0100 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0080 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0040 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0020 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0010 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0008 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0004 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0002 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0"), \
(word & 0x0001 ? TRACE_FG_GREEN_START"1" : TRACE_FG_RED_START"0")
```

u16\_t to binary converter macro.

Definition at line 150 of file gos\_kernel.c.

### 4.13.3 Function Documentation

#### 4.13.3.1 void\_t gos\_idleTask( void\_t )

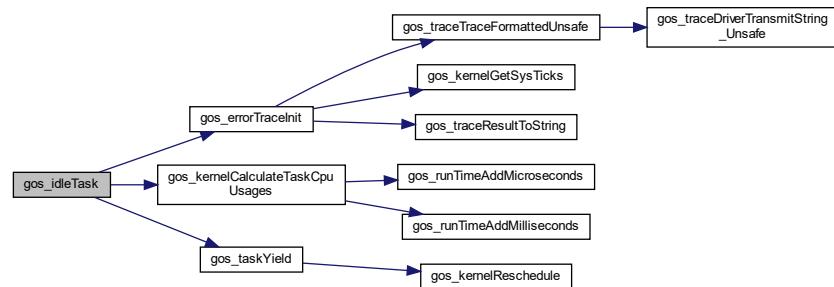
Kernel idle task.

This task is executed when there is no other ready task in the system. When executed, this function refreshes the CPU-usage statistics of tasks.

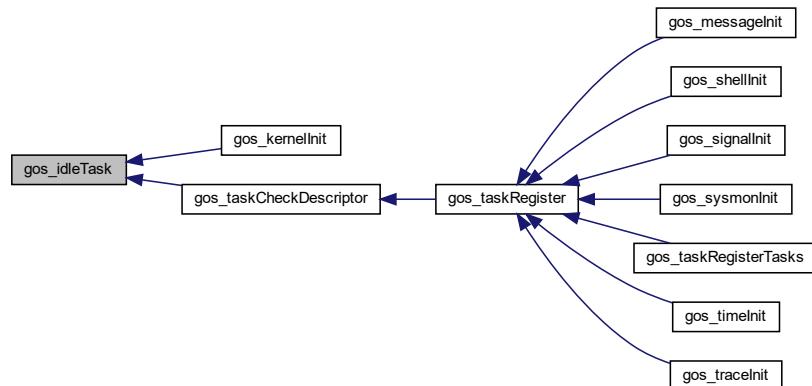
**Returns**

Definition at line 1314 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.3.2 GOS\_INLINE void\_t gos\_kernelCalculateTaskCpuUsages ( bool\_t *isResetRequired* )

Calculates the CPU usage for the tasks.

Based on the total system time range, it refreshes the CPU-usage statistics of tasks.

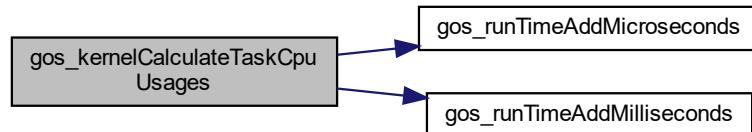
**Parameters**

|                        |                                                             |
|------------------------|-------------------------------------------------------------|
| <i>isResetRequired</i> | : Flag to indicate whether the measurement should be reset. |
|------------------------|-------------------------------------------------------------|

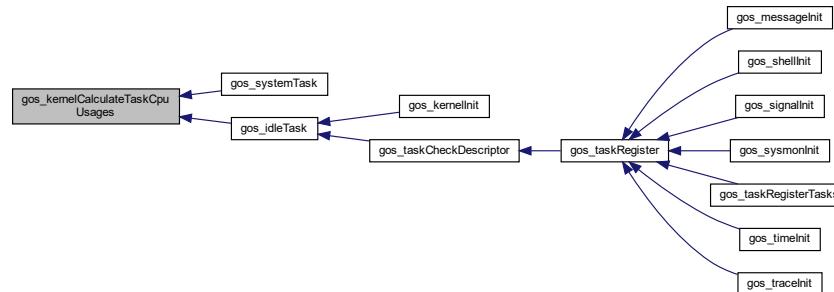
**Returns**

Definition at line 618 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.3.3 GOS\_STATIC void\_t gos\_kernelCheckTaskStack ( void\_t )

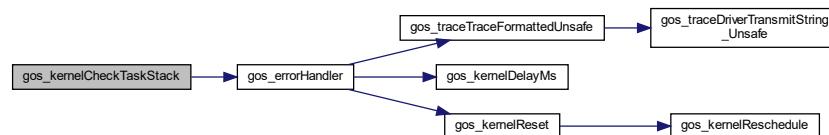
Checks the stack use of the current task.

Gets the current stack pointer value and checks whether it is lower than the allowed threshold value defined for the current stack. After the overflow check, it calculates the stack usage and updates the maximum stack usage for the current task. In case of stack overflow, it goes to system error.

##### Returns

Definition at line 961 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.3.4 GOS\_INLINE void\_t gos\_kernelDelayMs ( u16\_t milliseconds )

Blocking delay in millisecond range.

This function waits in a while loop until the given number of system ticks have elapsed.

##### Parameters

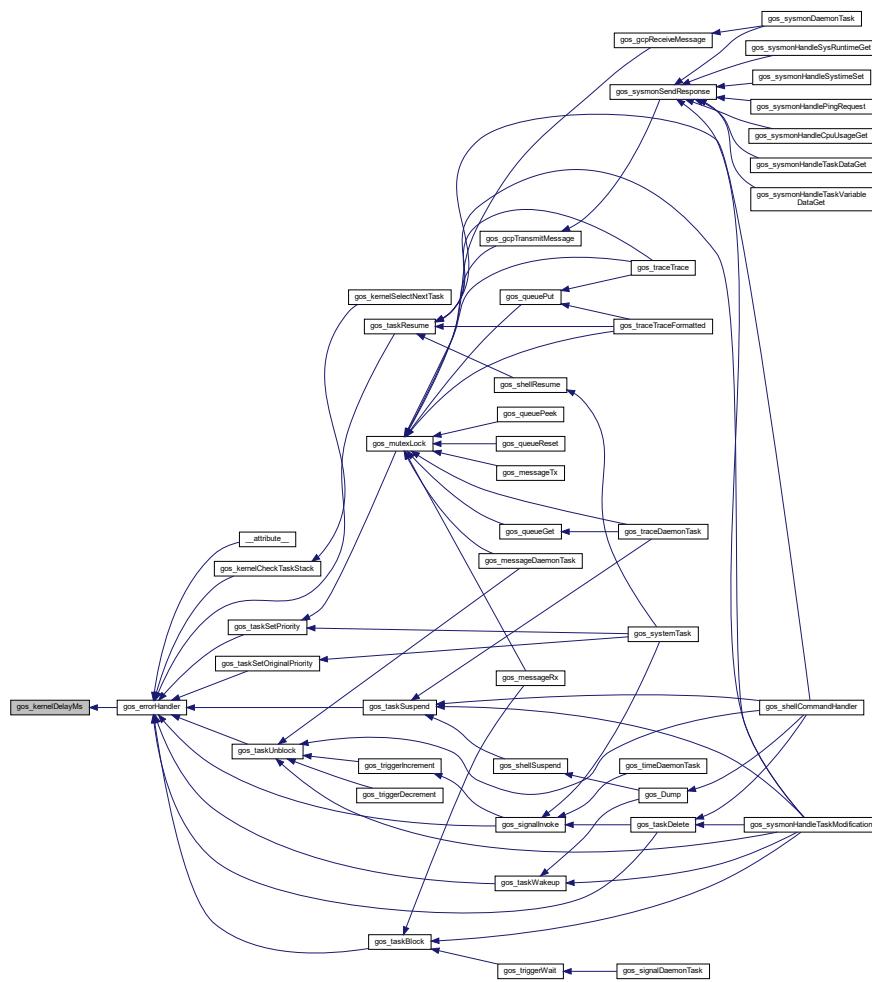
|                     |                         |
|---------------------|-------------------------|
| <i>milliseconds</i> | : Milliseconds to wait. |
|---------------------|-------------------------|

##### Returns

-

Definition at line 602 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.13.3.5 GOS\_INLINE void\_t gos\_kernelDelayUs ( u16\_t microseconds )

Blocking delay in microsecond range.

This function waits in a while loop until the given number of milliseconds have elapsed based on the system timer.

##### Parameters

|                           |                         |
|---------------------------|-------------------------|
| <code>microseconds</code> | : Microseconds to wait. |
|---------------------------|-------------------------|

**Returns**

-  
Definition at line 580 of file gos\_kernel.c.

Here is the call graph for this function:

**4.13.3.6 void\_t gos\_kernelDump( void\_t )**

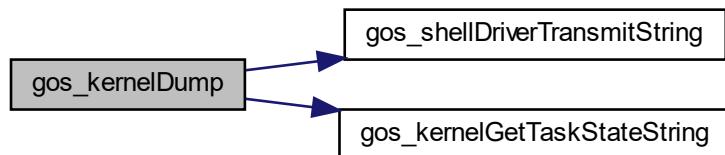
Kernel dump.

This function prints the kernel configuration and task data to the trace output.

**Returns**

-  
Definition at line 720 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.3.7 `u16_t gos_kernelGetCpuUsage( void_t )`

Returns the CPU usage.

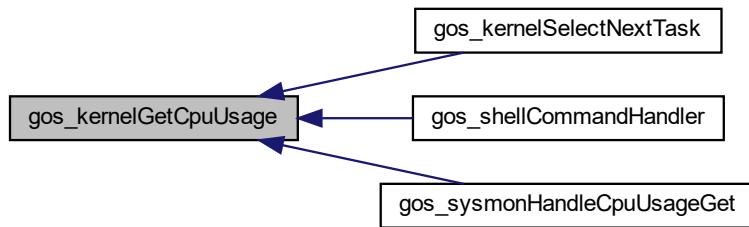
Refreshes the CPU statistics and returns the current CPU usage in [CPU%] \* 100 format that results in a range of 0...10000 where the last two digits represent two decimals. The usage is 100% - idle task%.

Returns

Overall CPU usage.

Definition at line 544 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.13.3.8 GOS\_UNUSED GOS\_STATIC u32\_t gos\_kernelGetCurrentPsp( void\_t )

Returns the current PSP.

Returns the current PSP.

Returns

Current PSP value.

Definition at line 1007 of file gos\_kernel.c.

#### 4.13.3.9 `gos_result_t gos_kernelGetMaxCpuLoad( u16_t * maxCpuLoad )`

Gets the maximum (global) CPU load. Returns the value of the maximum CPU load (limit).

Parameters

|                         |                                                  |
|-------------------------|--------------------------------------------------|
| <code>maxCpuLoad</code> | : Target variable to store the maximum CPU load. |
|-------------------------|--------------------------------------------------|

Returns

Result of maximum CPU load getting.

**Returns**

|                    |                                        |
|--------------------|----------------------------------------|
| <i>GOS_SUCCESS</i> | : Maximum CPU load getting successful. |
| <i>GOS_ERROR</i>   | : Target variable is NULL.             |

Definition at line 858 of file gos\_kernel.c.

**4.13.3.10 u32\_t gos\_kernelGetSysTicks( void\_t )**

Returns the system ticks.

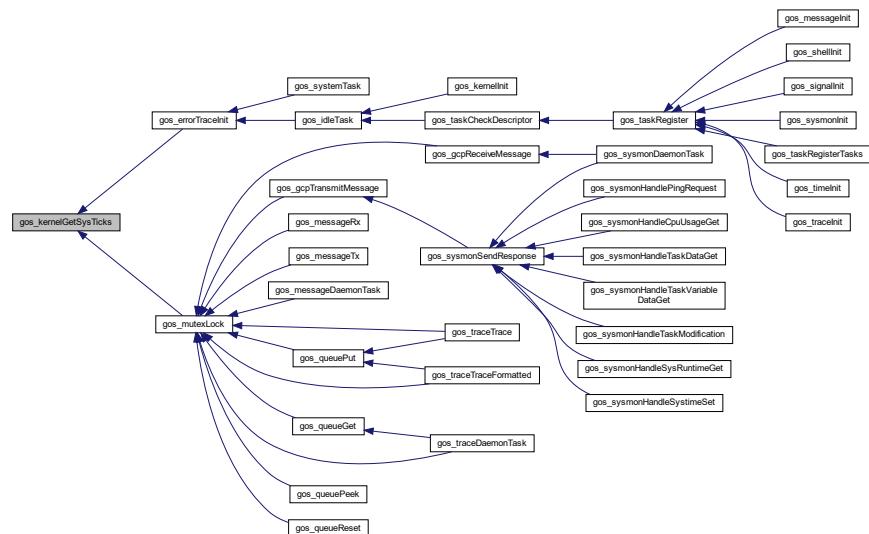
Returns the internal system tick counter value.

**Returns**

System ticks.

Definition at line 533 of file gos\_kernel.c.

Here is the caller graph for this function:

**4.13.3.11 GOS\_STATIC char\_t \* gos\_kernelGetTaskStateString( gos\_taskState\_t taskState )**

Translates the task state to a string.

Based on the task state it returns a string with a printable form of the task state.

**Parameters**

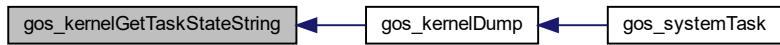
|                  |                                             |
|------------------|---------------------------------------------|
| <i>taskState</i> | : The task state variable to be translated. |
|------------------|---------------------------------------------|

**Returns**

String with the task state.

Definition at line 1167 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.13.3.12 gos\_result\_t gos\_kernellInit( void\_t )

This function initializes the kernel.

Initializes the internal task array, fills out the PSP for the idle task, and registers the kernel dump task and suspends it.

##### Returns

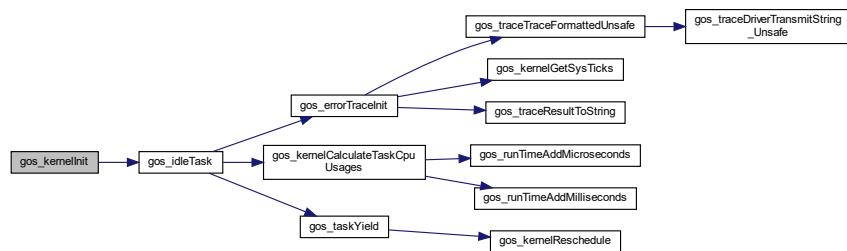
Result of initialization.

##### Return values

|                    |                                        |
|--------------------|----------------------------------------|
| <i>GOS_SUCCESS</i> | : Kernel initialization successful.    |
| <i>GOS_ERROR</i>   | : Kernel task suspension unsuccessful. |

Definition at line 287 of file gos\_kernel.c.

Here is the call graph for this function:



#### 4.13.3.13 bool\_t gos\_kernelsCallerIsr( void\_t )

Returns if the current task is ISR. Returns if the inlsr flag is greater than zero.

##### Returns

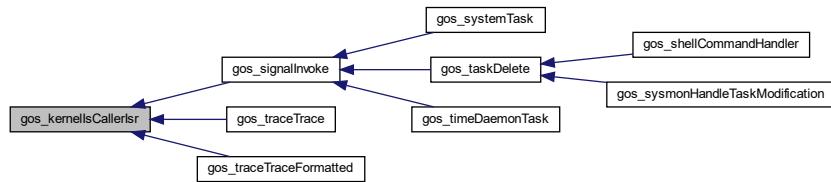
Whether the caller is ISR (Interrupt Service Routine).

##### Return values

|                        |                      |
|------------------------|----------------------|
| <code>GOS_TRUE</code>  | : Caller is ISR.     |
| <code>GOS_FALSE</code> | : Caller is not ISR. |

Definition at line 884 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.13.3.14 void\_t gos\_kernelPrivilegedModeSetRequired ( void\_t )

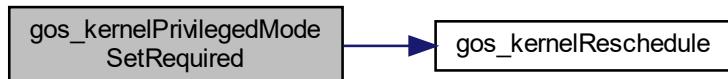
Requests a privileged mode setting.

Initiates an SVC call while setting the corresponding flag to set the execution mode to privileged (permanently). After this, it calls the corresponding hook function, so the caller can access special registers immediately.

Returns

Definition at line 567 of file gos\_kernel.c.

Here is the call graph for this function:



#### 4.13.3.15 GOS\_STATIC void\_t gos\_kernelProcessorReset ( void\_t )

Kernel processor reset.

Resets the processor.

Returns

Definition at line 1207 of file gos\_kernel.c.

#### 4.13.3.16 gos\_result\_t gos\_kernelRegisterPrivilegedHook ( *gos\_privilegedHook\_t privilegedHookFunction* )

Registers a privileged hook function.

Checks if a hook function has already been registered, and, if not, it registers the new hook function.

**Parameters**

|                               |                             |
|-------------------------------|-----------------------------|
| <i>privilegedHookFunction</i> | : Privileged hook function. |
|-------------------------------|-----------------------------|

**Returns**

Result of registration.

**Return values**

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Registration successful.                                                 |
| <i>GOS_ERROR</i>   | : Registration failed (hook function already exists or parameter is NULL). |

Definition at line 438 of file gos\_kernel.c.

**4.13.3.17 gos\_result\_t gos\_kernelRegisterSwapHook( gos\_taskSwapHook\_t swapHookFunction )**

Registers a task swap hook function.

Checks whether the param is NULL pointer and a hook function is already registered, and both conditions are false, it registers the hook function.

**Parameters**

|                         |                       |
|-------------------------|-----------------------|
| <i>swapHookFunction</i> | : Swap hook function. |
|-------------------------|-----------------------|

**Returns**

Result of registration.

**Return values**

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Registration successful.                                                 |
| <i>GOS_ERROR</i>   | : Registration failed (hook function already exists or parameter is NULL). |

Definition at line 386 of file gos\_kernel.c.

**4.13.3.18 gos\_result\_t gos\_kernelRegisterSysTickHook( gos\_sysTickHook\_t sysTickHookFunction )**

Registers a system tick hook function.

Checks whether the param is NULL pointer and a hook function is already registered, and both conditions are false, it registers the hook function.

**Parameters**

|                            |                              |
|----------------------------|------------------------------|
| <i>sysTickHookFunction</i> | : System tick hook function. |
|----------------------------|------------------------------|

**Returns**

Result of registration.

**Return values**

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <b>GOS_SUCCESS</b> | : Registration successful.                                                 |
| <b>GOS_ERROR</b>   | : Registration failed (hook function already exists or parameter is NULL). |

Definition at line 412 of file gos\_kernel.c.

4.13.3.19 **GOS\_INLINE void\_t gos\_kernelReschedule( gos\_kernel\_privilege\_t privilege )**

Reschedules the kernel.

Based on the privilege, it invokes a kernel reschedule event.

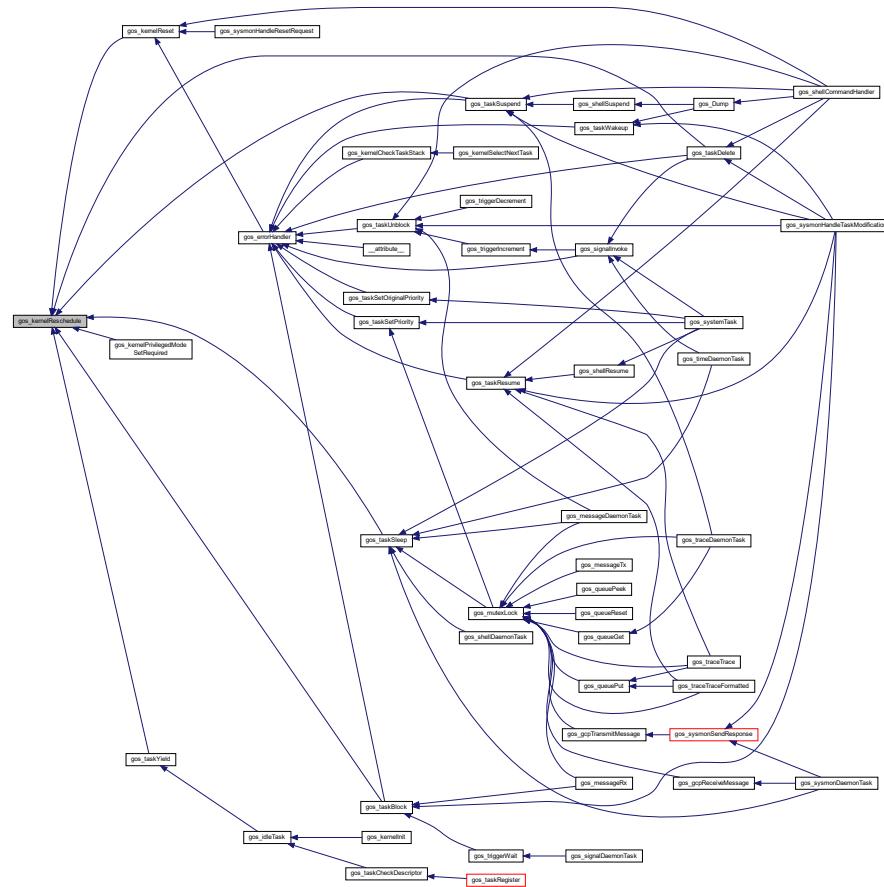
## Parameters

**privilege** : Privilege level.

## Returns

Definition at line 943 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.13.3.20 void t gos\_kernelReset( void t )

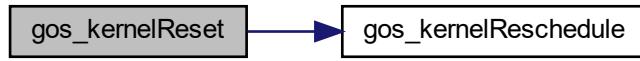
Resets the microcontroller.

Sets the reset required flag and gets privileged access to reset the controller.

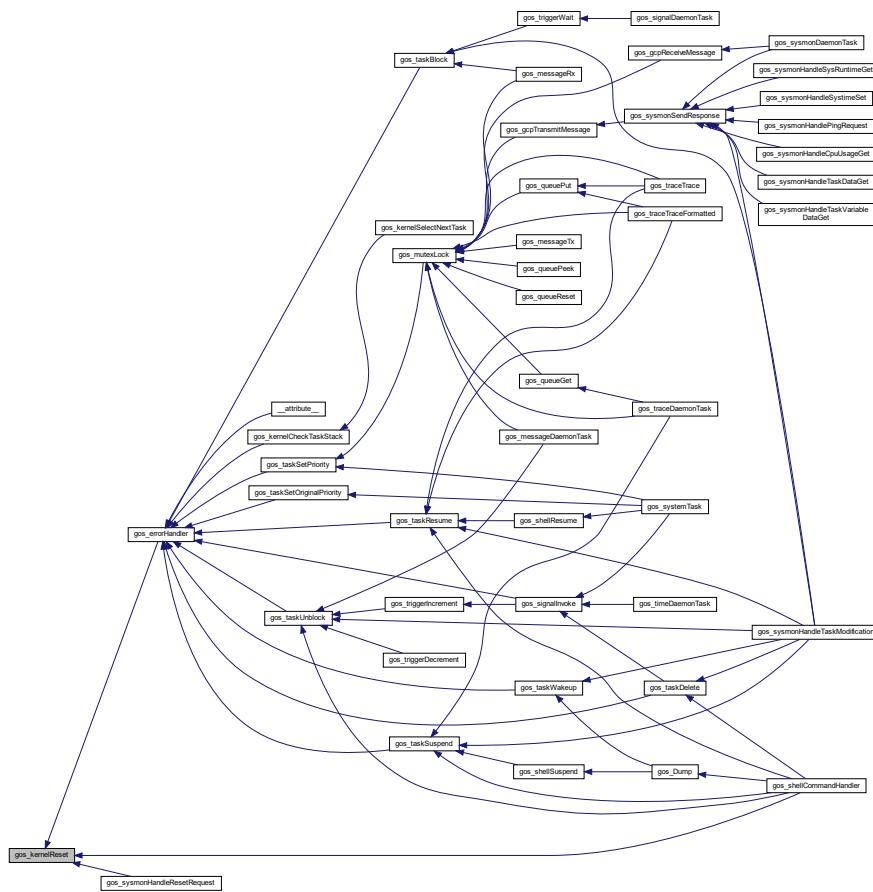
## Returns

Definition at line 555 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.3.21 **GOS\_UNUSED GOS\_STATIC void\_t gos\_kernelSaveCurrentPsp( u32\_t psp )**

Saves the current PSP.

Saves the current PSP.

**Parameters**

|            |                      |
|------------|----------------------|
| <i>psp</i> | : Current PSP value. |
|------------|----------------------|

**Returns**

-  
Definition at line 1023 of file gos\_kernel.c.

**4.13.3.22 GOS\_UNUSED GOS\_STATIC void\_t gos\_kernelSelectNextTask ( void\_t )**

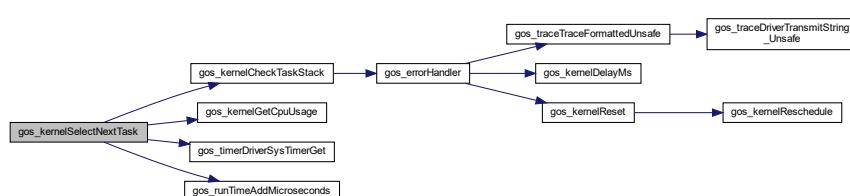
Selects the next task for execution.

Loops through the internal task descriptor array and first checks the sleeping tasks and wakes up the ones that passed their sleeping time. Then based on the priority of the ready tasks, it selects the one with the highest priority (lowest number in priority). If there is a swap-hook function registered, it calls it, and then it refreshes the task run-time statistics.

**Returns**

-  
Definition at line 1042 of file gos\_kernel.c.

Here is the call graph for this function:

**4.13.3.23 gos\_result\_t gos\_kernelSetMaxCpuLoad ( u16\_t maxCpuLoad )**

Sets the maximum (global) CPU load. Sets the value of the global CPU load above which the scheduler will start running the idle task until the CPU load falls below the limit value.

**Parameters**

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <i>maxCpuLoad</i> | : Desired maximum CPU load (0...10000 where 100 % = 10000). |
|-------------------|-------------------------------------------------------------|

**Returns**

Result of maximum CPU load setting.

**Return values**

|                          |                                        |
|--------------------------|----------------------------------------|
| <code>GOS_SUCCESS</code> | : Maximum CPU load setting successful. |
|--------------------------|----------------------------------------|

|                        |                                  |
|------------------------|----------------------------------|
| <code>GOS_ERROR</code> | : Desired value is out of range. |
|------------------------|----------------------------------|

Definition at line 832 of file gos\_kernel.c.

#### 4.13.3.24 `gos_result_t gos_kernelStart( void_t )`

Starts the kernel.

Prepares the PSP for the first task, changes to unprivileged level, initializes the system timer value, and starts executing the first task.

Returns

Result of kernel start.

Return values

|                        |                          |
|------------------------|--------------------------|
| <code>GOS_ERROR</code> | : First task terminated. |
|------------------------|--------------------------|

Definition at line 348 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.4 Variable Documentation

##### 4.13.4.1 `u8_t atomicCntr = 0u`

Atomic counter (incremented when GOS\_ATOMIC\_ENTER is called).

Definition at line 194 of file gos\_kernel.c.

##### 4.13.4.2 `u16_t cpuUseLimit = 10000`

CPU use limit.

Definition at line 209 of file gos\_kernel.c.

**4.13.4.3 u32\_t currentTaskIndex = 0u**

Current task index - for priority scheduling.

Definition at line 204 of file gos\_kernel.c.

**4.13.4.4 u8\_t inlsr = 0u**

In ISR counter.

Definition at line 189 of file gos\_kernel.c.

**4.13.4.5 GOS\_STATIC bool\_t isKernelRunning = GOS\_FALSE**

Flag to indicate whether kernel is running.

Definition at line 262 of file gos\_kernel.c.

**4.13.4.6 gos\_signalId\_t kernelDumpReadySignal**

Kernel dump ready signal.

Definition at line 179 of file gos\_kernel.c.

**4.13.4.7 gos\_signalId\_t kernelDumpSignal**

Kernel dump signal.

Definition at line 174 of file gos\_kernel.c.

**4.13.4.8 GOS\_STATIC gos\_privilegedHook\_t kernelPrivilegedHookFunction = NULL**

Kernel privileged execution mode set hook function.

Definition at line 242 of file gos\_kernel.c.

**4.13.4.9 GOS\_STATIC gos\_taskSwapHook\_t kernelSwapHookFunction = NULL**

Kernel swap hook function.

Definition at line 232 of file gos\_kernel.c.

**4.13.4.10 GOS\_STATIC gos\_sysTickHook\_t kernelSysTickHookFunction = NULL**

Kernel system tick hook function.

Definition at line 237 of file gos\_kernel.c.

**4.13.4.11 GOS\_STATIC gos\_runtime\_t monitoringTime = {0}**

Monitoring system time since last statistics calculation.

Definition at line 227 of file gos\_kernel.c.

**4.13.4.12 GOS\_STATIC u32\_t previousTick = 0u**

Previous tick value for sleep and block tick calculation.

Definition at line 257 of file gos\_kernel.c.

**4.13.4.13 u32\_t primask = 0u**

IRQ state for restoring after GOS\_ATOMIC\_EXIT.

Definition at line 199 of file gos\_kernel.c.

**4.13.4.14 GOS\_STATIC bool\_t privilegedModeSetRequired = GOS\_FALSE**

Flag to indicate whether privileged mode set is required.

Definition at line 252 of file gos\_kernel.c.

**4.13.4.15 GOS\_STATIC bool\_t resetRequired = GOS\_FALSE**

Reset required flag.

Definition at line 247 of file gos\_kernel.c.

**4.13.4.16 u8\_t schedDisableCntr = 0u**

Scheduling disabled counter.

Definition at line 184 of file gos\_kernel.c.

**4.13.4.17 GOS\_STATIC u32\_t sysTicks = 0u**

System tick value.

Definition at line 217 of file gos\_kernel.c.

**4.13.4.18 GOS\_STATIC u16\_t sysTimerValue = 0u**

System timer value for run-time calculations.

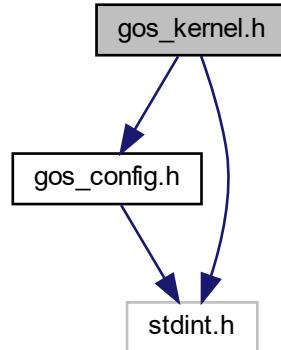
Definition at line 222 of file gos\_kernel.c.

## 4.14 gos\_kernel.h File Reference

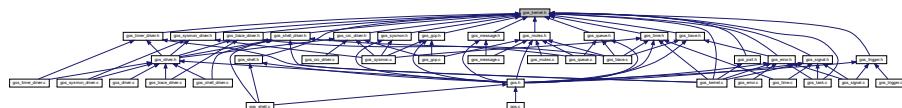
GOS kernel header.

```
#include <gos_config.h>
#include <stdint.h>
```

Include dependency graph for gos\_kernel.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define **NULL** ( void \* ) 0 )
- #define **RAM\_START** ( 0x20000000u )
- #define **RAM\_SIZE** ( 128 \* 1024 )
- #define **MAIN\_STACK** ( **RAM\_START** + **RAM\_SIZE** )
- #define **GLOBAL\_STACK** ( 0x1200 )
- #define **GOS\_DEFAULT\_TASK\_ID** ( (gos\_tid\_t)0x8000 )
- #define **GOS\_INVALID\_TASK\_ID** ( (gos\_tid\_t)0x0100 )
- #define **GOS\_TASK\_MAX\_PRIO\_LEVELS** ( **UINT8\_MAX** )
- #define **GOS\_TASK\_IDLE\_PRIO** ( **GOS\_TASK\_MAX\_PRIO\_LEVELS** )
- #define **GOS\_TASK\_MAX\_BLOCK\_TIME\_MS** ( 0xFFFFFFFFu )
- #define **GOS\_STATIC** static
- #define **GOS\_CONST** const
- #define **GOS\_INLINE** inline \_\_attribute\_\_ ((always\_inline))
- #define **GOS\_STATIC\_INLINE** GOS\_STATIC GOS\_INLINE
- #define **GOS\_EXTERN** extern
- #define **GOS\_NAKED** \_\_attribute\_\_ ((naked))
- #define **GOS\_UNUSED** \_\_attribute\_\_ ((unused))
- #define **GOS\_NOP** \_\_asm volatile ("NOP")
- #define **GOS\_ASM** \_\_asm volatile
- #define **GOS\_DISABLE\_SCHED**
- #define **GOS\_ENABLE\_SCHED**
- #define **GOS\_ISR\_ENTER**
- #define **GOS\_ISR\_EXIT**

- #define GOS\_ATOMIC\_ENTER
- #define GOS\_ATOMIC\_EXIT
- #define GOS\_PRIV\_TASK\_MANIPULATE ( 1 << 15 )
- #define GOS\_PRIV\_TASK\_PRIO\_CHANGE ( 1 << 14 )
- #define GOS\_PRIV\_TRACE ( 1 << 13 )
- #define GOS\_PRIV\_SIGNALING ( 1 << 11 )
- #define GOS\_PRIV\_RESERVED\_3 ( 1 << 10 )
- #define GOS\_PRIV\_RESERVED\_4 ( 1 << 9 )
- #define GOS\_PRIV\_RESERVED\_5 ( 1 << 8 )
- #define GOS\_CONCAT\_RESULT(finalResult, currentResult)

## TypeDefs

- typedef uint8\_t **bool\_t**  
*Boolean logic type.*
- typedef uint8\_t **u8\_t**  
*8-bit unsigned type.*
- typedef uint16\_t **u16\_t**  
*16-bit unsigned type.*
- typedef uint32\_t **u32\_t**  
*32-bit unsigned type.*
- typedef uint64\_t **u64\_t**  
*64-bit unsigned type.*
- typedef int8\_t **s8\_t**  
*8-bit signed type.*
- typedef int16\_t **s16\_t**  
*16-bit signed type.*
- typedef int32\_t **s32\_t**  
*32-bit signed type.*
- typedef int64\_t **s64\_t**  
*64-bit signed type.*
- typedef char **char\_t**  
*8-bit character type.*
- typedef float **float\_t**  
*Single precision float type.*
- typedef double **double\_t**  
*Double precision float type.*
- typedef void **void\_t**  
*Void type.*
- typedef u16\_t **gos\_tid\_t**  
*Task ID type.*
- typedef char\_t **gos\_taskName\_t** [CFG\_TASK\_MAX\_NAME\_LENGTH]  
*Task name type.*
- typedef void\_t(\* **gos\_task\_t** )(void\_t)  
*Task function type.*
- typedef u8\_t **gos\_taskPrio\_t**  
*Task priority type.*
- typedef u32\_t **gos\_taskSleepTick\_t**  
*Sleep tick type.*
- typedef u32\_t **gos\_blockMaxTick\_t**  
*Block max. tick type.*

- **typedef u32\_t gos\_taskAddress\_t**  
*Memory address type.*
- **typedef u32\_t gos\_taskRunCounter\_t**  
*Run counter type.*
- **typedef u64\_t gos\_taskRunTime\_t**  
*Run-time type.*
- **typedef u32\_t gos\_taskCSCounter\_t**  
*Context-switch counter type.*
- **typedef u16\_t gos\_taskStackSize\_t**  
*Task stack size type.*
- **typedef void\_t(\* gos\_taskSwapHook\_t)(gos\_tid\_t, gos\_tid\_t)**  
*Task swap hook type.*
- **typedef void\_t(\* gos\_taskIdleHook\_t)(void\_t)**  
*Task idle hook type.*
- **typedef void\_t(\* gos\_taskSleepHook\_t)(gos\_tid\_t)**  
*Task sleep hook type.*
- **typedef void\_t(\* gos\_taskWakeupHook\_t)(gos\_tid\_t)**  
*Task wake-up hook type.*
- **typedef void\_t(\* gos\_taskSuspendHook\_t)(gos\_tid\_t)**  
*Task suspend hook type.*
- **typedef void\_t(\* gos\_taskResumeHook\_t)(gos\_tid\_t)**  
*Task resume hook type.*
- **typedef void\_t(\* gos\_taskBlockHook\_t)(gos\_tid\_t)**  
*Task block hook type.*
- **typedef void\_t(\* gos\_taskUnblockHook\_t)(gos\_tid\_t)**  
*Task unblock hook type.*
- **typedef void\_t(\* gos\_taskDeleteHook\_t)(gos\_tid\_t)**  
*Task delete hook type.*
- **typedef void\_t(\* gos\_sysTickHook\_t)(void\_t)**  
*System tick hook type.*
- **typedef void\_t(\* gos\_privilegedHook\_t)(void\_t)**  
*Privileged mode hook type.*
- **typedef u16\_t gos\_microsecond\_t**  
*Microsecond type.*
- **typedef u16\_t gos\_millisecond\_t**  
*Millisecond type.*
- **typedef u8\_t gos\_second\_t**  
*Second type.*
- **typedef u8\_t gos\_minute\_t**  
*Minute type.*
- **typedef u8\_t gos\_hour\_t**  
*Hour type.*
- **typedef u16\_t gos\_day\_t**  
*Day type.*
- **typedef u8\_t gos\_month\_t**  
*Month type.*
- **typedef u16\_t gos\_year\_t**  
*Year type.*

## Enumerations

- enum `gos_taskState_t` {
 `GOS_TASK_READY` = 0b01010, `GOS_TASK_SLEEPING` = 0b10110, `GOS_TASK_BLOCKED` = 0b11001,
 `GOS_TASK_SUSPENDED` = 0b00101,
 `GOS_TASK_ZOMBIE` = 0b01101 }
- enum `gos_taskPrivilegeLevel_t` { `GOS_TASK_PRIVILEGE_SUPERVISOR` = 0xFFFF, `GOS_TASK_PRIVILEGE_KERNEL` = 0xFF00, `GOS_TASK_PRIVILEGE_USER` = 0x00FF, `GOS_TASK_PRIVILEGED_USER` = 0x20FF }
- enum `gos_result_t` { `GOS_SUCCESS` = 0b01010101, `GOS_ERROR` = 0b10101110, `GOS_BUSY` = 0b10110001 }
- enum `gos_boolValue_t` { `GOS_TRUE` = 0b00110110, `GOS_FALSE` = 0b01001001 }
- enum `gos_kernel_privilege_t` { `GOS_PRIVILEGED` = 0b10110, `GOS_UNPRIVILEGED` = 0b01001 }

## Functions

- struct `__attribute__((packed))`
- `gos_result_t gos_kernelInit (void_t)`

*This function initializes the kernel.*
- `gos_result_t gos_taskRegisterTasks (gos_taskDescriptor_t *taskDescriptors, u16_t arraySize)`

*This function registers an array of tasks for scheduling.*
- `gos_result_t gos_taskRegister (gos_taskDescriptor_t *taskDescriptor, gos_tid_t *taskId)`

*This function registers a task for scheduling.*
- `gos_result_t gos_taskSleep (gos_taskSleepTick_t sleepTicks)`

*Sends the current task to sleeping state.*
- `gos_result_t gos_taskWakeup (gos_tid_t taskId)`

*Wakes up the given task.*
- `gos_result_t gos_taskSuspend (gos_tid_t taskId)`

*Sends the given task to suspended state.*
- `gos_result_t gos_taskResume (gos_tid_t taskId)`

*Resumes the given task.*
- `gos_result_t gos_taskBlock (gos_tid_t taskId, gos_blockMaxTick_t blockTicks)`

*Sends the given task to blocked state.*
- `gos_result_t gos_taskUnblock (gos_tid_t taskId)`

*Unblocks the given task.*
- `gos_result_t gos_taskDelete (gos_tid_t taskId)`

*Deletes the given task from the scheduling array.*
- `gos_result_t gos_taskSetPriority (gos_tid_t taskId, gos_taskPrio_t taskPriority)`

*Sets the current priority of the given task to the given value (for temporary change).*
- `gos_result_t gos_taskSetOriginalPriority (gos_tid_t taskId, gos_taskPrio_t taskPriority)`

*Sets the original priority of the given task to the given value (for permanent change).*
- `gos_result_t gos_taskGetPriority (gos_tid_t taskId, gos_taskPrio_t *taskPriority)`

*Gets the current priority of the given task.*
- `gos_result_t gos_taskGetOriginalPriority (gos_tid_t taskId, gos_taskPrio_t *taskPriority)`

*Gets the original priority of the given task.*
- `gos_result_t gos_taskAddPrivilege (gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)`

*Adds the given privileges to the given task.*
- `gos_result_t gos_taskRemovePrivilege (gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)`

*Removes the given privileges from the given task.*
- `gos_result_t gos_taskSetPrivileges (gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)`

*Sets the given privileges for the given task.*
- `gos_result_t gos_taskGetPrivileges (gos_tid_t taskId, gos_taskPrivilegeLevel_t *privileges)`

- `gos_result_t gos_taskGetName (gos_tid_t taskId, gos_taskName_t taskName)`  
*Gets the task name of the task with the given ID.*
- `gos_result_t gos_taskGetId (gos_taskName_t taskName, gos_tid_t *taskId)`  
*Gets the task ID of the task with the given name.*
- `gos_result_t gos_taskGetCurrentId (gos_tid_t *taskId)`  
*Returns the ID of the currently running task.*
- `gos_result_t gos_taskGetData (gos_tid_t taskId, gos_taskDescriptor_t *taskData)`  
*Returns the task data of the given task.*
- `gos_result_t gos_taskGetDataByIndex (u16_t taskIndex, gos_taskDescriptor_t *taskData)`  
*Returns the task data of the given task.*
- `gos_result_t gos_taskGetNumber (u16_t *pTaskNum)`  
*Returns the number of registered tasks.*
- `gos_result_t gos_taskYield (void_t)`  
*Yields the current task.*
- `gos_result_t gos_kernelRegisterSwapHook (gos_taskSwapHook_t swapHookFunction)`  
*Registers a task swap hook function.*
- `gos_result_t gos_kernelRegisterIdleHook (gos_taskIdleHook_t idleHookFunction)`  
*Registers an idle hook function.*
- `gos_result_t gos_kernelRegisterSysTickHook (gos_sysTickHook_t sysTickHookFunction)`  
*Registers a system tick hook function.*
- `gos_result_t gos_kernelRegisterPrivilegedHook (gos_privilegedHook_t privilegedHookFunction)`  
*Registers a privileged hook function.*
- `gos_result_t gos_taskSubscribeDeleteSignal (void_t(*deleteSignalHandler)(u16_t))`  
*Subscribes the given handler to the task delete signal.*
- `gos_result_t gos_kernelSubscribeDumpReadySignal (void_t(*dumpReadySignalHandler)(u16_t))`  
*Subscribes the given handler to the dump ready signal.*
- `u32_t gos_kernelGetSysTicks (void_t)`  
*Returns the system ticks.*
- `u16_t gos_kernelGetCpuUsage (void_t)`  
*Returns the CPU usage.*
- `gos_result_t gos_kernelStart (void_t)`  
*Starts the kernel.*
- `void_t gos_kernelReset (void_t)`  
*Resets the microcontroller.*
- `void_t gos_kernelPrivilegedModeSetRequired (void_t)`  
*Requests a privileged mode setting.*
- `void_t gos_kernelDelayUs (u16_t microseconds)`  
*Blocking delay in microsecond range.*
- `void_t gos_kernelDelayMs (u16_t milliseconds)`  
*Blocking delay in millisecond range.*
- `void_t gos_kernelCalculateTaskCpuUsages (bool_t isResetRequired)`  
*Calculates the CPU usage for the tasks.*
- `void_t gos_kernelDump (void_t)`  
*Kernel dump.*
- `gos_result_t gos_kernelSetMaxCpuLoad (u16_t maxCpuLoad)`  
*Sets the maximum (global) CPU load. Sets the value of the global CPU load above which the scheduler will start running the idle task until the CPU load falls below the limit value.*
- `gos_result_t gos_kernelGetMaxCpuLoad (u16_t *maxCpuLoad)`  
*Gets the maximum (global) CPU load. Returns the value of the maximum CPU load (limit).*
- `bool_t gos_kernelsIsCallerLsr (void_t)`

*Returns if the current task is ISR. Returns if the inlsr flag is greater than zero.*

- `void_t gos_kernelReschedule (gos_kernel_privilege_t privilege)`  
*Reschedules the kernel.*
- `__attribute__ ((weak)) gos_result_t gos_platformDriverInit(void_t)`  
*Platform driver initializer. Used for the platform-specific driver initializations.*

## Variables

- `gos_runtime_t`
- `gos_taskDescriptor_t`

### 4.14.1 Detailed Description

GOS kernel header.

#### Author

Ahmed Gazar

#### Date

2024-06-13

#### Version

1.21

The GOS kernel is the core of the GOS system. It contains the basic type definitions of the system (these are used across the OS in other services), and it handles the system tick, tasks, and scheduling. The kernel module also provides an interface for registering tasks, changing their states, registering hook functions, disabling the scheduler. It also provides a service to dump the kernel configuration and task information on the log output.

Definition in file [gos\\_kernel.h](#).

### 4.14.2 Macro Definition Documentation

#### 4.14.2.1 `#define GLOBAL_STACK ( 0x1200 )`

Global stack.

Definition at line 142 of file [gos\\_kernel.h](#).

#### 4.14.2.2 `#define GOS_ASM __asm volatile`

ASM.

Definition at line 212 of file [gos\\_kernel.h](#).

#### 4.14.2.3 `#define GOS_ATOMIC_ENTER`

#### Value:

```
{
 GOS_EXTERN \
 GOS_EXTERN \
 if (atomicCntr == 0) \
 { \
 GOS_ASM \
 GOS_ASM; \
 GOS_ASM \
 GOS_ASM \
 GOS_ASM \
 } \
 atomicCntr+ \
 GOS_DISABLE_SCHED \
}
}
```

Atomic operation enter - disable interrupts and kernel rescheduling.

Definition at line 250 of file gos\_kernel.h.

#### 4.14.2.4 #define GOS\_ATOMIC\_EXIT

**Value:**

```
{
 GOS_EXTERN \
 GOS_EXTERN \
 if (atomicCntr > 0) \
 { \
 atomicCntr \
 } \
 if (atomicCntr == 0) \
 { \
 GOS_ASM \
 GOS_ASM; \
 GOS_ASM \
 GOS_ASM \
 GOS_ASM \
 } \
 GOS_ENABLE_SCHED \
}
}
```

Atomic operation exit - enable interrupts kernel rescheduling.

Definition at line 267 of file gos\_kernel.h.

#### 4.14.2.5 #define GOS\_CONCAT\_RESULT( finalResult, currentResult )

**Value:**

```
{
 \
 if (finalResult == \
 GOS_SUCCESS) \
 { \
 finalResult = currentResult; \
 } \
 else \
 { \
 }
```

```
finalResult =
GOS_ERROR; \
} \
}
```

Result concatenating macro.

Definition at line 322 of file gos\_kernel.h.

#### 4.14.2.6 #define GOS\_CONST const

Constant macro.

Definition at line 177 of file gos\_kernel.h.

#### 4.14.2.7 #define GOS\_DEFAULT\_TASK\_ID ( (gos\_tid\_t)0x8000 )

Default task ID.

Definition at line 147 of file gos\_kernel.h.

#### 4.14.2.8 #define GOS\_DISABLE\_SCHED

##### Value:

```
{
 u8_t schedDisableCntr; \
} \
GOS_EXTERN
++; \
}
```

Disable scheduling.

Definition at line 217 of file gos\_kernel.h.

#### 4.14.2.9 #define GOS\_ENABLE\_SCHED

##### Value:

```
{
 u8_t schedDisableCntr; \
} \
GOS_EXTERN
if (schedDisableCntr > 0)
{ schedDisableCntr--; } \
}
```

Enable scheduling.

Definition at line 224 of file gos\_kernel.h.

#### 4.14.2.10 #define GOS\_EXTERN extern

Extern macro.

Definition at line 192 of file gos\_kernel.h.

#### 4.14.2.11 #define GOS\_INLINE inline \_\_attribute\_\_((always\_inline))

Inline macro.

Definition at line 182 of file gos\_kernel.h.

4.14.2.12 #define GOS\_INVALID\_TASK\_ID ( (gos\_tid\_t)0x0100 )

Invalid task ID.

Definition at line 152 of file gos\_kernel.h.

4.14.2.13 #define GOS\_ISR\_ENTER

**Value:**

```
{
 \ GOS_EXTERN
 u8_t inIsr; \
 if (inIsr == 0) { GOS_DISABLE_SCHED }
 inIsr++;
}
```

Interrupt Service Routine enter.

Definition at line 233 of file gos\_kernel.h.

4.14.2.14 #define GOS\_ISR\_EXIT

**Value:**

```
{
 \ GOS_EXTERN
 u8_t inIsr; \
 if (inIsr > 0) { inIsr--; }
 if (inIsr == 0) { GOS_ENABLE_SCHED } \
}
```

Interrupt service routine exit.

Definition at line 241 of file gos\_kernel.h.

4.14.2.15 #define GOS\_NAKED \_\_attribute\_\_ ((naked))

Naked.

Definition at line 197 of file gos\_kernel.h.

4.14.2.16 #define GOS\_NOP \_\_asm volatile ("NOP")

NOP.

Definition at line 207 of file gos\_kernel.h.

4.14.2.17 #define GOS\_PRIV\_RESERVED\_3 (1 << 10)

Kernel reserved.

Definition at line 307 of file gos\_kernel.h.

4.14.2.18 #define GOS\_PRIV\_RESERVED\_4 (1 << 9)

Kernel reserved.

Definition at line 312 of file gos\_kernel.h.

4.14.2.19 `#define GOS_PRIV_RESERVED_5 ( 1 << 8 )`

Kernel reserved.

Definition at line 317 of file gos\_kernel.h.

4.14.2.20 `#define GOS_PRIV_SIGNALING ( 1 << 11 )`

Task signal invoking privilege flag.

Definition at line 302 of file gos\_kernel.h.

4.14.2.21 `#define GOS_PRIV_TASK_MANIPULATE ( 1 << 15 )`

Task manipulation privilege flag.

Definition at line 287 of file gos\_kernel.h.

4.14.2.22 `#define GOS_PRIV_TASK_PRIO_CHANGE ( 1 << 14 )`

Task priority change privilege flag.

Definition at line 292 of file gos\_kernel.h.

4.14.2.23 `#define GOS_PRIV_TRACE ( 1 << 13 )`

Tracing privilege flag.

Definition at line 297 of file gos\_kernel.h.

4.14.2.24 `#define GOS_STATIC static`

Static macro.

Definition at line 172 of file gos\_kernel.h.

4.14.2.25 `#define GOS_STATIC_INLINE GOS_STATIC GOS_INLINE`

Static in-line macro.

Definition at line 187 of file gos\_kernel.h.

4.14.2.26 `#define GOS_TASK_IDLE_PRIO ( GOS_TASK_MAX_PRIO_LEVELS )`

Idle task priority.

Definition at line 162 of file gos\_kernel.h.

4.14.2.27 `#define GOS_TASK_MAX_BLOCK_TIME_MS ( 0xFFFFFFFFu )`

Task maximum block time.

Definition at line 167 of file gos\_kernel.h.

4.14.2.28 `#define GOS_TASK_MAX_PRIO_LEVELS ( UINT8_MAX )`

Maximum task priority levels.

Definition at line 157 of file gos\_kernel.h.

4.14.2.29 `#define GOS_UNUSED __attribute__ ((unused))`

Unused.

Definition at line 202 of file gos\_kernel.h.

4.14.2.30 `#define MAIN_STACK ( RAM_START + RAM_SIZE )`

Main stack.

Definition at line 137 of file gos\_kernel.h.

4.14.2.31 `#define NULL ( void * ) 0 )`

NULL pointer.

Definition at line 121 of file gos\_kernel.h.

4.14.2.32 `#define RAM_SIZE ( 128 * 1024 )`

RAM size (128kB).

Definition at line 132 of file gos\_kernel.h.

4.14.2.33 `#define RAM_START ( 0x20000000u )`

RAM start address.

Definition at line 127 of file gos\_kernel.h.

## 4.14.3 Enumeration Type Documentation

4.14.3.1 `enum gos_boolValue_t`

Boolean values.

### Note

Hamming distance of true and false value: 7.

### Enumerator

**GOS\_TRUE** True value.

**GOS\_FALSE** False value.

Definition at line 437 of file gos\_kernel.h.

#### 4.14.3.2 enum gos\_kernel\_privilege\_t

Kernel privilege levels.

Enumerator

**GOS\_PRIVILEGED** GOS\_PRIVILEDGED.  
**GOS\_UNPRIVILEGED** GOS\_UNPRIVILEDGED.

Definition at line 446 of file gos\_kernel.h.

#### 4.14.3.3 enum gos\_result\_t

Result type enumerator.

Note

Hamming distance of

- success and error: 7
- success and busy: 4
- error and busy: 5

Enumerator

**GOS\_SUCCESS** Success.  
**GOS\_ERROR** Error.  
**GOS\_BUSY** Busy.

Definition at line 426 of file gos\_kernel.h.

#### 4.14.3.4 enum gos\_taskPrivilegeLevel\_t

Task privilege level enumerator.

Enumerator

**GOS\_TASK\_PRIVILEGE\_SUPERVISOR** Task supervisor privilege level.  
**GOS\_TASK\_PRIVILEGE\_KERNEL** Task kernel privilege level.  
**GOS\_TASK\_PRIVILEGE\_USER** Task user privilege level.  
**GOS\_TASK\_PRIVILEGED\_USER** User with logging right.

Definition at line 396 of file gos\_kernel.h.

#### 4.14.3.5 enum gos\_taskState\_t

Task state enumerator.

Note

Hamming distance of

- ready and sleeping: 3
- ready and blocked: 3
- ready and suspended: 4
- ready and zombie: 3

- sleeping and blocked: 4
- sleeping and suspended: 3
- sleeping and zombie: 4
- blocked and suspended: 3
- blocked and zombie: 2
- suspended and zombie: 1

**Enumerator**

**GOS\_TASK\_READY** Task is ready to be scheduled.

**GOS\_TASK\_SLEEPING** Task is sleeping (waiting for wake-up ticks).

**GOS\_TASK\_BLOCKED** Task is blocked (waiting for resource).

**GOS\_TASK\_SUSPENDED** Task is suspended (not to be scheduled), but can be resumed.

**GOS\_TASK\_ZOMBIE** Task deleted (physically existing in memory, but cannot be resumed).

Definition at line 384 of file gos\_kernel.h.

#### 4.14.4 Function Documentation

##### 4.14.4.1 struct \_\_attribute\_\_ ( (packed) )

Run-time type.

Task descriptor structure.

Task variable data message structure.

Ping message structure.

CPU usage message structure.

Task data get message structure.

Task data message structure.

Task modify message structure.

Task modify message result structure.

System runtime get message result structure.

System time set message structure.

System time set message result structure. < Microseconds.

< Milliseconds.

< Seconds.

< Minutes.

< Hours.

< Days.

< Task function.

< Task state.

< Task previous state (for restoration).

< Task priority.

< Task original priority.

< Task privilege level.

< Task name.  
 < Task ID (internal).  
 < Task sleep ticks.  
 < Task sleep tick counter.  
 < Task block ticks.  
 < Task block tick counter.  
 < Task PSP.  
 < Task run counter.  
 < Task context-switch counter.  
 < Task allocated stack size.  
 < Task max. stack size usage.  
 < Task run-time.  
 < Task monitoring run-time (not erased).  
 < Task CPU usage limit in [% x 100].  
 < Task CPU usage max value in [% x 100].  
 < Task processor usage in [% x 100].  
 < Task CPU usage monitoring value in [% x 100].  
 < Task stack overflow threshold address.

Definition at line 464 of file gos\_kernel.h.

#### 4.14.4.2 \_\_attribute\_\_ ( (weak) )

Platform driver initializer. Used for the platform-specific driver initializations.

User application initializer. Used for the application-related initializations.

This function is weak and therefore should be over-defined by the user. It prints a warning message to the log output in case it is not over-defined.

##### Returns

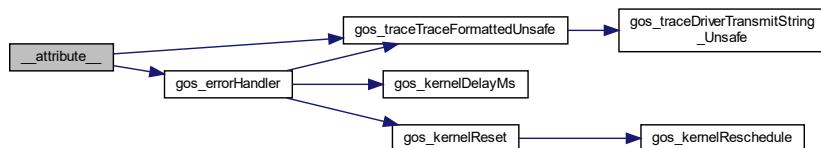
-

##### Return values

|           |   |   |
|-----------|---|---|
| GOS_ERROR | : | - |
|-----------|---|---|

Definition at line 250 of file gos.c.

Here is the call graph for this function:



#### 4.14.4.3 void\_t gos\_kernelCalculateTaskCpuUsages ( bool\_t isResetRequired )

Calculates the CPU usage for the tasks.

Based on the total system time range, it refreshes the CPU-usage statistics of tasks.

##### Parameters

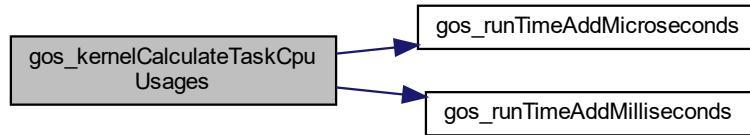
|                        |                                                             |
|------------------------|-------------------------------------------------------------|
| <i>isResetRequired</i> | : Flag to indicate whether the measurement should be reset. |
|------------------------|-------------------------------------------------------------|

##### Returns

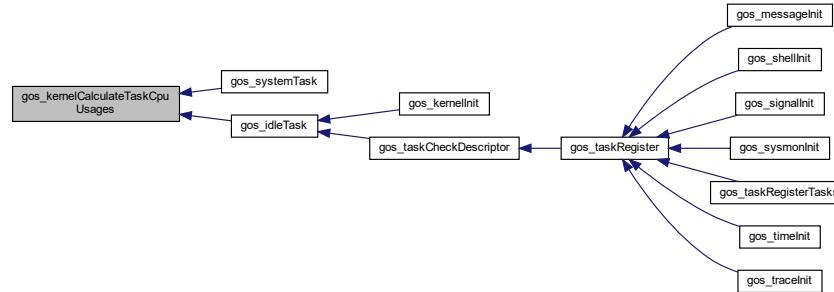
-

Definition at line 618 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.4 void\_t gos\_kernelDelayMs ( u16\_t milliseconds )

Blocking delay in millisecond range.

This function waits in a while loop until the given number of system ticks have elapsed.

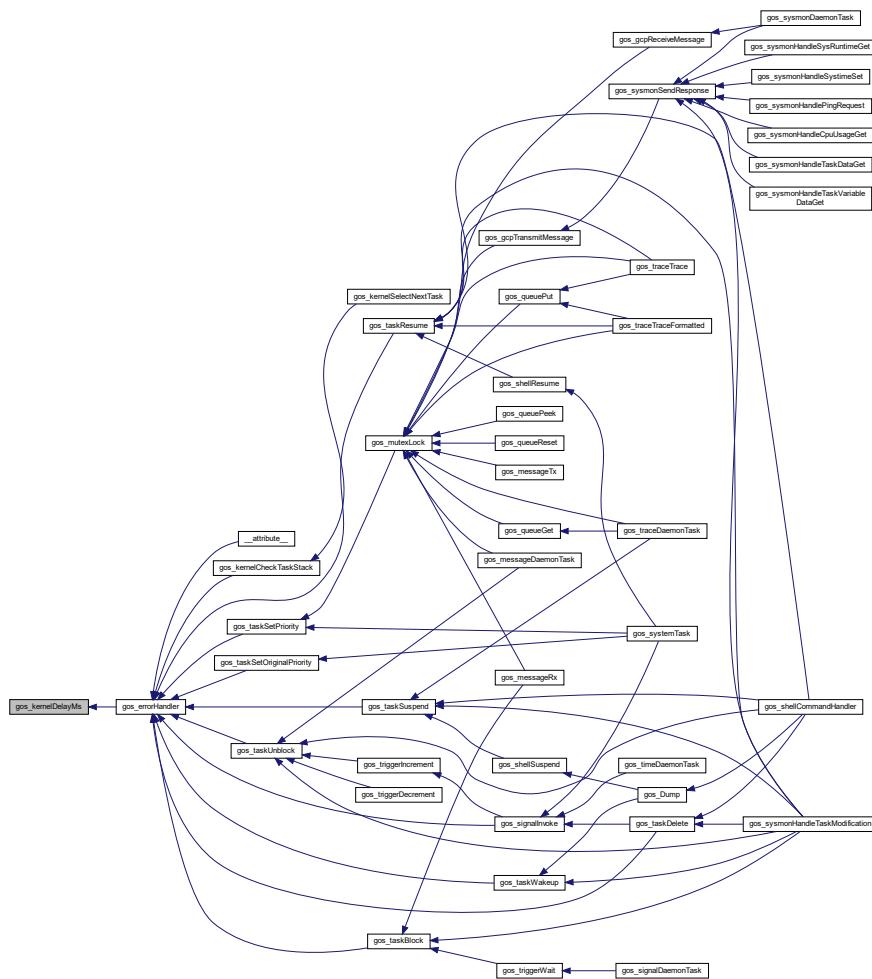
##### Parameters

|                     |                         |
|---------------------|-------------------------|
| <i>milliseconds</i> | : Milliseconds to wait. |
|---------------------|-------------------------|

## Returns

Definition at line 602 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.14.4.5 void\_t gos\_kernelDelayUs( u16\_t microseconds )

Blocking delay in microsecond range.

This function waits in a while loop until the given number of milliseconds have elapsed based on the system timer.

## Parameters

*microseconds* : Microseconds to wait.

## Returns

Definition at line 580 of file gos\_kernel.c.

Here is the call graph for this function:



#### 4.14.4.6 void\_t gos\_kernelDump( void\_t )

Kernel dump.

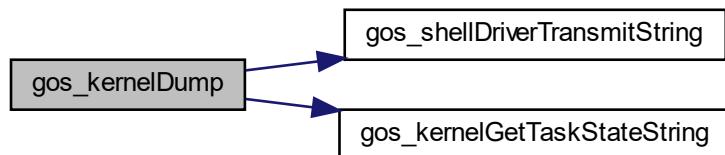
This function prints the kernel configuration and task data to the trace output.

##### Returns

-

Definition at line 720 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.7 u16\_t gos\_kernelGetCpuUsage( void\_t )

Returns the CPU usage.

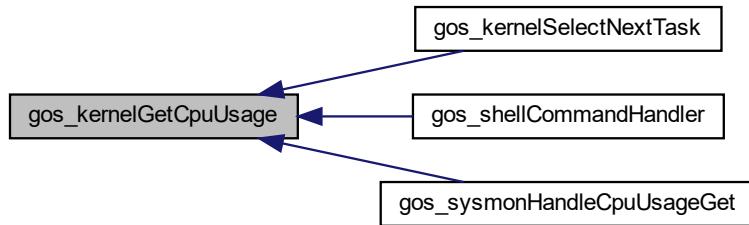
Refreshes the CPU statistics and returns the current CPU usage in [CPU%] \* 100 format that results in a range of 0...10000 where the last two digits represent two decimals. The usage is 100% - idle task%.

**Returns**

Overall CPU usage.

Definition at line 544 of file gos\_kernel.c.

Here is the caller graph for this function:

**4.14.4.8 gos\_result\_t gos\_kernelGetMaxCpuLoad ( u16\_t \* maxCpuLoad )**

Gets the maximum (global) CPU load. Returns the value of the maximum CPU load (limit).

**Parameters**

|                         |                                                  |
|-------------------------|--------------------------------------------------|
| <code>maxCpuLoad</code> | : Target variable to store the maximum CPU load. |
|-------------------------|--------------------------------------------------|

**Returns**

Result of maximum CPU load getting.

**Return values**

|                          |                                        |
|--------------------------|----------------------------------------|
| <code>GOS_SUCCESS</code> | : Maximum CPU load getting successful. |
| <code>GOS_ERROR</code>   | : Target variable is NULL.             |

Definition at line 858 of file gos\_kernel.c.

**4.14.4.9 u32\_t gos\_kernelGetSysTicks ( void\_t )**

Returns the system ticks.

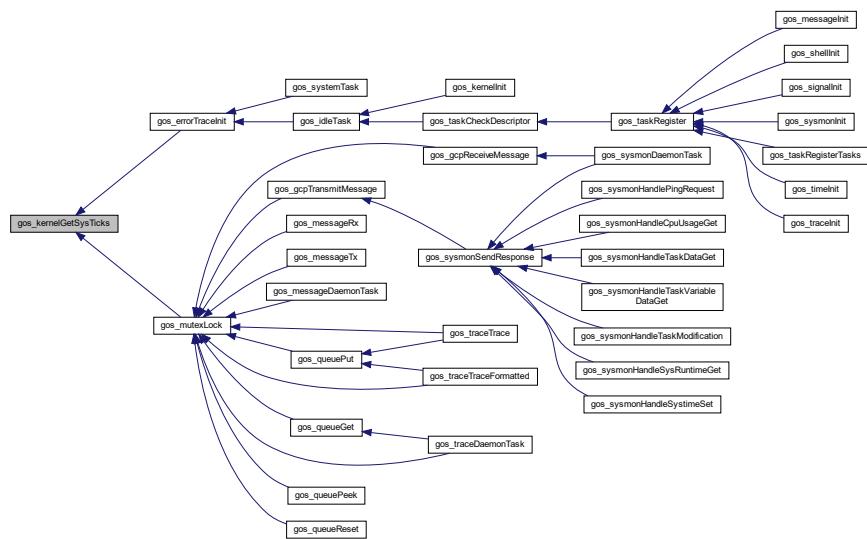
Returns the internal system tick counter value.

## Returns

## System ticks.

Definition at line 533 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.14.4.10 gos\_result\_t gos\_kernellnit ( void\_t )

This function initializes the kernel.

Initializes the internal task array, fills out the PSP for the idle task, and registers the kernel dump task and suspends it.

## Returns

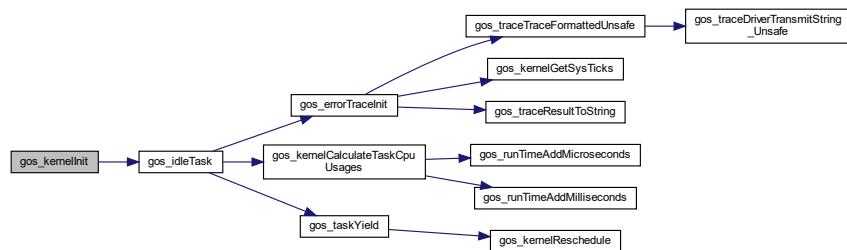
## Result of initialization.

## Return values

|                    |                                        |
|--------------------|----------------------------------------|
| <b>GOS_SUCCESS</b> | : Kernel initialization successful.    |
| <b>GOS_ERROR</b>   | : Kernel task suspension unsuccessful. |

Definition at line 287 of file gos\_kernel.c.

Here is the call graph for this function:



#### 4.14.4.11 `bool_t gos_kernelsCallerIsr( void_t )`

Returns if the current task is ISR. Returns if the inlsr flag is greater than zero.

##### Returns

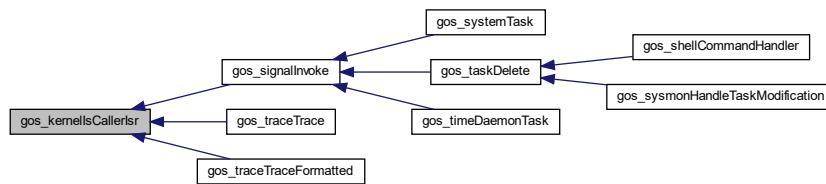
Whether the caller is ISR (Interrupt Service Routine).

##### Return values

|                        |                      |
|------------------------|----------------------|
| <code>GOS_TRUE</code>  | : Caller is ISR.     |
| <code>GOS_FALSE</code> | : Caller is not ISR. |

Definition at line 884 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.14.4.12 `void_t gos_kernelPrivilegedModeSetRequired( void_t )`

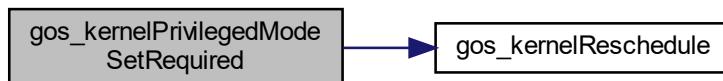
Requests a privileged mode setting.

Initiates an SVC call while setting the corresponding flag to set the execution mode to privileged (permanently). After this, it calls the corresponding hook function, so the caller can access special registers immediately.

##### Returns

Definition at line 567 of file gos\_kernel.c.

Here is the call graph for this function:



#### 4.14.4.13 `gos_result_t gos_kernelRegisterIdleHook( gos_taskIdleHook_t idleHookFunction )`

Registers an idle hook function.

Checks whether the param is NULL pointer and a hook function is already registered, and both conditions are false, it registers the hook function.

**Parameters**

|                         |                       |
|-------------------------|-----------------------|
| <i>idleHookFunction</i> | : Idle hook function. |
|-------------------------|-----------------------|

**Returns**

Result of registration.

**Return values**

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Registration successful.                                                 |
| <i>GOS_ERROR</i>   | : Registration failed (hook function already exists or parameter is NULL). |

**4.14.4.14 gos\_result\_t gos\_kernelRegisterPrivilegedHook ( gos\_privilegedHook\_t privilegedHookFunction )**

Registers a privileged hook function.

Checks if a hook function has already been registered, and, if not, it registers the new hook function.

**Parameters**

|                               |                             |
|-------------------------------|-----------------------------|
| <i>privilegedHookFunction</i> | : Privileged hook function. |
|-------------------------------|-----------------------------|

**Returns**

Result of registration.

**Return values**

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Registration successful.                                                 |
| <i>GOS_ERROR</i>   | : Registration failed (hook function already exists or parameter is NULL). |

Definition at line 438 of file gos\_kernel.c.

**4.14.4.15 gos\_result\_t gos\_kernelRegisterSwapHook ( gos\_taskSwapHook\_t swapHookFunction )**

Registers a task swap hook function.

Checks whether the param is NULL pointer and a hook function is already registered, and both conditions are false, it registers the hook function.

**Parameters**

|                         |                       |
|-------------------------|-----------------------|
| <i>swapHookFunction</i> | : Swap hook function. |
|-------------------------|-----------------------|

**Returns**

Result of registration.

**Return values**

|                    |                            |
|--------------------|----------------------------|
| <i>GOS_SUCCESS</i> | : Registration successful. |
|--------------------|----------------------------|

|                  |                                                                            |
|------------------|----------------------------------------------------------------------------|
| <i>GOS_ERROR</i> | : Registration failed (hook function already exists or parameter is NULL). |
|------------------|----------------------------------------------------------------------------|

Definition at line 386 of file gos\_kernel.c.

#### 4.14.4.16 `gos_result_t gos_kernelRegisterSysTickHook ( gos_sysTickHook_t sysTickHookFunction )`

Registers a system tick hook function.

Checks whether the param is NULL pointer and a hook function is already registered, and both conditions are false, it registers the hook function.

##### Parameters

|                            |                              |
|----------------------------|------------------------------|
| <i>sysTickHookFunction</i> | : System tick hook function. |
|----------------------------|------------------------------|

##### Returns

Result of registration.

##### Return values

|                    |                                                                            |
|--------------------|----------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Registration successful.                                                 |
| <i>GOS_ERROR</i>   | : Registration failed (hook function already exists or parameter is NULL). |

Definition at line 412 of file gos\_kernel.c.

#### 4.14.4.17 `void_t gos_kernelReschedule ( gos_kernel_privilege_t privilege )`

Reschedules the kernel.

Based on the privilege, it invokes a kernel reschedule event.

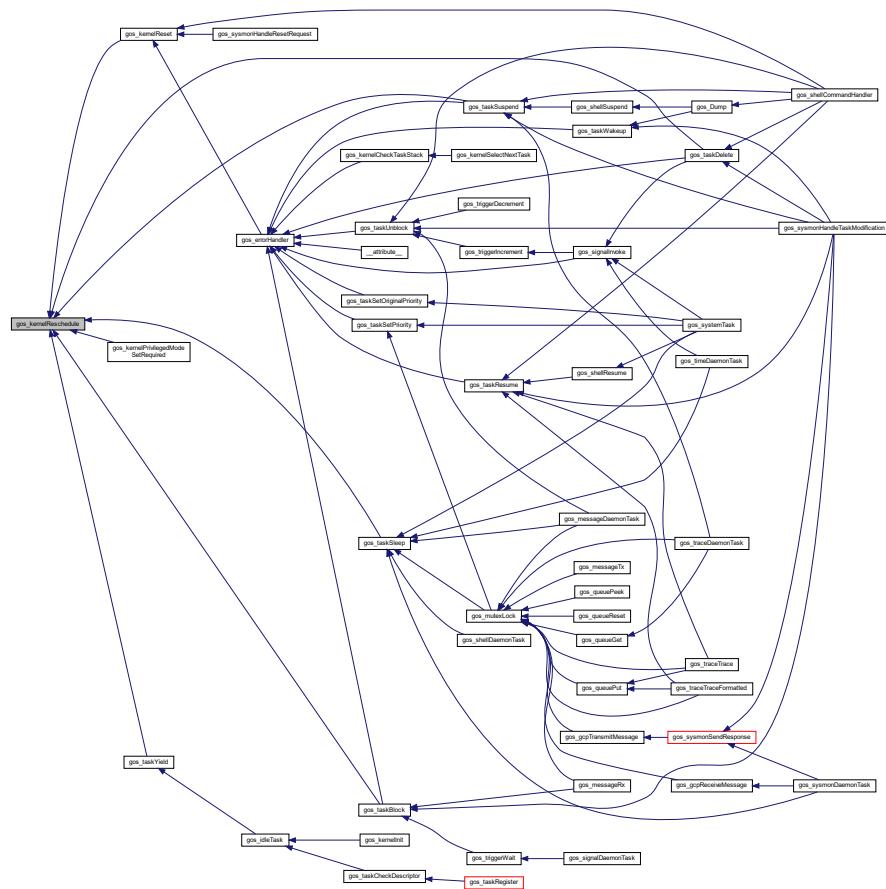
##### Parameters

|                  |                    |
|------------------|--------------------|
| <i>privilege</i> | : Privilege level. |
|------------------|--------------------|

## Returns

Definition at line 943 of file gos\_kernel.c.

Here is the caller graph for this function:



#### 4.14.4.18 void\_t gos\_kernelReset( void\_t )

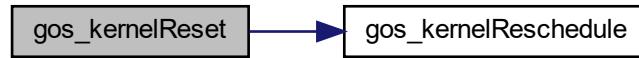
Resets the microcontroller.

Sets the reset required flag and gets privileged access to reset the controller.

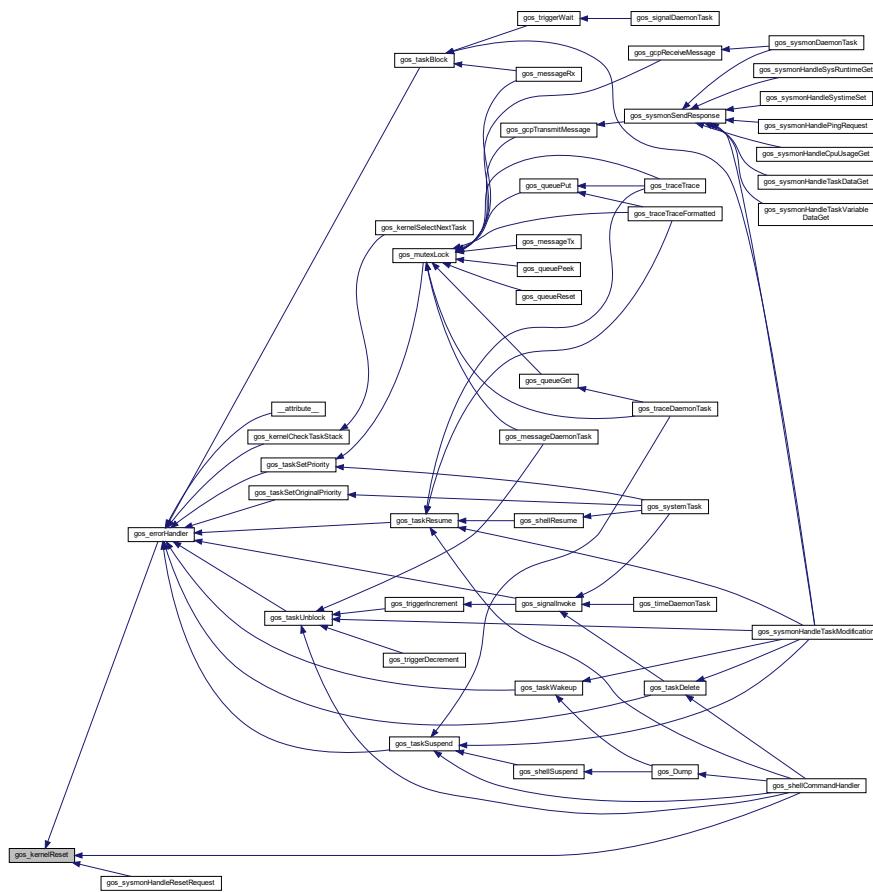
## Returns

Definition at line 555 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.14.4.19 gos\_result\_t gos\_kernelSetMaxCpuLoad ( u16\_t maxCpuLoad )

Sets the maximum (global) CPU load. Sets the value of the global CPU load above which the scheduler will start running the idle task until the CPU load falls below the limit value.

**Parameters**

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| <i>maxCpuLoad</i> | : Desired maximum CPU load (0...10000 where 100 % = 10000). |
|-------------------|-------------------------------------------------------------|

**Returns**

Result of maximum CPU load setting.

**Return values**

|                    |                                        |
|--------------------|----------------------------------------|
| <i>GOS_SUCCESS</i> | : Maximum CPU load setting successful. |
| <i>GOS_ERROR</i>   | : Desired value is out of range.       |

Definition at line 832 of file gos\_kernel.c.

**4.14.4.20 gos\_result\_t gos\_kernelStart( void\_t )**

Starts the kernel.

Prepares the PSP for the first task, changes to unprivileged level, initializes the system timer value, and starts executing the first task.

**Returns**

Result of kernel start.

**Return values**

|                  |                          |
|------------------|--------------------------|
| <i>GOS_ERROR</i> | : First task terminated. |
|------------------|--------------------------|

Definition at line 348 of file gos\_kernel.c.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.14.4.21 gos\_result\_t gos\_kernelSubscribeDumpReadySignal( void\_t(\*)(u16\_t) dumpReadySignalHandler )**

Subscribes the given handler to the dump ready signal.

Subscribes the given handler to the dump ready signal.

**Parameters**

|                               |                                       |
|-------------------------------|---------------------------------------|
| <i>dumpReadySignalHandler</i> | : Dump ready signal handler function. |
|-------------------------------|---------------------------------------|

**Returns**

Result of subscription.

**Return values**

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Subscription successful.                       |
| <i>GOS_ERROR</i>   | : Subscription failed or signal handler is NULL. |

**4.14.4.22 gos\_result\_t gos\_taskAddPrivilege ( gos\_tid\_t taskId, gos\_taskPrivilegeLevel\_t privileges )**

Adds the given privileges to the given task.

Checks the caller task if it has the privilege to modify task privileges and if so, it adds the given privileges to the given task.

**Parameters**

|                   |                                             |
|-------------------|---------------------------------------------|
| <i>taskId</i>     | : ID of the task to give the privileges to. |
| <i>privileges</i> | : Privileges to be added.                   |

**Returns**

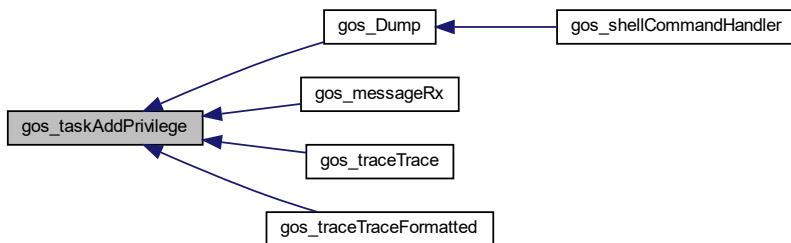
Result of privilege adding.

**Return values**

|                    |                                                                                    |
|--------------------|------------------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Privileges added successfully.                                                   |
| <i>GOS_ERROR</i>   | : Invalid task ID or caller does not have the privilege to modify task privileges. |

Definition at line 873 of file gos\_task.c.

Here is the caller graph for this function:

**4.14.4.23 gos\_result\_t gos\_taskBlock ( gos\_tid\_t taskId, gos\_blockMaxTick\_t blockTicks )**

Sends the given task to blocked state.

Checks the given task ID and its state, modified it to blocked, and if there is a block hook function registered, it calls it. If the blocked function is the currently running one, it invokes a rescheduling.

**Parameters**

|               |                                 |
|---------------|---------------------------------|
| <i>taskId</i> | : ID of the task to be blocked. |
|---------------|---------------------------------|

**Returns**

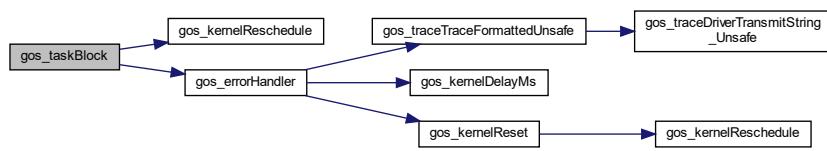
Result of task blocking.

**Return values**

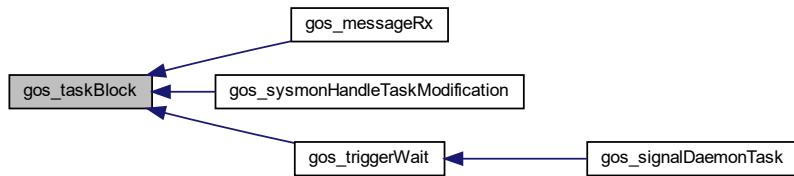
|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>GOS_SUCESS</i> | : Task blocked successfully.                      |
| <i>GOS_ERROR</i>  | : Task ID is invalid, or task state is not ready. |

Definition at line 507 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.14.4.24 gos\_result\_t gos\_taskDelete( gos\_tid\_t taskId )**

Deletes the given task from the scheduling array.

Checks the given task ID and its state, modifies it to zombie, and if there is a delete hook function registered, it calls it.

**Parameters**

|               |                                 |
|---------------|---------------------------------|
| <i>taskId</i> | : ID of the task to be deleted. |
|---------------|---------------------------------|

**Returns**

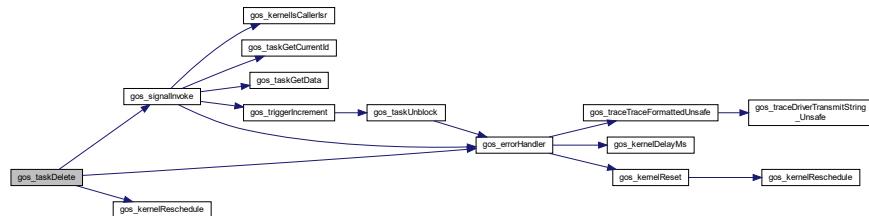
Result of deletion.

## Return values

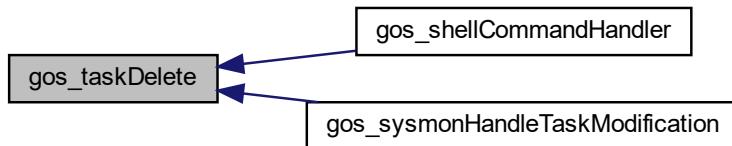
|                    |                              |
|--------------------|------------------------------|
| <b>GOS_SUCCESS</b> | : Task deleted successfully. |
| <b>GOS_ERROR</b>   | : Task is already a zombie.  |

Definition at line 639 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.14.4.25 gos\_result\_t gos\_taskGetCurrentId( gos\_tid\_t \* taskId )

Returns the ID of the currently running task.

Returns the ID of the currently running task.

## Parameters

*taskId* : Pointer to a task ID variable to store the current task ID.

### Returns

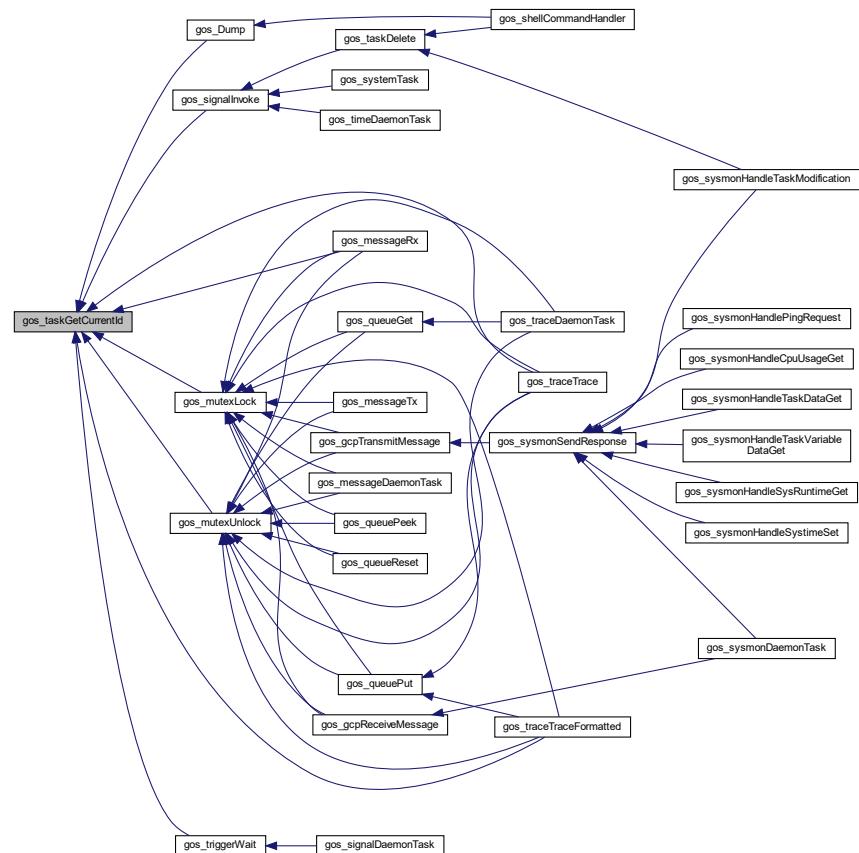
Result of current task ID get.

### Return values

|                    |                                          |
|--------------------|------------------------------------------|
| <i>GOS_SUCCESS</i> | : Current task ID returned successfully. |
| <i>GOS_ERROR</i>   | : Task ID pointer is NULL.               |

Definition at line 1074 of file gos\_task.c.

Here is the caller graph for this function:



#### 4.14.4.26 *gos\_result\_t gos\_taskGetData( gos\_tid\_t taskId, gos\_taskDescriptor\_t \*taskData )*

Returns the task data of the given task.

Based on the task ID, it copies the content of the internal task descriptor array element to the given task descriptor.

### Parameters

|                 |                                                            |
|-----------------|------------------------------------------------------------|
| <i>taskId</i>   | : ID of the task to get the data of.                       |
| <i>taskData</i> | : Pointer to the task descriptor to save the task data in. |

### Returns

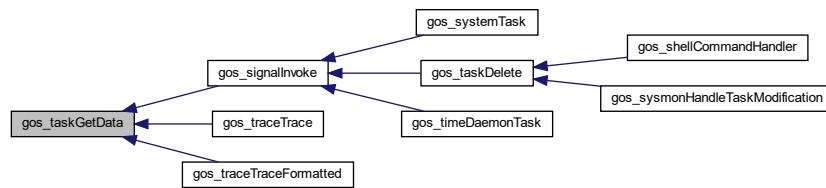
Result of task data get.

**Return values**

|                    |                                                 |
|--------------------|-------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Task data copied successfully.                |
| <i>GOS_ERROR</i>   | : Invalid task ID or task data pointer is NULL. |

Definition at line 1102 of file gos\_task.c.

Here is the caller graph for this function:

**4.14.4.27 *gos\_result\_t gos\_taskGetDataByIndex( u16\_t taskIndex, gos\_taskDescriptor\_t \*taskData )***

Returns the task data of the given task.

Based on the task index, it copies the content of the internal task descriptor array element to the given task descriptor.

**Parameters**

|                  |                                                            |
|------------------|------------------------------------------------------------|
| <i>taskIndex</i> | : Index of the task to get the data of.                    |
| <i>taskData</i>  | : Pointer to the task descriptor to save the task data in. |

**Returns**

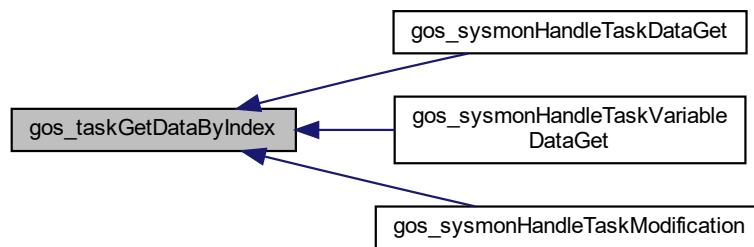
Result of task data get.

**Return values**

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Task data copied successfully.                   |
| <i>GOS_ERROR</i>   | : Invalid task index or task data pointer is NULL. |

Definition at line 1135 of file gos\_task.c.

Here is the caller graph for this function:



#### 4.14.4.28 gos\_result\_t gos\_taskGetId( gos\_taskName\_t taskName, gos\_tid\_t \* taskId )

Gets the task ID of the task with the given name.

This function loops through the internal task array and tries to find the given task name to get the corresponding task ID.

##### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>taskName</i> | : Name of the task (string).                              |
| <i>taskId</i>   | : Pointer to a task ID variable to store the returned ID. |

##### Returns

Success of task ID get.

##### Return values

|                    |                               |
|--------------------|-------------------------------|
| <i>GOS_SUCCESS</i> | : Task ID found successfully. |
| <i>GOS_ERROR</i>   | : Task name not found.        |

Definition at line 1043 of file gos\_task.c.

Here is the caller graph for this function:



#### 4.14.4.29 gos\_result\_t gos\_taskGetName( gos\_tid\_t taskId, gos\_taskName\_t taskName )

Gets the task name of the task with the given ID.

Copies the task name corresponding with the given task ID to the task name variable.

##### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>taskId</i>   | : Pointer to a task ID variable to store the returned ID. |
| <i>taskName</i> | : Task name pointer to store the returned task name.      |

##### Returns

Success of task name get.

##### Return values

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Task name found successfully.                  |
| <i>GOS_ERROR</i>   | : Invalid task ID or task name variable is NULL. |

Definition at line 1012 of file gos\_task.c.

#### 4.14.4.30 gos\_result\_t gos\_taskGetNumber( u16\_t \* pTaskNum )

Returns the number of registered tasks.

Loops through the internal task array and counts the entries where a task function is registered.

**Parameters**

|                 |                                          |
|-----------------|------------------------------------------|
| <i>pTaskNum</i> | : Variable to store the number of tasks. |
|-----------------|------------------------------------------|

**Returns**

Success of task number counting.

**Return values**

|                    |                                     |
|--------------------|-------------------------------------|
| <i>GOS_SUCCESS</i> | : Task number counted successfully. |
| <i>GOS_ERROR</i>   | : Target variable is NULL pointer.  |

Definition at line 1166 of file gos\_task.c.

**4.14.4.31 gos\_result\_t gos\_taskGetOriginalPriority ( gos\_tid\_t taskId, gos\_taskPrio\_t \* taskPriority )**

Gets the original priority of the given task.

Checks the given parameters and saves the original priority in the given variable.

**Parameters**

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| <i>taskId</i>       | : ID of the task to get the priority of.                   |
| <i>taskPriority</i> | : Pointer to a priority variable to store the priority in. |

**Returns**

Result of priority getting.

**Return values**

|                    |                                                 |
|--------------------|-------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Original priority getting successfully.       |
| <i>GOS_ERROR</i>   | : Invalid task ID or priority variable is NULL. |

Definition at line 842 of file gos\_task.c.

**4.14.4.32 gos\_result\_t gos\_taskGetPriority ( gos\_tid\_t taskId, gos\_taskPrio\_t \* taskPriority )**

Gets the current priority of the given task.

Checks the given parameters and saves the current priority in the given variable.

**Parameters**

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| <i>taskId</i>       | : ID of the task to get the priority of.                   |
| <i>taskPriority</i> | : Pointer to a priority variable to store the priority in. |

**Returns**

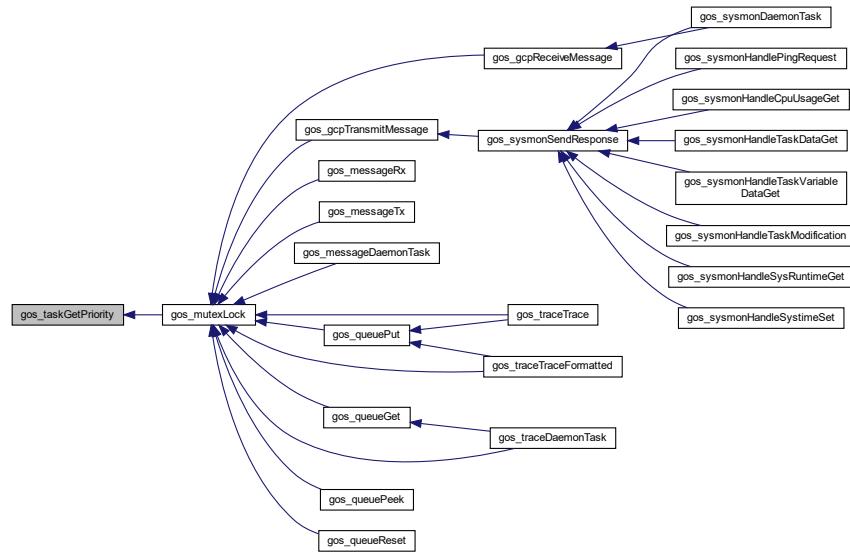
Result of priority getting.

**Return values**

|                    |                                                 |
|--------------------|-------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Current priority getting successfully.        |
| <i>GOS_ERROR</i>   | : Invalid task ID or priority variable is NULL. |

Definition at line 811 of file gos\_task.c.

Here is the caller graph for this function:



#### 4.14.4.33 gos\_result\_t gos\_taskGetPrivileges ( gos\_tid\_t taskId, gos\_taskPrivilegeLevel\_t \* privileges )

Gets the privileges of the given task.

Returns the privilege flags of the given task.

##### Parameters

|                         |                                            |
|-------------------------|--------------------------------------------|
| <code>taskId</code>     | : ID of the task to get the privileges of. |
| <code>privileges</code> | : Variable to store the privilege flags.   |

##### Returns

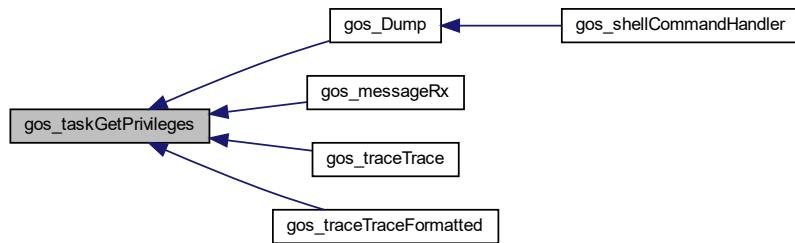
Result of privilege getting.

##### Return values

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Privileges get successful.                             |
| <code>GOS_ERROR</code>   | : Invalid task ID or privilege variable is NULL pointer. |

Definition at line 963 of file `gos_task.c`.

Here is the caller graph for this function:



#### 4.14.4.34 `gos_result_t gos_taskRegister ( gos_taskDescriptor_t * taskDescriptor, gos_tid_t * taskId )`

This function registers a task for scheduling.

Checks the task descriptor parameters and then tries to find the next empty slot in the internal task array. When it is found, it registers the task in that slot.

##### Parameters

|                             |                                                            |
|-----------------------------|------------------------------------------------------------|
| <code>taskDescriptor</code> | : Pointer to a task descriptor structure.                  |
| <code>taskId</code>         | : Pointer to a variable to hold to assigned task ID value. |

##### Returns

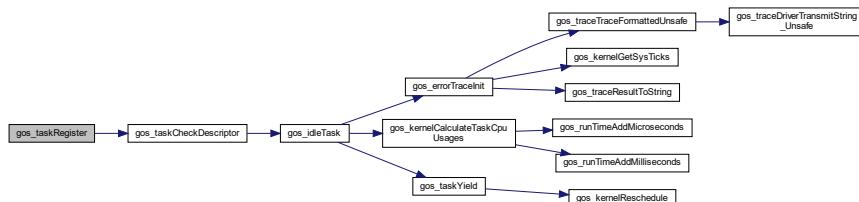
Result of task registration.

##### Return values

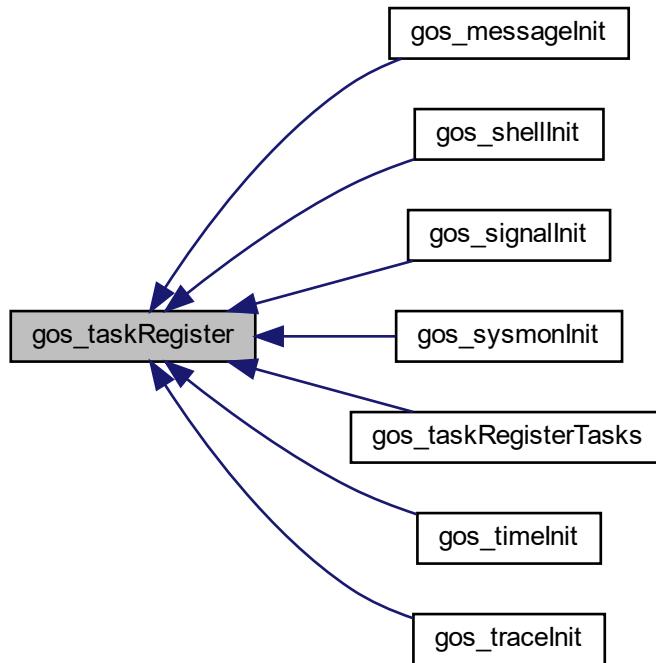
|                          |                                                                                                                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Task registered successfully.                                                                                                                                                   |
| <code>GOS_ERROR</code>   | : Invalid task descriptor (NULL function pointer, invalid priority level, invalid stack size, idle task registration, or stack size is not 4-byte-aligned) or task array is full. |

Definition at line 159 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.35 `gos_result_t gos_taskRegisterTasks ( gos_taskDescriptor_t * taskDescriptors, u16_t arraySize )`

This function registers an array of tasks for scheduling.

Checks the task descriptor array pointer and registers the tasks one by one.

##### Parameters

|                        |                                                 |
|------------------------|-------------------------------------------------|
| <i>taskDescriptors</i> | : Pointer to a task descriptor structure array. |
| <i>arraySize</i>       | : Size of the array in bytes.                   |

##### Returns

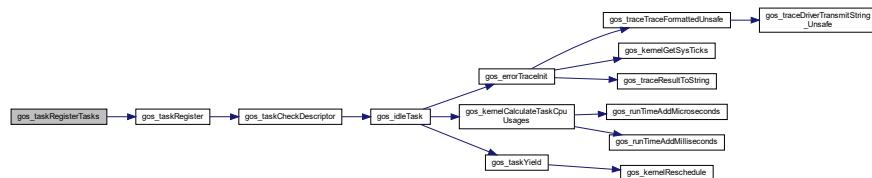
Result of task registration.

##### Return values

|                    |                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Tasks registered successfully.                                                                                                                                                                               |
| <i>GOS_ERROR</i>   | : Invalid task descriptor (NULL function pointer, invalid priority level, invalid stack size, idle task registration, or stack size is not 4-byte-aligned) in one of the array elements or task array is full. |

Definition at line 111 of file gos\_task.c.

Here is the call graph for this function:



#### 4.14.4.36 gos\_result\_t gos\_taskRemovePrivilege ( gos\_tid\_t taskId, gos\_taskPrivilegeLevel\_t privileges )

Removes the given privileges from the given task.

Checks the caller task if it has the privilege to modify task privileges and if so, it removes the given privileges from the given task.

##### Parameters

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <i>taskId</i>     | : ID of the task to remove the privileges from. |
| <i>privileges</i> | : Privileges to be removed.                     |

##### Returns

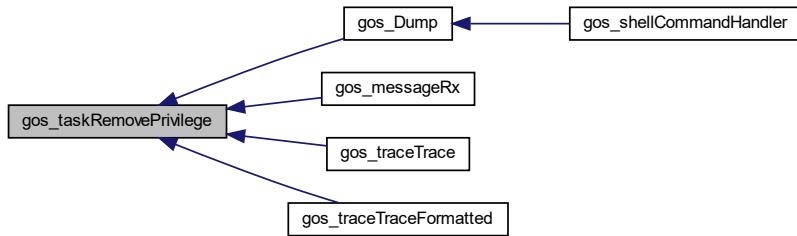
Result of privilege removing.

##### Return values

|                    |                                                                                    |
|--------------------|------------------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Privileges removed successfully.                                                 |
| <i>GOS_ERROR</i>   | : Invalid task ID or caller does not have the privilege to modify task privileges. |

Definition at line 903 of file gos\_task.c.

Here is the caller graph for this function:



#### 4.14.4.37 gos\_result\_t gos\_taskResume ( gos\_tid\_t taskId )

Resumes the given task.

Checks the given task ID and its state, modified it to ready, and if there is a resume hook function registered, it calls it.

**Parameters**

|               |                                 |
|---------------|---------------------------------|
| <i>taskId</i> | : ID of the task to be resumed. |
|---------------|---------------------------------|

**Returns**

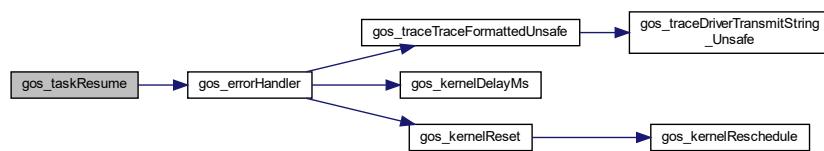
Result of task resumption.

**Return values**

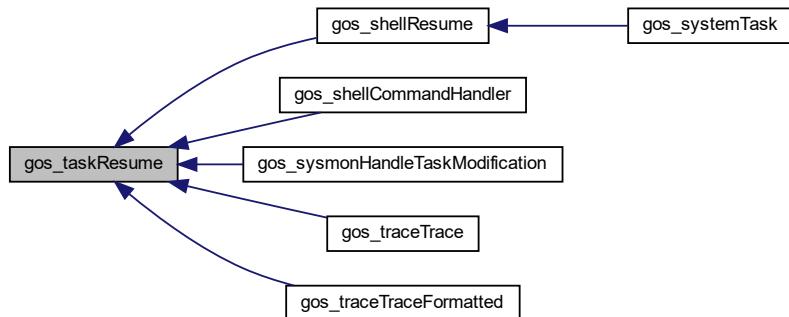
|                   |                                                 |
|-------------------|-------------------------------------------------|
| <i>GOS_SUCESS</i> | : Task resumed successfully.                    |
| <i>GOS_ERROR</i>  | : Task ID is invalid, or task is not suspended. |

Definition at line 456 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.38 `gos_result_t gos_taskSetOriginalPriority ( gos_tid_t taskId, gos_taskPrio_t taskPriority )`

Sets the original priority of the given task to the given value (for permanent change).

Checks the given parameters and sets the original priority of the given task.

**Parameters**

|               |                                             |
|---------------|---------------------------------------------|
| <i>taskId</i> | : ID of the task to change the priority of. |
|---------------|---------------------------------------------|

|                     |                              |
|---------------------|------------------------------|
| <i>taskPriority</i> | : The desired task priority. |
|---------------------|------------------------------|

**Returns**

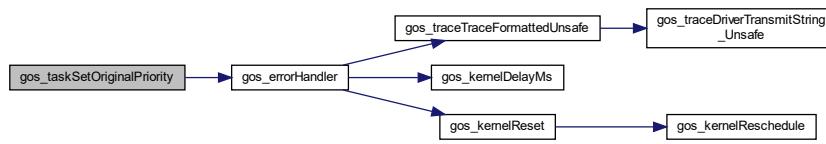
Result of priority change.

**Return values**

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>GOS_SUCCESS</i> | : Original priority changed successfully. |
| <i>GOS_ERROR</i>   | : Invalid task ID or priority.            |

Definition at line 765 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.39 `gos_result_t gos_taskSetPriority( gos_tid_t taskId, gos_taskPrio_t taskPriority )`

Sets the current priority of the given task to the given value (for temporary change).

Checks the given parameters and sets the current priority of the given task.

**Parameters**

|                     |                                             |
|---------------------|---------------------------------------------|
| <i>taskId</i>       | : ID of the task to change the priority of. |
| <i>taskPriority</i> | : The desired task priority.                |

**Returns**

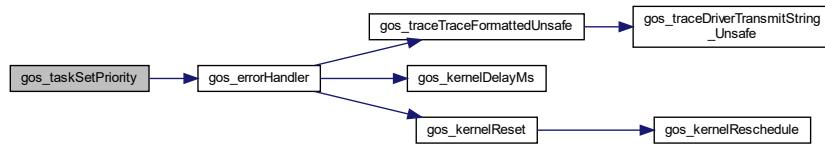
Result of priority change.

**Return values**

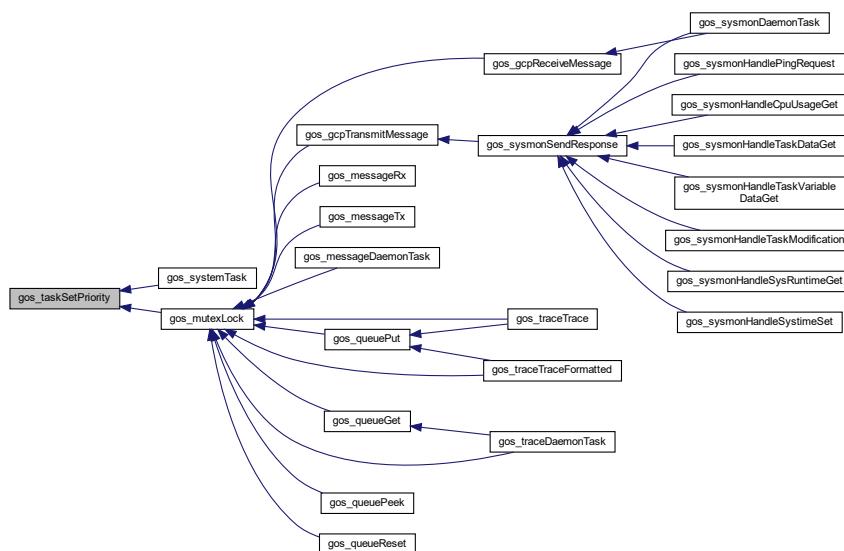
|                          |                                          |
|--------------------------|------------------------------------------|
| <code>GOS_SUCCESS</code> | : Current priority changed successfully. |
| <code>GOS_ERROR</code>   | : Invalid task ID or priority.           |

Definition at line 719 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.40 `gos_result_t gos_taskSetPrivileges( gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges )`

Sets the given privileges for the given task.

Checks the caller task if it has the privilege to modify task privileges and if so, it sets the given privileges for the given task.

##### Parameters

|                         |                                             |
|-------------------------|---------------------------------------------|
| <code>taskId</code>     | : ID of the task to set the privileges for. |
| <code>privileges</code> | : Privileges to be set.                     |

##### Returns

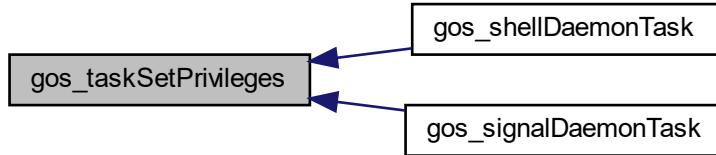
Result of privilege setting.

## Return values

|                    |                                                                                    |
|--------------------|------------------------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Privileges set successfully.                                                     |
| <i>GOS_ERROR</i>   | : Invalid task ID or caller does not have the privilege to modify task privileges. |

Definition at line 933 of file gos\_task.c.

Here is the caller graph for this function:



#### 4.14.4.41 `gos_result_t gos_taskSleep( gos_taskSleepTick_t sleepTicks )`

Sends the current task to sleeping state.

Checks the current task and its state, modifies it to sleeping, and if there is a sleep hook function registered, it calls it. Then, it invokes a rescheduling.

## Parameters

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| <i>sleepTicks</i> | : Minimum number of ticks until the task should remain in sleeping state. |
|-------------------|---------------------------------------------------------------------------|

## Returns

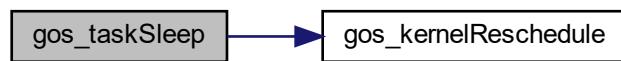
Result of task sleeping.

## Return values

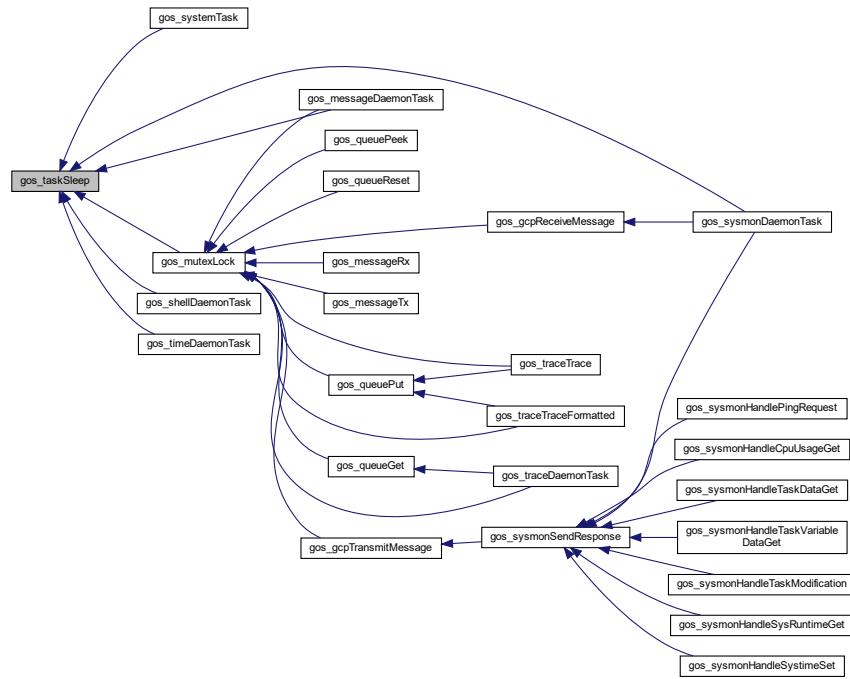
|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Task successfully sent to sleeping state.                  |
| <i>GOS_ERROR</i>   | : Function called from idle task or task state is not ready. |

Definition at line 282 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.42 gos\_result\_t gos\_taskSubscribeDeleteSignal ( void\_t(\*)(u16\_t) deleteSignalHandler )

Subscribes the given handler to the task delete signal.

Subscribes the given handler to the task delete signal.

##### Parameters

|                                  |                                   |
|----------------------------------|-----------------------------------|
| <code>deleteSignalHandler</code> | : Delete signal handler function. |
|----------------------------------|-----------------------------------|

##### Returns

Result of subscription.

##### Return values

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Subscription successful.                       |
| <code>GOS_ERROR</code>   | : Subscription failed or signal handler is NULL. |

#### 4.14.4.43 gos\_result\_t gos\_taskSuspend ( gos\_tid\_t taskId )

Sends the given task to suspended state.

Checks the given task ID and its state, modified it to suspended, and if there is a suspend hook function registered, it calls it. If the suspended function is the currently running one, it invokes a rescheduling.

**Parameters**

|               |                                   |
|---------------|-----------------------------------|
| <i>taskId</i> | : ID of the task to be suspended. |
|---------------|-----------------------------------|

**Returns**

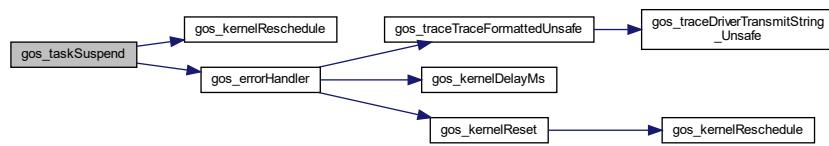
Result of task suspension.

**Return values**

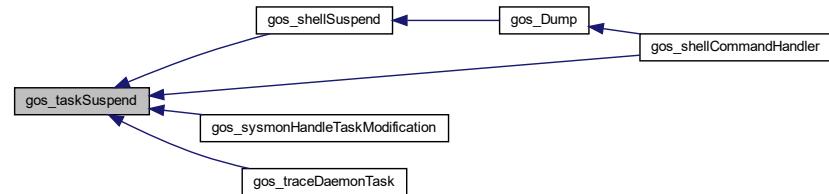
|                   |                                                               |
|-------------------|---------------------------------------------------------------|
| <i>GOS_SUCESS</i> | : Task suspended successfully.                                |
| <i>GOS_ERROR</i>  | : Task ID is invalid, or task state is not ready or sleeping. |

Definition at line 382 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.44 `gos_result_t gos_taskUnblock( gos_tid_t taskId )`

Unblocks the given task.

Checks the given task ID and its state, modified it to ready, and if there is an unblock hook function registered, it calls it.

**Parameters**

|               |                                   |
|---------------|-----------------------------------|
| <i>taskId</i> | : ID of the task to be unblocked. |
|---------------|-----------------------------------|

**Returns**

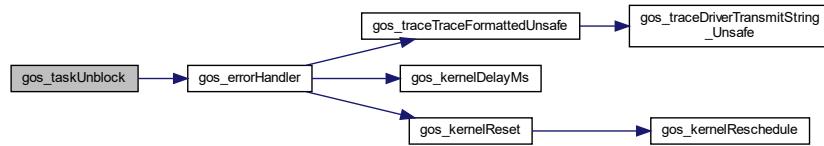
Result of task unblocking.

## Return values

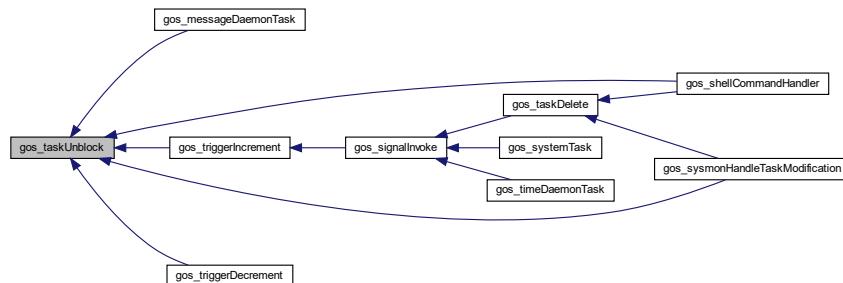
|                   |                                               |
|-------------------|-----------------------------------------------|
| <i>GOS_SUCESS</i> | : Task unblocked successfully.                |
| <i>GOS_ERROR</i>  | : Task ID is invalid, or task is not blocked. |

Definition at line 582 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.45 `gos_result_t gos_taskWakeup( gos_tid_t taskId )`

Wakes up the given task.

Checks the current task and its state, modifies it to ready, and if there is a wake-up hook function registered, it calls it.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>taskId</i> | : ID of the task to be waken up. |
|---------------|----------------------------------|

## Returns

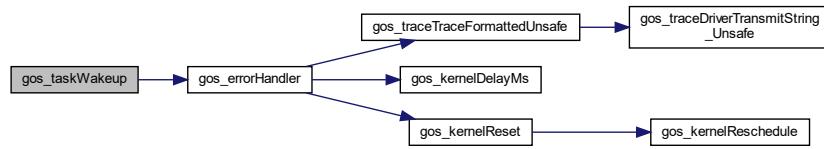
Result of task wake-up.

## Return values

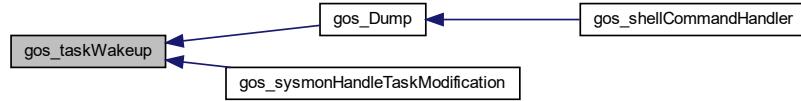
|                    |                                                |
|--------------------|------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Task waken up successfully.                  |
| <i>GOS_ERROR</i>   | : Task ID is invalid, or task is not sleeping. |

Definition at line 331 of file gos\_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.46 `gos_result_t gos_taskYield( void_t )`

Yields the current task.

Invokes rescheduling.

##### Returns

Result of task yield.

##### Return values

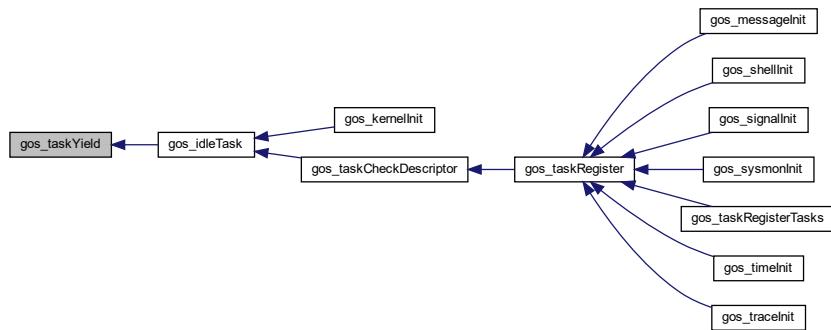
|                          |                     |
|--------------------------|---------------------|
| <code>GOS_SUCCESS</code> | : Yield successful. |
|--------------------------|---------------------|

Definition at line 995 of file `gos_task.c`.

Here is the call graph for this function:



Here is the caller graph for this function:

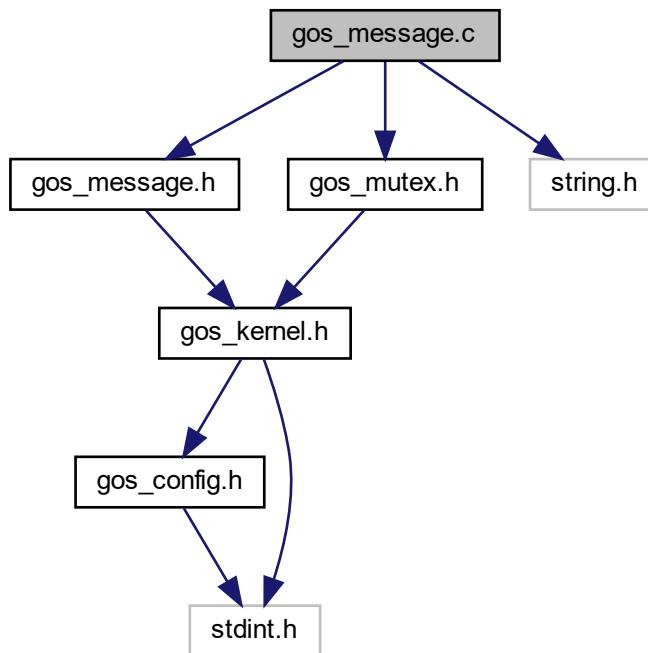


## 4.15 gos\_message.c File Reference

GOS message service source.

```
#include <gos_message.h>
#include <gos_mutex.h>
#include <string.h>
```

Include dependency graph for gos\_message.c:



## Data Structures

- struct [gos\\_messageWaiterDesc\\_t](#)

## Macros

- #define [GOS\\_MESSAGE\\_DAEMON\\_POLL\\_TIME\\_MS](#) (50u)

## Functions

- [GOS\\_STATIC void\\_t gos\\_messageDaemonTask \(void\\_t\)](#)  
*Message daemon task.*
- [gos\\_result\\_t gos\\_messageInit \(void\\_t\)](#)  
*Initializes the message service.*
- [GOS\\_INLINE gos\\_result\\_t gos\\_messageRx \(gos\\_messageId\\_t \\*messagelIdArray, gos\\_message\\_t \\*target, gos\\_messageTimeout\\_t tmo\)](#)  
*Receives the selected messages.*
- [GOS\\_INLINE gos\\_result\\_t gos\\_messageTx \(gos\\_message\\_t \\*message\)](#)  
*Transmits a message.*

## Variables

- [GOS\\_STATIC gos\\_message\\_t messageArray \[CFG\\_MESSAGE\\_MAX\\_NUMBER\]](#)
- [GOS\\_STATIC gos\\_messageWaiterDesc\\_t messageWaiterArray \[CFG\\_MESSAGE\\_MAX\\_WAITERS\]](#)
- [GOS\\_STATIC gos\\_tid\\_t messageDaemonTaskId](#)
- [GOS\\_STATIC gos\\_messageIndex\\_t nextMessageIndex](#)
- [GOS\\_STATIC gos\\_messageWaiterIndex\\_t nextWaiterIndex](#)
- [GOS\\_STATIC gos\\_mutex\\_t messageMutex](#)
- [GOS\\_STATIC gos\\_taskDescriptor\\_t messageDaemonTaskDesc](#)

### 4.15.1 Detailed Description

GOS message service source.

#### Author

Ahmed Gazar

#### Date

2023-11-01

#### Version

1.9

For a more detailed description of this service, please refer to [gos\\_message.h](#)

Definition in file [gos\\_message.c](#).

## 4.15.2 Macro Definition Documentation

### 4.15.2.1 `#define GOS_MESSAGE_DAEMON_POLL_TIME_MS ( 50u )`

Message daemon poll time in [ms].

Definition at line 78 of file gos\_message.c.

## 4.15.3 Function Documentation

### 4.15.3.1 `GOS_STATIC void_t gos_messageDaemonTask ( void_t )`

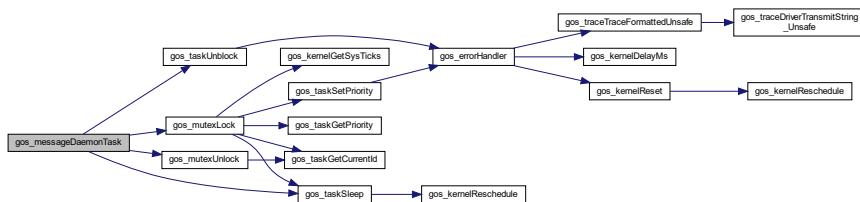
Message daemon task.

This task is responsible for message transfer between tasks. It loops through the internal message array and the waiter task array and if a message and a waiter matches, it copies the message to the target buffer and unblocks the previously blocked task.

#### Returns

Definition at line 375 of file gos\_message.c.

Here is the call graph for this function:



### 4.15.3.2 `gos_result_t gos_messageInit ( void_t )`

Initializes the message service.

Initializes the internal message and waiter arrays and registers the message daemon task.

#### Returns

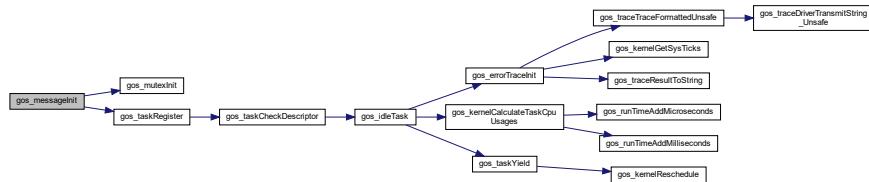
Result of initialization.

#### Return values

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>GOS_SUCCESS</code> | : Initialization successful.               |
| <code>GOS_ERROR</code>   | : Message daemon task registration failed. |

Definition at line 149 of file gos\_message.c.

Here is the call graph for this function:



#### 4.15.3.3 GOS\_INLINE gos\_result\_t gos\_messageRx ( gos\_messageId\_t \* messageIdArray, gos\_message\_t \* target, gos\_messageTimeout\_t tmo )

Receives the selected messages.

Based on the selected message IDs, this function receives the first available message. Until the message is received, the caller task is put to blocked state. The caller will be unblocked if the message is received or the timeout elapsed.

##### Parameters

|                       |                                                                                                                                                   |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>messageIdArray</i> | : Array of messages IDs the function should receive. Must be by a 0 element!                                                                      |
| <i>target</i>         | : Pointer to the target message structure. Received data will be placed here.                                                                     |
| <i>tmo</i>            | : Timeout value in [ms]. The resolution of timeout is 10ms! Do not use endless timeout if it is not guaranteed that the message will be received! |

##### Returns

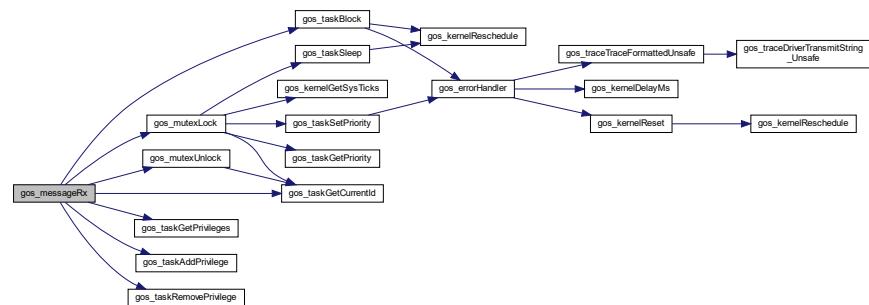
Result of message reception.

##### Return values

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Reception successful, data placed in the target structure. |
| <i>GOS_ERROR</i>   | : Reception failed because of invalid parameters or timeout. |

Definition at line 192 of file gos\_message.c.

Here is the call graph for this function:



#### 4.15.3.4 GOS\_INLINE gos\_result\_t gos\_messageTx ( gos\_message\_t \* message )

Transmits a message.

Copies the message in the internal message array for the message daemon to transfer it to the recipient task.

**Parameters**

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>message</i> | : Pointer to the message structure to be transmitted. |
|----------------|-------------------------------------------------------|

**Returns**

Result of message transmission.

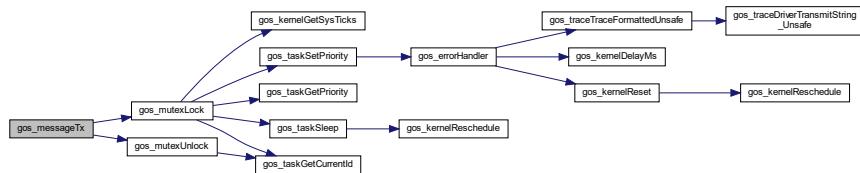
**Return values**

|                    |                                                             |
|--------------------|-------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Message transmission initiated successfully.              |
| <i>GOS_ERROR</i>   | : Invalid message pointer or data or message array is full. |

Function code.

Definition at line 305 of file gos\_message.c.

Here is the call graph for this function:



#### 4.15.4 Variable Documentation

##### 4.15.4.1 GOS\_STATIC gos\_message\_t messageArray[CFG\_MESSAGE\_MAX\_NUMBER]

Internal message array.

Definition at line 102 of file gos\_message.c.

##### 4.15.4.2 GOS\_STATIC gos\_taskDescriptor\_t messageDaemonTaskDesc

**Initial value:**

```
=
{
 .taskFunction = gos_messageDaemonTask,
 .taskName = "gos_message_daemon",
 .taskPriority = CFG_TASK_MESSAGE_DAEMON_PRIO,
 .taskStackSize = CFG_TASK_MESSAGE_DAEMON_STACK,
 .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL
}
```

Message daemon task descriptor.

Definition at line 137 of file gos\_message.c.

##### 4.15.4.3 GOS\_STATIC gos\_tid\_t messageDaemonTaskId

Message daemon task ID.

Definition at line 112 of file gos\_message.c.

#### 4.15.4.4 GOS\_STATIC gos\_mutex\_t messageMutex

Message mutex to protect the internal arrays as shared resources.

Definition at line 127 of file gos\_message.c.

#### 4.15.4.5 GOS\_STATIC gos\_messageWaiterDesc\_t messageWaiterArray[CFG\_MESSAGE\_MAX\_WAITERS]

Internal message waiter array.

Definition at line 107 of file gos\_message.c.

#### 4.15.4.6 GOS\_STATIC gos\_messageIndex\_t nextMessageIndex

Index of the next message slot (for circular buffer).

Definition at line 117 of file gos\_message.c.

#### 4.15.4.7 GOS\_STATIC gos\_messageWaiterIndex\_t nextWaiterIndex

Index of the next waiter slot (for circular buffer).

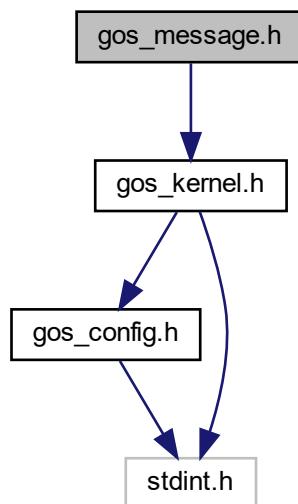
Definition at line 122 of file gos\_message.c.

## 4.16 gos\_message.h File Reference

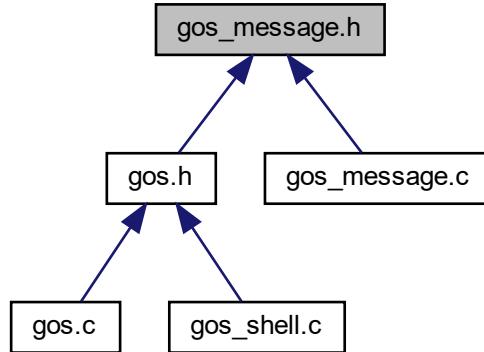
GOS message service header.

```
#include <gos_kernel.h>
```

Include dependency graph for gos\_message.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [gos\\_message\\_t](#)

## Macros

- #define [GOS\\_MESSAGE\\_ENDLESS\\_TMO](#) ( UINT16\_MAX )
- #define [GOS\\_MESSAGE\\_INVALID\\_ID](#) ( UINT16\_MAX )

## Typedefs

- typedef u16\_t [gos\\_messageId\\_t](#)
- typedef u16\_t [gos\\_messageTimeout\\_t](#)
- typedef u8\_t [gos\\_messageSize\\_t](#)  
*Message size type.*
- typedef u8\_t [gos\\_messageIndex\\_t](#)  
*Message index type.*
- typedef u8\_t [gos\\_messageWaiterIndex\\_t](#)  
*Message waiter index type.*
- typedef u8\_t [gos\\_messageIdIndex\\_t](#)  
*Message ID index type.*

## Functions

- [gos\\_result\\_t gos\\_messageInit \(void\\_t\)](#)  
*Initializes the message service.*
- [gos\\_result\\_t gos\\_messageRx \(gos\\_messageId\\_t \\*messageIdArray, gos\\_message\\_t \\*target, gos\\_messageTimeout\\_t tmo\)](#)  
*Receives the selected messages.*
- [gos\\_result\\_t gos\\_messageTx \(gos\\_message\\_t \\*message\)](#)  
*Transmits a message.*

#### 4.16.1 Detailed Description

GOS message service header.

Author

Ahmed Gazar

Date

2022-11-15

Version

1.2

Message service is a way of inter-task communication provided by the operating system. With the use of messages, data can be passed between different tasks. The data sent through the message service is only limited in size which is a configuration parameter. Message passing is handled by the background daemon task thus ensuring that big data transfers do not block the system. Messages are first copied into an internal message array and whenever a waiter and a message is matched, it gets copied into the target buffer. If a message is not received, it will be stuck in the internal buffer and it causes the message service to halt if the internal circular buffer gets back to the position of the stuck message. Reception of messages can happen with a given timeout or with an endless timeout. Either way the caller task will go to blocked state until the message is received or the timeout elapses, so it will not be scheduled in the meantime. Each message contains a message ID. When receiving messages, the task can define a list of IDs for reception as a filter. This way one task can receive more than one message but a message can only be received by one task.

Definition in file [gos\\_message.h](#).

#### 4.16.2 Macro Definition Documentation

##### 4.16.2.1 #define GOS\_MESSAGE\_ENDLESS\_TMO ( UINT16\_MAX )

Endless timeout.

Definition at line 78 of file gos\_message.h.

##### 4.16.2.2 #define GOS\_MESSAGE\_INVALID\_ID ( UINT16\_MAX )

Invalid message ID.

Definition at line 83 of file gos\_message.h.

#### 4.16.3 Typedef Documentation

##### 4.16.3.1 typedef u16\_t gos\_messageId\_t

Message identifier type.

Definition at line 91 of file gos\_message.h.

##### 4.16.3.2 typedef u16\_t gos\_messageTimeout\_t

Message timeout type.

Definition at line 96 of file gos\_message.h.

## 4.16.4 Function Documentation

### 4.16.4.1 gos\_result\_t gos\_messageInit( void\_t )

Initializes the message service.

Initializes the internal message and waiter arrays and registers the message daemon task.

#### Returns

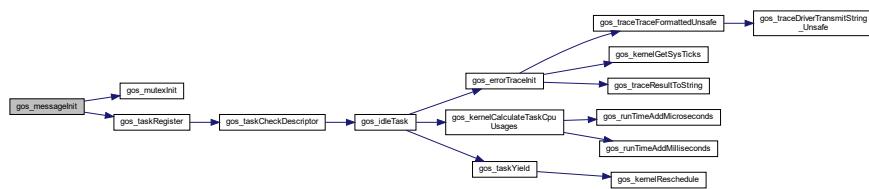
Result of initialization.

#### Return values

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>GOS_SUCCESS</i> | : Initialization successful.               |
| <i>GOS_ERROR</i>   | : Message daemon task registration failed. |

Definition at line 149 of file gos\_message.c.

Here is the call graph for this function:



### 4.16.4.2 gos\_result\_t gos\_messageRx( gos\_messageId\_t \* messageIdArray, gos\_message\_t \* target, gos\_messageTimeout\_t tmo )

Receives the selected messages.

Based on the selected message IDs, this function receives the first available message. Until the message is received, the caller task is put to blocked state. The caller will be unblocked if the message is received or the timeout elapsed.

#### Parameters

|                       |                                                                                                                                                   |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>messageIdArray</i> | : Array of messages IDs the function should receive. Must be by a 0 element!                                                                      |
| <i>target</i>         | : Pointer to the target message structure. Received data will be placed here.                                                                     |
| <i>tmo</i>            | : Timeout value in [ms]. The resolution of timeout is 10ms! Do not use endless timeout if it is not guaranteed that the message will be received! |

#### Returns

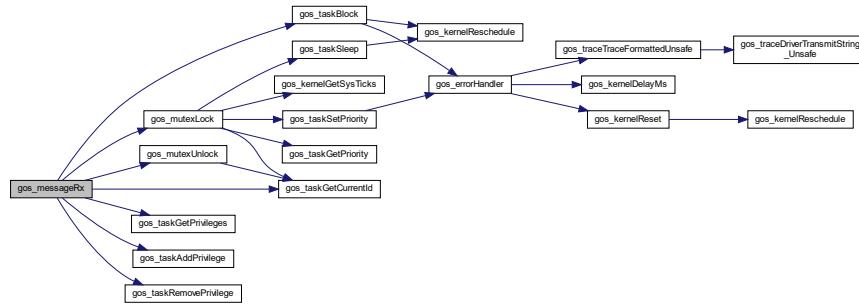
Result of message reception.

#### Return values

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Reception successful, data placed in the target structure. |
| <i>GOS_ERROR</i>   | : Reception failed because of invalid parameters or timeout. |

Definition at line 192 of file gos\_message.c.

Here is the call graph for this function:



#### 4.16.4.3 `gos_result_t gos_messageTx( gos_message_t * message )`

Transmits a message.

Copies the message in the internal message array for the message daemon to transfer it to the recipient task.

##### Parameters

|                      |                                                       |
|----------------------|-------------------------------------------------------|
| <code>message</code> | : Pointer to the message structure to be transmitted. |
|----------------------|-------------------------------------------------------|

##### Returns

Result of message transmission.

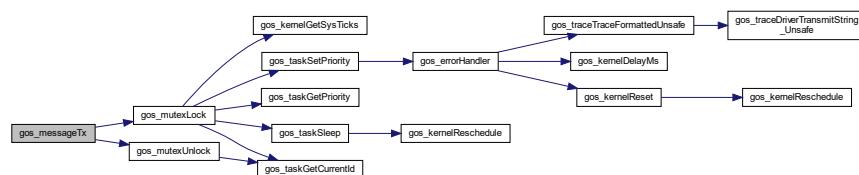
##### Return values

|                          |                                                             |
|--------------------------|-------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Message transmission initiated successfully.              |
| <code>GOS_ERROR</code>   | : Invalid message pointer or data or message array is full. |

Function code.

Definition at line 305 of file `gos_message.c`.

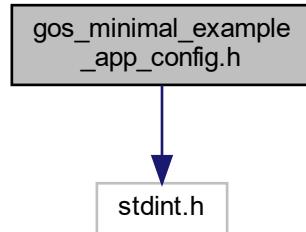
Here is the call graph for this function:



## 4.17 `gos_minimal_example_app_config.h` File Reference

GOS minimal example application configuration header.

```
#include <stdint.h>
Include dependency graph for gos_minimal_example_app_config.h:
```



## Macros

- #define GOS\_CFG\_OVERCONFIG
- #define ARM\_CORTEX\_M4 ( 1 )
- #define CFG\_TARGET\_CPU ( ARM\_CORTEX\_M4 )
- #define CFG\_SCHED\_COOPERATIVE ( 0 )
- #define CFG\_USE\_PRIO\_INHERITANCE ( 1 )
- #define CFG\_TASK\_MAX\_NAME\_LENGTH ( 32 )
- #define CFG\_TASK\_MAX\_NUMBER ( 16 )
- #define CFG\_TASK\_MIN\_STACK\_SIZE ( 0x200 )
- #define CFG\_TASK\_MAX\_STACK\_SIZE ( 0x4000 )
- #define CFG\_IDLE\_TASK\_STACK\_SIZE ( 0x400 )
- #define CFG\_SYSTEM\_TASK\_STACK\_SIZE ( 0x400 )
- #define CFG\_TASK\_SIGNAL\_DAEMON\_STACK ( 0x300 )
- #define CFG\_TASK\_PROC\_DAEMON\_STACK ( 0x300 )
- #define CFG\_TASK\_TIME\_DAEMON\_STACK ( 0x300 )
- #define CFG\_TASK\_MESSAGE\_DAEMON\_STACK ( 0x300 )
- #define CFG\_TASK\_SHELL\_DAEMON\_STACK ( 0x300 )
- #define CFG\_TASK\_TRACE\_DAEMON\_STACK ( 0x400 )
- #define CFG\_TASK\_SYSMON\_DAEMON\_STACK ( 0x300 )
- #define CFG\_TASK\_TRACE\_DAEMON\_PRIO ( 193 )
- #define CFG\_TASK\_MESSAGE\_DAEMON\_PRIO ( 198 )
- #define CFG\_TASK\_SIGNAL\_DAEMON\_PRIO ( 197 )
- #define CFG\_TASK\_PROC\_DAEMON\_PRIO ( 194 )
- #define CFG\_TASK\_SHELL\_DAEMON\_PRIO ( 192 )
- #define CFG\_TASK\_TIME\_DAEMON\_PRIO ( 196 )
- #define CFG\_TASK\_SYS\_PRIO ( 195 )
- #define CFG\_TASK\_SYSMON\_DAEMON\_PRIO ( 40 )
- #define CFG\_PROC\_USE\_SERVICE ( 0 )
- #define CFG\_PROC\_MAX\_PRIO\_LEVELS ( UINT8\_MAX )
- #define CFG\_PROC\_IDLE\_PRIO ( CFG\_PROC\_MAX\_PRIO\_LEVELS )
- #define CFG\_PROC\_MAX\_NAME\_LENGTH ( 24 )
- #define CFG\_PROC\_MAX\_NUMBER ( 4 )
- #define CFG\_QUEUE\_MAX\_NUMBER ( 4 )
- #define CFG\_QUEUE\_MAX\_ELEMENTS ( 40 )
- #define CFG\_QUEUE\_MAX\_LENGTH ( 200 )

- `#define CFG_QUEUE_USE_NAME ( 1 )`
- `#define CFG_QUEUE_MAX_NAME_LENGTH ( 24 )`
- `#define CFG_SIGNAL_MAX_NUMBER ( 6 )`
- `#define CFG_SIGNAL_MAX_SUBSCRIBERS ( 6 )`
- `#define CFG_MESSAGE_MAX_NUMBER ( 4 )`
- `#define CFG_MESSAGE_MAX_LENGTH ( 80 )`
- `#define CFG_MESSAGE_MAX_WAITERS ( 10 )`
- `#define CFG_MESSAGE_MAX_WAITER_IDS ( 8 )`
- `#define CFG_MESSAGE_MAX_ADDRESSEES ( 8 )`
- `#define CFG_SHELL_USE_SERVICE ( 1 )`
- `#define CFG_SHELL_MAX_COMMAND_NUMBER ( 16 )`
- `#define CFG_SHELL_MAX_COMMAND_LENGTH ( 20 )`
- `#define CFG_SHELL_MAX_PARAMS_LENGTH ( 128 )`
- `#define CFG_SHELL_COMMAND_BUFFER_SIZE ( 200 )`
- `#define CFG_GCP_CHANNELS_MAX_NUMBER ( 3 )`
- `#define CFG_TRACE_MAX_LENGTH ( 200 )`
- `#define CFG_SYSMON_USE_SERVICE ( 0 )`
- `#define CFG_SYSMON_GCP_CHANNEL_NUM ( 0 )`
- `#define CFG_SYSMON_MAX_USER_MESSAGES ( 6 )`
- `#define CFG_RESET_ON_ERROR ( 1 )`
- `#define CFG_RESET_ON_ERROR_DELAY_MS ( 3000 )`

#### 4.17.1 Detailed Description

GOS minimal example application configuration header.

**Author**

Ahmed Gazar

**Date**

2023-09-25

**Version**

1.0

This header contains the kernel and service configurations of the operating system.

Definition in file `gos_minimal_example_app_config.h`.

#### 4.17.2 Macro Definition Documentation

##### 4.17.2.1 `#define ARM_CORTEX_M4 ( 1 )`

ARM Cortex-M4

Definition at line 48 of file `gos_minimal_example_app_config.h`.

##### 4.17.2.2 `#define CFG_GCP_CHANNELS_MAX_NUMBER ( 3 )`

GCP maximum number of channels.

Definition at line 278 of file `gos_minimal_example_app_config.h`.

4.17.2.3 `#define CFG_IDLE_TASK_STACK_SIZE ( 0x400 )`

Idle task stack size.

Definition at line 93 of file gos\_minimal\_example\_app\_config.h.

4.17.2.4 `#define CFG_MESSAGE_MAX_ADDRESSEES ( 8 )`

Maximum number of message addressees.

Definition at line 246 of file gos\_minimal\_example\_app\_config.h.

4.17.2.5 `#define CFG_MESSAGE_MAX_LENGTH ( 80 )`

Maximum length of a message in bytes.

Definition at line 234 of file gos\_minimal\_example\_app\_config.h.

4.17.2.6 `#define CFG_MESSAGE_MAX_NUMBER ( 4 )`

Maximum number of messages handled at once.

Definition at line 230 of file gos\_minimal\_example\_app\_config.h.

4.17.2.7 `#define CFG_MESSAGE_MAX_WAITER_IDS ( 8 )`

Maximum number of message IDs a task can wait for (includes the terminating 0).

Definition at line 242 of file gos\_minimal\_example\_app\_config.h.

4.17.2.8 `#define CFG_MESSAGE_MAX_WAITERS ( 10 )`

Maximum number of message waiters.

Definition at line 238 of file gos\_minimal\_example\_app\_config.h.

4.17.2.9 `#define CFG_PROC_IDLE_PRIO ( CFG_PROC_MAX_PRIO_LEVELS )`

Idle process priority.

Definition at line 177 of file gos\_minimal\_example\_app\_config.h.

4.17.2.10 `#define CFG_PROC_MAX_NAME_LENGTH ( 24 )`

Maximum process name length.

Definition at line 181 of file gos\_minimal\_example\_app\_config.h.

4.17.2.11 `#define CFG_PROC_MAX_NUMBER ( 4 )`

Maximum number of processes.

Definition at line 185 of file gos\_minimal\_example\_app\_config.h.

4.17.2.12 #define CFG\_PROC\_MAX\_PRIO\_LEVELS ( UINT8\_MAX )

Maximum process priority levels.

Definition at line 173 of file gos\_minimal\_example\_app\_config.h.

4.17.2.13 #define CFG\_PROC\_USE\_SERVICE ( 0 )

Process service use flag.

Definition at line 169 of file gos\_minimal\_example\_app\_config.h.

4.17.2.14 #define CFG\_QUEUE\_MAX\_ELEMENTS ( 40 )

Maximum number of queue elements.

Definition at line 197 of file gos\_minimal\_example\_app\_config.h.

4.17.2.15 #define CFG\_QUEUE\_MAX\_LENGTH ( 200 )

Maximum queue length.

Definition at line 201 of file gos\_minimal\_example\_app\_config.h.

4.17.2.16 #define CFG\_QUEUE\_MAX\_NAME\_LENGTH ( 24 )

Maximum queue name length.

Definition at line 209 of file gos\_minimal\_example\_app\_config.h.

4.17.2.17 #define CFG\_QUEUE\_MAX\_NUMBER ( 4 )

Maximum number of queues.

Definition at line 193 of file gos\_minimal\_example\_app\_config.h.

4.17.2.18 #define CFG\_QUEUE\_USE\_NAME ( 1 )

Queue use name flag.

Definition at line 205 of file gos\_minimal\_example\_app\_config.h.

4.17.2.19 #define CFG\_RESET\_ON\_ERROR ( 1 )

Flag to indicate if the system should reset on error.

Definition at line 311 of file gos\_minimal\_example\_app\_config.h.

4.17.2.20 #define CFG\_RESET\_ON\_ERROR\_DELAY\_MS ( 3000 )

Delay time before system reset.

Definition at line 315 of file gos\_minimal\_example\_app\_config.h.

4.17.2.21 #define CFG\_SCHED\_COOPERATIVE ( 0 )

Cooperative scheduling flag.

Definition at line 61 of file gos\_minimal\_example\_app\_config.h.

4.17.2.22 #define CFG\_SHELL\_COMMAND\_BUFFER\_SIZE ( 200 )

Command buffer size.

Definition at line 270 of file gos\_minimal\_example\_app\_config.h.

4.17.2.23 #define CFG\_SHELL\_MAX\_COMMAND\_LENGTH ( 20 )

Maximum command length.

Definition at line 262 of file gos\_minimal\_example\_app\_config.h.

4.17.2.24 #define CFG\_SHELL\_MAX\_COMMAND\_NUMBER ( 16 )

Maximum number of shell commands.

Definition at line 258 of file gos\_minimal\_example\_app\_config.h.

4.17.2.25 #define CFG\_SHELL\_MAX\_PARAMS\_LENGTH ( 128 )

Maximum parameters length.

Definition at line 266 of file gos\_minimal\_example\_app\_config.h.

4.17.2.26 #define CFG\_SHELL\_USE\_SERVICE ( 1 )

Shell service use flag.

Definition at line 254 of file gos\_minimal\_example\_app\_config.h.

4.17.2.27 #define CFG\_SIGNAL\_MAX\_NUMBER ( 6 )

Maximum number of signals.

Definition at line 218 of file gos\_minimal\_example\_app\_config.h.

4.17.2.28 #define CFG\_SIGNAL\_MAX\_SUBSCRIBERS ( 6 )

Maximum number of signal subscribers.

Definition at line 222 of file gos\_minimal\_example\_app\_config.h.

4.17.2.29 #define CFG\_SYSMON\_GCP\_CHANNEL\_NUM ( 0 )

Define sysmon GCP channel number.

Definition at line 298 of file gos\_minimal\_example\_app\_config.h.

---

4.17.2.30 #define CFG\_SYSMON\_MAX\_USER\_MESSAGES ( 6 )

Maximum number of user messages.

Definition at line 303 of file gos\_minimal\_example\_app\_config.h.

4.17.2.31 #define CFG\_SYSMON\_USE\_SERVICE ( 0 )

Sysmon use service flag.

Definition at line 294 of file gos\_minimal\_example\_app\_config.h.

4.17.2.32 #define CFG\_SYSTEM\_TASK\_STACK\_SIZE ( 0x400 )

System task stack size.

Definition at line 97 of file gos\_minimal\_example\_app\_config.h.

4.17.2.33 #define CFG\_TARGET\_CPU ( ARM\_CORTEX\_M4 )

Target CPU.

Definition at line 53 of file gos\_minimal\_example\_app\_config.h.

4.17.2.34 #define CFG\_TASK\_MAX\_NAME\_LENGTH ( 32 )

Maximum task name length.

Definition at line 73 of file gos\_minimal\_example\_app\_config.h.

4.17.2.35 #define CFG\_TASK\_MAX\_NUMBER ( 16 )

Maximum number of tasks.

Definition at line 77 of file gos\_minimal\_example\_app\_config.h.

4.17.2.36 #define CFG\_TASK\_MAX\_STACK\_SIZE ( 0x4000 )

Maximum task stack size.

Definition at line 89 of file gos\_minimal\_example\_app\_config.h.

4.17.2.37 #define CFG\_TASK\_MESSAGE\_DAEMON\_PRIO ( 198 )

Message daemon task priority.

Definition at line 137 of file gos\_minimal\_example\_app\_config.h.

4.17.2.38 #define CFG\_TASK\_MESSAGE\_DAEMON\_STACK ( 0x300 )

Message daemon task stack size.

Definition at line 113 of file gos\_minimal\_example\_app\_config.h.

4.17.2.39 #define CFG\_TASK\_MIN\_STACK\_SIZE ( 0x200 )

Minimum task stack size.

Definition at line 85 of file gos\_minimal\_example\_app\_config.h.

4.17.2.40 #define CFG\_TASK\_PROC\_DAEMON\_PRIO ( 194 )

Process daemon task priority.

Definition at line 145 of file gos\_minimal\_example\_app\_config.h.

4.17.2.41 #define CFG\_TASK\_PROC\_DAEMON\_STACK ( 0x300 )

Process daemon task stack size.

Definition at line 105 of file gos\_minimal\_example\_app\_config.h.

4.17.2.42 #define CFG\_TASK\_SHELL\_DAEMON\_PRIO ( 192 )

Shell daemon task priority.

Definition at line 149 of file gos\_minimal\_example\_app\_config.h.

4.17.2.43 #define CFG\_TASK\_SHELL\_DAEMON\_STACK ( 0x300 )

Shell daemon task stack size.

Definition at line 117 of file gos\_minimal\_example\_app\_config.h.

4.17.2.44 #define CFG\_TASK\_SIGNAL\_DAEMON\_PRIO ( 197 )

Signal daemon task priority.

Definition at line 141 of file gos\_minimal\_example\_app\_config.h.

4.17.2.45 #define CFG\_TASK\_SIGNAL\_DAEMON\_STACK ( 0x300 )

Signal daemon task stack size.

Definition at line 101 of file gos\_minimal\_example\_app\_config.h.

4.17.2.46 #define CFG\_TASK\_SYS\_PRIO ( 195 )

System task priority.

Definition at line 157 of file gos\_minimal\_example\_app\_config.h.

4.17.2.47 #define CFG\_TASK\_SYSMON\_DAEMON\_PRIO ( 40 )

Sysmon daemon task priority.

Definition at line 161 of file gos\_minimal\_example\_app\_config.h.

4.17.2.48 #define CFG\_TASK\_SYSMON\_DAEMON\_STACK ( 0x300 )

Sysmon daemon task stack size.

Definition at line 125 of file gos\_minimal\_example\_app\_config.h.

4.17.2.49 #define CFG\_TASK\_TIME\_DAEMON\_PRIO ( 196 )

Time daemon task priority.

Definition at line 153 of file gos\_minimal\_example\_app\_config.h.

4.17.2.50 #define CFG\_TASK\_TIME\_DAEMON\_STACK ( 0x300 )

Time daemon task stack size.

Definition at line 109 of file gos\_minimal\_example\_app\_config.h.

4.17.2.51 #define CFG\_TASK\_TRACE\_DAEMON\_PRIO ( 193 )

Trace daemon task priority.

Definition at line 133 of file gos\_minimal\_example\_app\_config.h.

4.17.2.52 #define CFG\_TASK\_TRACE\_DAEMON\_STACK ( 0x400 )

Log daemon task stack size.

Definition at line 121 of file gos\_minimal\_example\_app\_config.h.

4.17.2.53 #define CFG\_TRACE\_MAX\_LENGTH ( 200 )

Trace maximum (line) length.

Definition at line 286 of file gos\_minimal\_example\_app\_config.h.

4.17.2.54 #define CFG\_USE\_PRIO\_INHERITANCE ( 1 )

Priority inheritance flag for lock.

Definition at line 65 of file gos\_minimal\_example\_app\_config.h.

4.17.2.55 #define GOS\_CFG\_OVERCONFIG

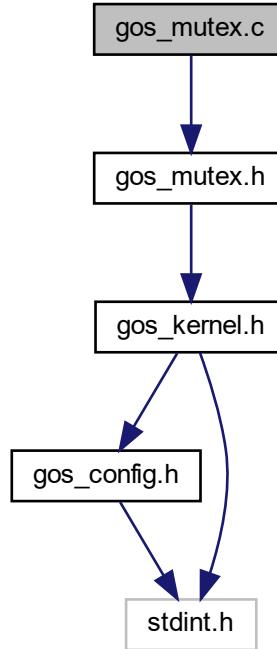
Overconfiguration macro.

Definition at line 41 of file gos\_minimal\_example\_app\_config.h.

## 4.18 gos\_mutex.c File Reference

GOS mutex service source.

```
#include <gos_mutex.h>
Include dependency graph for gos_mutex.c:
```



## Macros

- `#define MUTEX_LOCK_SLEEP_MS ( 2u )`

## Functions

- `gos_result_t gos_mutexInit (gos_mutex_t *pMutex)`  
*Initializes the mutex instance.*
- `gos_result_t gos_mutexLock (gos_mutex_t *pMutex, u32_t timeout)`  
*Tries to lock the given mutex with the given timeout.*
- `gos_result_t gos_mutexUnlock (gos_mutex_t *pMutex)`  
*Unlocks the mutex instance.*

### 4.18.1 Detailed Description

GOS mutex service source.

#### Author

Ahmed Gazar

**Date**

2024-04-02

**Version**

1.8

For a more detailed description of this service, please refer to [gos\\_mutex.h](#)

Definition in file [gos\\_mutex.c](#).

## 4.18.2 Macro Definition Documentation

### 4.18.2.1 #define MUTEX\_LOCK\_SLEEP\_MS ( 2u )

Sleep time in [ms] before re-attempting to lock mutex.

Definition at line 71 of file gos\_mutex.c.

## 4.18.3 Function Documentation

### 4.18.3.1 gos\_result\_t gos\_mutexInit( gos\_mutex\_t \* pMutex )

Initializes the mutex instance.

Sets the mutex state to unlocked.

**Parameters**

|                     |                                           |
|---------------------|-------------------------------------------|
| <code>pMutex</code> | : Pointer to the mutex to be initialized. |
|---------------------|-------------------------------------------|

**Returns**

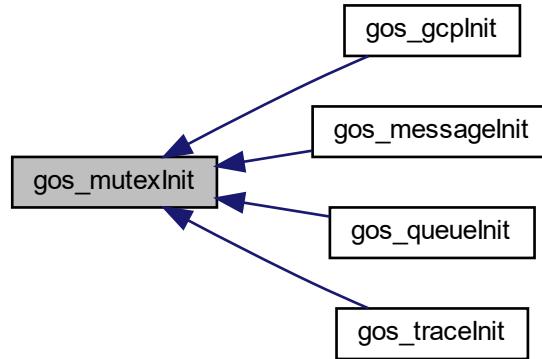
Result of initialization.

**Return values**

|                          |                                   |
|--------------------------|-----------------------------------|
| <code>GOS_SUCCESS</code> | : Mutex initialized successfully. |
| <code>GOS_ERROR</code>   | : Mutex pointer is NULL.          |

Definition at line 76 of file gos\_mutex.c.

Here is the caller graph for this function:



#### 4.18.3.2 gos\_result\_t gos\_mutexLock ( gos\_mutex\_t \* pMutex, u32\_t timeout )

Tries to lock the given mutex with the given timeout.

Waits in an unblocking way until the mutex is unlocked or the timeout value is reached. If the mutex becomes unlocked within the timeout value, it locks it.

##### Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>pMutex</i>  | : Pointer to the mutex to be locked. |
| <i>timeout</i> | : Timeout value.                     |

##### Returns

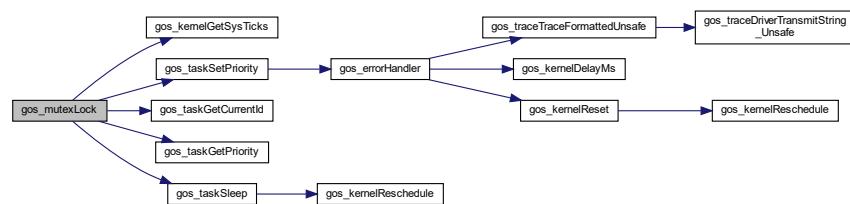
Result of mutex locking.

##### Return values

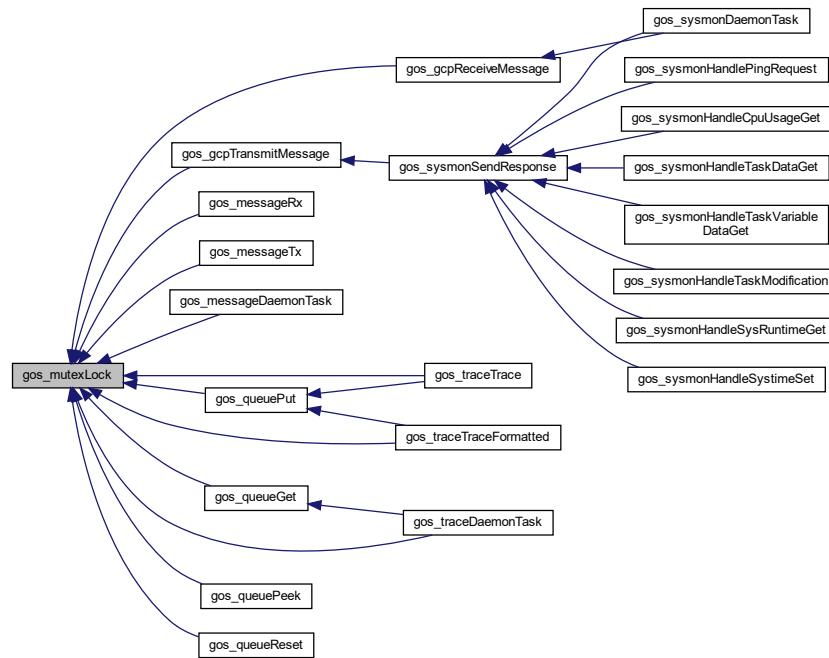
|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Mutex locked successfully.                          |
| <i>GOS_ERROR</i>   | : Mutex could not be locked within the timeout value. |

Definition at line 103 of file gos\_mutex.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.3 `gos_result_t gos_mutexUnlock( gos_mutex_t * pMutex )`

Unlocks the mutex instance.

Sets the mutex state to unlocked.

##### Parameters

|                     |                                        |
|---------------------|----------------------------------------|
| <code>pMutex</code> | : Pointer to the mutex to be unlocked. |
|---------------------|----------------------------------------|

##### Returns

Result of mutex unlocking.

##### Return values

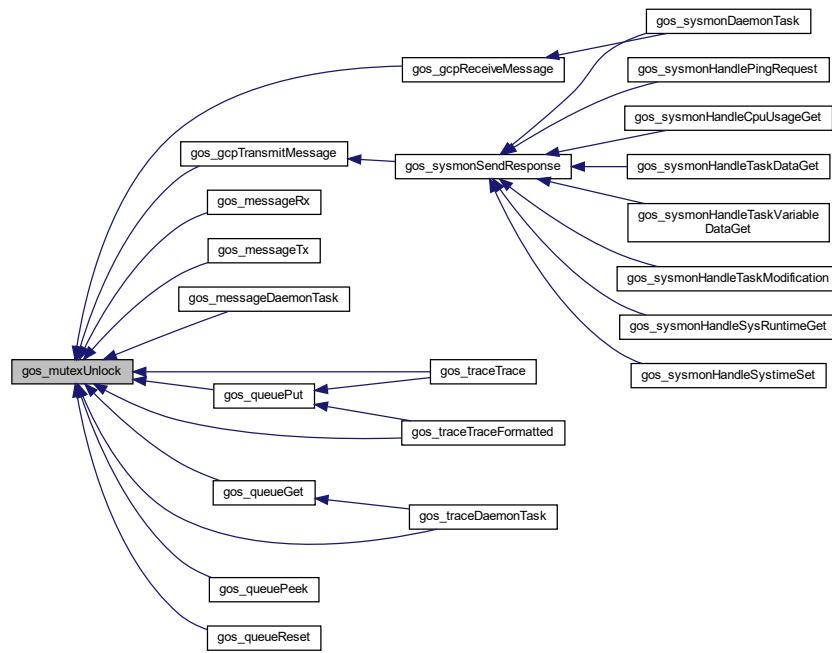
|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Unlocking successful.                                  |
| <code>GOS_ERROR</code>   | : Mutex is NULL or caller is not the owner of the mutex. |

Definition at line 202 of file `gos_mutex.c`.

Here is the call graph for this function:



Here is the caller graph for this function:

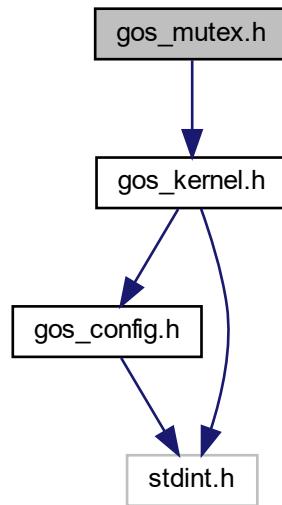


## 4.19 gos\_mutex.h File Reference

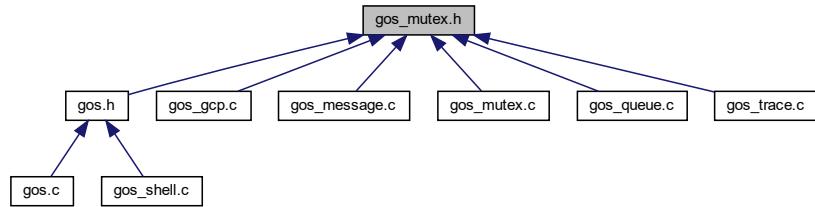
GOS mutex service header.

```
#include <gos_kernel.h>
```

Include dependency graph for gos\_mutex.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `gos_mutex_t`

## Macros

- `#define GOS_MUTEX_ENDLESS_TMO ( 0xFFFFFFFFFu )`
- `#define GOS_MUTEX_NO_TMO ( 0x00000000u )`

## Enumerations

- enum `gos_mutexState_t` { `GOS_MUTEX_UNLOCKED` = 0b11010010, `GOS_MUTEX_LOCKED` = 0b01101011 }

## Functions

- `gos_result_t gos_mutexInit (gos_mutex_t *pMutex)`  
*Initializes the mutex instance.*
- `gos_result_t gos_mutexLock (gos_mutex_t *pMutex, u32_t timeout)`  
*Tries to lock the given mutex with the given timeout.*
- `gos_result_t gos_mutexUnlock (gos_mutex_t *pMutex)`  
*Unlocks the mutex instance.*

### 4.19.1 Detailed Description

GOS mutex service header.

#### Author

Ahmed Gazar

#### Date

2023-09-14

**Version**

1.1

Mutex (Mutual Exclusion) service is provided for protecting shared resources. A mutex has two states: locked or unlocked. When a task calls the lock function, the service checks the mutex state, and it locks the mutex if it is unlocked. If a mutex is locked by another task, it waits in a non-blocking way (yields) for the mutex to be unlocked.

Definition in file [gos\\_mutex.h](#).

## 4.19.2 Macro Definition Documentation

### 4.19.2.1 `#define GOS_MUTEX_ENDLESS_TMO ( 0xFFFFFFFFFu )`

Mutex endless timeout.

Definition at line 66 of file [gos\\_mutex.h](#).

### 4.19.2.2 `#define GOS_MUTEX_NO_TMO ( 0x00000000u )`

Mutex no timeout.

Definition at line 71 of file [gos\\_mutex.h](#).

## 4.19.3 Enumeration Type Documentation

### 4.19.3.1 `enum gos_mutexState_t`

Mutex state type.

**Enumerator**

**GOS\_MUTEX\_UNLOCKED** Mutex unlocked.

**GOS\_MUTEX\_LOCKED** Mutex locked.

Definition at line 79 of file [gos\\_mutex.h](#).

## 4.19.4 Function Documentation

### 4.19.4.1 `gos_result_t gos_mutexInit( gos_mutex_t * pMutex )`

Initializes the mutex instance.

Sets the mutex state to unlocked.

**Parameters**

|                     |                                           |
|---------------------|-------------------------------------------|
| <code>pMutex</code> | : Pointer to the mutex to be initialized. |
|---------------------|-------------------------------------------|

**Returns**

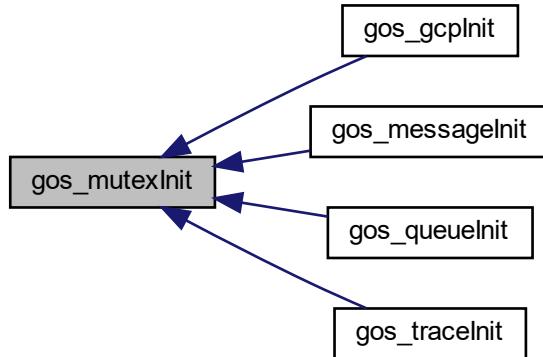
Result of initialization.

## Return values

|                    |                                   |
|--------------------|-----------------------------------|
| <i>GOS_SUCCESS</i> | : Mutex initialized successfully. |
| <i>GOS_ERROR</i>   | : Mutex pointer is NULL.          |

Definition at line 76 of file gos\_mutex.c.

Here is the caller graph for this function:



#### 4.19.4.2 `gos_result_t gos_mutexLock( gos_mutex_t * pMutex, u32_t timeout )`

Tries to lock the given mutex with the given timeout.

Waits in an unblocking way until the mutex is unlocked or the timeout value is reached. If the mutex becomes unlocked within the timeout value, it locks it.

## Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>pMutex</i>  | : Pointer to the mutex to be locked. |
| <i>timeout</i> | : Timeout value.                     |

**Returns**

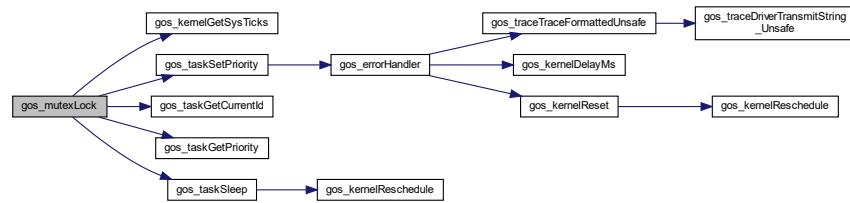
Result of mutex locking.

**Return values**

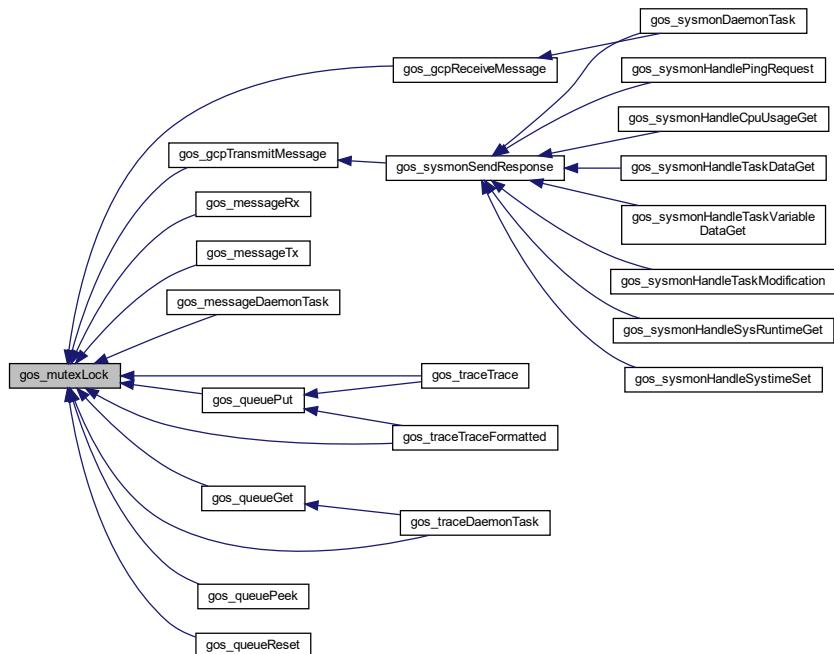
|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Mutex locked successfully.                          |
| <i>GOS_ERROR</i>   | : Mutex could not be locked within the timeout value. |

Definition at line 103 of file gos\_mutex.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.4.3 `gos_result_t gos_mutexUnlock( gos_mutex_t * pMutex )`

Unlocks the mutex instance.

Sets the mutex state to unlocked.

**Parameters**

|               |                                        |
|---------------|----------------------------------------|
| <i>pMutex</i> | : Pointer to the mutex to be unlocked. |
|---------------|----------------------------------------|

**Returns**

Result of mutex unlocking.

**Return values**

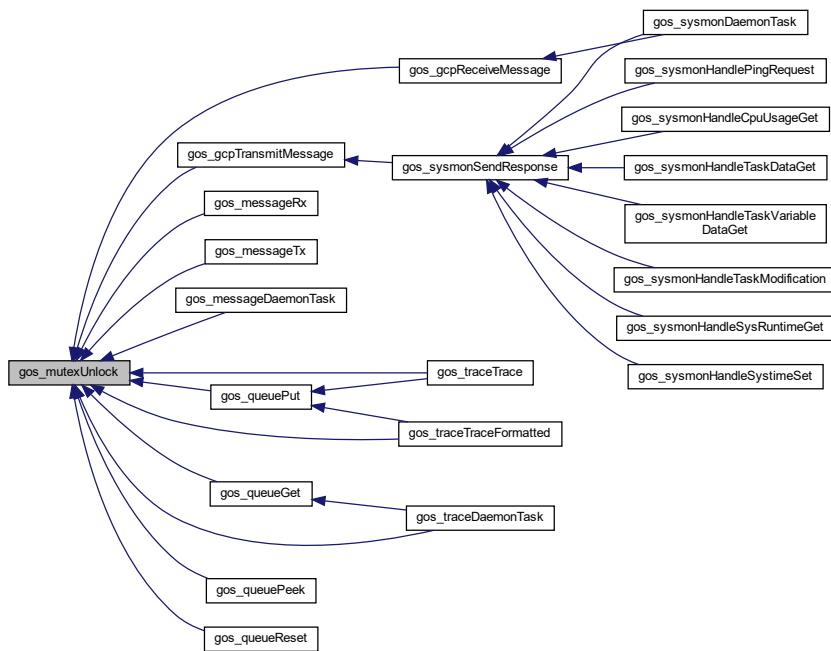
|                    |                                                          |
|--------------------|----------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Unlocking successful.                                  |
| <i>GOS_ERROR</i>   | : Mutex is NULL or caller is not the owner of the mutex. |

Definition at line 202 of file gos\_mutex.c.

Here is the call graph for this function:



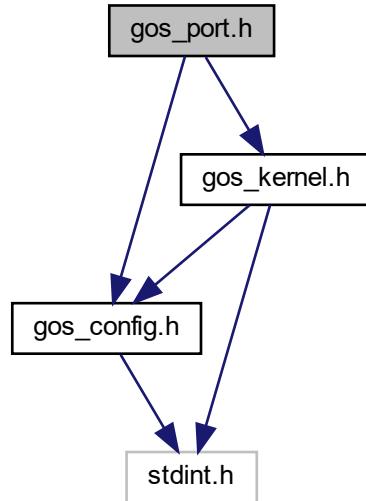
Here is the caller graph for this function:



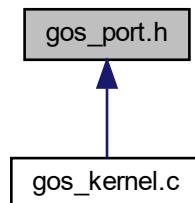
## 4.20 gos\_port.h File Reference

GOS port header.

```
#include "gos_kernel.h"
#include "gos_config.h"
Include dependency graph for gos_port.h:
```



This graph shows which files directly or indirectly include this file:



## Macros

- #define gos\_ported\_procReset()
- #define gos\_ported\_reschedule(privilege)
- #define gos\_ported\_pendSVHandler GOS\_NAKED PendSV\_Handler
- #define gos\_ported\_doContextSwitch()
- #define gos\_ported\_svcHandler GOS\_NAKED SVC\_Handler
- #define gos\_ported\_handleSVC()
- #define gos\_ported\_svcHandlerMain gos\_kernelSVC\_HandlerMain
- #define gos\_ported\_handleSVCMain(sp)
- #define gos\_ported\_sysTickInterrupt SysTick\_Handler

- #define gos\_ported\_kernelStartInit()
- #define gos\_ported\_enableFaultHandlers()

#### 4.20.1 Detailed Description

GOS port header.

**Author**

Ahmed Gazar

**Date**

2023-07-12

**Version**

1.0

This header contains the platform-specific ported definitions of the OS.

Definition in file [gos\\_port.h](#).

#### 4.20.2 Macro Definition Documentation

##### 4.20.2.1 #define gos\_ported\_doContextSwitch( )

**Value:**

```
{
 \
 /* Save LR back to main, must do this firstly. */
 GOS_ASM("PUSH {LR}");
 \
 /* Save the context of current task. */
 /* Get current PSP. */
 GOS_ASM("MRS R0, PSP");
 /* Save R4 to R11 to PSP Frame Stack. */
 GOS_ASM("STMDB R0!, {R4-R11}"); /* R0 is updated after decrement */
 /* Save current value of PSP. */
 GOS_ASM("BL gos_kernelSaveCurrentPsp"); /* R0 is first argument */
 \
 /* Do scheduling. */
 /* Select next task. */
 GOS_ASM("BL gos_kernelSelectNextTask");
 \
 /* Retrieve the context of next task. */
 /* Get its past PSP value. */
 GOS_ASM("BL gos_kernelGetCurrentPsp"); /* return PSP is in R0 */
 /* Retrieve R4-R11 from PSP Frame Stack. */
 GOS_ASM("LDMIA R0!, {R4-R11}"); /* R0 is updated after increment */
 /* Update PSP. */
 GOS_ASM("MSR PSP, R0");
 \
 /* Exit. */
 GOS_ASM("POP {LR}");
 \
 GOS_ASM("BX LR");
}
```

Context-switch function.

Definition at line 82 of file [gos\\_port.h](#).

## 4.20.2.2 #define gos\_ported\_enableFaultHandlers( )

**Value:**

```
(\
{ \
 SHCSR |= (1 << 16); /* Memory Fault */ \
 SHCSR |= (1 << 17); /* Bus Fault */ \
 SHCSR |= (1 << 18); /* Usage Fault */ \
} \
)
```

Fault handler enable function.

Definition at line 204 of file gos\_port.h.

## 4.20.2.3 #define gos\_ported\_handleSVC( )

**Value:**

```
(\
{ \
 /* Check LR to know which stack is used. */ \
 GOS_ASM("TST LR, 4"); \
 /* 2 next instructions are conditional. */ \
 GOS_ASM("ITE EQ"); \
 /* Save MSP if bit 2 is 0. */ \
 GOS_ASM("MRSEQ R0, MSP"); \
 /* Save PSP if bit 2 is 1. */ \
 GOS_ASM("MRSNE R0, PSP"); \
 \
 /* Check if reset is required. */ \
 if (resetRequired == GOS_TRUE) \
 { \
 resetRequired = GOS_FALSE; \
 gos_kernelProcessorReset(); \
 } \
 \
 /* Pass R0 as the argument. */ \
 GOS_ASM("B gos_kernelSVC_HandlerMain"); \
} \
)
```

SVC handler function.

Definition at line 121 of file gos\_port.h.

## 4.20.2.4 #define gos\_ported\_handleSVCMain( sp )

**Value:**

```
(\
{ \
 /* Get the address of the instruction saved in PC. */ \
 u8_t* pInstruction = (u8_t*)(sp[6]); \
 \
 /* Go back 2 bytes (16-bit opcode). */ \
 pInstruction -= 2; \
 \
 /* Get the opcode, in little-endian. */ \
 u8_t svcNum = *pInstruction; \
 \
 switch (svcNum) \
 { \
 case 0xFF: \
 /* Trigger PendSV. */ \
 ICSR |= (1 << 28); \
 break; \
 default: break; \
 } \
} \
)
```

SVC handler main function.

Definition at line 152 of file gos\_port.h.

#### 4.20.2.5 #define gos\_ported\_kernelStartInit( )

**Value:**

```
(\
{ \
 /* Prepare PSP of the first task. */
 GOS_ASM("BL gos_kernelGetCurrentPsp"); /* return PSP in R0 */
 GOS_ASM("MSR PSP, R0"); /* set PSP */

 /* Change to use PSP. */
 GOS_ASM("MRS R0, CONTROL");
 GOS_ASM("ORR R0, R0, #2"); /* set bit[1] SPSEL */
 GOS_ASM("MSR CONTROL, R0");

 /* Move to unprivileged level. */
 GOS_ASM("MRS R0, CONTROL");
 GOS_ASM("ORR R0, R0, #1"); /* Set bit[0] nPRIV */
 GOS_ASM("MSR CONTROL, R0");
 /* Right after here, access is limited. */
}
)
```

Kernel start initialization function.

Definition at line 182 of file gos\_port.h.

#### 4.20.2.6 #define gos\_ported\_pendSVHandler GOS\_NAKED PendSV\_Handler

Pend SV handler function name.

Definition at line 77 of file gos\_port.h.

#### 4.20.2.7 #define gos\_ported\_procReset( )

**Value:**

```
(\
{ \
 \
 GOS_ASM ("dsb 0xF:::";"memory");
 \
 (<u32_t>(0xE000ED0CUL) = (<u32_t>((0x5FAUL << 16U) | (*(<u32_t*>(0xE000ED0CUL) & (7UL <<
 8U)) | (1UL << 2U));
 GOS_ASM ("dsb 0xF:::";"memory");
 \
}
)
```

Processor reset function.

Definition at line 48 of file gos\_port.h.

#### 4.20.2.8 #define gos\_ported\_reschedule( privilege )

**Value:**

```
(\
{ \
 if (privilege == GOS_PRIVILEGED)
 {
 /* Trigger PendSV directly. */
 ICSR |= (1 << 28);
 }
 else
 {
 /* Call Supervisor exception to get Privileged access. */
 GOS_ASM("SVC #255");
 }
}
)
```

Reschedule function.

Definition at line 59 of file gos\_port.h.

#### 4.20.2.9 #define gos\_ported\_svcHandler GOS\_NAKED SVC\_Handler

SVC handler function name.

Definition at line 116 of file gos\_port.h.

#### 4.20.2.10 #define gos\_ported\_svcHandlerMain gos\_kernelSVC\_HandlerMain

SVC handler main function name.

Definition at line 147 of file gos\_port.h.

#### 4.20.2.11 #define gos\_ported\_sysTickInterrupt SysTick\_Handler

System tick handler function name.

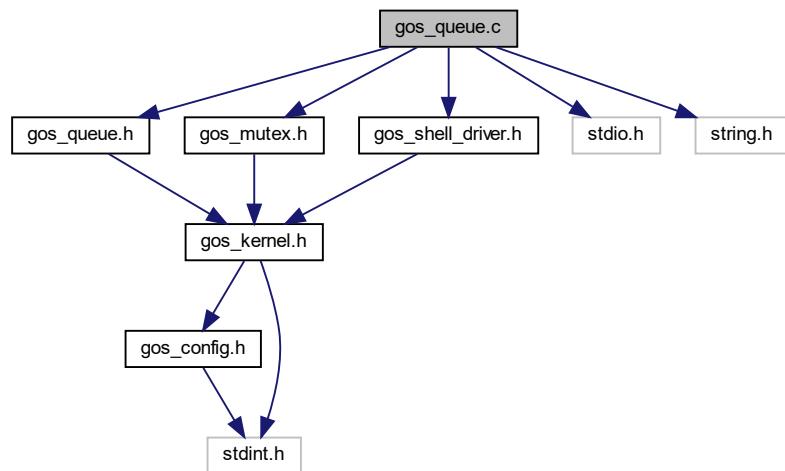
Definition at line 177 of file gos\_port.h.

## 4.21 gos\_queue.c File Reference

GOS queue service source.

```
#include <gos_queue.h>
#include <gos_mutex.h>
#include <gos_shell_driver.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for gos\_queue.c:



## Data Structures

- struct `gos_queueElement_t`
- struct `gos_queue_t`

## Macros

- `#define DUMP_SEPARATOR "+-----+\r\n"`

## Functions

- `gos_result_t gos_queueInit (void_t)`  
*This function initializes the queue service.*
- `gos_result_t gos_queueCreate (gos_queueDescriptor_t *pQueueDescriptor)`  
*This function creates a new queue.*
- `gos_result_t gos_queuePut (gos_queueId_t queueId, void_t *element, gos_queueLength_t elementSize, u32_t timeout)`  
*This function puts an element in the given queue.*
- `gos_result_t gos_queueGet (gos_queueId_t queueId, void_t *target, gos_queueLength_t targetSize, u32_t timeout)`  
*This function gets the next element from the given queue.*
- `gos_result_t gos_queuePeek (gos_queueId_t queueId, void_t *target, gos_queueLength_t targetSize, u32_t timeout)`  
*This function gets the next element from the given queue without removing it.*
- `gos_result_t gos_queueReset (gos_queueId_t queueId)`  
*Resets the given queue.*
- `gos_result_t gos_queueRegisterFullHook (gos_queueFullHook fullHook)`  
*This function registers a queue full hook function.*
- `gos_result_t gos_queueRegisterEmptyHook (gos_queueEmptyHook emptyHook)`  
*This function registers a queue empty hook function.*
- `gos_result_t gos_queueGetName (gos_queueId_t queueId, gos_queueName_t queueName)`  
*This function gets the name of the given queue.*
- `gos_result_t gos_queueGetElementNumber (gos_queueId_t queueId, gos_queueIndex_t *elementNumber)`  
*This function gets the number of elements in the given queue.*
- `void_t gos_queueDump (void_t)`  
*Queue dump.*

## Variables

- `GOS_STATIC gos_queue_t queues [CFG_QUEUE_MAX_NUMBER]`
- `GOS_STATIC gos_queueLength_t readCounters [CFG_QUEUE_MAX_NUMBER]`
- `GOS_STATIC gos_queueLength_t writeCounters [CFG_QUEUE_MAX_NUMBER]`
- `GOS_STATIC gos_mutex_t queueMutex`
- `GOS_STATIC gos_queueFullHook queueFullHook = NULL`
- `GOS_STATIC gos_queueEmptyHook queueEmptyHook = NULL`

### 4.21.1 Detailed Description

GOS queue service source.

#### Author

Ahmed Gazar

#### Date

2024-04-02

#### Version

1.8

For a more detailed description of this service, please refer to [gos\\_queue.h](#)

Definition in file [gos\\_queue.c](#).

### 4.21.2 Macro Definition Documentation

#### 4.21.2.1 `#define DUMP_SEPARATOR "+-----+\r\n"`

Dump separator line.

Definition at line 83 of file gos\_queue.c.

### 4.21.3 Function Documentation

#### 4.21.3.1 `gos_result_t gos_queueCreate( gos_queueDescriptor_t * pQueueDescriptor )`

This function creates a new queue.

This function loops through the internal queue array and registers the new queue in the next free slot.

#### Parameters

|                               |                                                         |
|-------------------------------|---------------------------------------------------------|
| <code>pQueueDescriptor</code> | : Pointer to queue descriptor variable with queue data. |
|-------------------------------|---------------------------------------------------------|

#### Returns

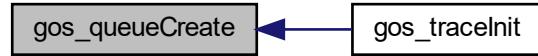
Result of queue creation.

#### Return values

|                          |                                                            |
|--------------------------|------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Queue creation successful.                               |
| <code>GOS_ERROR</code>   | : Queue descriptor is NULL pointer or queue array is full. |

Definition at line 181 of file gos\_queue.c.

Here is the caller graph for this function:



#### 4.21.3.2 void\_t gos\_queueDump( void\_t )

Queue dump.

Prints the queue data of all queues to the trace output.

Returns

-

Definition at line 598 of file `gos_queue.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.21.3.3 gos\_result\_t gos\_queueGet( gos\_queueId\_t queueId, void\_t \* target, gos\_queueLength\_t targetSize, u32\_t timeout )

This function gets the next element from the given queue.

This function checks the queue state and gets the next element from the queue.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <i>queueId</i>    | : Queue ID.                        |
| <i>target</i>     | : Pointer to target variable.      |
| <i>targetSize</i> | : Size of target.                  |
| <i>timeout</i>    | : Timeout for locking queue mutex. |

**Returns**

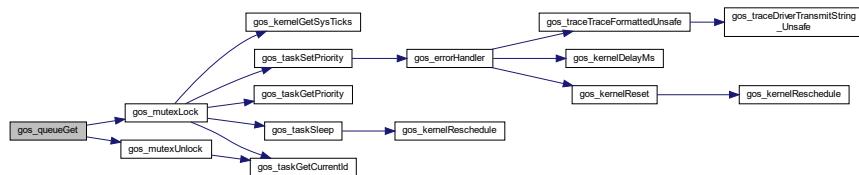
Result of element getting.

**Return values**

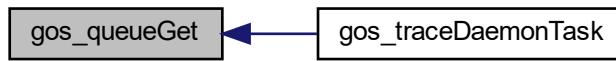
|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Element successfully moved from queue to target.         |
| <i>GOS_ERROR</i>   | : Invalid queue ID, invalid target size or queue is empty. |

Definition at line 310 of file gos\_queue.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.21.3.4 `gos_result_t gos_queueGetElementNumber( gos_queueId_t queueId, gos_queueIndex_t *elementNumber )`

This function gets the number of elements in the given queue.

This function copies the number of elements in the queue belonging to the given ID to the given variable.

**Parameters**

|                      |                                                       |
|----------------------|-------------------------------------------------------|
| <i>queueId</i>       | : Queue ID.                                           |
| <i>elementNumber</i> | : Pointer to variable to store the element number in. |

**Returns**

Result of queue element number getting.

**Return values**

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Element number getting successful.                   |
| <i>GOS_ERROR</i>   | : Invalid queue ID or element number variable is NULL. |

Definition at line 564 of file gos\_queue.c.

**4.21.3.5 gos\_result\_t gos\_queueGetName ( gos\_queueId\_t queueId, gos\_queueName\_t queueName )**

This function gets the name of the given queue.

This function copies the name of the queue belonging to the given ID to the given variable.

**Parameters**

|                  |               |
|------------------|---------------|
| <i>queueId</i>   | : Queue ID.   |
| <i>queueName</i> | : Queue name. |

**Returns**

Result of queue name getting.

**Return values**

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Name getting successful.                         |
| <i>GOS_ERROR</i>   | : Invalid queue ID or queue name variable is NULL. |

Definition at line 525 of file gos\_queue.c.

**4.21.3.6 gos\_result\_t gos\_queueInit ( void\_t )**

This function initializes the queue service.

Initializes the internal queue array, creates the queue lock, and registers the queue dump task in the kernel.

**Returns**

Result of initialization.

**Return values**

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Initialization successful.                                 |
| <i>GOS_ERROR</i>   | : Lock creation, task registration or task suspension error. |

Definition at line 147 of file gos\_queue.c.

Here is the call graph for this function:



4.21.3.7 `gos_result_t gos_queuePeek( gos_queueld_t queueId, void_t * target, gos_queueLength_t targetSize, u32_t timeout )`

This function gets the next element from the given queue without removing it.

This function checks the queue state and returns the next element from the queue without modifying the queue counters.

#### Parameters

|                         |                                    |
|-------------------------|------------------------------------|
| <code>queueId</code>    | : Queue ID.                        |
| <code>target</code>     | : Pointer to target variable.      |
| <code>targetSize</code> | : Size of target.                  |
| <code>timeout</code>    | : Timeout for locking queue mutex. |

#### Returns

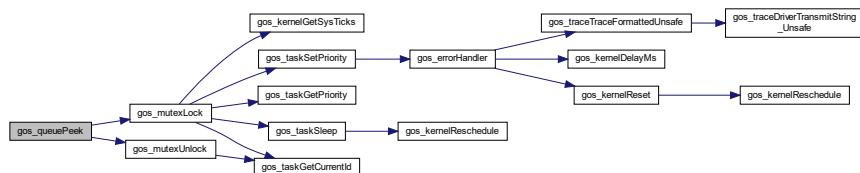
Result of element getting.

#### Return values

|                          |                                                            |
|--------------------------|------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Element successfully copied from queue to target.        |
| <code>GOS_ERROR</code>   | : Invalid queue ID, invalid target size or queue is empty. |

Definition at line 385 of file gos\_queue.c.

Here is the call graph for this function:



4.21.3.8 `gos_result_t gos_queuePut( gos_queueld_t queueId, void_t * element, gos_queueLength_t elementSize, u32_t timeout )`

This function puts an element in the given queue.

This function checks the queue state and places the given element in the next queue element.

#### Parameters

|                          |                                    |
|--------------------------|------------------------------------|
| <code>queueId</code>     | : Queue ID.                        |
| <code>element</code>     | : Pointer to element.              |
| <code>elementSize</code> | : Size of element.                 |
| <code>timeout</code>     | : Timeout for locking queue mutex. |

#### Returns

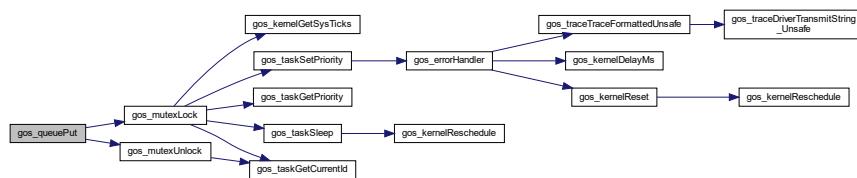
Result of element putting.

## Return values

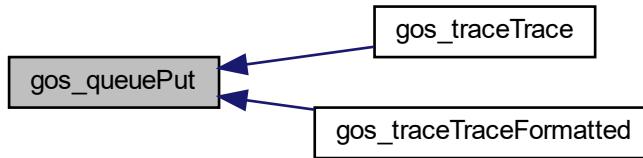
|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Element successfully put in the queue.                   |
| <i>GOS_ERROR</i>   | : Invalid queue ID, invalid element size or queue is full. |

Definition at line 231 of file gos\_queue.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.21.3.9 `gos_result_t gos_queueRegisterEmptyHook ( gos_queueEmptyHook emptyHook )`

This function registers a queue empty hook function.

This function checks whether a hook has been already registered, and if not, it saves the given hook function.

## Parameters

|                  |                  |
|------------------|------------------|
| <i>emptyHook</i> | : Hook function. |
|------------------|------------------|

## Returns

Result of hook registration.

## Return values

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Hook registration successful.                     |
| <i>GOS_ERROR</i>   | : Hook already exists or parameter is NULL pointer. |

Definition at line 499 of file gos\_queue.c.

#### 4.21.3.10 `gos_result_t gos_queueRegisterFullHook ( gos_queueFullHook fullHook )`

This function registers a queue full hook function.

This function checks whether a hook has been already registered, and if not, it saves the given hook function.

**Parameters**

|                 |                  |
|-----------------|------------------|
| <i>fullHook</i> | : Hook function. |
|-----------------|------------------|

**Returns**

Result of hook registration.

**Return values**

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Hook registration successful.                     |
| <i>GOS_ERROR</i>   | : Hook already exists or parameter is NULL pointer. |

Definition at line 473 of file gos\_queue.c.

**4.21.3.11 gos\_result\_t gos\_queueReset( gos\_queueId\_t queueId )**

Resets the given queue.

Sets the read and write counter to zero, making the queue empty.

**Parameters**

|                |             |
|----------------|-------------|
| <i>queueId</i> | : Queue ID. |
|----------------|-------------|

**Returns**

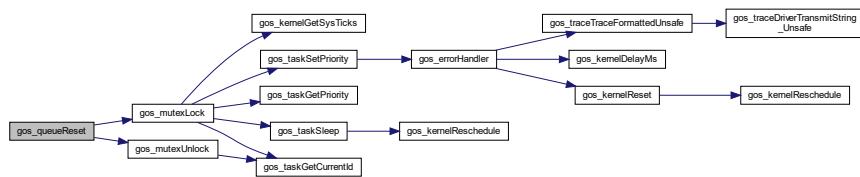
Result of queue resetting.

**Return values**

|                    |                     |
|--------------------|---------------------|
| <i>GOS_SUCCESS</i> | : Reset successful. |
| <i>GOS_ERROR</i>   | : Invalid queue ID. |

Definition at line 435 of file gos\_queue.c.

Here is the call graph for this function:

**4.21.4 Variable Documentation****4.21.4.1 GOS\_STATIC gos\_queueEmptyHook queueEmptyHook = NULL**

Queue empty hook.

Definition at line 142 of file gos\_queue.c.

**4.21.4.2 GOS\_STATIC gos\_queueFullHook queueFullHook = NULL**

Queue full hook.

Definition at line 137 of file gos\_queue.c.

#### 4.21.4.3 GOS\_STATIC gos\_mutex\_t queueMutex

Queue mutex.

Definition at line 132 of file gos\_queue.c.

#### 4.21.4.4 GOS\_STATIC gos\_queue\_t queues[CFG\_QUEUE\_MAX\_NUMBER]

Internal queue array.

Definition at line 117 of file gos\_queue.c.

#### 4.21.4.5 GOS\_STATIC gos\_queueLength\_t readCounters[CFG\_QUEUE\_MAX\_NUMBER]

Read counter array (next queue element to read).

Definition at line 122 of file gos\_queue.c.

#### 4.21.4.6 GOS\_STATIC gos\_queueLength\_t writeCounters[CFG\_QUEUE\_MAX\_NUMBER]

Write counter array (next queue element to write).

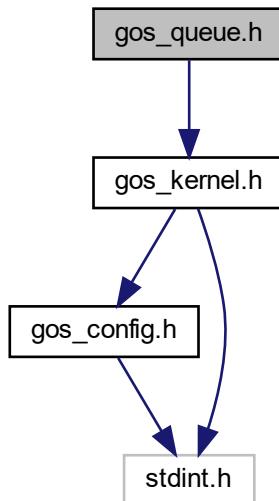
Definition at line 127 of file gos\_queue.c.

## 4.22 gos\_queue.h File Reference

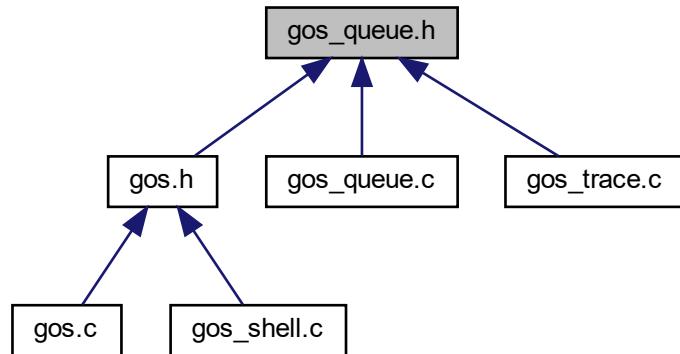
GOS queue service header.

```
#include <gos_kernel.h>
```

Include dependency graph for gos\_queue.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `gos_queueDescriptor_t`

## Macros

- `#define GOS_DEFAULT_QUEUE_ID ( (gos_queueId_t) 0x3000 )`
- `#define GOS_INVALID_QUEUE_ID ( (gos_queueId_t) 0x0300 )`

## TypeDefs

- `typedef u8_t gos_queueByte_t`
- `typedef u8_t gos_queueLength_t`  
*Queue length type.*
- `typedef u8_t gos_queueIndex_t`  
*Queue index type.*
- `typedef u16_t gos_queueId_t`  
*Queue ID type.*
- `typedef void_t(*) gos_queueFullHook )(gos_queueId_t)`
- `typedef void_t(*) gos_queueEmptyHook )(gos_queueId_t)`

## Functions

- `gos_result_t gos_queueInit (void_t)`  
*This function initializes the queue service.*
- `gos_result_t gos_queueCreate (gos_queueDescriptor_t *pQueueDescriptor)`  
*This function creates a new queue.*
- `gos_result_t gos_queuePut (gos_queueId_t queueId, void_t *element, gos_queueLength_t elementSize, u32_t timeout)`  
*This function puts an element in the given queue.*

- `gos_result_t gos_queueGet (gos_queueId_t queueId, void_t *target, gos_queueLength_t targetSize, u32_t timeout)`  
*This function gets the next element from the given queue.*
- `gos_result_t gos_queuePeek (gos_queueId_t queueId, void_t *target, gos_queueLength_t targetSize, u32_t timeout)`  
*This function gets the next element from the given queue without removing it.*
- `gos_result_t gos_queueReset (gos_queueId_t queueId)`  
*Resets the given queue.*
- `gos_result_t gos_queueRegisterFullHook (gos_queueFullHook fullHook)`  
*This function registers a queue full hook function.*
- `gos_result_t gos_queueRegisterEmptyHook (gos_queueEmptyHook emptyHook)`  
*This function registers a queue empty hook function.*
- `gos_result_t gos_queueGetName (gos_queueId_t queueId, gos_queueName_t queueName)`  
*This function gets the name of the given queue.*
- `gos_result_t gos_queueGetElementNumber (gos_queueId_t queueId, gos_queueIndex_t *elementNumber)`  
*This function gets the number of elements in the given queue.*
- `void_t gos_queueDump (void_t)`  
*Queue dump.*

#### 4.22.1 Detailed Description

GOS queue service header.

##### Author

Ahmed Gazar

##### Date

2024-04-02

##### Version

1.6

Queue service is one of the inter-task communication solutions offered by the OS. A queue can hold a given number of elements with a given element size. The queue is implemented as a FIFO. Queues are unsafe in a way that any task with the queue ID in their knowledge can put and get elements to/from the queue. So queues are most suitable for serializing data that are coming from multiple resources but processes by a single task. It is also possible to peek the queue, meaning that the next element of the queue can be requested without deleting it from the queue. This way multiple tasks can process data coming from a queue in a cooperative way, but they rely on each others data processing.

Definition in file [gos\\_queue.h](#).

#### 4.22.2 Macro Definition Documentation

##### 4.22.2.1 `#define GOS_DEFAULT_QUEUE_ID ( (gos_queueId_t) 0x3000 )`

Default queue ID.

Definition at line 77 of file [gos\\_queue.h](#).

4.22.2.2 `#define GOS_INVALID_QUEUE_ID ( (gos_queueId_t) 0x0300 )`

Invalid queue ID.

Definition at line 82 of file gos\_queue.h.

### 4.22.3 Typedef Documentation

4.22.3.1 `typedef u8_t gos_queueByte_t`

One byte in a queue element.

Definition at line 90 of file gos\_queue.h.

4.22.3.2 `typedef void_t(* gos_queueEmptyHook)(gos_queueId_t)`

Queue full hook type.

Definition at line 116 of file gos\_queue.h.

4.22.3.3 `typedef void_t(* gos_queueFullHook)(gos_queueId_t)`

Queue full hook type.

Definition at line 111 of file gos\_queue.h.

### 4.22.4 Function Documentation

4.22.4.1 `gos_result_t gos_queueCreate( gos_queueDescriptor_t * pQueueDescriptor )`

This function creates a new queue.

This function loops through the internal queue array and registers the new queue in the next free slot.

#### Parameters

|                               |                                                         |
|-------------------------------|---------------------------------------------------------|
| <code>pQueueDescriptor</code> | : Pointer to queue descriptor variable with queue data. |
|-------------------------------|---------------------------------------------------------|

**Returns**

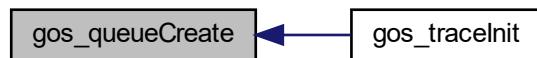
Result of queue creation.

**Return values**

|                          |                                                            |
|--------------------------|------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Queue creation successful.                               |
| <code>GOS_ERROR</code>   | : Queue descriptor is NULL pointer or queue array is full. |

Definition at line 181 of file gos\_queue.c.

Here is the caller graph for this function:



#### 4.22.4.2 `void_t gos_queueDump( void_t )`

Queue dump.

Prints the queue data of all queues to the trace output.

**Returns**

-

Definition at line 598 of file gos\_queue.c.

Here is the call graph for this function:



Here is the caller graph for this function:



**4.22.4.3 gos\_result\_t gos\_queueGet( gos\_queueId\_t queueId, void\_t \*target, gos\_queueLength\_t targetSize, u32\_t timeout )**

This function gets the next element from the given queue.

This function checks the queue state and gets the next element from the queue.

#### Parameters

|                   |                                    |
|-------------------|------------------------------------|
| <i>queueId</i>    | : Queue ID.                        |
| <i>target</i>     | : Pointer to target variable.      |
| <i>targetSize</i> | : Size of target.                  |
| <i>timeout</i>    | : Timeout for locking queue mutex. |

#### Returns

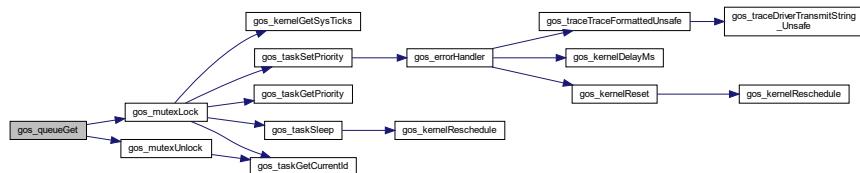
Result of element getting.

#### Return values

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Element successfully moved from queue to target.         |
| <i>GOS_ERROR</i>   | : Invalid queue ID, invalid target size or queue is empty. |

Definition at line 310 of file gos\_queue.c.

Here is the call graph for this function:



Here is the caller graph for this function:



**4.22.4.4 gos\_result\_t gos\_queueGetElementNumber( gos\_queueId\_t queueId, gos\_queueIndex\_t \*elementNumber )**

This function gets the number of elements in the given queue.

This function copies the number of elements in the queue belonging to the given ID to the given variable.

**Parameters**

|                      |                                                       |
|----------------------|-------------------------------------------------------|
| <i>queueId</i>       | : Queue ID.                                           |
| <i>elementNumber</i> | : Pointer to variable to store the element number in. |

**Returns**

Result of queue element number getting.

**Return values**

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Element number getting successful.                   |
| <i>GOS_ERROR</i>   | : Invalid queue ID or element number variable is NULL. |

Definition at line 564 of file gos\_queue.c.

**4.22.4.5 gos\_result\_t gos\_queueGetName ( gos\_queueId\_t queueId, gos\_queueName\_t queueName )**

This function gets the name of the given queue.

This function copies the name of the queue belonging to the given ID to the given variable.

**Parameters**

|                  |               |
|------------------|---------------|
| <i>queueId</i>   | : Queue ID.   |
| <i>queueName</i> | : Queue name. |

**Returns**

Result of queue name getting.

**Return values**

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Name getting successful.                         |
| <i>GOS_ERROR</i>   | : Invalid queue ID or queue name variable is NULL. |

Definition at line 525 of file gos\_queue.c.

**4.22.4.6 gos\_result\_t gos\_queueInit( void\_t )**

This function initializes the queue service.

Initializes the internal queue array, creates the queue lock, and registers the queue dump task in the kernel.

**Returns**

Result of initialization.

**Return values**

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Initialization successful.                                 |
| <i>GOS_ERROR</i>   | : Lock creation, task registration or task suspension error. |

Definition at line 147 of file gos\_queue.c.

Here is the call graph for this function:



#### 4.22.4.7 *gos\_result\_t gos\_queuePeek( gos\_queueld\_t queueId, void\_t \* target, gos\_queueLength\_t targetSize, u32\_t timeout )*

This function gets the next element from the given queue without removing it.

This function checks the queue state and returns the next element from the queue without modifying the queue counters.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <i>queueId</i>    | : Queue ID.                        |
| <i>target</i>     | : Pointer to target variable.      |
| <i>targetSize</i> | : Size of target.                  |
| <i>timeout</i>    | : Timeout for locking queue mutex. |

**Returns**

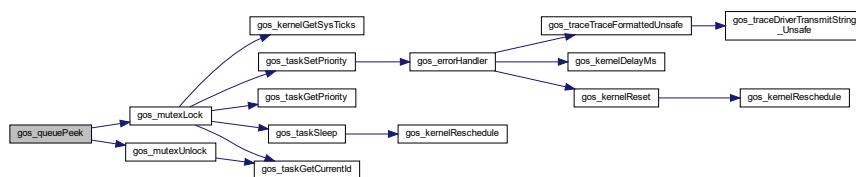
Result of element getting.

**Return values**

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Element successfully copied from queue to target.        |
| <i>GOS_ERROR</i>   | : Invalid queue ID, invalid target size or queue is empty. |

Definition at line 385 of file gos\_queue.c.

Here is the call graph for this function:



#### 4.22.4.8 `gos_result_t gos_queuePut( gos_queueId_t queueId, void_t *element, gos_queueLength_t elementSize, u32_t timeout )`

This function puts an element in the given queue.

This function checks the queue state and places the given element in the next queue element.

##### Parameters

|                          |                                    |
|--------------------------|------------------------------------|
| <code>queueId</code>     | : Queue ID.                        |
| <code>element</code>     | : Pointer to element.              |
| <code>elementSize</code> | : Size of element.                 |
| <code>timeout</code>     | : Timeout for locking queue mutex. |

##### Returns

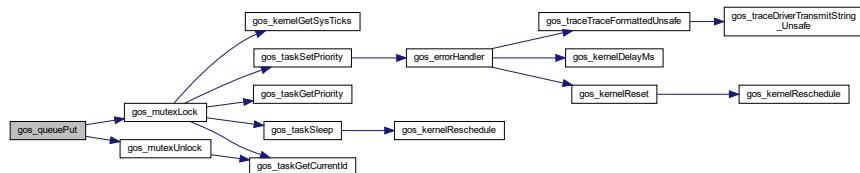
Result of element putting.

##### Return values

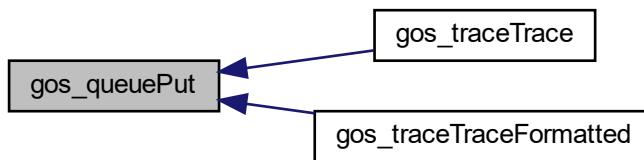
|                          |                                                            |
|--------------------------|------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Element successfully put in the queue.                   |
| <code>GOS_ERROR</code>   | : Invalid queue ID, invalid element size or queue is full. |

Definition at line 231 of file gos\_queue.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.22.4.9 `gos_result_t gos_queueRegisterEmptyHook( gos_queueEmptyHook emptyHook )`

This function registers a queue empty hook function.

This function checks whether a hook has been already registered, and if not, it saves the given hook function.

**Parameters**

|                  |                  |
|------------------|------------------|
| <i>emptyHook</i> | : Hook function. |
|------------------|------------------|

**Returns**

Result of hook registration.

**Return values**

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Hook registration successful.                     |
| <i>GOS_ERROR</i>   | : Hook already exists or parameter is NULL pointer. |

Definition at line 499 of file gos\_queue.c.

**4.22.4.10 gos\_result\_t gos\_queueRegisterFullHook( gos\_queueFullHook *fullHook* )**

This function registers a queue full hook function.

This function checks whether a hook has been already registered, and if not, it saves the given hook function.

**Parameters**

|                 |                  |
|-----------------|------------------|
| <i>fullHook</i> | : Hook function. |
|-----------------|------------------|

**Returns**

Result of hook registration.

**Return values**

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Hook registration successful.                     |
| <i>GOS_ERROR</i>   | : Hook already exists or parameter is NULL pointer. |

Definition at line 473 of file gos\_queue.c.

**4.22.4.11 gos\_result\_t gos\_queueReset( gos\_queueId\_t *queueId* )**

Resets the given queue.

Sets the read and write counter to zero, making the queue empty.

**Parameters**

|                |             |
|----------------|-------------|
| <i>queueId</i> | : Queue ID. |
|----------------|-------------|

**Returns**

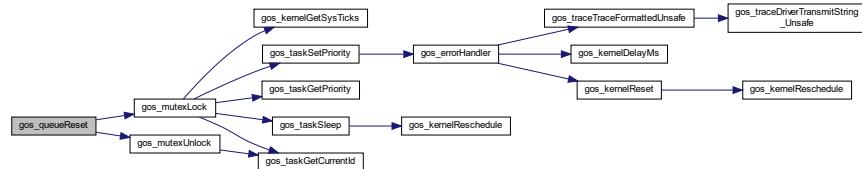
Result of queue resetting.

**Return values**

|                    |                     |
|--------------------|---------------------|
| <i>GOS_SUCCESS</i> | : Reset successful. |
| <i>GOS_ERROR</i>   | : Invalid queue ID. |

Definition at line 435 of file gos\_queue.c.

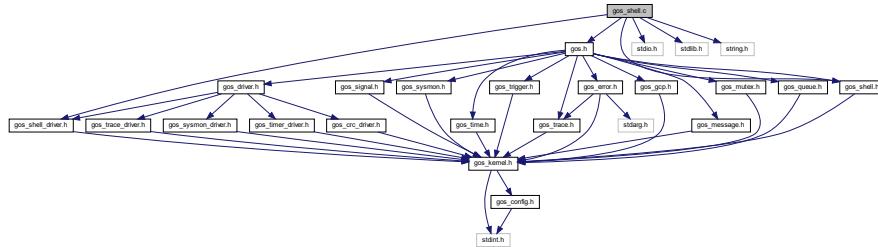
Here is the call graph for this function:



## 4.23 gos\_shell.c File Reference

GOS shell service source.

```
#include <gos.h>
#include <gos_shell.h>
#include <gos_shell_driver.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
Include dependency graph for gos_shell.c:
```



## Macros

- `#define GOS_SHELL_DAEMON_POLL_TIME_MS ( 50u )`
- `#define GOS_SHELL_DISPLAY_TEXT ("[\x1B[1m\x1B[33mgos shell\x1B[0m]>> ")`

## Functions

- `GOS_STATIC void_t gos_shellDaemonTask (void_t)`  
*Shell daemon task.*
- `GOS_STATIC void_t gos_shellCommandHandler (char_t *params)`  
*Shell command handler.*
- `gos_result_t gos_shellInit (void_t)`  
*This function initializes the shell service.*
- `gos_result_t gos_shellRegisterCommands (gos_shellCommand_t *commands, u16_t arraySize)`  
*This function registers an array of commands in the shell.*
- `gos_result_t gos_shellRegisterCommand (gos_shellCommand_t *command)`  
*This function registers a command in the shell.*
- `gos_result_t gos_shellSuspend (void_t)`  
*Suspends the shell daemon task.*

- [gos\\_result\\_t gos\\_shellResume \(void\\_t\)](#)  
*Resumes the shell daemon task.*
- [gos\\_result\\_t gos\\_shellEchoOn \(void\\_t\)](#)  
*Turns the shell echoing on.*
- [gos\\_result\\_t gos\\_shellEchoOff \(void\\_t\)](#)  
*Turns the shell echoing off.*

## Variables

- [GOS\\_STATIC gos\\_shellCommand\\_t shellCommands \[CFG\\_SHELL\\_MAX\\_COMMAND\\_NUMBER\]](#)
- [GOS\\_STATIC gos\\_tid\\_t shellDaemonTaskId](#)
- [GOS\\_STATIC char\\_t commandBuffer \[CFG\\_SHELL\\_COMMAND\\_BUFFER\\_SIZE\]](#)
- [GOS\\_STATIC u16\\_t commandBufferIndex](#)
- [GOS\\_STATIC bool\\_t useEcho](#)
- [GOS\\_STATIC char\\_t actualCommand \[CFG\\_SHELL\\_MAX\\_COMMAND\\_LENGTH\]](#)
- [GOS\\_STATIC char\\_t commandParams \[CFG\\_SHELL\\_MAX\\_PARAMS\\_LENGTH\]](#)
- [GOS\\_STATIC gos\\_taskDescriptor\\_t shellDaemonTaskDesc](#)
- [GOS\\_STATIC gos\\_shellCommand\\_t shellCommand](#)

### 4.23.1 Detailed Description

GOS shell service source.

#### Author

Ahmed Gazar

#### Date

2024-06-28

#### Version

1.9

For a more detailed description of this service, please refer to [gos\\_shell.h](#)

Definition in file [gos\\_shell.c](#).

### 4.23.2 Macro Definition Documentation

#### 4.23.2.1 #define GOS\_SHELL\_DAEMON\_POLL\_TIME\_MS ( 50u )

Shell daemon poll time [ms].

Definition at line 77 of file [gos\\_shell.c](#).

#### 4.23.2.2 #define GOS\_SHELL\_DISPLAY\_TEXT ("[\x1B[1m\x1B[33mgos shell\x1B[0m]>> ")

Shell display text.

Definition at line 82 of file [gos\\_shell.c](#).

### 4.23.3 Function Documentation

#### 4.23.3.1 GOS\_STATIC void\_t gos\_shellCommandHandler( char\_t \* params )

Shell command handler.

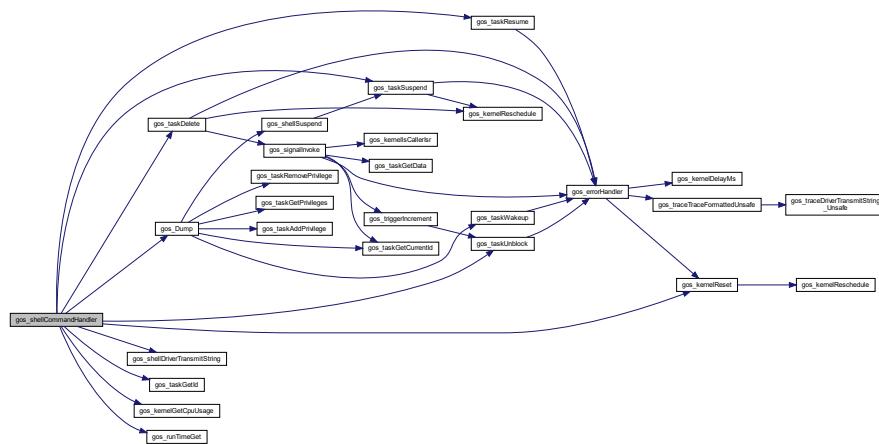
Handles the built-in shell command.

##### Returns

-

Definition at line 480 of file gos\_shell.c.

Here is the call graph for this function:



#### 4.23.3.2 GOS\_STATIC void\_t gos\_shellDaemonTask( void\_t )

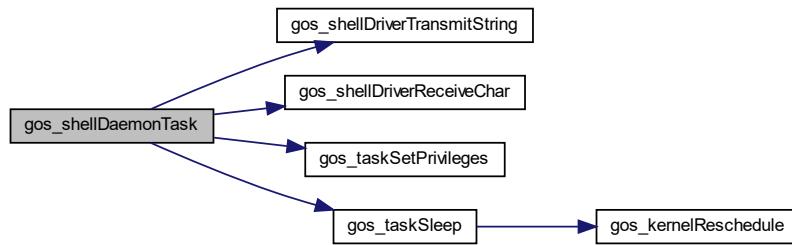
Shell daemon task.

Receives a character from the log serial line, if the echoing is on, then sends the same character back. When an enter key is received, it processes the command typed in, and loops through the command array. When the command is found, it calls the command handler function with the parameter list as a string.

**Returns**

-  
Definition at line 349 of file gos\_shell.c.

Here is the call graph for this function:

**4.23.3.3 gos\_result\_t gos\_shellEchoOff( void\_t )**

Turns the shell echoing off.

Resets the internal echo flag.

**Returns**

Result of turning echoing off.

**Return values**

|                    |                                    |
|--------------------|------------------------------------|
| <i>GOS_SUCCESS</i> | : Echoing turned off successfully. |
|--------------------|------------------------------------|

Definition at line 324 of file gos\_shell.c.

**4.23.3.4 gos\_result\_t gos\_shellEchoOn( void\_t )**

Turns the shell echoing on.

Sets the internal echo flag.

**Returns**

Result of turning echoing on.

**Return values**

|                    |                                   |
|--------------------|-----------------------------------|
| <i>GOS_SUCCESS</i> | : Echoing turned on successfully. |
|--------------------|-----------------------------------|

Definition at line 306 of file gos\_shell.c.

**4.23.3.5 gos\_result\_t gos\_shellInit( void\_t )**

This function initializes the shell service.

Initializes the internal command structure, and registers the shell daemon task in the kernel.

## Returns

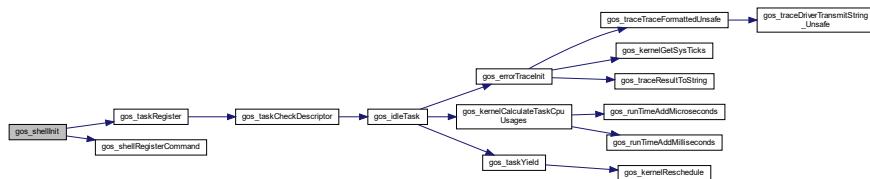
## Result of initialization.

## Return values

|                    |                                          |
|--------------------|------------------------------------------|
| <i>GOS_SUCCESS</i> | : Shell initialization successful.       |
| <i>GOS_ERROR</i>   | : Shell daemon task registration failed. |

Definition at line 153 of file gos\_shell.c.

Here is the call graph for this function:



4.23.3.6 gos\_result\_t gos\_shellRegisterCommand ( gos\_shellCommand\_t \* *command* )

This function registers a command in the shell.

Checks the command structure and registers the command in the next empty slot in the internal command array.

## Parameters

*command* : Pointer to the command structure to register.

## Returns

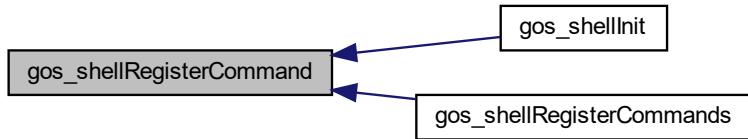
Result of shell command registration.

## Return values

|                    |                                                                       |
|--------------------|-----------------------------------------------------------------------|
| <b>GOS_SUCCESS</b> | : Command registration successful.                                    |
| <b>GOS_ERROR</b>   | : Command function or name is NULL or internal command array is full. |

Definition at line 235 of file gos\_shell.c.

Here is the caller graph for this function:



4.23.3.7 gos\_result\_t gos\_shellRegisterCommands ( gos\_shellCommand\_t \* *commands*, u16\_t *arraySize* )

This function registers an array of commands in the shell.

Checks the array pointer and registers the commands one by one.

**Parameters**

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>commands</i>  | : Pointer to the command structure array to register. |
| <i>arraySize</i> | : Size of the array in bytes.                         |

**Returns**

Result of shell command registration.

**Return values**

|                    |                                                                       |
|--------------------|-----------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Command registration successful.                                    |
| <i>GOS_ERROR</i>   | : Command function or name is NULL or internal command array is full. |

Definition at line 187 of file gos\_shell.c.

Here is the call graph for this function:

**4.23.3.8 gos\_result\_t gos\_shellResume( void\_t )**

Resumes the shell daemon task.

Resumes the shell daemon task.

**Returns**

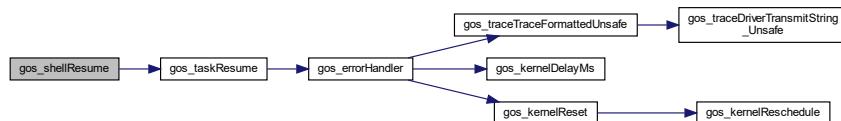
Result of shell resumption.

**Return values**

|                    |                               |
|--------------------|-------------------------------|
| <i>GOS_SUCCESS</i> | : Shell resumed successfully. |
| <i>GOS_ERROR</i>   | : Task resumption failed.     |

Definition at line 288 of file gos\_shell.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.23.3.9 gos\_result\_t gos\_shellSuspend( void\_t )

Suspends the shell daemon task.

Suspends the shell daemon task.

##### Returns

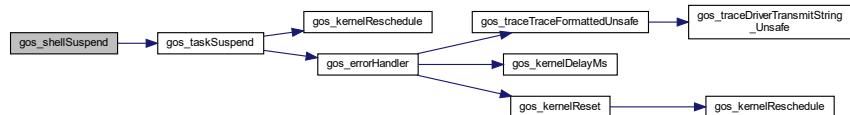
Result of shell suspension.

##### Return values

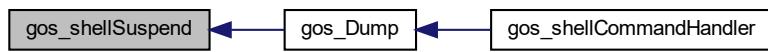
|                    |                                 |
|--------------------|---------------------------------|
| <i>GOS_SUCCESS</i> | : Shell suspended successfully. |
| <i>GOS_ERROR</i>   | : Task suspension failed.       |

Definition at line 270 of file gos\_shell.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.23.4 Variable Documentation

##### 4.23.4.1 GOS\_STATIC char\_t actualCommand[CFG\_SHELL\_MAX\_COMMAND\_LENGTH]

Actual command buffer.

Definition at line 115 of file gos\_shell.c.

#### 4.23.4.2 GOS\_STATIC char\_t commandBuffer[CFG\_SHELL\_COMMAND\_BUFFER\_SIZE]

Command buffer.

Definition at line 100 of file gos\_shell.c.

#### 4.23.4.3 GOS\_STATIC u16\_t commandBufferIndex

Command buffer index.

Definition at line 105 of file gos\_shell.c.

#### 4.23.4.4 GOS\_STATIC char\_t commandParams[CFG\_SHELL\_MAX\_PARAMS\_LENGTH]

Command parameters buffer.

Definition at line 120 of file gos\_shell.c.

#### 4.23.4.5 GOS\_STATIC gos\_shellCommand\_t shellCommand

##### Initial value:

```
=
{
 .command = "shell",
 .commandHandler = gos_shellCommandHandler,
 .commandHandlerPrivileges = GOS_TASK_PRIVILEGE_KERNEL
}
```

Shell info command.

Definition at line 143 of file gos\_shell.c.

#### 4.23.4.6 GOS\_STATIC gos\_shellCommand\_t shellCommands[CFG\_SHELL\_MAX\_COMMAND\_NUMBER]

Shell command array.

Definition at line 90 of file gos\_shell.c.

#### 4.23.4.7 GOS\_STATIC gos\_taskDescriptor\_t shellDaemonTaskDesc

##### Initial value:

```
=
{
 .taskFunction = gos_shellDaemonTask,
 .taskName = "gos_shell_daemon",
 .taskPriority = CFG_TASK_SHELL_DAEMON_PRIO,
 .taskStackSize = CFG_TASK_SHELL_DAEMON_STACK,
 .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL
}
```

Shell daemon task descriptor.

Definition at line 131 of file gos\_shell.c.

#### 4.23.4.8 GOS\_STATIC gos\_tid\_t shellDaemonTaskId

Shell daemon task ID.

Definition at line 95 of file gos\_shell.c.

## 4.23.4.9 GOS\_STATIC bool\_t useEcho

Shell echo flag.

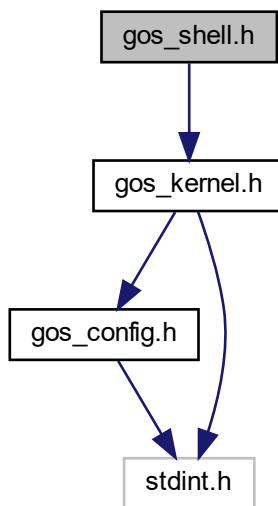
Definition at line 110 of file gos\_shell.c.

## 4.24 gos\_shell.h File Reference

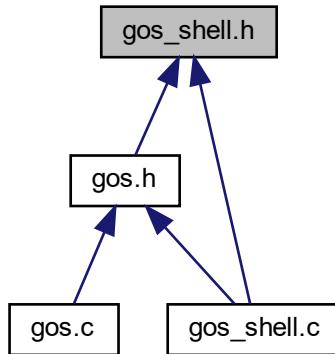
GOS shell service header.

```
#include <gos_kernel.h>
```

Include dependency graph for gos\_shell.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `gos_shellCommand_t`

## Typedefs

- `typedef void_t(* gos_shellFunction )(char_t *params)`
- `typedef u8_t gos_shellCommandIndex_t`

*Shell command index type.*

## Functions

- `gos_result_t gos_shellInit (void_t)`  
*This function initializes the shell service.*
- `gos_result_t gos_shellRegisterCommands (gos_shellCommand_t *commands, u16_t arraySize)`  
*This function registers an array of commands in the shell.*
- `gos_result_t gos_shellRegisterCommand (gos_shellCommand_t *command)`  
*This function registers a command in the shell.*
- `gos_result_t gos_shellSuspend (void_t)`  
*Suspends the shell daemon task.*
- `gos_result_t gos_shellResume (void_t)`  
*Resumes the shell daemon task.*
- `gos_result_t gos_shellEchoOn (void_t)`  
*Turns the shell echoing on.*
- `gos_result_t gos_shellEchoOff (void_t)`  
*Turns the shell echoing off.*

### 4.24.1 Detailed Description

GOS shell service header.

**Author**

Ahmed Gazar

**Date**

2023-07-12

**Version**

1.3

The shell service provides an easy interface to receive and process commands in a terminal. Optionally the characters can be echoed back. The user can register different commands with callback functions. When the enter key is hit, the shell daemon processes the input. If it finds the input string in the command array, it calls the corresponding callback function and passes the parameter list as a string to the callback for further processing.

Definition in file [gos\\_shell.h](#).

## 4.24.2 Typedef Documentation

### 4.24.2.1 `typedef void_t(* gos_shellFunction)(char_t *params)`

Shell function type.

Definition at line 70 of file [gos\\_shell.h](#).

## 4.24.3 Function Documentation

### 4.24.3.1 `gos_result_t gos_shellEchoOff( void_t )`

Turns the shell echoing off.

Resets the internal echo flag.

**Returns**

Result of turning echoing off.

**Return values**

|                          |                                    |
|--------------------------|------------------------------------|
| <code>GOS_SUCCESS</code> | : Echoing turned off successfully. |
|--------------------------|------------------------------------|

Definition at line 324 of file [gos\\_shell.c](#).

### 4.24.3.2 `gos_result_t gos_shellEchoOn( void_t )`

Turns the shell echoing on.

Sets the internal echo flag.

**Returns**

Result of turning echoing on.

**Return values**

|                    |                                   |
|--------------------|-----------------------------------|
| <i>GOS_SUCCESS</i> | : Echoing turned on successfully. |
|--------------------|-----------------------------------|

Definition at line 306 of file gos\_shell.c.

**4.24.3.3 gos\_result\_t gos\_shellInit ( void\_t )**

This function initializes the shell service.

Initializes the internal command structure, and registers the shell daemon task in the kernel.

**Returns**

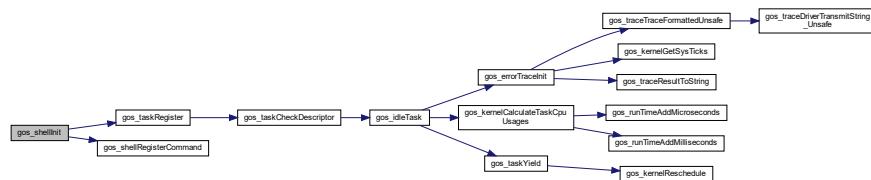
Result of initialization.

**Return values**

|                    |                                          |
|--------------------|------------------------------------------|
| <i>GOS_SUCCESS</i> | : Shell initialization successful.       |
| <i>GOS_ERROR</i>   | : Shell daemon task registration failed. |

Definition at line 153 of file gos\_shell.c.

Here is the call graph for this function:

**4.24.3.4 gos\_result\_t gos\_shellRegisterCommand ( gos\_shellCommand\_t \* command )**

This function registers a command in the shell.

Checks the command structure and registers the command in the next empty slot in the internal command array.

**Parameters**

|                |                                                 |
|----------------|-------------------------------------------------|
| <i>command</i> | : Pointer to the command structure to register. |
|----------------|-------------------------------------------------|

**Returns**

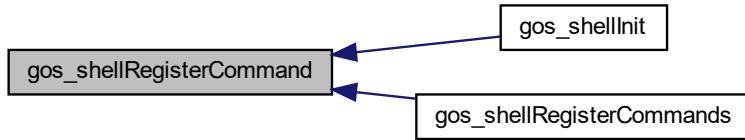
Result of shell command registration.

**Return values**

|                    |                                                                       |
|--------------------|-----------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Command registration successful.                                    |
| <i>GOS_ERROR</i>   | : Command function or name is NULL or internal command array is full. |

Definition at line 235 of file gos\_shell.c.

Here is the caller graph for this function:



#### 4.24.3.5 gos\_result\_t gos\_shellRegisterCommands ( gos\_shellCommand\_t \* commands, u16\_t arraySize )

This function registers an array of commands in the shell.

Checks the array pointer and registers the commands one by one.

##### Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>commands</i>  | : Pointer to the command structure array to register. |
| <i>arraySize</i> | : Size of the array in bytes.                         |

##### Returns

Result of shell command registration.

##### Return values

|                    |                                                                       |
|--------------------|-----------------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : Command registration successful.                                    |
| <i>GOS_ERROR</i>   | : Command function or name is NULL or internal command array is full. |

Definition at line 187 of file gos\_shell.c.

Here is the call graph for this function:



#### 4.24.3.6 gos\_result\_t gos\_shellResume ( void\_t )

Resumes the shell daemon task.

Resumes the shell daemon task.

##### Returns

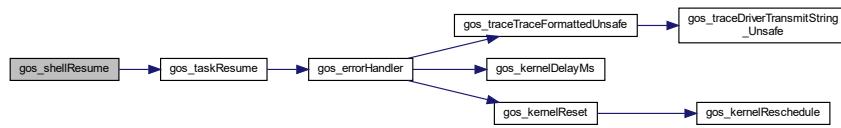
Result of shell resumption.

**Return values**

|                    |                               |
|--------------------|-------------------------------|
| <i>GOS_SUCCESS</i> | : Shell resumed successfully. |
| <i>GOS_ERROR</i>   | : Task resumption failed.     |

Definition at line 288 of file gos\_shell.c.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.24.3.7 gos\_result\_t gos\_shellSuspend( void\_t )**

Suspends the shell daemon task.

Suspends the shell daemon task.

**Returns**

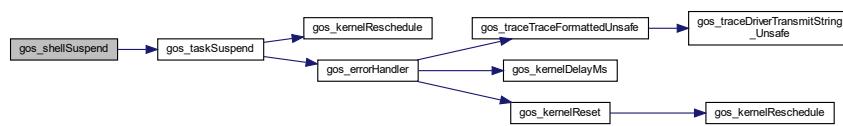
Result of shell suspension.

**Return values**

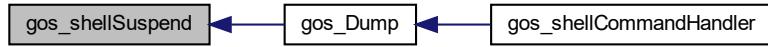
|                    |                                 |
|--------------------|---------------------------------|
| <i>GOS_SUCCESS</i> | : Shell suspended successfully. |
| <i>GOS_ERROR</i>   | : Task suspension failed.       |

Definition at line 270 of file gos\_shell.c.

Here is the call graph for this function:



Here is the caller graph for this function:

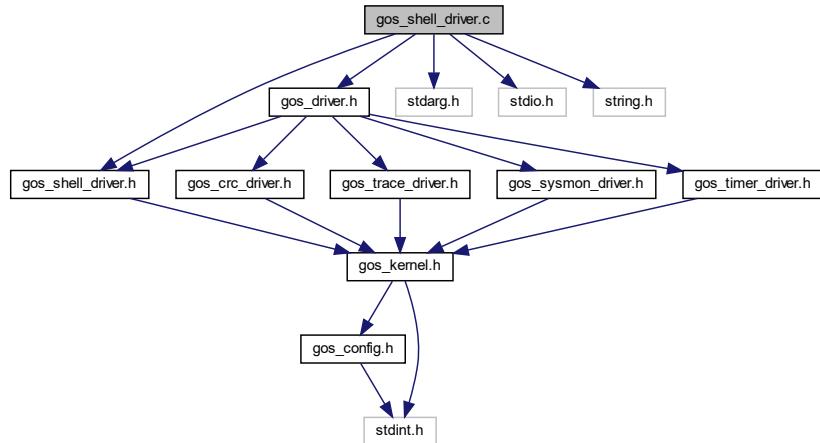


## 4.25 gos\_shell\_driver.c File Reference

GOS SHELL driver source.

```
#include "gos_shell_driver.h"
#include "gos_driver.h"
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for gos\_shell\_driver.c:



## Functions

- [gos\\_result\\_t gos\\_shellDriverReceiveChar \(char\\_t \\*pChar\)](#)  
*Receives a character.*
- [gos\\_result\\_t gos\\_shellDriverTransmitString \(char\\_t \\*pString....\)](#)  
*Transmits a string.*

## Variables

- [GOS\\_STATIC char\\_t formattedBuffer \[CFG\\_SHELL\\_COMMAND\\_BUFFER\\_SIZE\]](#)
- [GOS\\_EXTERN gos\\_driver\\_functions\\_t driverFunctions](#)

### 4.25.1 Detailed Description

GOS SHELL driver source.

#### Author

Ahmed Gazar

#### Date

2023-06-17

#### Version

1.1

For a more detailed description of this driver, please refer to [gos\\_shell\\_driver.h](#)

Definition in file [gos\\_shell\\_driver.c](#).

### 4.25.2 Function Documentation

#### 4.25.2.1 gos\_result\_t gos\_shellDriverReceiveChar ( char\_t \* pChar )

Receives a character.

If registered, it calls the custom character receiver function.

#### Parameters

|              |                                                                       |
|--------------|-----------------------------------------------------------------------|
| <i>pChar</i> | : Pointer to a character variable to store the received character in. |
|--------------|-----------------------------------------------------------------------|

#### Returns

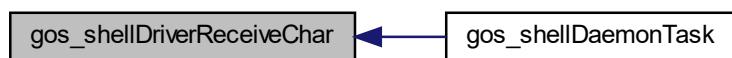
Result of character reception.

#### Return values

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : According to user implementation.                           |
| <i>GOS_ERROR</i>   | : According to user implementation / function not registered. |

Definition at line 75 of file [gos\\_shell\\_driver.c](#).

Here is the caller graph for this function:



#### 4.25.2.2 gos\_result\_t gos\_shellDriverTransmitString ( char\_t \* pString, ... )

Transmits a string.

If registered, it calls the custom string transmitter function.

**Parameters**

|                |                                              |
|----------------|----------------------------------------------|
| <i>pString</i> | : Pointer to the string to be transmitted.   |
| ...            | : Variable parameters for formatted strings. |

**Returns**

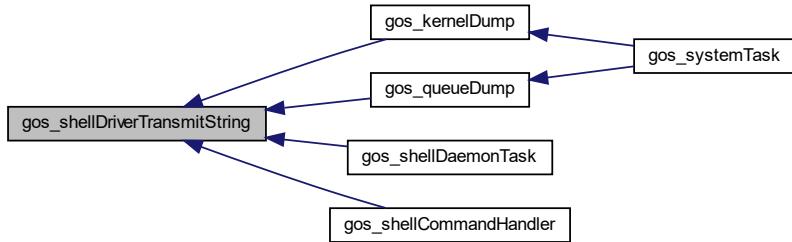
Result of string transmission.

**Return values**

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : According to user implementation.                           |
| <i>GOS_ERROR</i>   | : According to user implementation / function not registered. |

Definition at line 100 of file gos\_shell\_driver.c.

Here is the caller graph for this function:

**4.25.3 Variable Documentation****4.25.3.1 GOS\_STATIC char\_t formattedBuffer[CFG\_SHELL\_COMMAND\_BUFFER\_SIZE]**

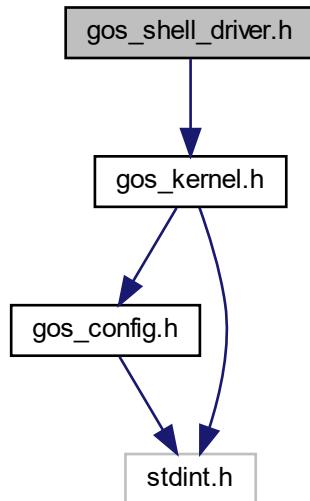
Buffer for formatted strings.

Definition at line 65 of file gos\_shell\_driver.c.

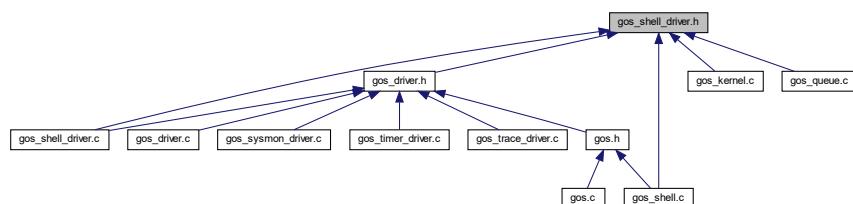
**4.26 gos\_shell\_driver.h File Reference**

GOS SHELL driver header.

```
#include "gos_kernel.h"
Include dependency graph for gos_shell_driver.h:
```



This graph shows which files directly or indirectly include this file:



## Typedefs

- `typedef gos_result_t(* gos_shellDriverReceiveChar_t )(char_t *)`
- `typedef gos_result_t(* gos_shellDriverTransmitString_t )(char_t *)`

## Functions

- `gos_result_t gos_shellDriverReceiveChar (char_t *pChar)`  
*Receives a character.*
- `gos_result_t gos_shellDriverTransmitString (char_t *pString,...)`  
*Transmits a string.*

### 4.26.1 Detailed Description

GOS SHELL driver header.

**Author**

Ahmed Gazar

**Date**

2023-06-17

**Version**

1.1

This driver provides a skeleton for the driver for the shell service.

Definition in file [gos\\_shell\\_driver.h](#).

## 4.26.2 Typedef Documentation

### 4.26.2.1 `typedef gos_result_t(* gos_shellDriverReceiveChar_t)(char_t *)`

Shell driver receive character function type.

Definition at line 63 of file gos\_shell\_driver.h.

### 4.26.2.2 `typedef gos_result_t(* gos_shellDriverTransmitString_t)(char_t *)`

Shell driver transmit string function type.

Definition at line 68 of file gos\_shell\_driver.h.

## 4.26.3 Function Documentation

### 4.26.3.1 `gos_result_t gos_shellDriverReceiveChar( char_t * pChar )`

Receives a character.

If registered, it calls the custom character receiver function.

**Parameters**

|                    |                                                                       |
|--------------------|-----------------------------------------------------------------------|
| <code>pChar</code> | : Pointer to a character variable to store the received character in. |
|--------------------|-----------------------------------------------------------------------|

**Returns**

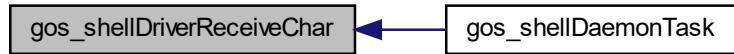
Result of character reception.

**Return values**

|                          |                                                               |
|--------------------------|---------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : According to user implementation.                           |
| <code>GOS_ERROR</code>   | : According to user implementation / function not registered. |

Definition at line 75 of file gos\_shell\_driver.c.

Here is the caller graph for this function:



#### 4.26.3.2 gos\_result\_t gos\_shellDriverTransmitString ( char\_t \* pString, ... )

Transmits a string.

If registered, it calls the custom string transmitter function.

##### Parameters

|                |                                              |
|----------------|----------------------------------------------|
| <i>pString</i> | : Pointer to the string to be transmitted.   |
| ...            | : Variable parameters for formatted strings. |

##### Returns

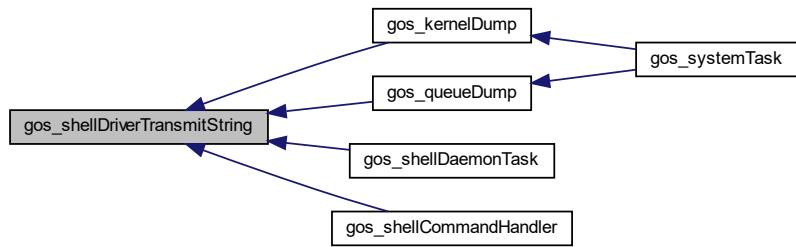
Result of string transmission.

##### Return values

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>GOS_SUCCESS</i> | : According to user implementation.                           |
| <i>GOS_ERROR</i>   | : According to user implementation / function not registered. |

Definition at line 100 of file gos\_shell\_driver.c.

Here is the caller graph for this function:

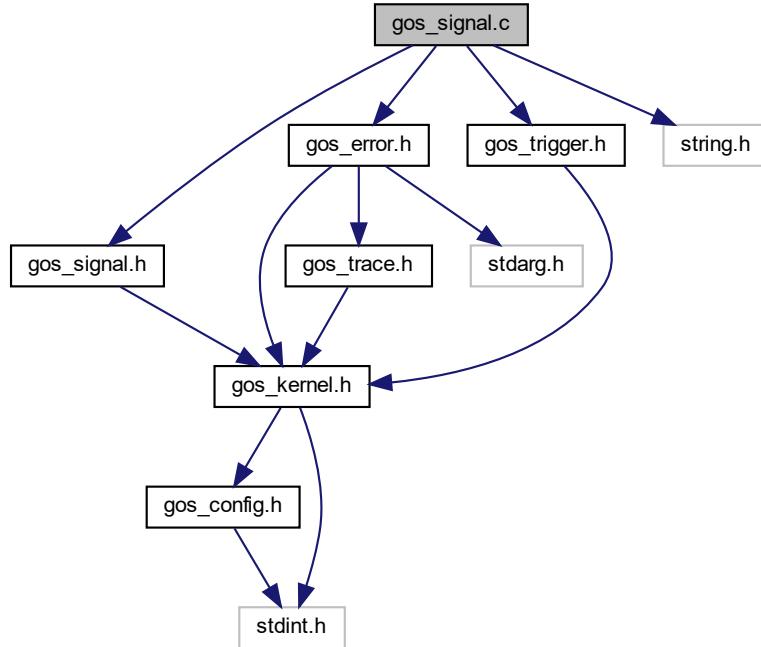


## 4.27 gos\_signal.c File Reference

GOS signal service source.

```
#include <gos_signal.h>
#include <gos_error.h>
#include <gos_trigger.h>
#include <string.h>
```

Include dependency graph for gos\_signal.c:



## Data Structures

- struct [gos\\_signalDescriptor\\_t](#)
- struct [gos\\_signallInvokeDescriptor](#)

## Functions

- [GOS\\_STATIC void\\_t gos\\_signalDaemonTask \(void\\_t\)](#)  
*Signal daemon task.*
- [gos\\_result\\_t gos\\_signallInit \(void\\_t\)](#)  
*Initializes the signal service.*
- [gos\\_result\\_t gos\\_signalCreate \(gos\\_signallId\\_t \\*pSignal\)](#)  
*Creates a new signal.*
- [gos\\_result\\_t gos\\_signalSubscribe \(gos\\_signallId\\_t signallId, gos\\_signalHandler\\_t signalHandler, gos\\_taskPrivilegeLevel\\_t signalHandlerPrivileges\)](#)  
*Subscribes to the given signal.*
- [GOS\\_INLINE gos\\_result\\_t gos\\_signallInvoke \(gos\\_signallId\\_t signallId, gos\\_signalSenderId\\_t senderId\)](#)  
*Invokes the given signal.*

## Variables

- [GOS\\_STATIC gos\\_signalDescriptor\\_t signalArray \[CFG\\_SIGNAL\\_MAX\\_NUMBER\]](#)
- [GOS\\_STATIC gos\\_trigger\\_t signallInvokeTrigger](#)
- [GOS\\_EXTERN gos\\_signallId\\_t kernelTaskDeleteSignal](#)
- [GOS\\_EXTERN gos\\_signallId\\_t kernelDumpReadySignal](#)
- [GOS\\_STATIC gos\\_taskDescriptor\\_t signalDaemonTaskDescriptor](#)

#### 4.27.1 Detailed Description

GOS signal service source.

**Author**

Ahmed Gazar

**Date**

2023-10-04

**Version**

1.8

For a more detailed description of this service, please refer to [gos\\_signal.h](#)

Definition in file [gos\\_signal.c](#).

#### 4.27.2 Function Documentation

##### 4.27.2.1 `gos_result_t gos_signalCreate( gos_signalId_t * pSignal )`

Creates a new signal.

Finds the next free slot in the signal array and registers the new signal there.

**Parameters**

|                      |                                   |
|----------------------|-----------------------------------|
| <code>pSignal</code> | : Pointer to a signal identifier. |
|----------------------|-----------------------------------|

**Returns**

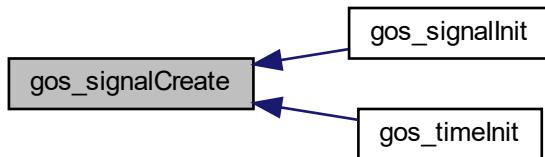
Success of signal creation.

**Return values**

|                          |                        |
|--------------------------|------------------------|
| <code>GOS_SUCCESS</code> | : Creation successful. |
| <code>GOS_ERROR</code>   | : Signal array full.   |

Definition at line 168 of file [gos\\_signal.c](#).

Here is the caller graph for this function:



#### 4.27.2.2 GOS\_STATIC void\_t gos\_signalDaemonTask( void\_t )

Signal daemon task.

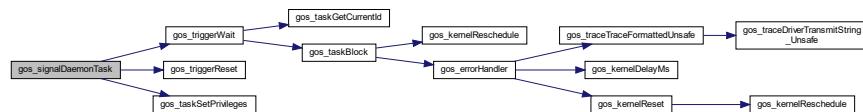
Polls the signal invoke queue, and completes the necessary signal invokings.

**Returns**

-

Definition at line 294 of file gos\_signal.c.

Here is the call graph for this function:



#### 4.27.2.3 gos\_result\_t gos\_signalInit( void\_t )

Initializes the signal service.

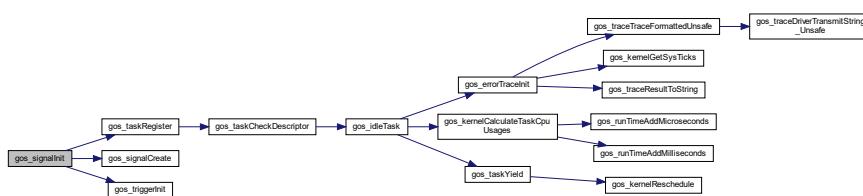
Initializes the internal signal array, creates a signal queue, registers the signal daemon task, creates the kernel dump signal, and subscribes the necessary components for the dump signal.

**Returns**

-

Definition at line 131 of file gos\_signal.c.

Here is the call graph for this function:



#### 4.27.2.4 GOS\_INLINE gos\_result\_t gos\_signalInvoke( gos\_signalId\_t signalId, gos\_signalSenderId\_t senderId )

Invokes the given signal.

Places the given signal in the invoke queue (for the signal daemon to actually invoke the signal in the background).

**Parameters**

|                 |                                        |
|-----------------|----------------------------------------|
| <i>signalId</i> | : Signal identifier.                   |
| <i>senderId</i> | : Sender identifier (or data to pass). |

**Returns**

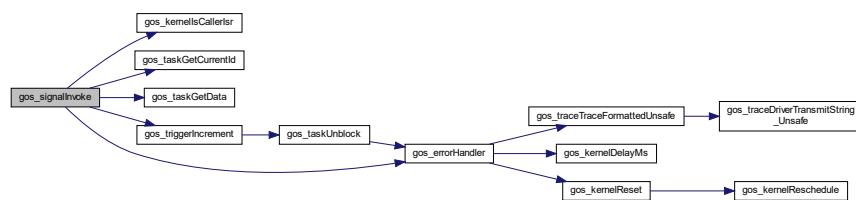
Success of signal invoking.

**Return values**

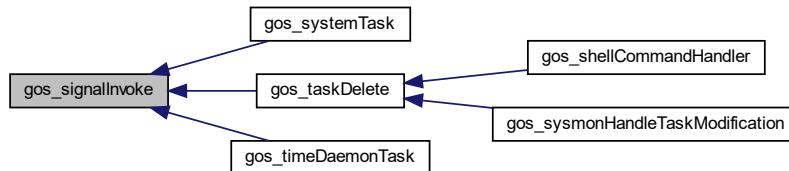
|                    |                                       |
|--------------------|---------------------------------------|
| <i>GOS_SUCCESS</i> | : Invoking successful.                |
| <i>GOS_ERROR</i>   | : Invalid signal ID or signal unused. |

Definition at line 245 of file gos\_signal.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.2.5 `gos_result_t gos_signalSubscribe ( gos_signalId_t signalId, gos_signalHandler_t signalHandler, gos_taskPrivilegeLevel_t signalHandlerPrivileges )`

Subscribes to the given signal.

Finds the next free slot in the signal handler array and registers the signal handler there.

**Parameters**

|                                |                                    |
|--------------------------------|------------------------------------|
| <i>signalId</i>                | : Signal identifier.               |
| <i>signalHandler</i>           | : Signal handler function pointer. |
| <i>signalHandlerPrivileges</i> | : Signal handler privilege level.  |

**Returns**

Success of signal subscription.

Return values

|                          |                                                                          |
|--------------------------|--------------------------------------------------------------------------|
| <code>GOS_SUCCESS</code> | : Subscription successful.                                               |
| <code>GOS_ERROR</code>   | : Invalid signal ID, signal handler NULL pointer, or handler array full. |

Definition at line 202 of file gos\_signal.c.

### 4.27.3 Variable Documentation

#### 4.27.3.1 GOS\_STATIC gos\_signalDescriptor\_t signalArray[CFG\_SIGNAL\_MAX\_NUMBER]

Internal signal descriptor array.

Definition at line 98 of file gos\_signal.c.

#### 4.27.3.2 GOS\_STATIC gos\_taskDescriptor\_t signalDaemonTaskDescriptor

**Initial value:**

```
=
{
 .taskFunction = gos_signalDaemonTask,
 .taskName = "gos_signal_daemon",
 .taskStackSize = CFG_TASK_SIGNAL_DAEMON_STACK,
 .taskPriority = CFG_TASK_SIGNAL_DAEMON_PRIO,
 .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL
}
```

Signal daemon task descriptor.

Definition at line 119 of file gos\_signal.c.

#### 4.27.3.3 GOS\_STATIC gos\_trigger\_t signalInvokeTrigger

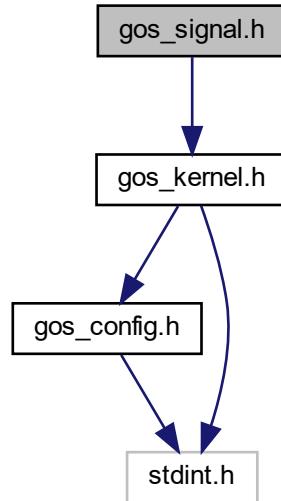
Invoke trigger (to count the number of signal invokings).

Definition at line 103 of file gos\_signal.c.

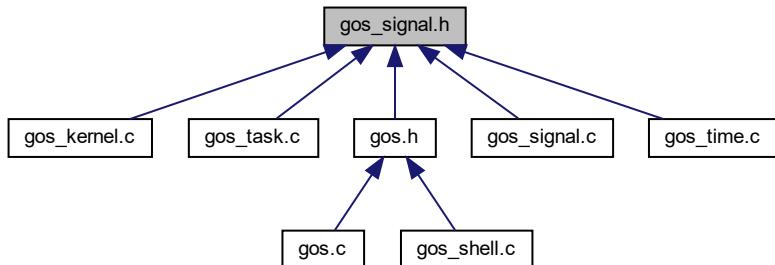
## 4.28 gos\_signal.h File Reference

GOS signal service header.

```
#include <gos_kernel.h>
Include dependency graph for gos_signal.h:
```



This graph shows which files directly or indirectly include this file:



## Typedefs

- `typedef u8_t gos_signalId_t`  
*Signal ID.*
- `typedef u16_t gos_signalSenderId_t`  
*Signal sender ID.*
- `typedef u8_t gos_signalIndex_t`  
*Signal index for loops.*
- `typedef u8_t gos_signalHandlerIndex_t`  
*Signal handler index type.*
- `typedef void_t(* gos_signalHandler_t )(gos_signalSenderId_t)`  
*Signal handler function type.*

## Functions

- `gos_result_t gos_signalInit (void_t)`  
*Initializes the signal service.*
- `gos_result_t gos_signalCreate (gos_signallId_t *pSignal)`  
*Creates a new signal.*
- `gos_result_t gos_signalSubscribe (gos_signallId_t signallId, gos_signalHandler_t signalHandler, gos_taskPrivilegeLevel_t signalHandlerPrivileges)`  
*Subscribes to the given signal.*
- `gos_result_t gos_signalInvoke (gos_signallId_t signallId, gos_signalSenderId_t senderId)`  
*Invokes the given signal.*

### 4.28.1 Detailed Description

GOS signal service header.

#### Author

Ahmed Gazar

#### Date

2022-11-15

#### Version

1.1

Signal service is a way of inter-task or inter-process communication provided by the operating system. Signals can be created, subscribed to and invoked. When a signal is invoked, it is placed into an invoke queue and the signal daemon task will handle the invoke requests in the background. Thus signals are not instantly invoked. When a signal is invoked, all the subscribed functions get called. A signal can be invoked without any subscribers (in this case no function will be called).

Definition in file [gos\\_signal.h](#).

### 4.28.2 Function Documentation

#### 4.28.2.1 `gos_result_t gos_signalCreate ( gos_signallId_t * pSignal )`

Creates a new signal.

Finds the next free slot in the signal array and registers the new signal there.

##### Parameters

|                      |                                   |
|----------------------|-----------------------------------|
| <code>pSignal</code> | : Pointer to a signal identifier. |
|----------------------|-----------------------------------|

##### Returns

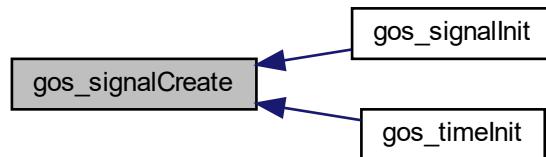
Success of signal creation.

## Return values

|                    |                        |
|--------------------|------------------------|
| <i>GOS_SUCCESS</i> | : Creation successful. |
| <i>GOS_ERROR</i>   | : Signal array full.   |

Definition at line 168 of file gos\_signal.c.

Here is the caller graph for this function:



#### 4.28.2.2 `gos_result_t gos_signallInit( void_t )`

Initializes the signal service.

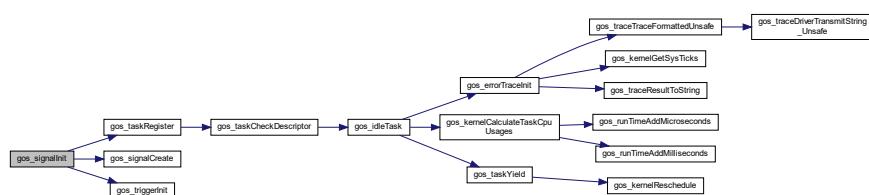
Initializes the internal signal array, creates a signal queue, registers the signal daemon task, creates the kernel dump signal, and subscribes the necessary components for the dump signal.

## Returns

-

Definition at line 131 of file gos\_signal.c.

Here is the call graph for this function:



#### 4.28.2.3 `gos_result_t gos_signallInvoke( gos_signalId_t signalId, gos_signalSenderId_t senderId )`

Invokes the given signal.

Places the given signal in the invoke queue (for the signal daemon to actually invoke the signal in the background).

**Parameters**

|                 |                                        |
|-----------------|----------------------------------------|
| <i>signalId</i> | : Signal identifier.                   |
| <i>senderId</i> | : Sender identifier (or data to pass). |

**Returns**

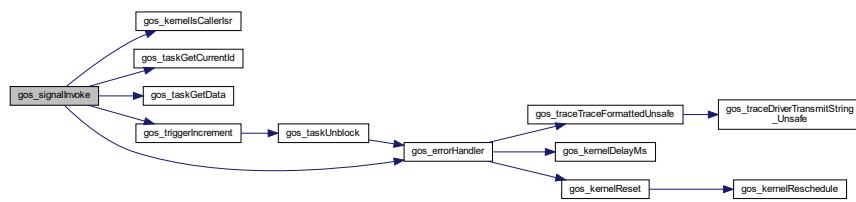
Success of signal invoking.

**Return values**

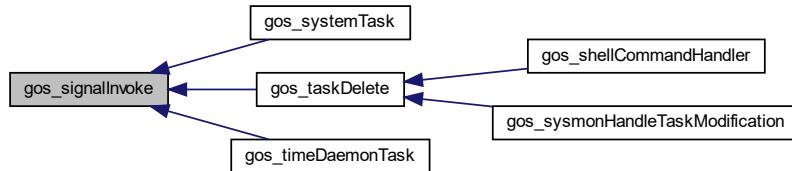
|                    |                                       |
|--------------------|---------------------------------------|
| <i>GOS_SUCCESS</i> | : Invoking successful.                |
| <i>GOS_ERROR</i>   | : Invalid signal ID or signal unused. |

Definition at line 245 of file gos\_signal.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.28.2.4 `gos_result_t gos_signalSubscribe( gos_signalId_t signalId, gos_signalHandler_t signalHandler, gos_taskPrivilegeLevel_t signalHandlerPrivileges )`

Subscribes to the given signal.

Finds the next free slot in the signal handler array and registers the signal handler there.

**Parameters**

|                                |                                    |
|--------------------------------|------------------------------------|
| <i>signalId</i>                | : Signal identifier.               |
| <i>signalHandler</i>           | : Signal handler function pointer. |
| <i>signalHandlerPrivileges</i> | : Signal handler privilege level.  |

**Returns**

Success of signal subscription.

## Return values

|                    |                                                                          |
|--------------------|--------------------------------------------------------------------------|
| <b>GOS_SUCCESS</b> | : Subscription successful.                                               |
| <b>GOS_ERROR</b>   | : Invalid signal ID, signal handler NULL pointer, or handler array full. |

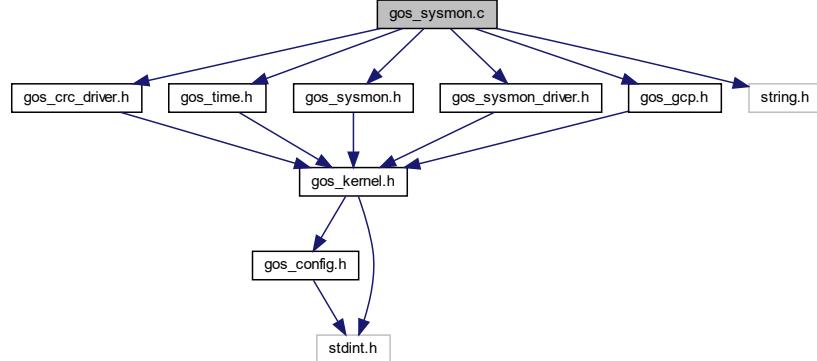
Definition at line 202 of file gos\_signal.c.

## 4.29 gos\_sysmon.c File Reference

## GOS system monitoring service source.

```
#include <gos_crc_driver.h>
#include <gos_time.h>
#include <gos_sysmon.h>
#include <gos_sysmon_driver.h>
#include <gos_gcp.h>
#include <string.h>
```

Include dependency graph for gos\_sysmon.c:



## Data Structures

- struct gos sysmonLut t

## Macros

- #define RECEIVE BUFFER SIZE ( 128u )

## TypeDefs

- ```
• typedef void t(* gos_sysmonMessageHandler t )(gos_sysmonMessageEnum t)
```

Enumerations

- enum gos_sysmonMessageId_t {
 GOS_SYSMON_MSG_UNKNOWN_ID = 0, GOS_SYSMON_MSG_PING_ID = 0x1010, GOS_SYSMON_MSG_PING_RESP_ID = 0x50A0, GOS_SYSMON_MSG_CPU_USAGE_GET_ID = 0x1023,
 GOS_SYSMON_MSG_CPU_USAGE_GET_RESP_ID = 0x5C20, GOS_SYSMON_MSG_TASK_GET_DA_ID = 0x1024, GOS_SYSMON_MSG_TASK_SET_DA_ID = 0x1025,

```

TA_ID = 0x1B67, GOS_SYSMON_MSG_TASK_GET_DATA_RESP_ID = 0x5F8A, GOS_SYSMON_MSG_
_TASK_GET_VAR_DATA_ID = 0x12D5,
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP_ID = 0x596B, GOS_SYSMON_MSG_TASK_MO_
DIFY_STATE_ID = 0xA917, GOS_SYSMON_MSG_TASK_MODIFY_STATE_RESP_ID = 0x4AB2, GOS_
SYSMON_MSG_SYSRUNTIME_GET_ID = 0x33AF,
GOS_SYSMON_MSG_SYSRUNTIME_GET_RESP_ID = 0xD91E, GOS_SYSMON_MSG_SYSTIME_SE_
T_ID = 0xBB53, GOS_SYSMON_MSG_SYSTIME_SET_RESP_ID = 0x174C, GOS_SYSMON_MSG_RE_
SET_REQ_ID = 0x0A78 }

• enum gos_sysmonMessageEnum_t {
    GOS_SYSMON_MSG_UNKNOWN = 0, GOS_SYSMON_MSG_PING, GOS_SYSMON_MSG_PING_RE_
SP, GOS_SYSMON_MSG_CPU_USAGE_GET,
    GOS_SYSMON_MSG_CPU_USAGE_GET_RESP, GOS_SYSMON_MSG_TASK_GET_DATA, GOS_SYS_
MON_MSG_TASK_GET_DATA_RESP, GOS_SYSMON_MSG_TASK_GET_VAR_DATA,
    GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP, GOS_SYSMON_MSG_TASK_MODIFY_STATE,
    GOS_SYSMON_MSG_TASK_MODIFY_STATE_RESP, GOS_SYSMON_MSG_SYSRUNTIME_GET,
    GOS_SYSMON_MSG_SYSRUNTIME_GET_RESP, GOS_SYSMON_MSG_SYSTIME_SET, GOS_SYS_
MON_MSG_SYSTIME_SET_RESP, GOS_SYSMON_MSG_RESET_REQ,
    GOS_SYSMON_MSG_NUM_OF_MESSAGES }
}

• enum gos_sysmonMessagePv_t {
    GOS_SYSMON_MSG_UNKNOWN_PV = 1, GOS_SYSMON_MSG_PING_PV = 1, GOS_SYSMON_MSG_
_PING_ACK_PV = 1, GOS_SYSMON_MSG_CPU_USAGE_GET_PV = 1,
    GOS_SYSMON_MSG_CPU_USAGE_GET_RESP_PV = 1, GOS_SYSMON_MSG_TASK_GET_DATA_PV
= 1, GOS_SYSMON_MSG_TASK_GET_DATA_RESP_PV = 1, GOS_SYSMON_MSG_TASK_GET_VAR_
DATA_PV = 1,
    GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP_PV = 1, GOS_SYSMON_MSG_TASK_MODIFY_
STATE_PV = 1, GOS_SYSMON_MSG_TASK_MODIFY_STATE_RESP_PV = 1, GOS_SYSMON_MSG_
SYSRUNTIME_GET_PV = 1,
    GOS_SYSMON_MSG_SYSRUNTIME_GET_RESP_PV = 1, GOS_SYSMON_MSG_SYSTIME_SET_PV
= 1, GOS_SYSMON_MSG_SYSTIME_SET_RESP_PV = 1, GOS_SYSMON_MSG_RESET_REQ_PV = 1 }
}

• enum gos_sysmonMessageResult_t { GOS_SYSMON_MSG_RES_OK = 40, GOS_SYSMON_MSG_RE_
S_ERROR = 99, GOS_SYSMON_MSG_INV_PV = 35, GOS_SYSMON_MSG_INV_PAYLOAD_CRC = 28
}

• enum gos_sysmonTaskModifyType_t {
    GOS_SYSMON_TASK_MOD_TYPE_SUSPEND = 12, GOS_SYSMON_TASK_MOD_TYPE_RESUME =
34, GOS_SYSMON_TASK_MOD_TYPE_DELETE = 49, GOS_SYSMON_TASK_MOD_TYPE_BLOCK = 52,
    GOS_SYSMON_TASK_MOD_TYPE_UNBLOCK = 63, GOS_SYSMON_TASK_MOD_TYPE_WAKEUP = 74
}

```

Functions

- struct __attribute__((packed))
- **GOS_STATIC void_t gos_sysmonDaemonTask (void_t)**

System monitoring daemon task.
- **GOS_STATIC gos_sysmonMessageEnum_t gos_sysmonGetLutIndex (gos_sysmonMessageId_t message_
Id)**

Gets the LUT index of the given message.
- **GOS_STATIC void_t gos_sysmonSendResponse (gos_sysmonMessageEnum_t lutIndex)**

Sends the response to the given message.
- **GOS_STATIC void_t gos_sysmonHandlePingRequest (gos_sysmonMessageEnum_t lutIndex)**

Handles the ping request.
- **GOS_STATIC void_t gos_sysmonHandleCpuUsageGet (gos_sysmonMessageEnum_t lutIndex)**

Handles the CPU usage get request.
- **GOS_STATIC void_t gos_sysmonHandleTaskDataGet (gos_sysmonMessageEnum_t lutIndex)**

Handles the task data get request.
- **GOS_STATIC void_t gos_sysmonHandleTaskVariableDataGet (gos_sysmonMessageEnum_t lutIndex)**

- Handles the task variable data get request.
- **GOS_STATIC void_t gos_sysmonHandleTaskModification (gos_sysmonMessageEnum_t lutIndex)**
Handles the task modification request.
- **GOS_STATIC void_t gos_sysmonHandleSysRuntimeGet (gos_sysmonMessageEnum_t lutIndex)**
Handles the system runtime get request.
- **GOS_STATIC void_t gos_sysmonHandleSystimeSet (gos_sysmonMessageEnum_t lutIndex)**
Handles the system time set request.
- **GOS_STATIC void_t gos_sysmonHandleResetRequest (gos_sysmonMessageEnum_t lutIndex)**
Handles the system reset request.
- **GOS_STATIC gos_sysmonMessageResult_t gos_sysmonCheckMessage (gos_sysmonMessageEnum_t lutIndex)**
Checks the high-level message parameters.
- **gos_result_t gos_sysmonInit (void_t)**
This function initializes the system monitoring service.
- **gos_result_t gos_sysmonRegisterUserMessage (gos_sysmonUserMessageDescriptor_t *pDesc)**
This function registers a custom user sysmon message.

Variables

- **gos_sysmonTaskData_t**
- **gos_sysmonTaskVariableData**
- **gos_sysmonPingMessage_t**
- **gos_sysmonCpuUsageMessage_t**
- **gos_sysmonTaskDataGetMessage_t**
- **gos_sysmonTaskDataMessage_t**
- **gos_sysmonTaskVariableDataMessage_t**
- **gos_sysmonTaskModifyMessage_t**
- **gos_sysmonTaskModifyResultMessage_t**
- **gos_sysmonSysruntimeGetResultMessage_t**
- **gos_sysmonSystimeSetMessage_t**
- **gos_sysmonSystimeSetResultMessage_t**
- **GOS_STATIC u8_t receiveBuffer [RECEIVE_BUFFER_SIZE]**
- **GOS_STATIC gos_sysmonPingMessage_t pingMessage = {0}**
- **GOS_STATIC gos_sysmonCpuUsageMessage_t cpuMessage = {0}**
- **GOS_STATIC gos_sysmonTaskDataGetMessage_t taskDataGetMsg = {0}**
- **GOS_STATIC gos_sysmonTaskDataMessage_t taskDataMsg = {0}**
- **GOS_STATIC gos_sysmonTaskVariableDataMessage_t taskVariableDataMsg = {0}**
- **GOS_STATIC gos_taskDescriptor_t taskDesc = {0}**
- **GOS_STATIC gos_sysmonTaskModifyMessage_t taskModifyMessage = {0}**
- **GOS_STATIC gos_sysmonTaskModifyResultMessage_t taskModifyResultMessage = {0}**
- **GOS_STATIC gos_sysmonSysruntimeGetResultMessage_t sysRuntimeGetResultMessage = {0}**
- **GOS_STATIC gos_sysmonSystimeSetMessage_t sysTimeSetMessage = {0}**
- **GOS_STATIC gos_sysmonSystimeSetResultMessage_t sysTimeSetResultMessage = {0}**
- **GOS_STATIC gos_sysmonUserMessageDescriptor_t userMessages [CFG_SYSMON_MAX_USER_MESSAGES]**

- `GOS_STATIC` `gos_taskDescriptor_t sysmonDaemonTaskDesc`
- `GOS_STATIC GOS_CONST gos_sysmonLut_t sysmonLut [GOS_SYSMON_MSG_NUM_OF_MESSAGES]`

4.29.1 Detailed Description

GOS system monitoring service source.

Author

Ahmed Gazar

Date

2024-07-18

Version

1.5

For a more detailed description of this service, please refer to [gos_sysmon.h](#)

Definition in file [gos_sysmon.c](#).

4.29.2 Macro Definition Documentation

4.29.2.1 `#define RECEIVE_BUFFER_SIZE (128u)`

Receive buffer size.

Definition at line 73 of file [gos_sysmon.c](#).

4.29.3 Typedef Documentation

4.29.3.1 `typedef void_t(* gos_sysmonMessageHandler_t)(gos_sysmonMessageEnum_t)`

Message handler function type.

Definition at line 292 of file [gos_sysmon.c](#).

4.29.4 Enumeration Type Documentation

4.29.4.1 `enum gos_sysmonMessageEnum_t`

Enumerator

`GOS_SYSMON_MSG_UNKNOWN` Unknown message LUT index.

`GOS_SYSMON_MSG_PING` Ping message LUT index.

`GOS_SYSMON_MSG_PING_RESP` Ping response message LUT index.

`GOS_SYSMON_MSG_CPU_USAGE_GET` CPU usage get message LUT index.

`GOS_SYSMON_MSG_CPU_USAGE_GET_RESP` CPU usage get response message LUT index.

`GOS_SYSMON_MSG_TASK_GET_DATA` Task data get message LUT index.

`GOS_SYSMON_MSG_TASK_GET_DATA_RESP` Task data get response message LUT index.

`GOS_SYSMON_MSG_TASK_GET_VAR_DATA` Task variable data get message LUT index.

GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP Task variable data get response message LUT index.

GOS_SYSMON_MSG_TASK MODIFY STATE Task modify message LUT index.

GOS_SYSMON_MSG_TASK MODIFY STATE RESP Task modify response message LUT index.

GOS_SYSMON_MSG_SYSRUNTIME_GET System runtime get message LUT index.

GOS_SYSMON_MSG_SYSRUNTIME_GET RESP System runtime get response message LUT index.

GOS_SYSMON_MSG_SYSTIME_SET System time set message LUT index.

GOS_SYSMON_MSG_SYSTIME_SET RESP System time set response message LUT index.

GOS_SYSMON_MSG_RESET_REQ System reset message LUT index.

GOS_SYSMON_MSG_NUM_OF_MESSAGES Number of messages.

Definition at line 101 of file gos_sysmon.c.

4.29.4.2 enum gos_sysmonMessageId_t

System monitoring message ID enum.

Enumerator

GOS_SYSMON_MSG_UNKNOWN_ID Unknown message ID.

GOS_SYSMON_MSG_PING_ID Ping message ID.

GOS_SYSMON_MSG_PING RESP ID Ping response message ID.

GOS_SYSMON_MSG_CPU_USAGE_GET_ID CPU usage get message ID.

GOS_SYSMON_MSG_CPU_USAGE_GET RESP ID CPU usage get response message ID.

GOS_SYSMON_MSG_TASK_GET_DATA_ID Task data get message ID.

GOS_SYSMON_MSG_TASK_GET_DATA RESP ID Task data get response message ID.

GOS_SYSMON_MSG_TASK_GET_VAR_DATA_ID Task variable data get message ID.

GOS_SYSMON_MSG_TASK_GET_VAR_DATA RESP ID Task variable data get response message ID.

GOS_SYSMON_MSG_TASK MODIFY STATE ID Task modify state message ID.

GOS_SYSMON_MSG_TASK MODIFY STATE RESP ID Task modify state response ID.

GOS_SYSMON_MSG_SYSRUNTIME_GET_ID System runtime get message ID.

GOS_SYSMON_MSG_SYSRUNTIME_GET RESP ID System runtime get response message ID.

GOS_SYSMON_MSG_SYSTIME_SET_ID System time set message ID.

GOS_SYSMON_MSG_SYSTIME_SET RESP ID System time set response ID.

GOS_SYSMON_MSG_RESET REQ ID System reset request ID.

Definition at line 81 of file gos_sysmon.c.

4.29.4.3 enum gos_sysmonMessagePv_t

System monitoring message protocol version enum.

Enumerator

GOS_SYSMON_MSG_UNKNOWN_PV Unknown protocol version.

GOS_SYSMON_MSG_PING_PV Ping message protocol version.

GOS_SYSMON_MSG_PING ACK_PV Ping acknowledge message protocol version.

GOS_SYSMON_MSG_CPU_USAGE_GET_PV CPU usage get message protocol version.

GOS_SYSMON_MSG_CPU_USAGE_GET RESP_PV CPU usage get response message protocol version.

GOS_SYSMON_MSG_TASK_GET_DATA_PV Task data get message protocol version.
GOS_SYSMON_MSG_TASK_GET_DATA_RESP_PV Task data get response message protocol version.
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_PV Task variable data get message protocol version.
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP_PV Task variable data get response message protocol version.
GOS_SYSMON_MSG_TASK MODIFY STATE_PV Task modify state message protocol version.
GOS_SYSMON_MSG_TASK MODIFY STATE RESP_PV Task modify state response message protocol version.
GOS_SYSMON_MSG_SYSRUNTIME_GET_PV System runtime get message protocol version.
GOS_SYSMON_MSG_SYSRUNTIME_GET_RESP_PV System runtime get response protocol version.
GOS_SYSMON_MSG_SYSTIME_SET_PV System time set message protocol version.
GOS_SYSMON_MSG_SYSTIME_SET_RESP_PV System time set response message protocol version.
GOS_SYSMON_MSG_RESET_REQ_PV System reset request message protocol version.

Definition at line 125 of file gos_sysmon.c.

4.29.4.4 enum gos_sysmonMessageResult_t

Message result enum.

Enumerator

GOS_SYSMON_MSG_RES_OK Message result OK.
GOS_SYSMON_MSG_RES_ERROR Message result ERROR.
GOS_SYSMON_MSG_INV_PV Invalid protocol version.
GOS_SYSMON_MSG_INV_PAYLOAD_CRC Invalid payload CRC.

Definition at line 148 of file gos_sysmon.c.

4.29.4.5 enum gos_sysmonTaskModifyType_t

State modification enum.

Enumerator

GOS_SYSMON_TASK_MOD_TYPE_SUSPEND Task suspend.
GOS_SYSMON_TASK_MOD_TYPE_RESUME Task resume.
GOS_SYSMON_TASK_MOD_TYPE_DELETE Task delete.
GOS_SYSMON_TASK_MOD_TYPE_BLOCK Task block.
GOS_SYSMON_TASK_MOD_TYPE_UNBLOCK Task unblock.
GOS_SYSMON_TASK_MOD_TYPE_WAKEUP Task wakeup.

Definition at line 159 of file gos_sysmon.c.

4.29.5 Function Documentation

4.29.5.1 struct __attribute__ ((packed))

Task data message structure.

Task variable data message structure.

Ping message structure.
CPU usage message structure.
Task data get message structure.
Task modify message structure.
Task modify message result structure.
System runtime get message result structure.
System time set message structure.
System time set message result structure. < Task state.
< Task priority.
< Task original priority.
< Task privilege level.
< Task name.
< Task ID (internal).
< Task context-switch counter.
< Task stack size.
< Task run-time.
< Task CPU usage limit in [% x 100].
< Task CPU usage max value in [% x 100].
< Task processor usage in [% x 100].
< Task max. stack usage.
< Task state.
< Task priority.
< Task context-switch counter.
< Task run-time.
< Task CPU usage max value in [% x 100].
< Task processor usage in [% x 100].
< Task stack usage.
< Message result.
< Message result.
< CPU usage x100[%].
< Task index.
< Message result.
< Task data.
< Message result.
< Task variable data.
< Task index.
< Task modification type.
< Parameter for functions where timeout is required.
< Message result.
< Message result.

< System runtime.

< Desired system time.

< Message result.

Definition at line 172 of file gos_sysmon.c.

4.29.5.2 GOS_STATIC gos_sysmonMessageResult_t gos_sysmonCheckMessage (gos_sysmonMessageEnum_t lutIndex)

Checks the high-level message parameters.

Checks the protocol version and the payload CRC value.

Parameters

<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

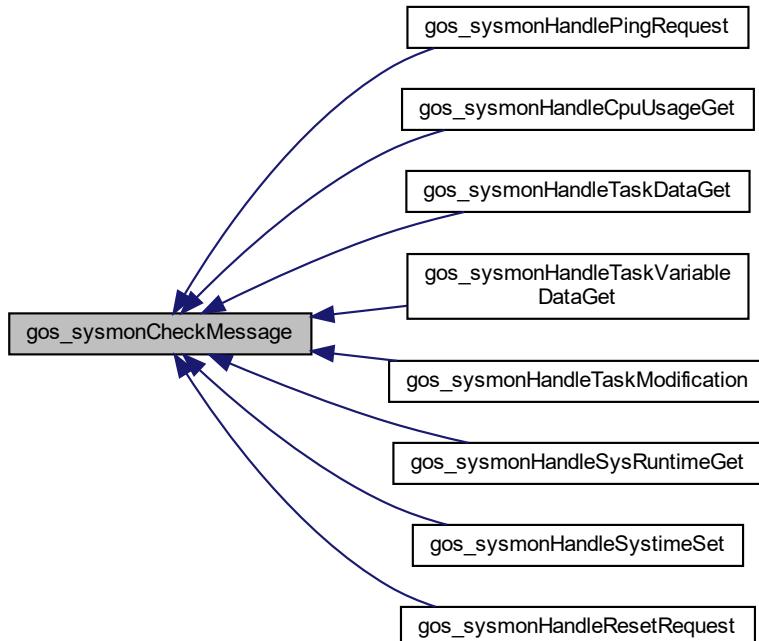
Result of message checking.

Return values

<i>GOS_SYSMON_MSG_R← ES_OK</i>	: Message OK.
<i>GOS_SYSMON_MSG_IN← V_PV</i>	: Invalid protocol version.
<i>GOS_SYSMON_MSG_IN← V_PAYLOAD_CRC</i>	: Payload CRC mismatch.

Definition at line 1173 of file gos_sysmon.c.

Here is the caller graph for this function:



4.29.5.3 GOS_STATIC void_t gos_sysmonDaemonTask (void_t)

System monitoring daemon task.

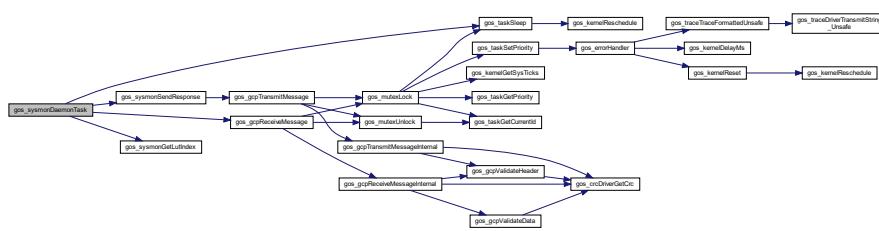
Serves the incoming system monitoring messages.

Returns

-

Definition at line 608 of file gos_sysmon.c.

Here is the call graph for this function:



4.29.5.4 GOS_STATIC gos_sysmonMessageEnum_t gos_sysmonGetLutIndex (gos_sysmonMessageId_t messageId)

Gets the LUT index of the given message.

Returns the look-up table index that belongs to the given message ID.

Parameters

<code>messageId</code>	: ID of the message to get the index for.
------------------------	---

Returns

Look-up table index of the message.

Definition at line 685 of file gos_sysmon.c.

Here is the caller graph for this function:



4.29.5.5 GOS_STATIC void_t gos_sysmonHandleCpuUsageGet (gos_sysmonMessageEnum_t lutIndex)

Handles the CPU usage get request.

Sends out the current CPU usage.

Parameters

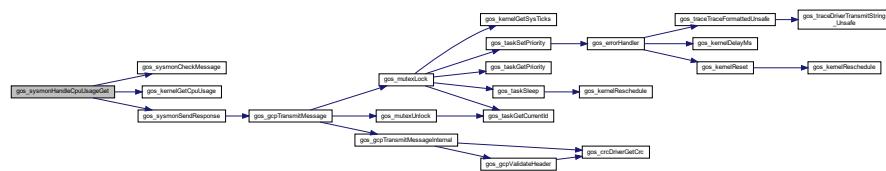
<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

-

Definition at line 755 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.6 GOS_STATIC void_t gos_sysmonHandlePingRequest (gos_sysmonMessageEnum_t lutIndex)**

Handles the ping request.

Sends a ping response.

Parameters

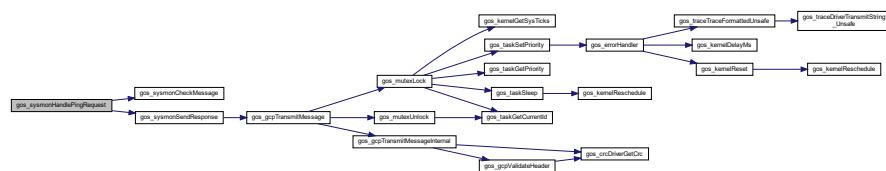
<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

-

Definition at line 738 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.7 GOS_STATIC void_t gos_sysmonHandleResetRequest (gos_sysmonMessageEnum_t lutIndex)**

Handles the system reset request.

Resets the system.

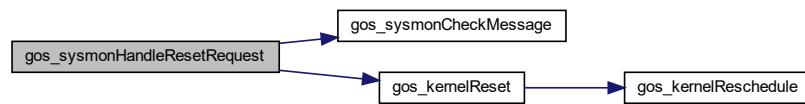
Parameters

<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

-
Definition at line 1146 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.8 GOS_STATIC void_t gos_sysmonHandleSysRuntimeGet(gos_sysmonMessageEnum_t lutIndex)**

Handles the system runtime get request.

Sends out the total system runtime since startup.

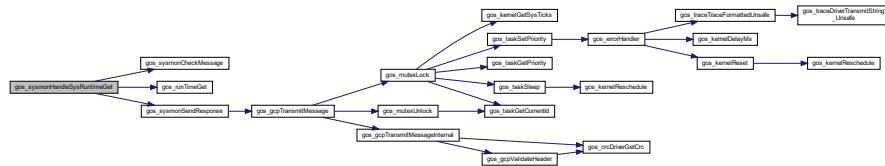
Parameters

<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

-
Definition at line 1078 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.9 GOS_STATIC void_t gos_sysmonHandleSystimeSet(gos_sysmonMessageEnum_t lutIndex)**

Handles the system time set request.

Sets the system time to the given value.

Parameters

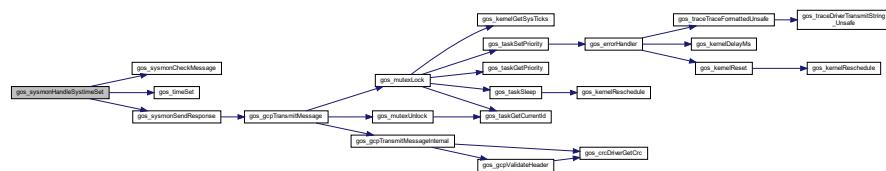
<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

-

Definition at line 1112 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.10 GOS_STATIC void_t gos_sysmonHandleTaskDataGet (gos_sysmonMessageEnum_t lutIndex)**

Handles the task data get request.

Sends out the static task data for the requested task.

Parameters

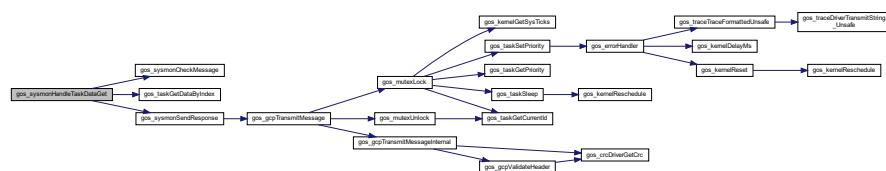
<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

-

Definition at line 782 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.11 GOS_STATIC void_t gos_sysmonHandleTaskModification (gos_sysmonMessageEnum_t lutIndex)**

Handles the task modification request.

Performs the requested kind of modification on the requested task.

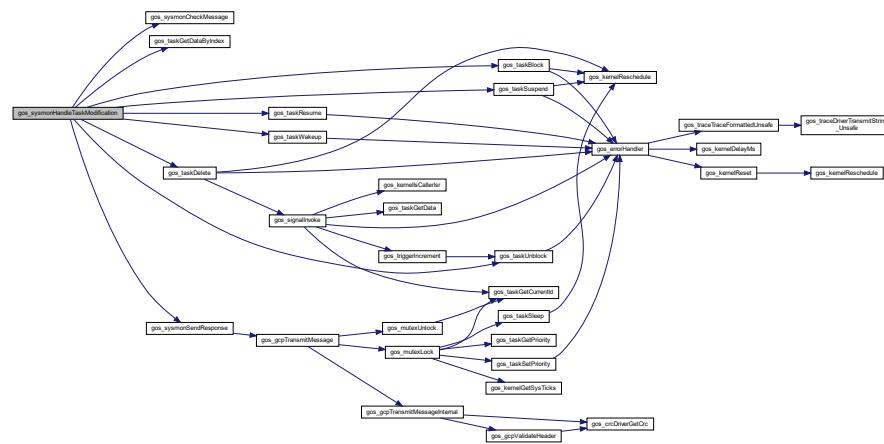
Parameters

<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

Definition at line 969 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.12 GOS_STATIC void_t gos_sysmonHandleTaskVariableDataGet(gos_sysmonMessageEnum_t lutIndex)**

Handles the task variable data get request.

Sends out the variable task data related to the request task.

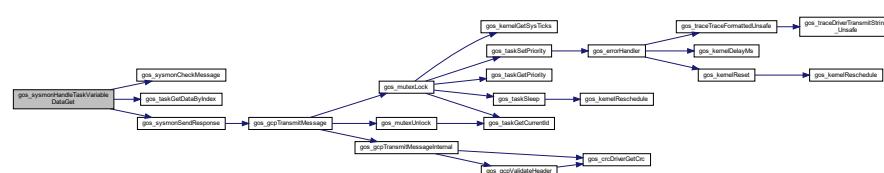
Parameters

<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

Returns

Definition at line 881 of file gos_sysmon.c.

Here is the call graph for this function:

**4.29.5.13 gos_result_t gos_sysmonInit(void_t)**

This function initializes the system monitoring service.

Registers the GCP physical driver and the sysmon daemon task in the kernel.

Returns

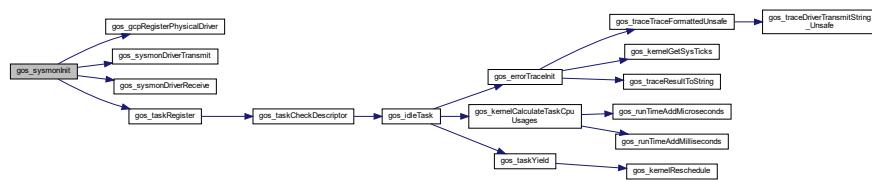
Result of initialization.

Return values

<i>GOS_SUCCESS</i>	: Sysmon initialization successful.
<i>GOS_ERROR</i>	: Sysmon daemon task registration failed.

Definition at line 532 of file gos_sysmon.c.

Here is the call graph for this function:



4.29.5.14 ***GOS_STATIC gos_result_t gos_sysmonRegisterUserMessage(gos_sysmonUserMessageDescriptor_t * pDesc)***

This function registers a custom user sysmon message.

Registers a custom sysmon message given by its ID and PV. The message will only be processed if the required ID is not an existing sysmon ID. When a message is received with the given ID and PV, the message content will be copied into the buffer defined in the descriptor structure, and the registered callback function will be called.

Recommended ID range: 0x6000 ... 0x9999.

Returns

Result of registration.

Return values

<i>GOS_SUCCESS</i>	: User message registered successfully.
<i>GOS_ERROR</i>	: Descriptor or callback is NULL or maximum number of user messages has been reached.

Definition at line 569 of file gos_sysmon.c.

4.29.5.15 ***GOS_STATIC void_t gos_sysmonSendResponse(gos_sysmonMessageEnum_t lutIndex)***

Sends the response to the given message.

Fills out the remaining transmit header parameters (except for the result field), and transmits the message with the corresponding payload.

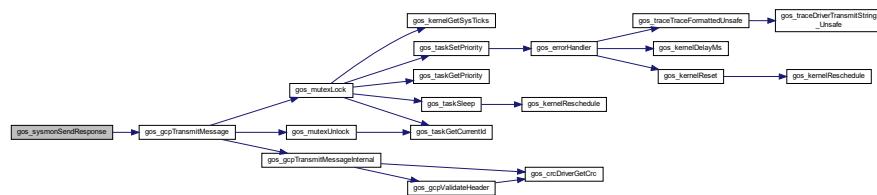
Parameters

<i>lutIndex</i>	: Look-up table index of the message.
-----------------	---------------------------------------

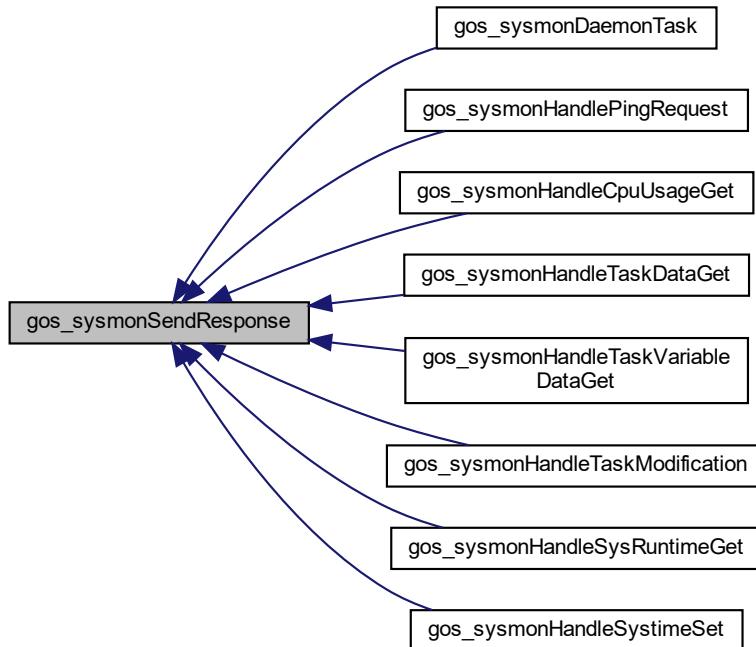
Returns

-
Definition at line 719 of file gos_sysmon.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.29.6 Variable Documentation

4.29.6.1 GOS_STATIC gos_sysmonCpuUsageMessage_t cpuMessage = {0}

CPU load message structure.

Definition at line 322 of file gos_sysmon.c.

4.29.6.2 GOS_STATIC gos_sysmonPingMessage_t pingMessage = {0}

Ping message structure.

Definition at line 317 of file gos_sysmon.c.

4.29.6.3 GOS_STATIC u8_t receiveBuffer[RECEIVE_BUFFER_SIZE]

Receive buffer.

Definition at line 312 of file gos_sysmon.c.

4.29.6.4 GOS_STATIC gos_taskDescriptor_t sysmonDaemonTaskDesc

Initial value:

```
=  
{  
    .taskFunction      = gos_sysmonDaemonTask,  
    .taskName         = "gos_sysmon_daemon",  
    .taskPriority     = CFG_TASK_SYSMON_DAEMON_PRIO,  
    .taskStackSize    = CFG_TASK_SYSMON_DAEMON_STACK,  
    .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL  
}
```

Sysmon daemon task descriptor.

Definition at line 393 of file gos_sysmon.c.

4.29.6.5 GOS_STATIC GOS_CONST gos_sysmonLut_t sysmonLut[GOS_SYSMON_MSG_NUM_OF_MESSAGES]

Sysmon look-up table.

Definition at line 405 of file gos_sysmon.c.

4.29.6.6 GOS_STATIC gos_sysmonSysruntimeGetResultMessage_t sysRuntimeGetResultMessage = {0}

System runtime get message.

Definition at line 357 of file gos_sysmon.c.

4.29.6.7 GOS_STATIC gos_sysmonSystimeSetMessage_t sysTimeSetMessage = {0}

System time set message.

Definition at line 362 of file gos_sysmon.c.

4.29.6.8 GOS_STATIC gos_sysmonSystimeSetResultMessage_t sysTimeSetResultMessage = {0}

System time set result message.

Definition at line 367 of file gos_sysmon.c.

4.29.6.9 GOS_STATIC gos_sysmonTaskDataGetMessage_t taskDataGetMsg = {0}

Task data get message structure.

Definition at line 327 of file gos_sysmon.c.

4.29.6.10 GOS_STATIC gos_sysmonTaskDataMessage_t taskDataMsg = {0}

Task data message structure.

Definition at line 332 of file gos_sysmon.c.

4.29.6.11 GOS_STATIC gos_taskDescriptor_t taskDesc = {0}

Task descriptor structure.

Definition at line 342 of file gos_sysmon.c.

4.29.6.12 GOS_STATIC gos_sysmonTaskModifyMessage_t taskModifyMessage = {0}

Task modify message.

Definition at line 347 of file gos_sysmon.c.

4.29.6.13 GOS_STATIC gos_sysmonTaskModifyResultMessage_t taskModifyResultMessage = {0}

Task modify result message.

Definition at line 352 of file gos_sysmon.c.

4.29.6.14 GOS_STATIC gos_sysmonTaskVariableDataMessage_t taskVariableDataMsg = {0}

Task variable data message structure.

Definition at line 337 of file gos_sysmon.c.

4.29.6.15 GOS_STATIC gos_sysmonUserMessageDescriptor_t userMessages[CFG_SYSMON_MAX_USER_MESSAGES]

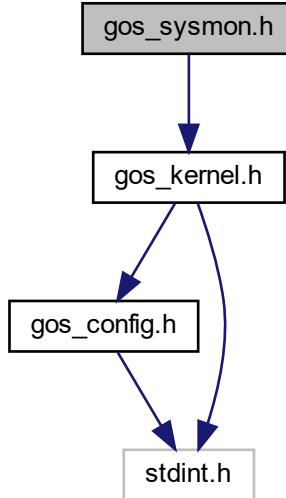
Sysmon user messages.

Definition at line 372 of file gos_sysmon.c.

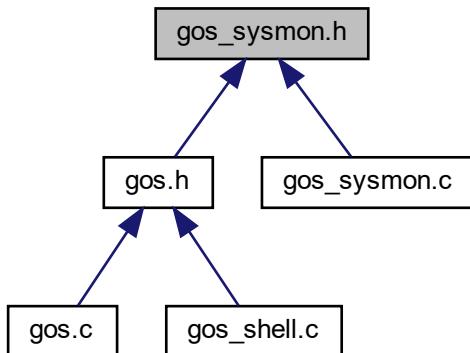
4.30 gos_sysmon.h File Reference

GOS system monitoring service header.

```
#include <gos_kernel.h>
Include dependency graph for gos_sysmon.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `gos_sysmonUserMessageDescriptor_t`

Typedefs

- typedef `void_t(*) gos_sysmonMessageReceivedCallback)(void_t)`

Functions

- [`gos_result_t gos_sysmonInit \(void_t\)`](#)
This function initializes the system monitoring service.
- [`gos_result_t gos_sysmonRegisterUserMessage \(gos_sysmonUserMessageDescriptor_t *pDesc\)`](#)
This function registers a custom user sysmon message.

4.30.1 Detailed Description

GOS system monitoring service header.

Author

Ahmed Gazar

Date

2023-07-12

Version

1.1

This service is used to send and receive system information to an external client such as a PC tool.

Definition in file [gos_sysmon.h](#).

4.30.2 Typedef Documentation

4.30.2.1 `typedef void_t(* gos_sysmonMessageReceivedCallback)(void_t)`

Sysmon message received callback function type.

Definition at line 65 of file [gos_sysmon.h](#).

4.30.3 Function Documentation

4.30.3.1 `gos_result_t gos_sysmonInit(void_t)`

This function initializes the system monitoring service.

Registers the GCP physical driver and the sysmon daemon task in the kernel.

Returns

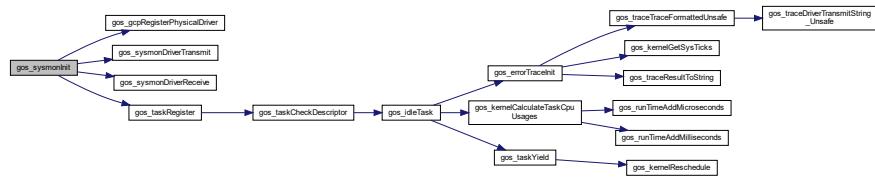
Result of initialization.

Return values

<code>GOS_SUCCESS</code>	: Sysmon initialization successful.
<code>GOS_ERROR</code>	: Sysmon daemon task registration failed.

Definition at line 532 of file [gos_sysmon.c](#).

Here is the call graph for this function:



4.30.3.2 gos_result_t gos_sysmonRegisterUserMessage (gos_sysmonUserMessageDescriptor_t * pDesc)

This function registers a custom user sysmon message.

Registers a custom sysmon message given by its ID and PV. The message will only be processed if the required ID is not an existing sysmon ID. When a message is received with the given ID and PV, the message content will be copied into the buffer defined in the descriptor structure, and the registered callback function will be called.

Recommended ID range: 0x6000 ... 0x9999.

Returns

Result of registration.

Return values

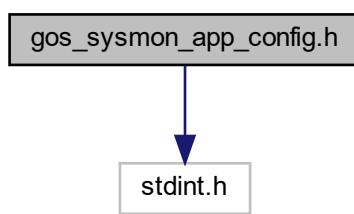
<code>GOS_SUCCESS</code>	: User message registered successfully.
<code>GOS_ERROR</code>	: Descriptor or callback is NULL or maximum number of user messages has been reached.

Definition at line 569 of file gos_sysmon.c.

4.31 gos_sysmon_app_config.h File Reference

GOS system monitoring application configuration header.

```
#include <stdint.h>
Include dependency graph for gos_sysmon_app_config.h:
```



Macros

- #define GOS_CFG_OVERCONFIG
- #define ARM_CORTEX_M4 (1)
- #define CFG_TARGET_CPU (ARM_CORTEX_M4)
- #define CFG_SCHED_COOPERATIVE (0)
- #define CFG_USE_PRIO_INHERITANCE (1)
- #define CFG_TASK_MAX_NAME_LENGTH (32)
- #define CFG_TASK_MAX_NUMBER (36)
- #define CFG_TASK_MIN_STACK_SIZE (0x200)
- #define CFG_TASK_MAX_STACK_SIZE (0x4000)
- #define CFG_IDLE_TASK_STACK_SIZE (0x400)
- #define CFG_SYSTEM_TASK_STACK_SIZE (0x400)
- #define CFG_TASK_SIGNAL_DAEMON_STACK (0x400)
- #define CFG_TASK_PROC_DAEMON_STACK (0x300)
- #define CFG_TASK_TIME_DAEMON_STACK (0x300)
- #define CFG_TASK_MESSAGE_DAEMON_STACK (0x300)
- #define CFG_TASK_SHELL_DAEMON_STACK (0x300)
- #define CFG_TASK_TRACE_DAEMON_STACK (0x400)
- #define CFG_TASK_SYSMON_DAEMON_STACK (0x400)
- #define CFG_TASK_TRACE_DAEMON_PRIO (193)
- #define CFG_TASK_MESSAGE_DAEMON_PRIO (198)
- #define CFG_TASK_SIGNAL_DAEMON_PRIO (197)
- #define CFG_TASK_PROC_DAEMON_PRIO (194)
- #define CFG_TASK_SHELL_DAEMON_PRIO (192)
- #define CFG_TASK_TIME_DAEMON_PRIO (196)
- #define CFG_TASK_SYS_PRIO (195)
- #define CFG_TASK_SYSMON_DAEMON_PRIO (191)
- #define CFG_PROC_USE_SERVICE (0)
- #define CFG_PROC_MAX_PRIO_LEVELS (UINT8_MAX)
- #define CFG_PROC_IDLE_PRIO (CFG_PROC_MAX_PRIO_LEVELS)
- #define CFG_PROC_MAX_NAME_LENGTH (24)
- #define CFG_PROC_MAX_NUMBER (4)
- #define CFG_QUEUE_MAX_NUMBER (4)
- #define CFG_QUEUE_MAX_ELEMENTS (40)
- #define CFG_QUEUE_MAX_LENGTH (200)
- #define CFG_QUEUE_USE_NAME (1)
- #define CFG_QUEUE_MAX_NAME_LENGTH (24)
- #define CFG_SIGNAL_MAX_NUMBER (6)
- #define CFG_SIGNAL_MAX_SUBSCRIBERS (6)
- #define CFG_MESSAGE_MAX_NUMBER (8)
- #define CFG_MESSAGE_MAX_LENGTH (80)
- #define CFG_MESSAGE_MAX_WAITERS (10)
- #define CFG_MESSAGE_MAX_WAITER_IDS (8)
- #define CFG_MESSAGE_MAX_ADDRESSEES (8)
- #define CFG_SHELL_USE_SERVICE (0)
- #define CFG_SHELL_MAX_COMMAND_NUMBER (16)
- #define CFG_SHELL_MAX_COMMAND_LENGTH (20)
- #define CFG_SHELL_MAX_PARAMS_LENGTH (128)
- #define CFG_SHELL_COMMAND_BUFFER_SIZE (200)
- #define CFG_GCP_CHANNELS_MAX_NUMBER (3)
- #define CFG_TRACE_MAX_LENGTH (200)
- #define CFG_SYSMON_USE_SERVICE (1)
- #define CFG_SYSMON_GCP_CHANNEL_NUM (0)
- #define CFG_SYSMON_MAX_USER_MESSAGES (6)
- #define CFG_RESET_ON_ERROR (1)
- #define CFG_RESET_ON_ERROR_DELAY_MS (3000)

4.31.1 Detailed Description

GOS system monitoring application configuration header.

Author

Ahmed Gazar

Date

2023-09-25

Version

1.0

This header contains the kernel and service configurations of the operating system.

Definition in file [gos_sysmon_app_config.h](#).

4.31.2 Macro Definition Documentation

4.31.2.1 #define ARM_CORTEX_M4 (1)

ARM Cortex-M4.

Definition at line 49 of file [gos_sysmon_app_config.h](#).

4.31.2.2 #define CFG_GCP_CHANNELS_MAX_NUMBER (3)

GCP maximum number of channels.

Definition at line 279 of file [gos_sysmon_app_config.h](#).

4.31.2.3 #define CFG_IDLE_TASK_STACK_SIZE (0x400)

Idle task stack size.

Definition at line 94 of file [gos_sysmon_app_config.h](#).

4.31.2.4 #define CFG_MESSAGE_MAX_ADDRESSEES (8)

Maximum number of message addressees.

Definition at line 247 of file [gos_sysmon_app_config.h](#).

4.31.2.5 #define CFG_MESSAGE_MAX_LENGTH (80)

Maximum length of a message in bytes.

Definition at line 235 of file [gos_sysmon_app_config.h](#).

4.31.2.6 #define CFG_MESSAGE_MAX_NUMBER (8)

Maximum number of messages handled at once.

Definition at line 231 of file [gos_sysmon_app_config.h](#).

4.31.2.7 #define CFG_MESSAGE_MAX_WAITER_IDS (8)

Maximum number of message IDs a task can wait for (includes the terminating 0).

Definition at line 243 of file gos_sysmon_app_config.h.

4.31.2.8 #define CFG_MESSAGE_MAX_WAITERS (10)

Maximum number of message waiters.

Definition at line 239 of file gos_sysmon_app_config.h.

4.31.2.9 #define CFG_PROC_IDLE_PRIO (CFG_PROC_MAX_PRIO_LEVELS)

Idle process priority.

Definition at line 178 of file gos_sysmon_app_config.h.

4.31.2.10 #define CFG_PROC_MAX_NAME_LENGTH (24)

Maximum process name length.

Definition at line 182 of file gos_sysmon_app_config.h.

4.31.2.11 #define CFG_PROC_MAX_NUMBER (4)

Maximum number of processes.

Definition at line 186 of file gos_sysmon_app_config.h.

4.31.2.12 #define CFG_PROC_MAX_PRIO_LEVELS (UINT8_MAX)

Maximum process priority levels.

Definition at line 174 of file gos_sysmon_app_config.h.

4.31.2.13 #define CFG_PROC_USE_SERVICE (0)

Process service use flag.

Definition at line 170 of file gos_sysmon_app_config.h.

4.31.2.14 #define CFG_QUEUE_MAX_ELEMENTS (40)

Maximum number of queue elements.

Definition at line 198 of file gos_sysmon_app_config.h.

4.31.2.15 #define CFG_QUEUE_MAX_LENGTH (200)

Maximum queue length.

Definition at line 202 of file gos_sysmon_app_config.h.

4.31.2.16 `#define CFG_QUEUE_MAX_NAME_LENGTH (24)`

Maximum queue name length.

Definition at line 210 of file gos_sysmon_app_config.h.

4.31.2.17 `#define CFG_QUEUE_MAX_NUMBER (4)`

Maximum number of queues.

Definition at line 194 of file gos_sysmon_app_config.h.

4.31.2.18 `#define CFG_QUEUE_USE_NAME (1)`

Queue use name flag.

Definition at line 206 of file gos_sysmon_app_config.h.

4.31.2.19 `#define CFG_RESET_ON_ERROR (1)`

Flag to indicate if the system should reset on error.

Definition at line 313 of file gos_sysmon_app_config.h.

4.31.2.20 `#define CFG_RESET_ON_ERROR_DELAY_MS (3000)`

Delay time before system reset.

Definition at line 317 of file gos_sysmon_app_config.h.

4.31.2.21 `#define CFG_SCHED_COOPERATIVE (0)`

Cooperative scheduling flag.

Definition at line 62 of file gos_sysmon_app_config.h.

4.31.2.22 `#define CFG_SHELL_COMMAND_BUFFER_SIZE (200)`

Command buffer size.

Definition at line 271 of file gos_sysmon_app_config.h.

4.31.2.23 `#define CFG_SHELL_MAX_COMMAND_LENGTH (20)`

Maximum command length.

Definition at line 263 of file gos_sysmon_app_config.h.

4.31.2.24 `#define CFG_SHELL_MAX_COMMAND_NUMBER (16)`

Maximum number of shell commands.

Definition at line 259 of file gos_sysmon_app_config.h.

4.31.2.25 #define CFG_SHELL_MAX_PARAMS_LENGTH (128)

Maximum parameters length.

Definition at line 267 of file gos_sysmon_app_config.h.

4.31.2.26 #define CFG_SHELL_USE_SERVICE (0)

Shell service use flag.

Definition at line 255 of file gos_sysmon_app_config.h.

4.31.2.27 #define CFG_SIGNAL_MAX_NUMBER (6)

Maximum number of signals.

Definition at line 219 of file gos_sysmon_app_config.h.

4.31.2.28 #define CFG_SIGNAL_MAX_SUBSCRIBERS (6)

Maximum number of signal subscribers.

Definition at line 223 of file gos_sysmon_app_config.h.

4.31.2.29 #define CFG_SYSMON_GCP_CHANNEL_NUM (0)

Define sysmon GCP channel number.

Definition at line 300 of file gos_sysmon_app_config.h.

4.31.2.30 #define CFG_SYSMON_MAX_USER_MESSAGES (6)

Maximum number of user messages.

Definition at line 305 of file gos_sysmon_app_config.h.

4.31.2.31 #define CFG_SYSMON_USE_SERVICE (1)

Sysmon use service flag.

Definition at line 295 of file gos_sysmon_app_config.h.

4.31.2.32 #define CFG_SYSTEM_TASK_STACK_SIZE (0x400)

System task stack size.

Definition at line 98 of file gos_sysmon_app_config.h.

4.31.2.33 #define CFG_TARGET_CPU (ARM_CORTEX_M4)

Target CPU.

Definition at line 54 of file gos_sysmon_app_config.h.

4.31.2.34 #define CFG_TASK_MAX_NAME_LENGTH (32)

Maximum task name length.

Definition at line 74 of file gos_sysmon_app_config.h.

4.31.2.35 #define CFG_TASK_MAX_NUMBER (36)

Maximum number of tasks.

Definition at line 78 of file gos_sysmon_app_config.h.

4.31.2.36 #define CFG_TASK_MAX_STACK_SIZE (0x4000)

Maximum task stack size.

Definition at line 90 of file gos_sysmon_app_config.h.

4.31.2.37 #define CFG_TASK_MESSAGE_DAEMON_PRIO (198)

Message daemon task priority.

Definition at line 138 of file gos_sysmon_app_config.h.

4.31.2.38 #define CFG_TASK_MESSAGE_DAEMON_STACK (0x300)

Message daemon task stack size.

Definition at line 114 of file gos_sysmon_app_config.h.

4.31.2.39 #define CFG_TASK_MIN_STACK_SIZE (0x200)

Minimum task stack size.

Definition at line 86 of file gos_sysmon_app_config.h.

4.31.2.40 #define CFG_TASK_PROC_DAEMON_PRIO (194)

Process daemon task priority.

Definition at line 146 of file gos_sysmon_app_config.h.

4.31.2.41 #define CFG_TASK_PROC_DAEMON_STACK (0x300)

Process daemon task stack size.

Definition at line 106 of file gos_sysmon_app_config.h.

4.31.2.42 #define CFG_TASK_SHELL_DAEMON_PRIO (192)

Shell daemon task priority.

Definition at line 150 of file gos_sysmon_app_config.h.

4.31.2.43 #define CFG_TASK_SHELL_DAEMON_STACK (0x300)

Shell daemon task stack size.

Definition at line 118 of file gos_sysmon_app_config.h.

4.31.2.44 #define CFG_TASK_SIGNAL_DAEMON_PRIO (197)

Signal daemon task priority.

Definition at line 142 of file gos_sysmon_app_config.h.

4.31.2.45 #define CFG_TASK_SIGNAL_DAEMON_STACK (0x400)

Signal daemon task stack size.

Definition at line 102 of file gos_sysmon_app_config.h.

4.31.2.46 #define CFG_TASK_SYS_PRIO (195)

System task priority.

Definition at line 158 of file gos_sysmon_app_config.h.

4.31.2.47 #define CFG_TASK_SYSMON_DAEMON_PRIO (191)

Sysmon daemon task priority.

Definition at line 162 of file gos_sysmon_app_config.h.

4.31.2.48 #define CFG_TASK_SYSMON_DAEMON_STACK (0x400)

Sysmon daemon task stack size.

Definition at line 126 of file gos_sysmon_app_config.h.

4.31.2.49 #define CFG_TASK_TIME_DAEMON_PRIO (196)

Time daemon task priority.

Definition at line 154 of file gos_sysmon_app_config.h.

4.31.2.50 #define CFG_TASK_TIME_DAEMON_STACK (0x300)

Time daemon task stack size.

Definition at line 110 of file gos_sysmon_app_config.h.

4.31.2.51 #define CFG_TASK_TRACE_DAEMON_PRIO (193)

Trace daemon task priority.

Definition at line 134 of file gos_sysmon_app_config.h.

4.31.2.52 `#define CFG_TASK_TRACE_DAEMON_STACK (0x400)`

Log daemon task stack size.

Definition at line 122 of file gos_sysmon_app_config.h.

4.31.2.53 `#define CFG_TRACE_MAX_LENGTH (200)`

Trace maximum (line) length.

Definition at line 287 of file gos_sysmon_app_config.h.

4.31.2.54 `#define CFG_USE_PRIO_INHERITANCE (1)`

Priority inheritance flag for lock.

Definition at line 66 of file gos_sysmon_app_config.h.

4.31.2.55 `#define GOS_CFG_OVERCONFIG`

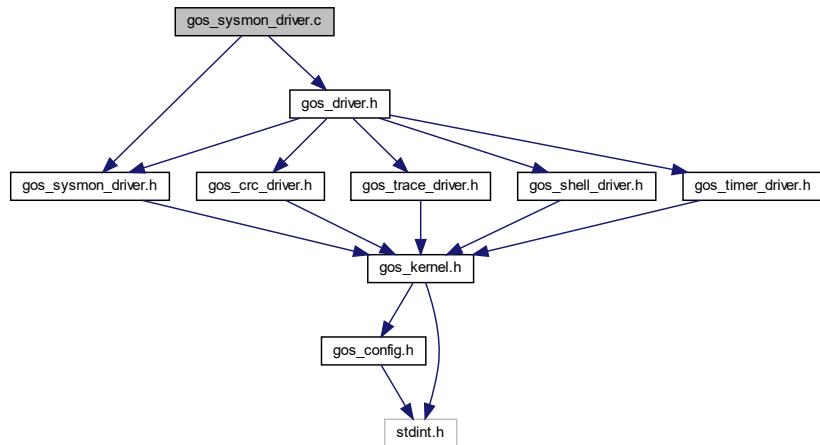
Overconfiguration macro.

Definition at line 42 of file gos_sysmon_app_config.h.

4.32 gos_sysmon_driver.c File Reference

GOS SYSMON driver source.

```
#include "gos_sysmon_driver.h"
#include "gos_driver.h"
Include dependency graph for gos_sysmon_driver.c:
```



Functions

- `gos_result_t gos_sysmonDriverReceive (u8_t *pBuffer, u16_t bufferSize)`
It receives to the given buffer.

- `gos_result_t gos_sysmonDriverTransmit (u8_t *pBuffer, u16_t bufferSize)`

It transmits the given buffer.

Variables

- `GOS_EXTERN gos_driver_functions_t driverFunctions`

4.32.1 Detailed Description

GOS SYSMON driver source.

Author

Ahmed Gazar

Date

2023-07-12

Version

1.0

For a more detailed description of this driver, please refer to [gos_sysmon_driver.h](#)

Definition in file [gos_sysmon_driver.c](#).

4.32.2 Function Documentation

4.32.2.1 `gos_result_t gos_sysmonDriverReceive (u8_t * pBuffer, u16_t bufferSize)`

It receives to the given buffer.

If registered, it calls the receiver function.

Parameters

<code>pBuffer</code>	: Target buffer to receive to.
<code>bufferSize</code>	: Size of the target buffer.

Returns

Result of reception.

Return values

<code>GOS_SUCCESS</code>	: According to user implementation.
<code>GOS_ERROR</code>	: According to user implementation / function not registered.

Definition at line 63 of file [gos_sysmon_driver.c](#).

Here is the caller graph for this function:



4.32.2.2 gos_result_t gos_sysmonDriverTransmit(u8_t * pBuffer, u16_t bufferSize)

It transmits the given buffer.

If registered, it calls the transmitter function.

Parameters

<i>pBuffer</i>	: Buffer to transmit.
<i>bufferSize</i>	: Size of the buffer.

Returns

Result of string transmission.

Return values

<i>GOS_SUCCESS</i>	: According to user implementation.
<i>GOS_ERROR</i>	: According to user implementation / function not registered.

Definition at line 88 of file gos_sysmon_driver.c.

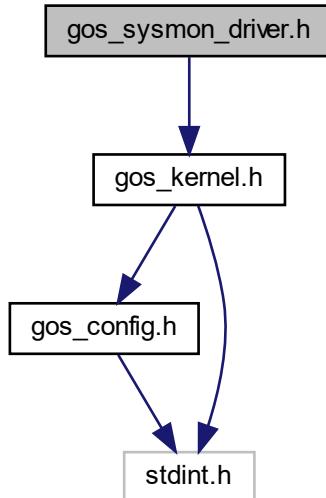
Here is the caller graph for this function:



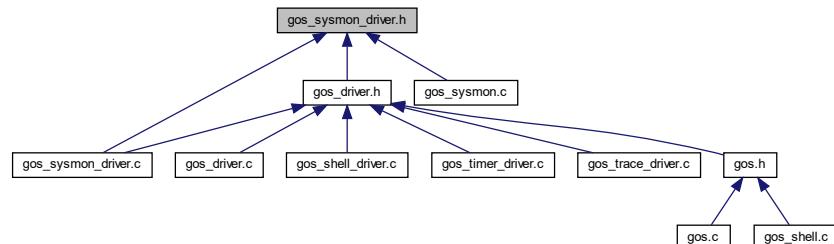
4.33 gos_sysmon_driver.h File Reference

GOS SYSMON driver header.

```
#include "gos_kernel.h"
Include dependency graph for gos_sysmon_driver.h:
```



This graph shows which files directly or indirectly include this file:



Typedefs

- `typedef gos_result_t(* gos_sysmonDriverTransmit_t)(u8_t *, u16_t)`
- `typedef gos_result_t(* gos_sysmonDriverReceive_t)(u8_t *, u16_t)`

Functions

- `gos_result_t gos_sysmonDriverReceive (u8_t *pBuffer, u16_t bufferSize)`
It receives to the given buffer.
- `gos_result_t gos_sysmonDriverTransmit (u8_t *pBuffer, u16_t bufferSize)`
It transmits the given buffer.

4.33.1 Detailed Description

GOS SYSMON driver header.

Author

Ahmed Gazar

Date

2023-07-12

Version

1.0

This driver provides a skeleton for the driver for the sysmon service.

Definition in file [gos_sysmon_driver.h](#).

4.33.2 Typedef Documentation

4.33.2.1 `typedef gos_result_t(* gos_sysmonDriverReceive_t)(u8_t *, u16_t)`

Shell driver transmit string function type.

Definition at line 66 of file gos_sysmon_driver.h.

4.33.2.2 `typedef gos_result_t(* gos_sysmonDriverTransmit_t)(u8_t *, u16_t)`

Shell driver receive character function type.

Definition at line 61 of file gos_sysmon_driver.h.

4.33.3 Function Documentation

4.33.3.1 `gos_result_t gos_sysmonDriverReceive(u8_t * pBuffer, u16_t bufferSize)`

It receives to the given buffer.

If registered, it calls the receiver function.

Parameters

<code>pBuffer</code>	: Target buffer to receive to.
<code>bufferSize</code>	: Size of the target buffer.

Returns

Result of reception.

Return values

<code>GOS_SUCCESS</code>	: According to user implementation.
--------------------------	-------------------------------------

<i>GOS_ERROR</i>	: According to user implementation / function not registered.
------------------	---

Definition at line 63 of file gos_sysmon_driver.c.

Here is the caller graph for this function:



4.33.3.2 `gos_result_t gos_sysmonDriverTransmit(u8_t * pBuffer, u16_t bufferSize)`

It transmits the given buffer.

If registered, it calls the transmitter function.

Parameters

<i>pBuffer</i>	: Buffer to transmit.
<i>bufferSize</i>	: Size of the buffer.

Returns

Result of string transmission.

Return values

<i>GOS_SUCCESS</i>	: According to user implementation.
<i>GOS_ERROR</i>	: According to user implementation / function not registered.

Definition at line 88 of file gos_sysmon_driver.c.

Here is the caller graph for this function:

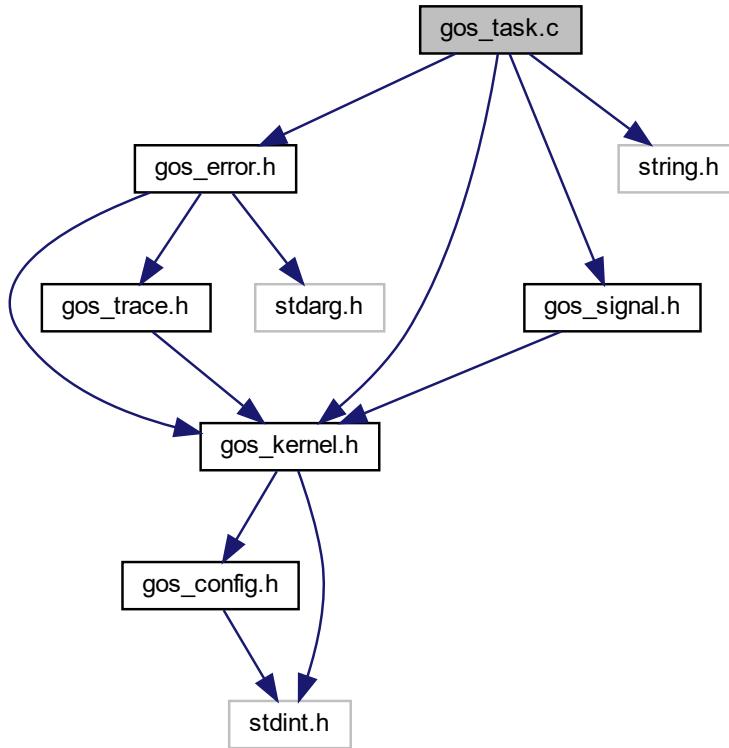


4.34 gos_task.c File Reference

GOS task source.

```
#include <gos_error.h>
#include <gos_kernel.h>
#include <gos_signal.h>
#include <string.h>
```

Include dependency graph for gos_task.c:



Functions

- [GOS_STATIC gos_result_t gos_taskCheckDescriptor \(gos_taskDescriptor_t *taskDescriptor\)](#)
Checks the validity of the task descriptor.
- [void_t gos_idleTask \(void_t\)](#)
Kernel idle task.
- [gos_result_t gos_taskRegisterTasks \(gos_taskDescriptor_t *taskDescriptors, u16_t arraySize\)](#)
This function registers an array of tasks for scheduling.
- [gos_result_t gos_taskRegister \(gos_taskDescriptor_t *taskDescriptor, gos_tid_t *taskId\)](#)
This function registers a task for scheduling.
- [GOS_INLINE gos_result_t gos_taskSleep \(gos_taskSleepTick_t sleepTicks\)](#)
Sends the current task to sleeping state.
- [GOS_INLINE gos_result_t gos_taskWakeup \(gos_tid_t taskId\)](#)
Wakes up the given task.
- [GOS_INLINE gos_result_t gos_taskSuspend \(gos_tid_t taskId\)](#)
Sends the given task to suspended state.
- [GOS_INLINE gos_result_t gos_taskResume \(gos_tid_t taskId\)](#)
Resumes the given task.
- [GOS_INLINE gos_result_t gos_taskBlock \(gos_tid_t taskId, gos_blockMaxTick_t blockTicks\)](#)
Sends the given task to blocked state.
- [GOS_INLINE gos_result_t gos_taskUnblock \(gos_tid_t taskId\)](#)

- **GOS_INLINE gos_result_t gos_taskDelete (gos_tid_t taskId)**
Deletes the given task from the scheduling array.
- **GOS_INLINE gos_result_t gos_taskSetPriority (gos_tid_t taskId, gos_taskPrio_t taskPriority)**
Sets the current priority of the given task to the given value (for temporary change).
- **GOS_INLINE gos_result_t gos_taskSetOriginalPriority (gos_tid_t taskId, gos_taskPrio_t taskPriority)**
Sets the original priority of the given task to the given value (for permanent change).
- **gos_result_t gos_taskGetPriority (gos_tid_t taskId, gos_taskPrio_t *taskPriority)**
Gets the current priority of the given task.
- **gos_result_t gos_taskGetOriginalPriority (gos_tid_t taskId, gos_taskPrio_t *taskPriority)**
Gets the original priority of the given task.
- **GOS_INLINE gos_result_t gos_taskAddPrivilege (gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)**
Adds the given privileges to the given task.
- **GOS_INLINE gos_result_t gos_taskRemovePrivilege (gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)**
Removes the given privileges from the given task.
- **GOS_INLINE gos_result_t gos_taskSetPrivileges (gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)**
Sets the given privileges for the given task.
- **GOS_INLINE gos_result_t gos_taskGetPrivileges (gos_tid_t taskId, gos_taskPrivilegeLevel_t *privileges)**
Gets the privileges of the given task.
- **GOS_INLINE gos_result_t gos_taskYield (void_t)**
Yields the current task.
- **gos_result_t gos_taskGetName (gos_tid_t taskId, gos_taskName_t taskName)**
Gets the task name of the task with the given ID.
- **gos_result_t gos_taskGetId (gos_taskName_t taskName, gos_tid_t *taskId)**
Gets the task ID of the task with the given name.
- **GOS_INLINE gos_result_t gos_taskGetCurrentId (gos_tid_t *taskId)**
Returns the ID of the currently running task.
- **gos_result_t gos_taskGetData (gos_tid_t taskId, gos_taskDescriptor_t *taskData)**
Returns the task data of the given task.
- **gos_result_t gos_taskGetDataByIndex (u16_t taskIndex, gos_taskDescriptor_t *taskData)**
Returns the task data of the given task.
- **gos_result_t gos_taskGetNumber (u16_t *pTaskNum)**
Returns the number of registered tasks.
- **gos_result_t gos_taskRegisterIdleHook (gos_taskIdleHook_t idleHookFunction)**
- **gos_result_t gos_taskSubscribeDeleteSignal (gos_signalHandler_t deleteSignalHandler)**

Variables

- **gos_signallt kernelTaskDeleteSignal**
- **GOS_STATIC gos_taskIdleHook_t kernelIdleHookFunction = NULL**
- **GOS_EXTERN u8_t inlsr**
- **GOS_EXTERN u32_t currentTaskIndex**
- **gos_taskDescriptor_t taskDescriptors [CFG_TASK_MAX_NUMBER]**

4.34.1 Detailed Description

GOS task source.

Author

Ahmed Gazar

Date

2024-06-13

Version

1.2

For a more detailed description of this module, please refer to [gos_kernel.h](#)

Definition in file [gos_task.c](#).

4.34.2 Function Documentation

4.34.2.1 void_t gos_idleTask(void_t)

Kernel idle task.

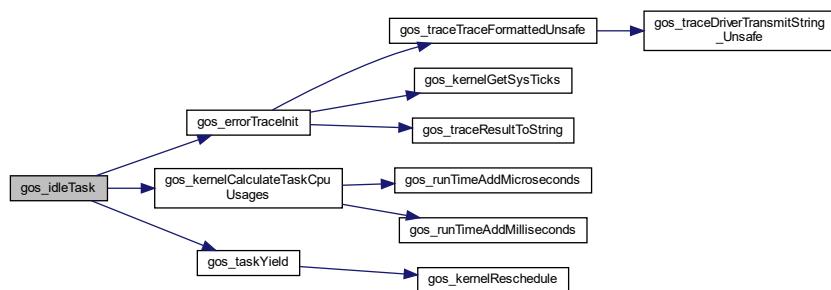
This task is executed when there is no other ready task in the system. When executed, this function refreshes the CPU-usage statistics of tasks.

Returns

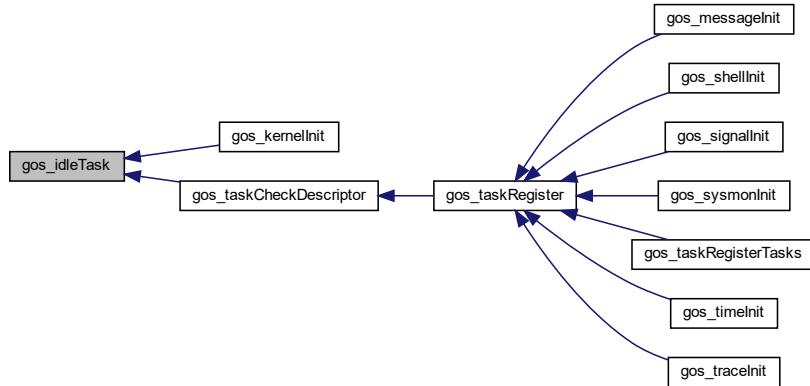
-

Definition at line 1314 of file [gos_task.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.2 GOS_INLINE gos_result_t gos_taskAddPrivilege(gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)

Adds the given privileges to the given task.

Checks the caller task if it has the privilege to modify task privileges and if so, it adds the given privileges to the given task.

Parameters

<i>taskId</i>	: ID of the task to give the privileges to.
<i>privileges</i>	: Privileges to be added.

Returns

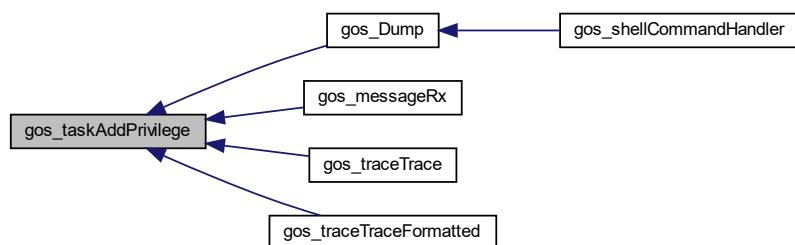
Result of privilege adding.

Return values

<i>GOS_SUCCESS</i>	: Privileges added successfully.
<i>GOS_ERROR</i>	: Invalid task ID or caller does not have the privilege to modify task privileges.

Definition at line 873 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.3 GOS_INLINE gos_result_t gos_taskBlock (gos_tid_t taskId, gos_blockMaxTick_t blockTicks)

Sends the given task to blocked state.

Checks the given task ID and its state, modified it to blocked, and if there is a block hook function registered, it calls it. If the blocked function is the currently running one, it invokes a rescheduling.

Parameters

<i>taskId</i>	: ID of the task to be blocked.
---------------	---------------------------------

Returns

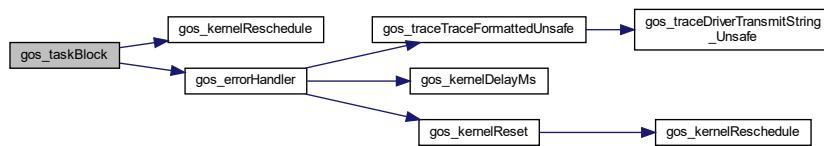
Result of task blocking.

Return values

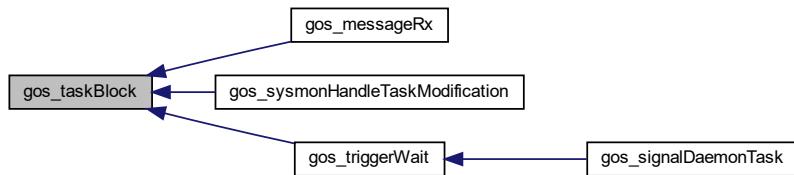
<i>GOS_SUCESS</i>	: Task blocked successfully.
<i>GOS_ERROR</i>	: Task ID is invalid, or task state is not ready.

Definition at line 507 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.4 GOS_STATIC gos_result_t gos_taskCheckDescriptor (gos_taskDescriptor_t * taskDescriptor)

Checks the validity of the task descriptor.

It checks the task function pointer, task priority value, stack, and size parameters and return with error if the parameters are incorrect.

Parameters

<code>taskDescriptor</code>	: Pointer to the task descriptor structure.
-----------------------------	---

Returns

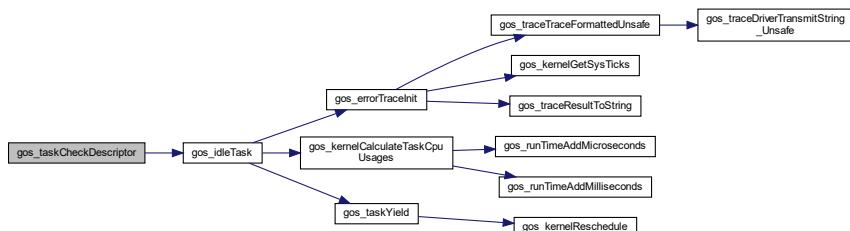
Result of task descriptor checking.

Return values

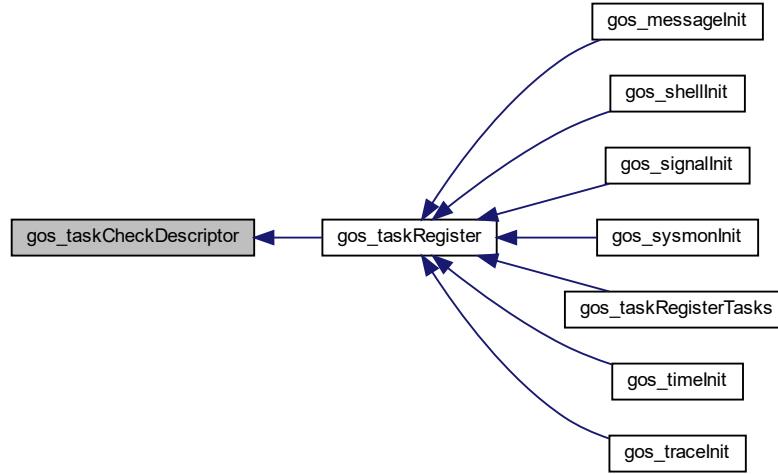
<code>GOS_SUCCESS</code>	: Descriptor contains valid data.
<code>GOS_ERROR</code>	: One or more parameter is invalid: <ul style="list-style-type: none"> • Task function is NULL pointer • Task function is the idle task • Priority exceeds the maximum priority level • Stack size is smaller than the minimum allowed • Stack size is greater than the maximum allowed • Stack size is not 4-byte aligned

Definition at line 1279 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.5 GOS_INLINE gos_result_t gos_taskDelete (gos_tid_t taskId)

Deletes the given task from the scheduling array.

Checks the given task ID and its state, modifies it to zombie, and if there is a delete hook function registered, it calls it.

Parameters

<code>taskId</code>	: ID of the task to be deleted.
---------------------	---------------------------------

Returns

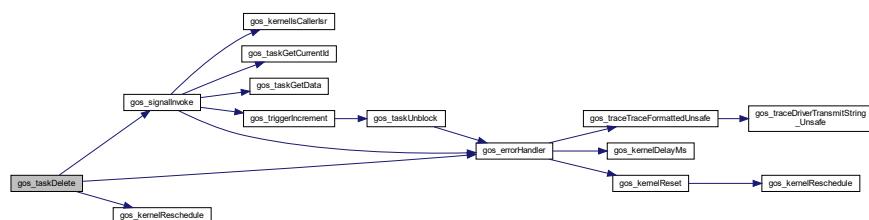
Result of deletion.

Return values

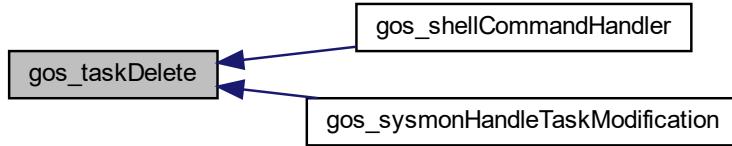
<code>GOS_SUCCESS</code>	: Task deleted successfully.
<code>GOS_ERROR</code>	: Task is already a zombie.

Definition at line 639 of file `gos_task.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.6 GOS_INLINE gos_result_t gos_taskGetCurrentId(gos_tid_t * taskId)

Returns the ID of the currently running task.

Returns the ID of the currently running task.

Parameters

<code>taskId</code>	: Pointer to a task ID variable to store the current task ID.
---------------------	---

Returns

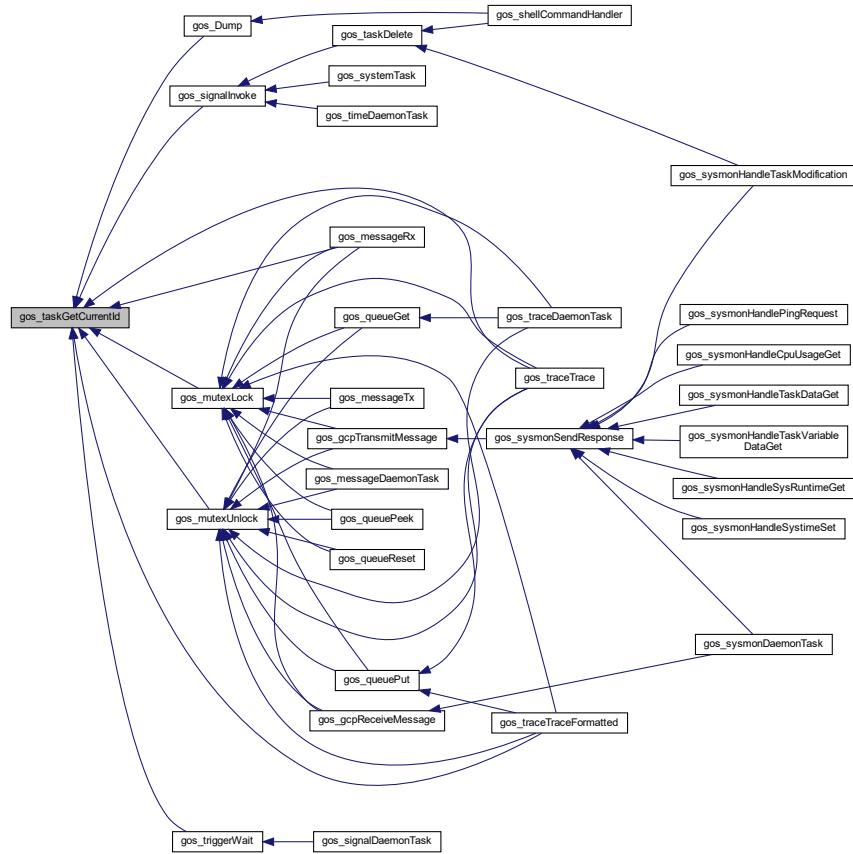
Result of current task ID get.

Return values

<code>GOS_SUCCESS</code>	: Current task ID returned successfully.
<code>GOS_ERROR</code>	: Task ID pointer is NULL.

Definition at line 1074 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.7 gos_result_t gos_taskGetData(gos_tid_t taskId, gos_taskDescriptor_t * taskData)

Returns the task data of the given task.

Based on the task ID, it copies the content of the internal task descriptor array element to the given task descriptor.

Parameters

<i>taskId</i>	: ID of the task to get the data of.
<i>taskData</i>	: Pointer to the task descriptor to save the task data in.

Returns

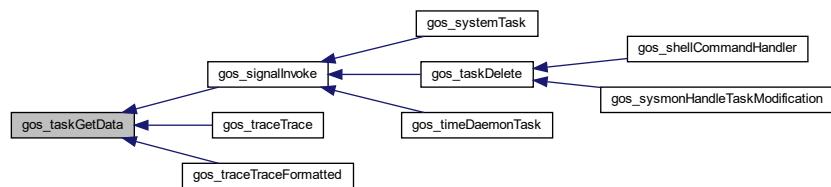
Result of task data get.

Return values

<i>GOS_SUCCESS</i>	: Task data copied successfully.
<i>GOS_ERROR</i>	: Invalid task ID or task data pointer is NULL.

Definition at line 1102 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.8 `gos_result_t gos_taskGetDataByIndex(u16_t taskIndex, gos_taskDescriptor_t *taskData)`

Returns the task data of the given task.

Based on the task index, it copies the content of the internal task descriptor array element to the given task descriptor.

Parameters

<i>taskIndex</i>	: Index of the task to get the data of.
<i>taskData</i>	: Pointer to the task descriptor to save the task data in.

Returns

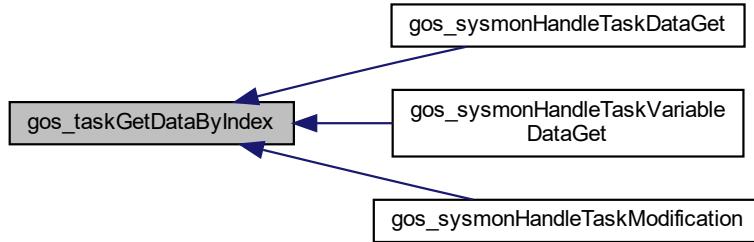
Result of task data get.

Return values

<i>GOS_SUCCESS</i>	: Task data copied successfully.
<i>GOS_ERROR</i>	: Invalid task index or task data pointer is NULL.

Definition at line 1135 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.9 `gos_result_t gos_taskGetId(gos_taskName_t taskName, gos_tid_t * taskId)`

Gets the task ID of the task with the given name.

This function loops through the internal task array and tries to find the given task name to get the corresponding task ID.

Parameters

<code>taskName</code>	: Name of the task (string).
<code>taskId</code>	: Pointer to a task ID variable to store the returned ID.

Returns

Success of task ID get.

Return values

<code>GOS_SUCCESS</code>	: Task ID found successfully.
<code>GOS_ERROR</code>	: Task name not found.

Definition at line 1043 of file `gos_task.c`.

Here is the caller graph for this function:



4.34.2.10 `gos_result_t gos_taskGetName(gos_tid_t taskId, gos_taskName_t taskName)`

Gets the task name of the task with the given ID.

Copies the task name corresponding with the given task ID to the task name variable.

Parameters

<i>taskId</i>	: Pointer to a task ID variable to store the returned ID.
<i>taskName</i>	: Task name pointer to store the returned task name.

Returns

Success of task name get.

Return values

<i>GOS_SUCCESS</i>	: Task name found successfully.
<i>GOS_ERROR</i>	: Invalid task ID or task name variable is NULL.

Definition at line 1012 of file gos_task.c.

4.34.2.11 gos_result_t gos_taskGetNumber(u16_t * pTaskNum)

Returns the number of registered tasks.

Loops through the internal task array and counts the entries where a task function is registered.

Parameters

<i>pTaskNum</i>	: Variable to store the number of tasks.
-----------------	--

Returns

Success of task number counting.

Return values

<i>GOS_SUCCESS</i>	: Task number counted successfully.
<i>GOS_ERROR</i>	: Target variable is NULL pointer.

Definition at line 1166 of file gos_task.c.

4.34.2.12 gos_result_t gos_taskGetOriginalPriority(gos_tid_t taskId, gos_taskPrio_t * taskPriority)

Gets the original priority of the given task.

Checks the given parameters and saves the original priority in the given variable.

Parameters

<i>taskId</i>	: ID of the task to get the priority of.
<i>taskPriority</i>	: Pointer to a priority variable to store the priority in.

Returns

Result of priority getting.

Return values

<i>GOS_SUCCESS</i>	: Original priority getting successfully.
<i>GOS_ERROR</i>	: Invalid task ID or priority variable is NULL.

Definition at line 842 of file gos_task.c.

4.34.2.13 gos_result_t gos_taskGetPriority (gos_tid_t taskId, gos_taskPrio_t * taskPriority)

Gets the current priority of the given task.

Checks the given parameters and saves the current priority in the given variable.

Parameters

<i>taskId</i>	: ID of the task to get the priority of.
<i>taskPriority</i>	: Pointer to a priority variable to store the priority in.

Returns

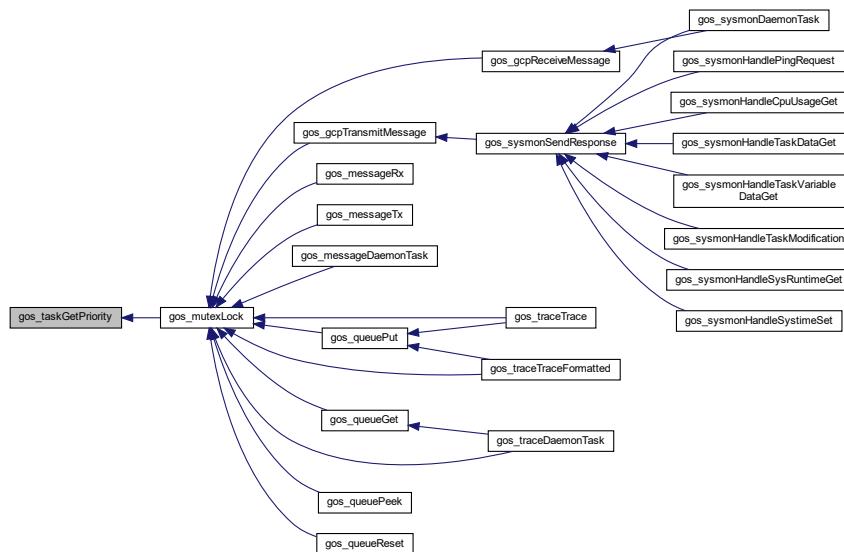
Result of priority getting.

Return values

<i>GOS_SUCCESS</i>	: Current priority getting successfully.
<i>GOS_ERROR</i>	: Invalid task ID or priority variable is NULL.

Definition at line 811 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.14 GOS_INLINE gos_result_t gos_taskGetPrivileges (*gos_tid_t taskId*, *gos_taskPrivilegeLevel_t *privileges*)

Gets the privileges of the given task.

Returns the privilege flags of the given task.

Parameters

<i>taskId</i>	: ID of the task to get the privileges of.
<i>privileges</i>	: Variable to store the privilege flags.

Returns

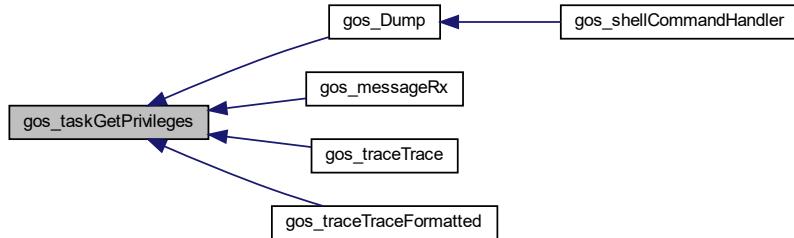
Result of privilege getting.

Return values

<i>GOS_SUCCESS</i>	: Privileges get successful.
<i>GOS_ERROR</i>	: Invalid task ID or privilege variable is NULL pointer.

Definition at line 963 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.15 `gos_result_t gos_taskRegister(gos_taskDescriptor_t *taskDescriptor, gos_tid_t *taskId)`

This function registers a task for scheduling.

Checks the task descriptor parameters and then tries to find the next empty slot in the internal task array. When it is found, it registers the task in that slot.

Parameters

<i>taskDescriptor</i>	: Pointer to a task descriptor structure.
<i>taskId</i>	: Pointer to a variable to hold to assigned task ID value.

Returns

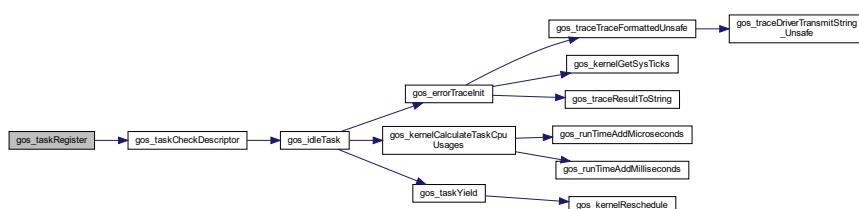
Result of task registration.

Return values

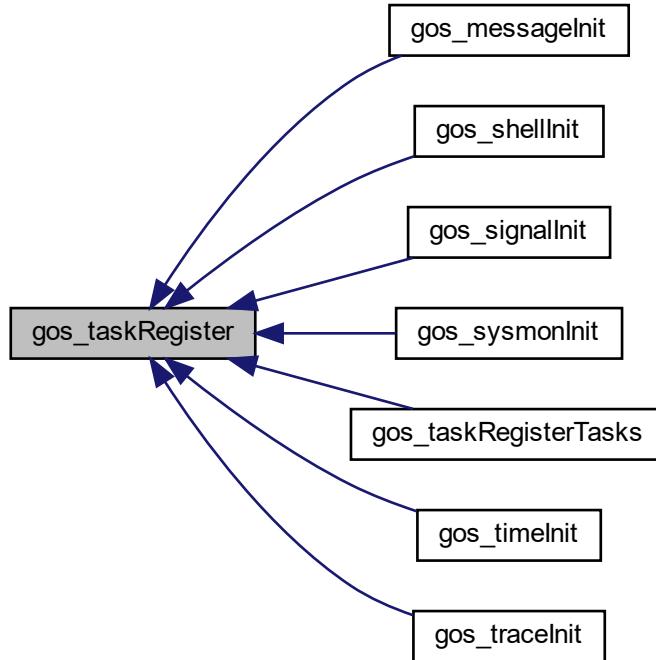
<i>GOS_SUCCESS</i>	: Task registered successfully.
<i>GOS_ERROR</i>	: Invalid task descriptor (NULL function pointer, invalid priority level, invalid stack size, idle task registration, or stack size is not 4-byte-aligned) or task array is full.

Definition at line 159 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.16 `gos_result_t gos_taskRegisterTasks (gos_taskDescriptor_t * taskDescriptors, u16_t arraySize)`

This function registers an array of tasks for scheduling.

Checks the task descriptor array pointer and registers the tasks one by one.

Parameters

<code>taskDescriptors</code>	: Pointer to a task descriptor structure array.
<code>arraySize</code>	: Size of the array in bytes.

Returns

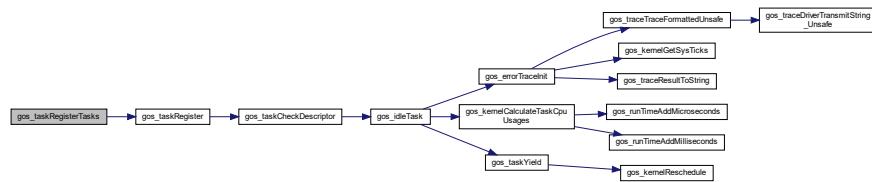
Result of task registration.

Return values

<code>GOS_SUCCESS</code>	: Tasks registered successfully.
<code>GOS_ERROR</code>	: Invalid task descriptor (NULL function pointer, invalid priority level, invalid stack size, idle task registration, or stack size is not 4-byte-aligned) in one of the array elements or task array is full.

Definition at line 111 of file `gos_task.c`.

Here is the call graph for this function:



4.34.2.17 GOS_INLINE gos_result_t gos_taskRemovePrivilege (gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)

Removes the given privileges from the given task.

Checks the caller task if it has the privilege to modify task privileges and if so, it removes the given privileges from the given task.

Parameters

<i>taskId</i>	: ID of the task to remove the privileges from.
<i>privileges</i>	: Privileges to be removed.

Returns

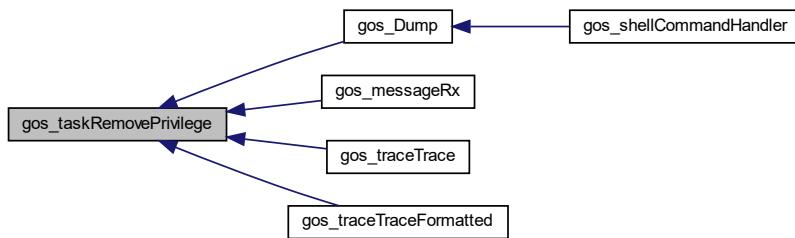
Result of privilege removing.

Return values

<i>GOS_SUCCESS</i>	: Privileges removed successfully.
<i>GOS_ERROR</i>	: Invalid task ID or caller does not have the privilege to modify task privileges.

Definition at line 903 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.18 GOS_INLINE gos_result_t gos_taskResume (gos_tid_t taskId)

Resumes the given task.

Checks the given task ID and its state, modified it to ready, and if there is a resume hook function registered, it calls it.

Parameters

<i>taskId</i>	: ID of the task to be resumed.
---------------	---------------------------------

Returns

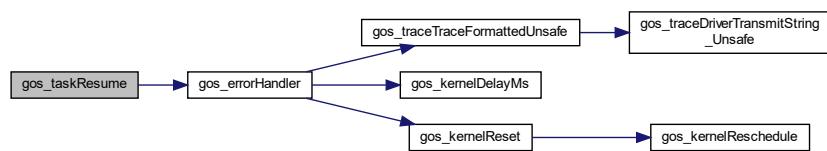
Result of task resumption.

Return values

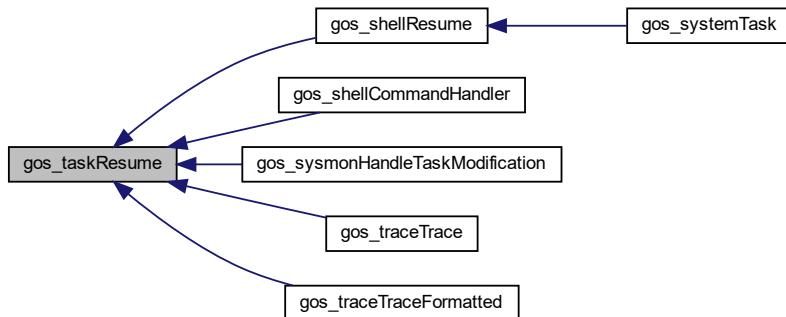
<i>GOS_SUCESS</i>	: Task resumed successfully.
<i>GOS_ERROR</i>	: Task ID is invalid, or task is not suspended.

Definition at line 456 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.34.2.19 GOS_INLINE gos_result_t gos_taskSetOriginalPriority(gos_tid_t taskId, gos_taskPrio_t taskPriority)**

Sets the original priority of the given task to the given value (for permanent change).

Checks the given parameters and sets the original priority of the given task.

Parameters

<i>taskId</i>	: ID of the task to change the priority of.
---------------	---

<i>taskPriority</i>	: The desired task priority.
---------------------	------------------------------

Returns

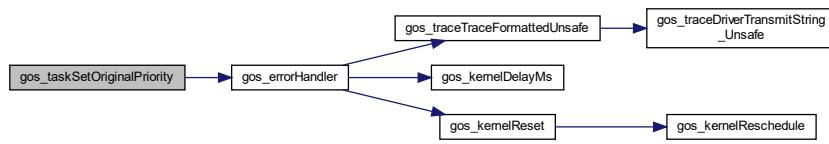
Result of priority change.

Return values

<i>GOS_SUCCESS</i>	: Original priority changed successfully.
<i>GOS_ERROR</i>	: Invalid task ID or priority.

Definition at line 765 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.34.2.20 GOS_INLINE gos_result_t gos_taskSetPriority (*gos_tid_t taskId*, *gos_taskPrio_t taskPriority*)**

Sets the current priority of the given task to the given value (for temporary change).

Checks the given parameters and sets the current priority of the given task.

Parameters

<i>taskId</i>	: ID of the task to change the priority of.
<i>taskPriority</i>	: The desired task priority.

Returns

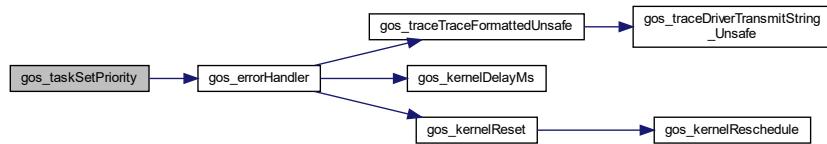
Result of priority change.

Return values

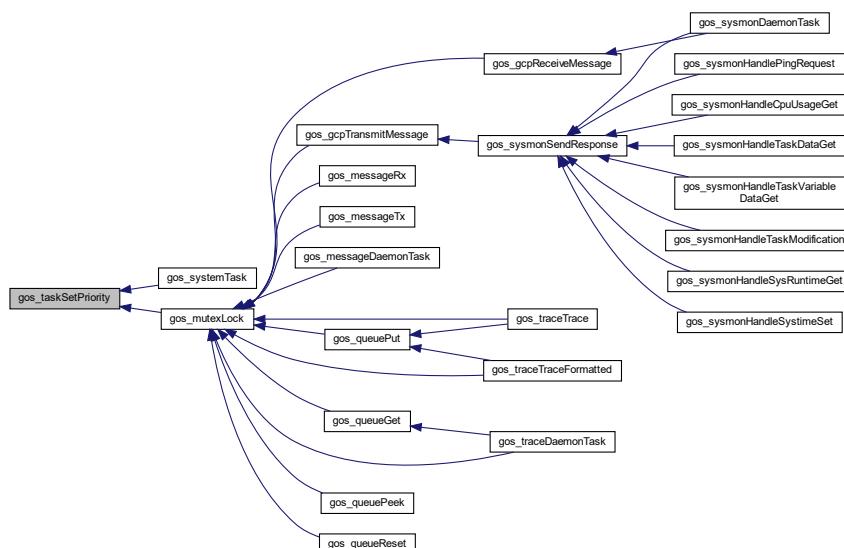
<code>GOS_SUCCESS</code>	: Current priority changed successfully.
<code>GOS_ERROR</code>	: Invalid task ID or priority.

Definition at line 719 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.21 GOS_INLINE gos_result_t gos_taskSetPrivileges(gos_tid_t taskId, gos_taskPrivilegeLevel_t privileges)

Sets the given privileges for the given task.

Checks the caller task if it has the privilege to modify task privileges and if so, it sets the given privileges for the given task.

Parameters

<code>taskId</code>	: ID of the task to set the privileges for.
<code>privileges</code>	: Privileges to be set.

Returns

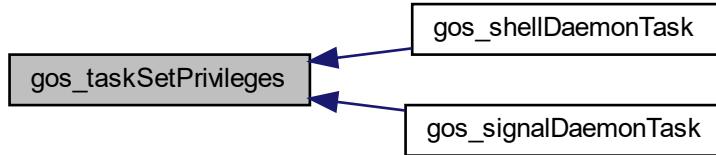
Result of privilege setting.

Return values

<i>GOS_SUCCESS</i>	: Privileges set successfully.
<i>GOS_ERROR</i>	: Invalid task ID or caller does not have the privilege to modify task privileges.

Definition at line 933 of file gos_task.c.

Here is the caller graph for this function:



4.34.2.22 GOS_INLINE gos_result_t gos_taskSleep (gos_taskSleepTick_t sleepTicks)

Sends the current task to sleeping state.

Checks the current task and its state, modifies it to sleeping, and if there is a sleep hook function registered, it calls it. Then, it invokes a rescheduling.

Parameters

<i>sleepTicks</i>	: Minimum number of ticks until the task should remain in sleeping state.
-------------------	---

Returns

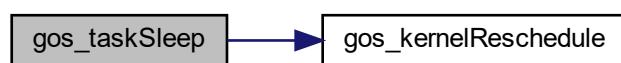
Result of task sleeping.

Return values

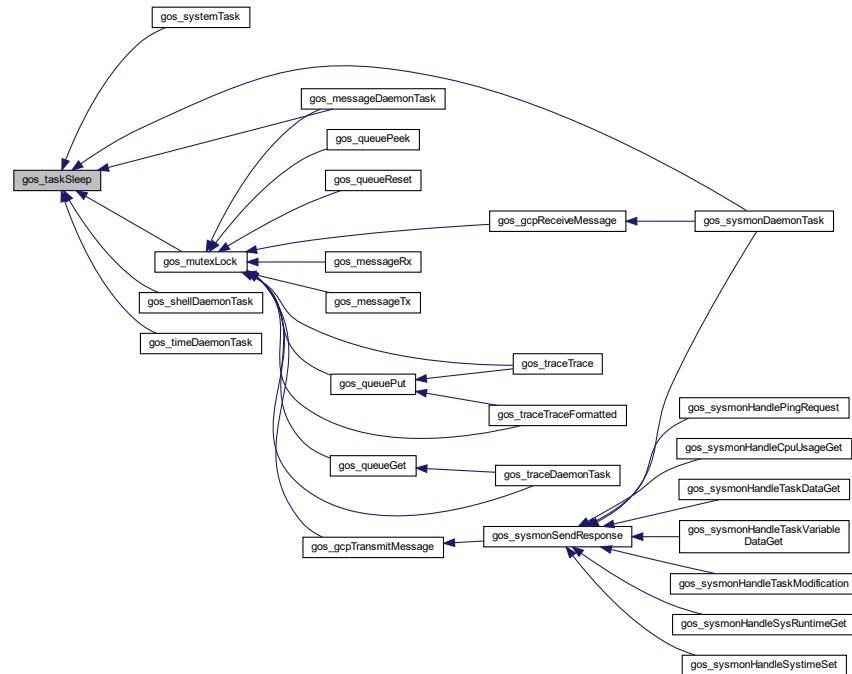
<i>GOS_SUCCESS</i>	: Task successfully sent to sleeping state.
<i>GOS_ERROR</i>	: Function called from idle task or task state is not ready.

Definition at line 282 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.23 GOS_INLINE gos_result_t gos_taskSuspend(gos_tid_t taskId)

Sends the given task to suspended state.

Checks the given task ID and its state, modified it to suspended, and if there is a suspend hook function registered, it calls it. If the suspended function is the currently running one, it invokes a rescheduling.

Parameters

<code>taskId</code>	: ID of the task to be suspended.
---------------------	-----------------------------------

Returns

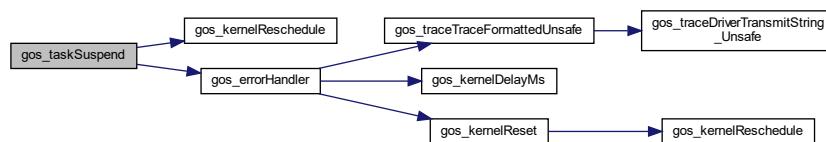
Result of task suspension.

Return values

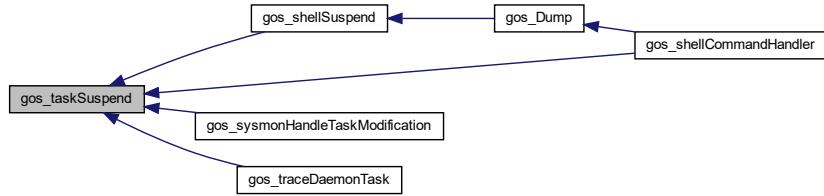
<code>GOS_SUCESS</code>	: Task suspended successfully.
<code>GOS_ERROR</code>	: Task ID is invalid, or task state is not ready or sleeping.

Definition at line 382 of file `gos_task.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.24 GOS_INLINE gos_result_t gos_taskUnblock(gos_tid_t taskId)

Unblocks the given task.

Checks the given task ID and its state, modified it to ready, and if there is an unblock hook function registered, it calls it.

Parameters

<i>taskId</i>	: ID of the task to be unblocked.
---------------	-----------------------------------

Returns

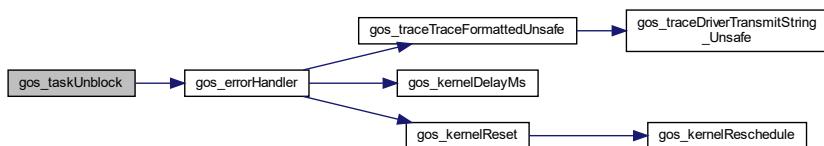
Result of task unblocking.

Return values

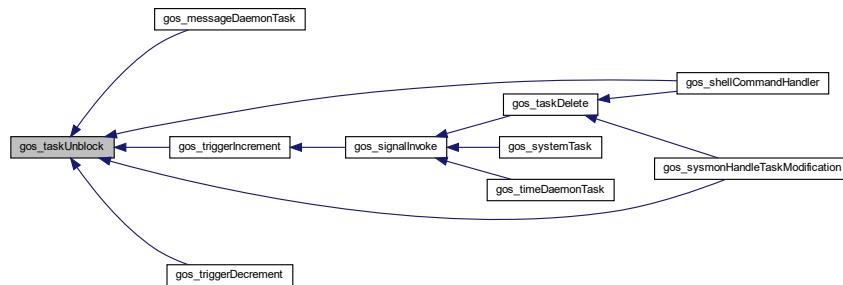
<i>GOS_SUCESS</i>	: Task unblocked successfully.
<i>GOS_ERROR</i>	: Task ID is invalid, or task is not blocked.

Definition at line 582 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.25 GOS_INLINE gos_result_t gos_taskWakeup (gos_tid_t taskId)

Wakes up the given task.

Checks the current task and its state, modifies it to ready, and if there is a wake-up hook function registered, it calls it.

Parameters

<code>taskId</code>	: ID of the task to be waken up.
---------------------	----------------------------------

Returns

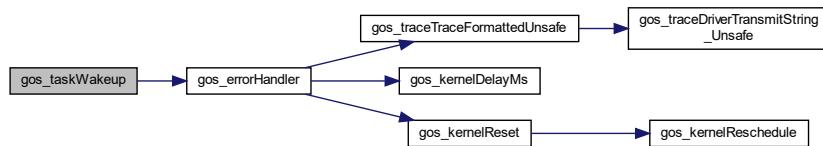
Result of task wake-up.

Return values

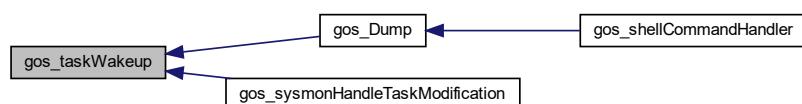
<code>GOS_SUCCESS</code>	: Task waken up successfully.
<code>GOS_ERROR</code>	: Task ID is invalid, or task is not sleeping.

Definition at line 331 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.2.26 GOS_INLINE gos_result_t gos_taskYield(void_t)

Yields the current task.

Invokes rescheduling.

Returns

Result of task yield.

Return values

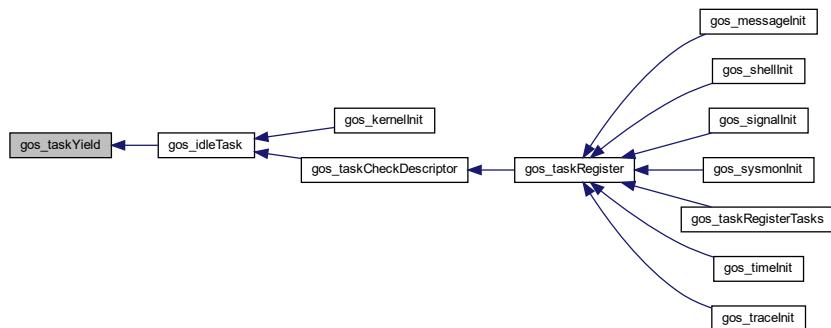
<code>GOS_SUCCESS</code>	: Yield successful.
--------------------------	---------------------

Definition at line 995 of file gos_task.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.34.3 Variable Documentation

4.34.3.1 GOS_STATIC gos_taskIdleHook_t kernelIdleHookFunction = NULL

Kernel idle hook function.

Definition at line 72 of file gos_task.c.

4.34.3.2 gos_signalId_t kernelTaskDeleteSignal

Kernel task delete signal.

Definition at line 64 of file gos_task.c.

4.34.3.3 gos_taskDescriptor_t taskDescriptors[CFG_TASK_MAX_NUMBER]

Initial value:

```
=
{
    [0] =
    {
        .taskFunction      = gos_idleTask,
        .taskId           = GOS_DEFAULT_TASK_ID,
        .taskPriority     = GOS_TASK_IDLE_PRIO,
        .taskName          = "gos_idle_task",
        .taskState         = GOS_TASK_READY,
        .taskStackSize     = CFG_IDLE_TASK_STACK_SIZE,
        .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL,
        .taskCpuUsageLimit = 10000
    }
}
```

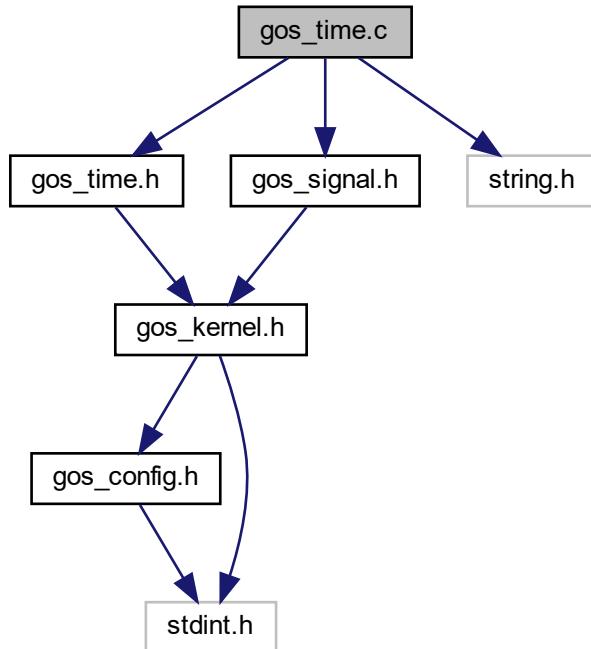
Internal task array.

Definition at line 93 of file gos_task.c.

4.35 gos_time.c File Reference

GOS time service source.

```
#include <gos_time.h>
#include <gos_signal.h>
#include <string.h>
Include dependency graph for gos_time.c:
```



Macros

- #define TIME_DEFAULT_YEAR (2023)
- #define TIME_DEFAULT_MONTH (GOS_TIME_JANUARY)
- #define TIME_DEFAULT_DAY (1)
- #define TIME_SLEEP_TIME_MS (500u)

Functions

- **GOS_STATIC void_t gos_timeDaemonTask (void_t)**
Time daemon task.
- **gos_result_t gos_timeInit (void_t)**
This function initializes the time service.
- **gos_result_t gos_timeGet (gos_time_t *pTime)**
This function gets the system time.
- **gos_result_t gos_timeSet (gos_time_t *pTime)**
This function sets the system time.
- **gos_result_t gos_runTimeGet (gos_runtime_t *pRunTime)**
This function gets the system run-time.
- **gos_result_t gos_timeCompare (gos_time_t *pTime1, gos_time_t *pTime2, gos_timeCompareResult_t *result)**
This function compares two time structures.
- **gos_result_t gos_timeAddMilliseconds (gos_time_t *pTime, u16_t milliseconds)**
This function adds the given number of milliseconds to the given time structure.
- **gos_result_t gos_timeAddSeconds (gos_time_t *pTime, u16_t seconds)**
This function adds the given number of seconds to the given time variable.
- **gos_result_t gos_runTimeAddMicroseconds (gos_runtime_t *pRunTime1, gos_runtime_t *pRunTime2, u16_t microseconds)**
This function adds the given number of microseconds to the given time variables.
- **gos_result_t gos_runTimeAddMilliseconds (gos_runtime_t *pRunTime, u16_t milliseconds)**
This function adds the given number of milliseconds to the given time variables.
- **gos_result_t gos_runTimeAddSeconds (gos_runtime_t *pRunTime, u32_t seconds)**
This function adds the given number of seconds to the given run-time variable.
- **gos_result_t gos_timeIncreaseSystemTime (u16_t milliseconds)**
This function increases the system time and runtime with the given value of milliseconds.

Variables

- **GOS_STATIC gos_time_t systemTime**
- **GOS_STATIC gos_runtime_t systemRunTime**
- **GOS_STATIC GOS_CONST gos_day_t dayLookupTable [GOS_TIME_NUMBER_OF_MONTHS]**
- **GOS_STATIC gos_tid_t timeDaemonTaskId**
- **gos_signallId_t timeSignallId**
- **GOS_STATIC gos_taskDescriptor_t timeDaemonTaskDesc**

4.35.1 Detailed Description

GOS time service source.

Author

Ahmed Gazar

Date

2023-10-04

Version

1.7

For a more detailed description of this service, please refer to [gos_time.h](#)

Definition in file [gos_time.c](#).

4.35.2 Macro Definition Documentation

4.35.2.1 #define TIME_DEFAULT_DAY (1)

Default day.

Definition at line 85 of file gos_time.c.

4.35.2.2 #define TIME_DEFAULT_MONTH (GOS_TIME_JANUARY)

Default month.

Definition at line 80 of file gos_time.c.

4.35.2.3 #define TIME_DEFAULT_YEAR (2023)

Default year.

Definition at line 75 of file gos_time.c.

4.35.2.4 #define TIME_SLEEP_TIME_MS (500u)

Time task sleep time in [ms].

Definition at line 90 of file gos_time.c.

4.35.3 Function Documentation

4.35.3.1 `gos_result_t gos_runTimeAddMicroseconds (gos_runtime_t * pRunTime1, gos_runtime_t * pRunTime2, u16_t microseconds)`

This function adds the given number of microseconds to the given time variables.

This function adds the given number of microseconds to the given time variables.

Parameters

<code>pRunTime1</code>	: Pointer to the time variable.
<code>pRunTime2</code>	: Pointer to the time variable.
<code>microseconds</code>	: Number of microseconds to add.

Returns

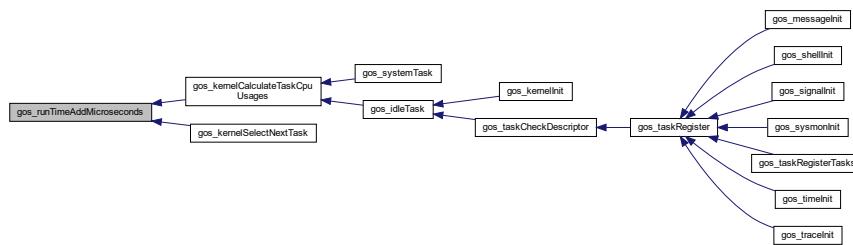
Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Microseconds added successfully.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 497 of file gos_time.c.

Here is the caller graph for this function:



4.35.3.2 *gos_result_t gos_runTimeAddMilliseconds (gos_runtime_t * pRunTime, u16_t milliseconds)*

This function adds the given number of milliseconds to the given time variables.

This function adds the given number of milliseconds to the given time variables.

Parameters

<i>pRunTime1</i>	: Pointer to the time variable.
<i>pRunTime2</i>	: Pointer to the time variable.
<i>milliseconds</i>	: Number of milliseconds to add.

Returns

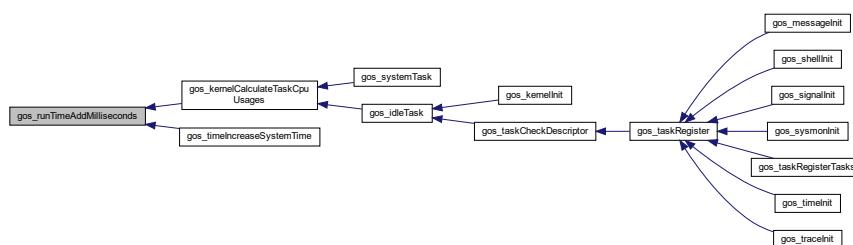
Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Milliseconds added successfully.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 637 of file gos_time.c.

Here is the caller graph for this function:



4.35.3.3 gos_result_t gos_runTimeAddSeconds (gos_runtime_t * pRunTime, u32_t seconds)

This function adds the given number of seconds to the given run-time variable.

This function adds the given number of seconds to the given run-time variable.

Parameters

<i>pRunTime</i>	: Pointer to a run-time variable.
<i>seconds</i>	: Number of seconds to add.

Returns

Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Seconds added successfully.
<i>GOS_ERROR</i>	: Run-time variable is NULL pointer.

Definition at line 706 of file gos_time.c.

4.35.3.4 gos_result_t gos_runTimeGet (gos_runtime_t * pRunTime)

This function gets the system run-time.

This function gets the system run-time.

Parameters

<i>pRunTime</i>	: Pointer to a run-time variable to store the system run-time in.
-----------------	---

Returns

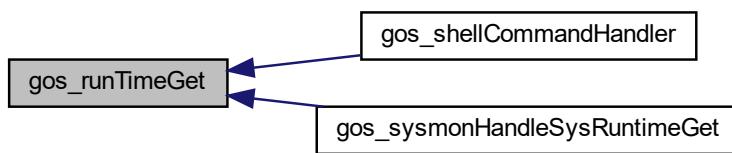
Result of run-time getting.

Return values

<i>GOS_SUCCESS</i>	: Run-time getting is successful.
<i>GOS_ERROR</i>	: Run-time variable is NULL pointer.

Definition at line 240 of file gos_time.c.

Here is the caller graph for this function:



4.35.3.5 gos_result_t gos_timeAddMilliseconds (gos_time_t * pTime, u16_t milliseconds)

This function adds the given number of milliseconds to the given time structure.

This function adds the given number of milliseconds to the given time structure.

Parameters

<i>pTime</i>	: Pointer to the time structure.
<i>milliseconds</i>	: Number of milliseconds to add.

Returns

Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Increasing successful.
<i>GOS_ERROR</i>	: Time structure is NULL pointer.

Definition at line 311 of file gos_time.c.

Here is the caller graph for this function:



4.35.3.6 *gos_result_t gos_timeAddSeconds(gos_time_t * pTime, u16_t seconds)*

This function adds the given number of seconds to the given time variable.

This function adds the given number of seconds to the given time variable.

Parameters

<i>pTime</i>	: Pointer to the time variable.
<i>seconds</i>	: Number of seconds to add.

Returns

Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Seconds added successfully.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 407 of file gos_time.c.

4.35.3.7 *gos_result_t gos_timeCompare(gos_time_t * pTime1, gos_time_t * pTime2, gos_timeCompareResult_t * result)*

This function compares two time structures.

This function compares two time structures.

Parameters

<i>pTime1</i>	: Pointer to the first time variable.
<i>pTime2</i>	: Pointer to the second time variable.
<i>result</i>	: Pointer to the comparison result variable.

Returns

Result of time comparison.

Return values

<i>GOS_SUCCESS</i>	: Time comparison successful.
<i>GOS_ERROR</i>	: Either time structure and/or result variable is NULL pointer.

Definition at line 267 of file gos_time.c.

4.35.3.8 GOS_STATIC void_t gos_timeDaemonTask(void_t)

Time daemon task.

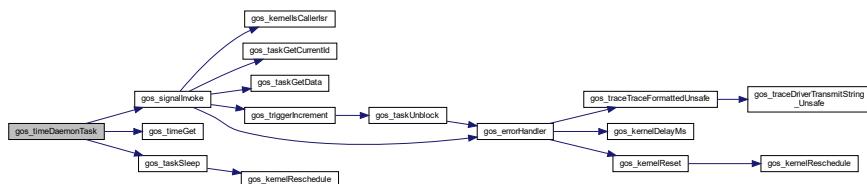
Increases the system time approximately every second and invokes the elapsed signals.

Returns

-

Definition at line 797 of file gos_time.c.

Here is the call graph for this function:

**4.35.3.9 gos_result_t gos_timeGet(gos_time_t * pTime)**

This function gets the system time.

This function copies the system time value to the given time variable.

Parameters

<i>pTime</i>	: Pointer to a time variable to store the system time value in.
--------------	---

Returns

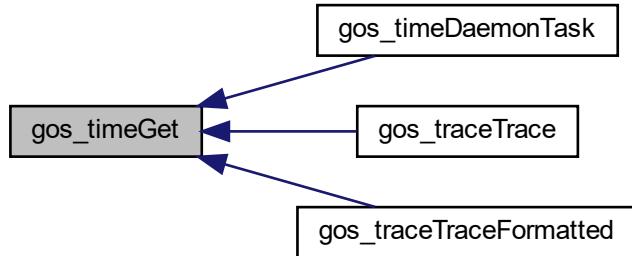
Result of time getting.

Return values

<i>GOS_SUCCESS</i>	: Time getting successful.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 186 of file gos_time.c.

Here is the caller graph for this function:



4.35.3.10 `gos_result_t gos_timeIncreaseSystemTime (u16_t milliseconds)`

This function increases the system time and runtime with the given value of milliseconds.

This function increases the system time and runtime with the given value of milliseconds.

Parameters

<i>milliseconds</i>	: Number of milliseconds to add to system time.
---------------------	---

Returns

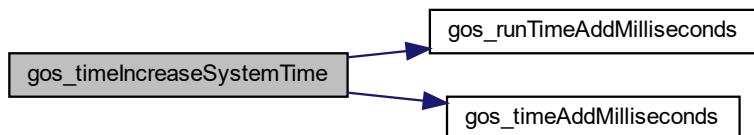
Result of system time increasing.

Return values

<i>GOS_SUCCESS</i>	: Increasing successful.
<i>GOS_ERROR</i>	: System time or runtime increasing failed.

Definition at line 767 of file gos_time.c.

Here is the call graph for this function:



4.35.3.11 gos_result_t gos_timeInit(void_t)

This function initializes the time service.

Creates the time signal and registers the time daemon in the kernel.

Returns

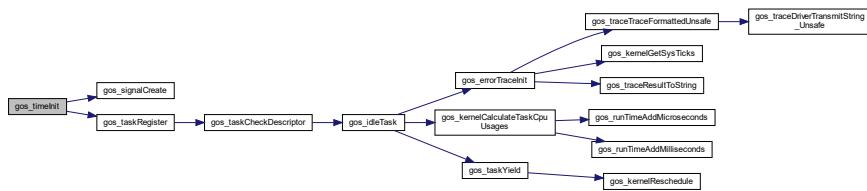
Result of initialization.

Return values

<i>GOS_SUCCESS</i>	: Initialization successful.
<i>GOS_ERROR</i>	: Time signal registration error or time daemon registration error.

Definition at line 159 of file gos_time.c.

Here is the call graph for this function:



4.35.3.12 gos_result_t gos_timeSet(gos_time_t * pTime)

This function sets the system time.

This function copies the time value from the given time variable to the system time variable.

Parameters

<i>pTime</i>	: Pointer to a time variable holding the desired time value.
--------------	--

Returns

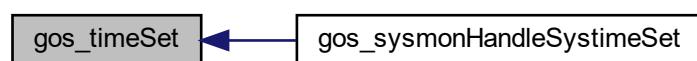
Result of time setting.

Return values

<i>GOS_SUCCESS</i>	: Time setting successful.
<i>GOS_ERROR</i>	: Time structure is NULL pointer.

Definition at line 213 of file gos_time.c.

Here is the caller graph for this function:



4.35.4 Variable Documentation

4.35.4.1 GOS_STATIC GOS_CONST gos_day_t dayLookupTable[GOS_TIME_NUMBER_OF_MONTHS]

Initial value:

```
=
{
    [GOS_TIME_JANUARY] = 31,
    [GOS_TIME_FEBRUARY] = 28,
    [GOS_TIME_MARCH] = 31,
    [GOS_TIME_APRIIL] = 30,
    [GOS_TIME_MAY] = 31,
    [GOS_TIME_JUNE] = 30,
    [GOS_TIME_JULY] = 31,
    [GOS_TIME_AUGUST] = 31,
    [GOS_TIME_SEPTMBER] = 30,
    [GOS_TIME_OCTOBER] = 31,
    [GOS_TIME_NOVEMBER] = 30,
    [GOS_TIME_DECEMBER] = 31
}
```

Number of days in each month - lookup table.

Definition at line 113 of file gos_time.c.

4.35.4.2 GOS_STATIC gos_runtime_t systemRunTime

System run-time.

Definition at line 108 of file gos_time.c.

4.35.4.3 GOS_STATIC gos_time_t systemTime

Initial value:

```
=
{
    .years = TIME_DEFAULT_YEAR,
    .months = TIME_DEFAULT_MONTH,
    .days = TIME_DEFAULT_DAY
}
```

System time.

Definition at line 98 of file gos_time.c.

4.35.4.4 GOS_STATIC gos_taskDescriptor_t timeDaemonTaskDesc

Initial value:

```
=
{
    .taskFunction = gos_timeDaemonTask,
    .taskName = "gos_time_daemon",
    .taskStackSize = CFG_TASK_TIME_DAEMON_STACK,
    .taskPriority = CFG_TASK_TIME_DAEMON_PRIO,
    .taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL
}
```

Time task descriptor.

Definition at line 147 of file gos_time.c.

4.35.4.5 GOS_STATIC gos_tid_t timeDaemonTaskId

Time task ID.

Definition at line 132 of file gos_time.c.

4.35.4.6 gos_signalId_t timeSignalId

Time signal ID. This signal is invoked when a second has elapsed.

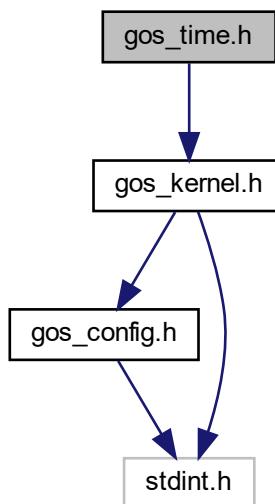
Definition at line 137 of file gos_time.c.

4.36 gos_time.h File Reference

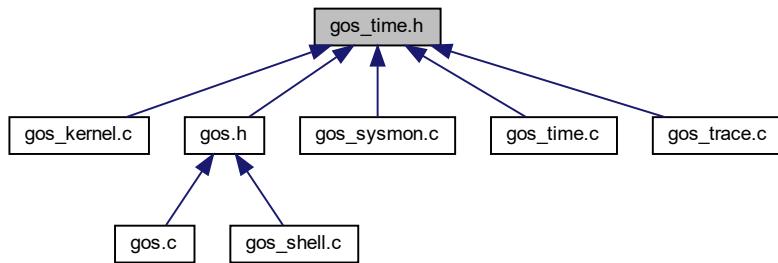
GOS time service header.

```
#include <gos_kernel.h>
```

Include dependency graph for gos_time.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum `gos_timeComprareResult_t` { `GOS_TIME_EARLIER`, `GOS_TIME_LATER`, `GOS_TIME_EQUAL` }
- enum `gos_timeMonthEnum_t` {
 `GOS_TIME_JANUARY` = 1, `GOS_TIME_FEBRUARY`, `GOS_TIME_MARCH`, `GOS_TIME_APRI`l,
 `GOS_TIME_MAY`, `GOS_TIME_JUNE`, `GOS_TIME_JULY`, `GOS_TIME_AUGUST`,
 `GOS_TIME_SEPTMBER`, `GOS_TIME_OCTOBER`, `GOS_TIME_NOVEMBER`, `GOS_TIME_DECEMBER`,
 `GOS_TIME_NUMBER_OF_MONTHS` = 12 }
- enum `gos_timeElapsedSenderId_t` {
 `GOS_TIME_SECOND_ELAPSED_SENDER_ID`, `GOS_TIME_MINUTE_ELAPSED_SENDER_ID`, `GOS_T`ime,
 `HOUR_ELAPSED_SENDER_ID`, `GOS_TIME_DAY_ELAPSED_SENDER_ID`,
 `GOS_TIME_MONTH_ELAPSED_SENDER_ID`, `GOS_TIME_YEAR_ELAPSED_SENDER_ID` }

Functions

- struct `__attribute__((packed))`
- `gos_result_t gos_timeInit (void_t)`

This function initializes the time service.
- `gos_result_t gos_timeGet (gos_time_t *pTime)`

This function gets the system time.
- `gos_result_t gos_timeSet (gos_time_t *pTime)`

This function sets the system time.
- `gos_result_t gos_timeCompare (gos_time_t *pTime1, gos_time_t *pTime2, gos_timeComprareResult_t *result)`

This function compares two time structures.
- `gos_result_t gos_timeIncreaseSystemTime (u16_t milliseconds)`

This function increases the system time and runtime with the given value of milliseconds.
- `gos_result_t gos_timeAddMilliseconds (gos_time_t *pTime, u16_t milliseconds)`

This function adds the given number of milliseconds to the given time structure.
- `gos_result_t gos_timeAddSeconds (gos_time_t *pTime, u16_t seconds)`

This function adds the given number of seconds to the given time variable.
- `gos_result_t gos_runTimeAddMicroseconds (gos_runtime_t *pRunTime1, gos_runtime_t *pRunTime2, u16_t microseconds)`

This function adds the given number of microseconds to the given time variables.
- `gos_result_t gos_runTimeAddMilliseconds (gos_runtime_t *pRunTime, u16_t milliseconds)`

This function adds the given number of milliseconds to the given time variables.
- `gos_result_t gos_runTimeAddSeconds (gos_runtime_t *pRunTime, u32_t seconds)`

This function adds the given number of seconds to the given run-time variable.

- [gos_result_t gos_runTimeGet \(gos_runtime_t *pRunTime\)](#)

This function gets the system run-time.

Variables

- [gos_time_t](#)

4.36.1 Detailed Description

GOS time service header.

Author

Ahmed Gazar

Date

2023-11-06

Version

1.6

Time service provides an easy interface to manipulate time structures, track the passage of time.

Definition in file [gos_time.h](#).

4.36.2 Enumeration Type Documentation

4.36.2.1 enum gos_timeCompareResult_t

Time comparison result enumerator.

Enumerator

GOS_TIME_EARLIER First time is earlier than second.

GOS_TIME_LATER First time is later than second.

GOS_TIME_EQUAL Times are equal.

Definition at line 90 of file gos_time.h.

4.36.2.2 enum gos_timeElapsedSenderId_t

Time elapsed sender IDs.

Enumerator

GOS_TIME_SECOND_ELAPSED_SENDER_ID Second elapsed sender ID.

GOS_TIME_MINUTE_ELAPSED_SENDER_ID Minute elapsed sender ID.

GOS_TIME_HOUR_ELAPSED_SENDER_ID Hour elapsed sender ID.

GOS_TIME_DAY_ELAPSED_SENDER_ID Day elapsed sender ID.

GOS_TIME_MONTH_ELAPSED_SENDER_ID Month elapsed sender ID.

GOS_TIME_YEAR_ELAPSED_SENDER_ID Year elapsed sender ID.

Definition at line 121 of file gos_time.h.

4.36.2.3 enum gos_timeMonthEnum_t

Month enumerator.

Enumerator

GOS_TIME_JANUARY January.

GOS_TIME_FEBRUARY February.

GOS_TIME_MARCH March.

GOS_TIME_APRIIL April.

GOS_TIME_MAY May.

GOS_TIME_JUNE June.

GOS_TIME_JULY July.

GOS_TIME_AUGUST August.

GOS_TIME_SEPTEMBER September.

GOS_TIME_OCTOBER October.

GOS_TIME_NOVEMBER November.

GOS_TIME_DECEMBER December.

GOS_TIME_NUMBER_OF_MONTHS Number of months in a year.

Definition at line 100 of file gos_time.h.

4.36.3 Function Documentation

4.36.3.1 struct __attribute__ ((packed))

Time type. < Milliseconds.

< Seconds.

< Minutes.

< Hours.

< Days.

< Months.

< Years.

Definition at line 76 of file gos_time.h.

4.36.3.2 gos_result_t gos_runTimeAddMicroseconds (gos_runtime_t * pRunTime1, gos_runtime_t * pRunTime2, u16_t microseconds)

This function adds the given number of microseconds to the given time variables.

This function adds the given number of microseconds to the given time variables.

Parameters

<i>pRunTime1</i>	: Pointer to the time variable.
<i>pRunTime2</i>	: Pointer to the time variable.
<i>microseconds</i>	: Number of microseconds to add.

Returns

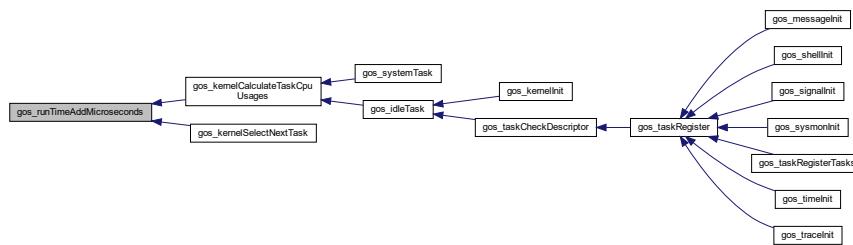
Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Microseconds added successfully.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 497 of file gos_time.c.

Here is the caller graph for this function:

4.36.3.3 *gos_result_t gos_runTimeAddMilliseconds (gos_runtime_t * pRunTime, u16_t milliseconds)*

This function adds the given number of milliseconds to the given time variables.

This function adds the given number of milliseconds to the given time variables.

Parameters

<i>pRunTime1</i>	: Pointer to the time variable.
<i>pRunTime2</i>	: Pointer to the time variable.
<i>milliseconds</i>	: Number of milliseconds to add.

Returns

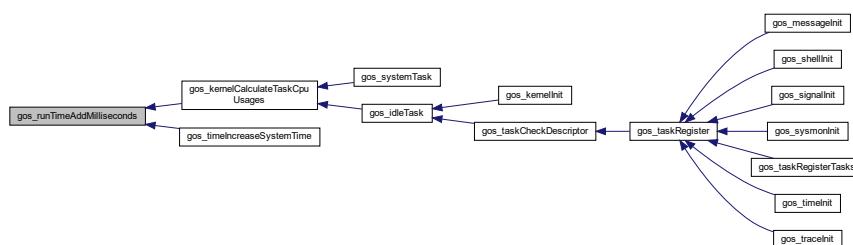
Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Milliseconds added successfully.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 637 of file gos_time.c.

Here is the caller graph for this function:



4.36.3.4 gos_result_t gos_runTimeAddSeconds (gos_runtime_t * pRunTime, u32_t seconds)

This function adds the given number of seconds to the given run-time variable.

This function adds the given number of seconds to the given run-time variable.

Parameters

<i>pRunTime</i>	: Pointer to a run-time variable.
<i>seconds</i>	: Number of seconds to add.

Returns

Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Seconds added successfully.
<i>GOS_ERROR</i>	: Run-time variable is NULL pointer.

Definition at line 706 of file gos_time.c.

4.36.3.5 gos_result_t gos_runTimeGet (gos_runtime_t * pRunTime)

This function gets the system run-time.

This function gets the system run-time.

Parameters

<i>pRunTime</i>	: Pointer to a run-time variable to store the system run-time in.
-----------------	---

Returns

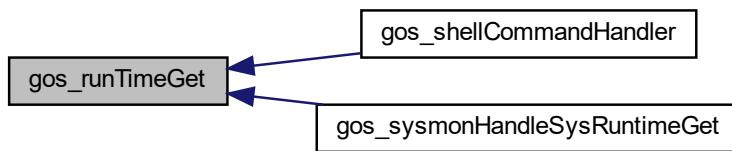
Result of run-time getting.

Return values

<i>GOS_SUCCESS</i>	: Run-time getting is successful.
<i>GOS_ERROR</i>	: Run-time variable is NULL pointer.

Definition at line 240 of file gos_time.c.

Here is the caller graph for this function:



4.36.3.6 gos_result_t gos_timeAddMilliseconds (gos_time_t * pTime, u16_t milliseconds)

This function adds the given number of milliseconds to the given time structure.

This function adds the given number of milliseconds to the given time structure.

Parameters

<i>pTime</i>	: Pointer to the time structure.
<i>milliseconds</i>	: Number of milliseconds to add.

Returns

Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Increasing successful.
<i>GOS_ERROR</i>	: Time structure is NULL pointer.

Definition at line 311 of file gos_time.c.

Here is the caller graph for this function:

**4.36.3.7 gos_result_t gos_timeAddSeconds(gos_time_t * pTime, u16_t seconds)**

This function adds the given number of seconds to the given time variable.

This function adds the given number of seconds to the given time variable.

Parameters

<i>pTime</i>	: Pointer to the time variable.
<i>seconds</i>	: Number of seconds to add.

Returns

Result of time increasing.

Return values

<i>GOS_SUCCESS</i>	: Seconds added successfully.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 407 of file gos_time.c.

4.36.3.8 gos_result_t gos_timeCompare(gos_time_t * pTime1, gos_time_t * pTime2, gos_timeCompareResult_t * result)

This function compares two time structures.

This function compares two time structures.

Parameters

<i>pTime1</i>	: Pointer to the first time variable.
<i>pTime2</i>	: Pointer to the second time variable.
<i>result</i>	: Pointer to the comparison result variable.

Returns

Result of time comparison.

Return values

<i>GOS_SUCCESS</i>	: Time comparison successful.
<i>GOS_ERROR</i>	: Either time structure and/or result variable is NULL pointer.

Definition at line 267 of file gos_time.c.

4.36.3.9 gos_result_t gos_timeGet (gos_time_t * *pTime*)

This function gets the system time.

This function copies the system time value to the given time variable.

Parameters

<i>pTime</i>	: Pointer to a time variable to store the system time value in.
--------------	---

Returns

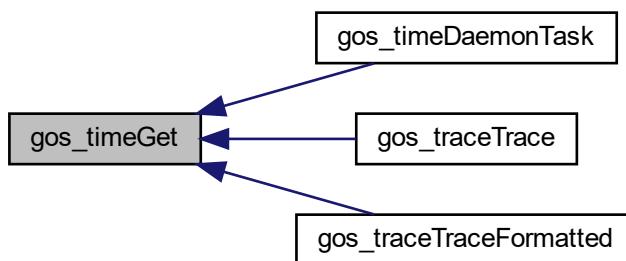
Result of time getting.

Return values

<i>GOS_SUCCESS</i>	: Time getting successful.
<i>GOS_ERROR</i>	: Time variable is NULL pointer.

Definition at line 186 of file gos_time.c.

Here is the caller graph for this function:

**4.36.3.10 gos_result_t gos_timeIncreaseSystemTime (u16_t *milliseconds*)**

This function increases the system time and runtime with the given value of milliseconds.

This function increases the system time and runtime with the given value of milliseconds.

Parameters

<i>milliseconds</i>	: Number of milliseconds to add to system time.
---------------------	---

Returns

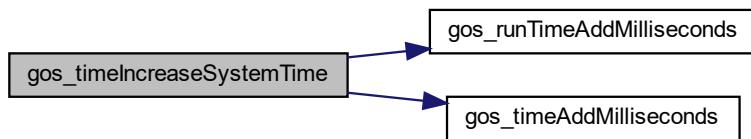
Result of system time increasing.

Return values

<i>GOS_SUCCESS</i>	: Increasing successful.
<i>GOS_ERROR</i>	: System time or runtime increasing failed.

Definition at line 767 of file gos_time.c.

Here is the call graph for this function:

**4.36.3.11 gos_result_t gos_timeInit(void_t)**

This function initializes the time service.

Creates the time signal and registers the time daemon in the kernel.

Returns

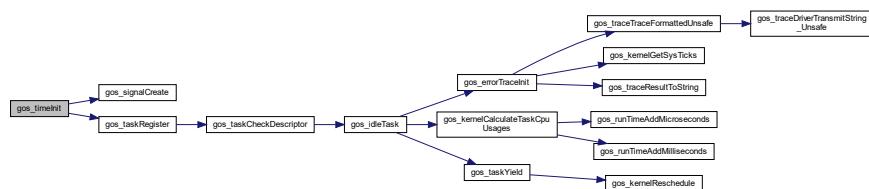
Result of initialization.

Return values

<i>GOS_SUCCESS</i>	: Initialization successful.
<i>GOS_ERROR</i>	: Time signal registration error or time daemon registration error.

Definition at line 159 of file gos_time.c.

Here is the call graph for this function:



4.36.3.12 gos_result_t gos_timeSet(gos_time_t * pTime)

This function sets the system time.

This function copies the time value from the given time variable to the system time variable.

Parameters

<i>pTime</i>	: Pointer to a time variable holding the desired time value.
--------------	--

Returns

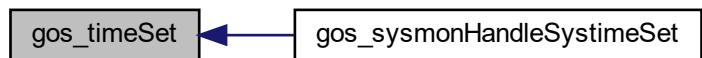
Result of time setting.

Return values

<i>GOS_SUCCESS</i>	: Time setting successful.
<i>GOS_ERROR</i>	: Time structure is NULL pointer.

Definition at line 213 of file gos_time.c.

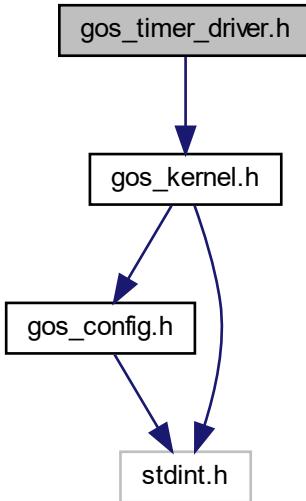
Here is the caller graph for this function:



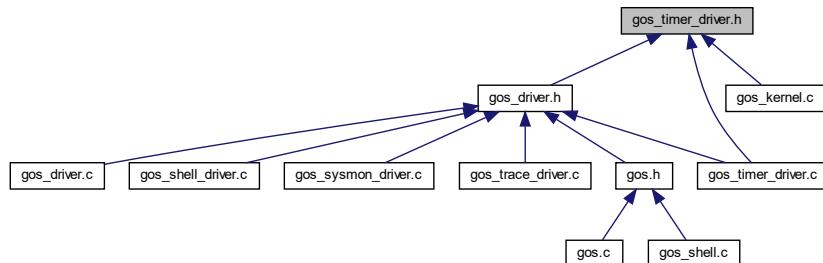
4.37 gos_timer_driver.h File Reference

GOS timer driver header.

```
#include "gos_kernel.h"
Include dependency graph for gos_timer_driver.h:
```



This graph shows which files directly or indirectly include this file:



Typedefs

- `typedef gos_result_t(* gos_timerDriverSysTimerGetVal_t)(u16_t *)`

Functions

- `gos_result_t gos_timerDriverSysTimerGet (u16_t *pValue)`
System timer value getter skeleton.

4.37.1 Detailed Description

GOS timer driver header.

Author

Ahmed Gazar

Date

2022-12-09

Version

1.0

This is the timer driver skeleton. It contains the required interface functions for the OS. These functions shall be implemented by the user in order to be able to use microsecond delay and task run-time monitoring.

Definition in file [gos_timer_driver.h](#).

4.37.2 Typedef Documentation

4.37.2.1 `typedef gos_result_t(* gos_timerDriverSysTimerGetVal_t)(u16_t *)`

System timer value get function type.

Definition at line 63 of file [gos_timer_driver.h](#).

4.37.3 Function Documentation

4.37.3.1 `gos_result_t gos_timerDriverSysTimerGet(u16_t * pValue)`

System timer value getter skeleton.

If registered, it calls the custom system timer getter function.

Parameters

<code>pValue</code>	: Pointer to the variable to store the timer value in.
---------------------	--

Returns

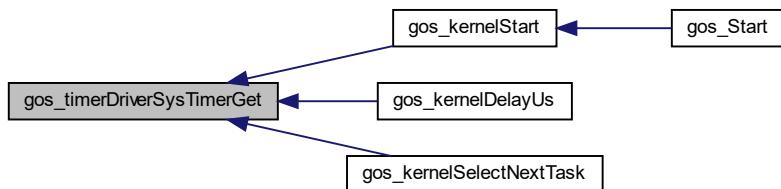
Result of system timer value getting.

Return values

<code>GOS_SUCCESS</code>	: According to user implementation.
<code>GOS_ERROR</code>	: According to user implementation / function not registered.

Definition at line 64 of file [gos_timer_driver.c](#).

Here is the caller graph for this function:

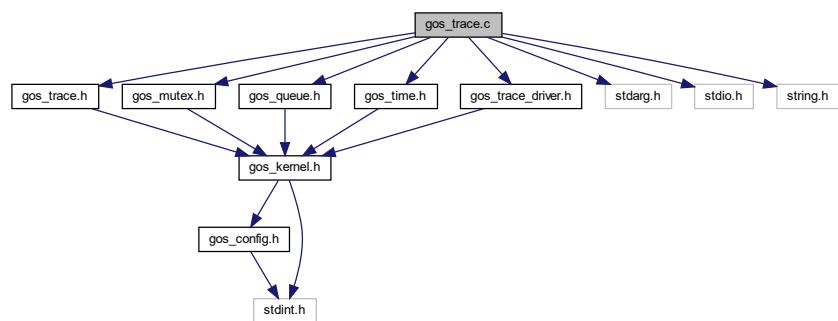


4.38 gos_trace.c File Reference

GOS trace service source.

```
#include <gos_trace.h>
#include <gos_mutex.h>
#include <gos_queue.h>
#include <gos_time.h>
#include <gos_trace_driver.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for gos_trace.c:



Macros

- #define GOS_TRACE_TIMESTAMP_FORMAT "[TRACE_FG_YELLOW_START"%04d-%02d-%02d%02d:%02d:%02d.%03d"TRACE_FORMAT_RESET"]t"
- #define GOS_TRACE_TIMESTAMP_LENGTH (44u)
- #define GOS_TRACE_QUEUE_TMO_MS (2000u)
- #define GOS_TRACE_MUTEX_TMO_MS (2000u)

Functions

- **GOS_STATIC void_t gos_traceDaemonTask (void_t)**
Trace daemon task.
- **gos_result_t gos_tracelInit (void_t)**
Initializes the trace service.
- **GOS_INLINE gos_result_t gos_traceTrace (bool_t addTimeStamp, char_t *traceMessage)**
Traces a given message.
- **gos_result_t gos_traceTraceFormatted (bool_t addTimeStamp, GOS_CONST char_t *traceFormat,...)**
Traces a given formatted message.
- **gos_result_t gos_traceTraceFormattedUnsafe (GOS_CONST char_t *traceFormat,...)**
Traces a given formatted message.

Variables

- **GOS_STATIC gos_queueDescriptor_t traceQueue**
- **GOS_STATIC char_t traceLine [CFG_TRACE_MAX_LENGTH]**

- `GOS_STATIC char_t formattedBuffer [CFG_TRACE_MAX_LENGTH]`
- `GOS_STATIC char_t timeStampBuffer [GOS_TRACE_TIMESTAMP_LENGTH]`
- `GOS_STATIC gos_mutex_t traceMutex`
- `GOS_STATIC gos_taskDescriptor_t traceDaemonTaskDesc`

4.38.1 Detailed Description

GOS trace service source.

Author

Ahmed Gazar

Date

2024-03-08

Version

1.12

For a more detailed description of this service, please refer to [gos_trace.h](#)

Definition in file [gos_trace.c](#).

4.38.2 Macro Definition Documentation

4.38.2.1 `#define GOS_TRACE_MUTEX_TMO_MS (2000u)`

Timeout value in [ms] for mutex operations.

Definition at line 95 of file [gos_trace.c](#).

4.38.2.2 `#define GOS_TRACE_QUEUE_TMO_MS (2000u)`

Timeout value in [ms] for queue operations.

Definition at line 90 of file [gos_trace.c](#).

4.38.2.3 `#define GOS_TRACE_TIMESTAMP_FORMAT "["TRACE_FG_YELLOW_START"%04d-%02d-%02d%02d:%02d:%02d.%03d"TRACE_FORMAT_RESET"]\n"`

Trace timestamp formatter.

Definition at line 80 of file [gos_trace.c](#).

4.38.2.4 `#define GOS_TRACE_TIMESTAMP_LENGTH (44u)`

Trace timestamp length.

Definition at line 85 of file [gos_trace.c](#).

4.38.3 Function Documentation

4.38.3.1 GOS_STATIC void_t gos_traceDaemonTask (void_t)

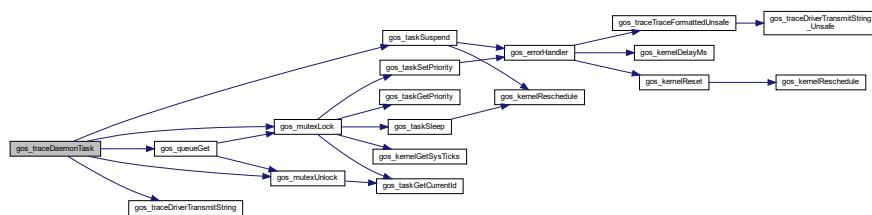
Trace daemon task.

Polls the trace queue and transmits the elements in the trace queue via the registered trace driver.

Returns

Definition at line 410 of file gos_trace.c.

Here is the call graph for this function:



4.38.3.2 gos_result_t gos_tracelnit (void_t)

Initializes the trace service.

Creates a trace queue and registers the trace daemon in the kernel.

Returns

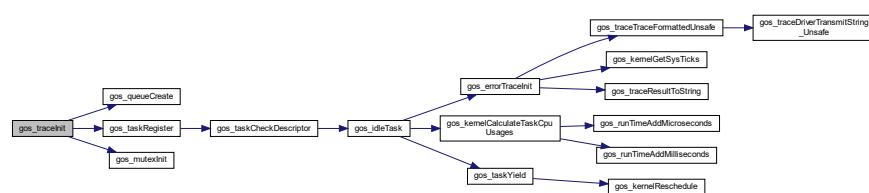
Result of initialization.

Return values

<i>GOS_SUCCESS</i>	: Initialization successful.
<i>GOS_ERROR</i>	: Queue creation or task registration error.

Definition at line 150 of file gos_trace.c.

Here is the call graph for this function:



4.38.3.3 GOS_INLINE gos_result_t gos_traceTrace (bool_t addTimeStamp, char_t * traceMessage)

Traces a given message.

Places the given message to the trace queue (for the trace daemon to print it).

Parameters

<i>addTimeStamp</i>	: Flag to indicate whether to add time-stamp or not.
<i>traceMessage</i>	: String to trace.

Returns

Result of tracing.

Return values

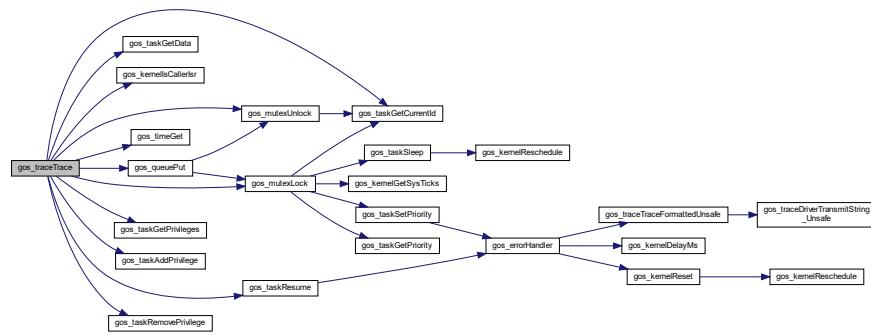
<i>GOS_SUCCESS</i>	: Tracing successful.
<i>GOS_ERROR</i>	: Queue put error.

Remarks

This function uses the queue service.

Definition at line 175 of file gos_trace.c.

Here is the call graph for this function:



4.38.3.4 `gos_result_t gos_traceTraceFormatted (bool_t addTimeStamp, GOS_CONST char_t * traceFormat, ...)`

Traces a given formatted message.

Prints the formatted message into a local buffer and places it to the trace queue (for the trace daemon to print it).

Parameters

<i>addTimeStamp</i>	: Flag to indicate whether to add time-stamp or not.
<i>traceFormat</i>	: Formatter string.
...	: Optional parameters.

Returns

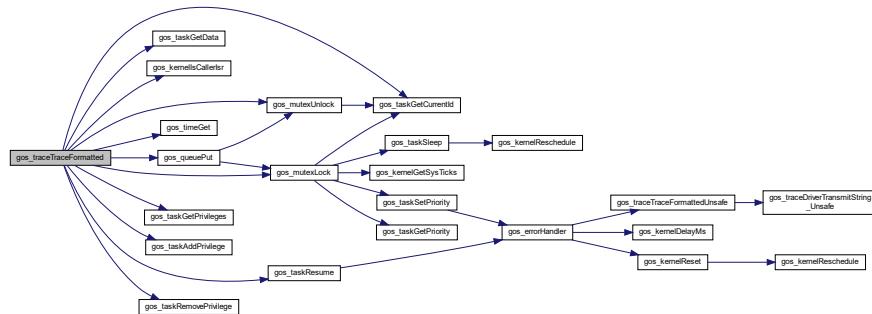
Result of formatted tracing.

Return values

<code>GOS_SUCCESS</code>	: Formatted tracing successful.
<code>GOS_ERROR</code>	: Queue put error.

Definition at line 273 of file gos_trace.c.

Here is the call graph for this function:



4.38.3.5 `gos_result_t gos_traceTraceFormattedUnsafe (GOS_CONST char_t * traceFormat, ...)`

Traces a given formatted message.

Prints the formatted message into a local buffer and transmits it via the trace port.

Parameters

<code>traceFormat</code>	: Formatter string.
...	: Optional parameters.

Returns

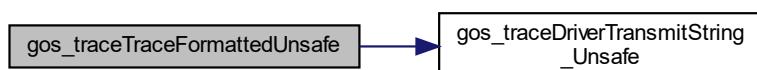
Result of formatted tracing.

Return values

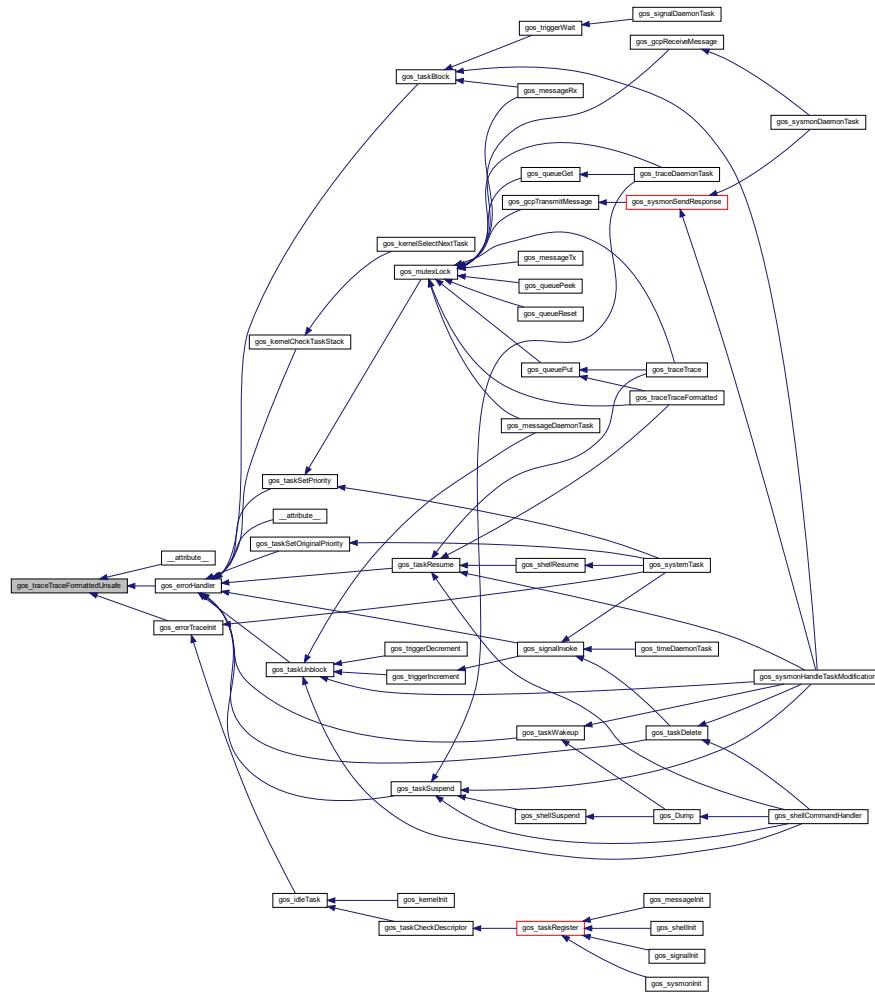
<code>GOS_SUCCESS</code>	: Message traced successfully.
<code>GOS_ERROR</code>	: Transmit error.

Definition at line 372 of file gos_trace.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.38.4 Variable Documentation

4.38.4.1 GOS_STATIC char_t formattedBuffer[CFG_TRACE_MAX_LENGTH]

Trace formatted buffer for message formatting.

Definition at line 118 of file gos_trace.c.

4.38.4.2 GOS_STATIC char_t timeStampBuffer[GOS_TRACE_TIMESTAMP_LENGTH]

Buffer for timestamp printing.

Definition at line 123 of file gos_trace.c.

4.38.4.3 GOS_STATIC gos_taskDescriptor_t traceDaemonTaskDesc

Initial value:

```
=
{
    .taskFunction      = gos_traceDaemonTask,
```

```
.taskName      = "gos_trace_daemon",
.taskPriority   = CFG_TASK_TRACE_DAEMON_PRIO,
.taskStackSize  = CFG_TASK_TRACE_DAEMON_STACK,
.taskPrivilegeLevel = GOS_TASK_PRIVILEGE_KERNEL
}
```

Trace daemon task descriptor.

Definition at line 138 of file gos_trace.c.

4.38.4.4 GOS_STATIC char_t traceLine[CFG_TRACE_MAX_LENGTH]

Trace line buffer.

Definition at line 113 of file gos_trace.c.

4.38.4.5 GOS_STATIC gos_mutex_t traceMutex

Trace mutex.

Definition at line 128 of file gos_trace.c.

4.38.4.6 GOS_STATIC gos_queueDescriptor_t traceQueue

Initial value:

```
=  
{
```

```
}
```

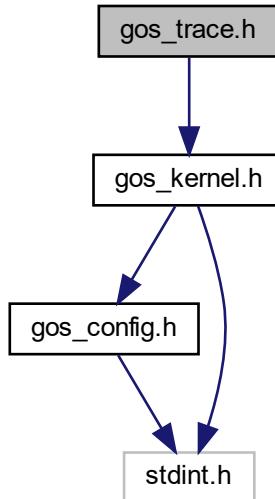
Trace queue.

Definition at line 103 of file gos_trace.c.

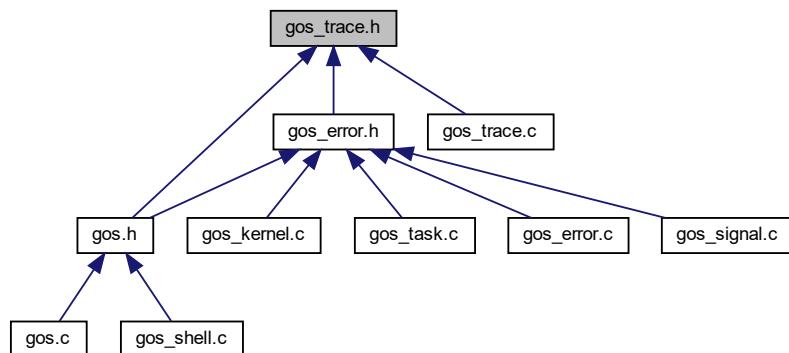
4.39 gos_trace.h File Reference

GOS trace service header.

```
#include <gos_kernel.h>
Include dependency graph for gos_trace.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- `#define TRACE_FORMAT_RESET "\x1B[0m"`
Reset formatting.
- `#define TRACE_FG_RED_START "\x1B[31m"`
Red foreground start.
- `#define TRACE_FG_GREEN_START "\x1B[32m"`
Green foreground start.
- `#define TRACE_FG_YELLOW_START "\x1B[33m"`

- `#define TRACE_FG_BLUE_START "\x1B[34m"`
Yellow foreground start.
- `#define TRACE_FG_MAGENTA_START "\x1B[35m"`
Blue foreground start.
- `#define TRACE_FG_CYAN_START "\x1B[36m"`
Magenta foreground start.
- `#define TRACE_FG_WHITE_START "\x1B[37m"`
Cyan foreground start.
- `#define TRACE_BG_RED_START "\x1B[41m"`
White foreground start.
- `#define TRACE_BG_GREEN_START "\x1B[42m"`
Red background start.
- `#define TRACE_BG_YELLOW_START "\x1B[43m"`
Green background start.
- `#define TRACE_BG_BLUE_START "\x1B[44m"`
Yellow background start.
- `#define TRACE_BG_MAGENTA_START "\x1B[45m"`
Blue background start.
- `#define TRACE_BG_CYAN_START "\x1B[46m"`
Magenta background start.
- `#define TRACE_BG_WHITE_START "\x1B[47m"`
Cyan background start.
- `#define TRACE_BOLD_START "\x1B[1m"`
White background start.
- `#define TRACE_ITALIC_START "\x1B[3m"`
Bold start.
- `#define TRACE_UNDERLINE_START "\x1B[4m"`
Italic start.
- `#define TRACE_STRIKETHROUGH_START "\x1B[9m"`
Underline start.
- `#define TRACE_STRIKETHROUGH_UNSAFE "\x1B[21m"`
Strikethrough start.

Functions

- `gos_result_t gos_tracelInit (void_t)`
Initializes the trace service.
- `gos_result_t gos_traceTrace (bool_t addTimeStamp, char_t *traceMessage)`
Traces a given message.
- `gos_result_t gos_traceTraceFormatted (bool_t addTimeStamp, GOS_CONST char_t *traceFormat,...)`
Traces a given formatted message.
- `gos_result_t gos_traceTraceFormattedUnsafe (GOS_CONST char_t *traceFormat,...)`
Traces a given formatted message.

4.39.1 Detailed Description

GOS trace service header.

Author

Ahmed Gazar

Date

2023-01-13

Version

2.1

Trace service is a simple interface to send out strings via the configured trace periphery.

Definition in file [gos_trace.h](#).

4.39.2 Function Documentation

4.39.2.1 gos_result_t gos_tracelinit (void_t)

Initializes the trace service.

Creates a trace queue and registers the trace daemon in the kernel.

Returns

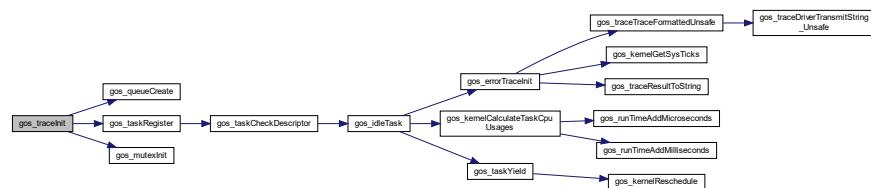
Result of initialization.

Return values

<i>GOS_SUCCESS</i>	: Initialization successful.
<i>GOS_ERROR</i>	: Queue creation or task registration error.

Definition at line 150 of file gos_trace.c.

Here is the call graph for this function:



4.39.2.2 gos_result_t gos_traceTrace (bool_t addTimeStamp, char_t * traceMessage)

Traces a given message.

Places the given message to the trace queue (for the trace daemon to print it).

Parameters

<i>addTimeStamp</i>	: Flag to indicate whether to add time-stamp or not.
<i>traceMessage</i>	: String to trace.

Returns

Result of tracing.

Return values

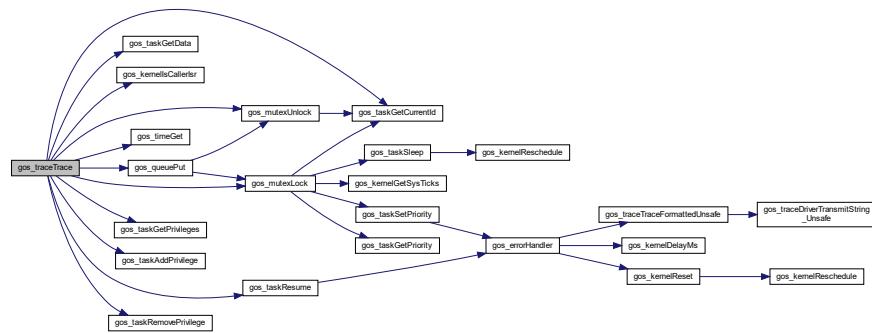
<i>GOS_SUCCESS</i>	: Tracing successful.
<i>GOS_ERROR</i>	: Queue put error.

Remarks

This function uses the queue service.

Definition at line 175 of file gos_trace.c.

Here is the call graph for this function:



4.39.2.3 `gos_result_t gos_traceTraceFormatted (bool_t addTimeStamp, GOS_CONST char_t * traceFormat, ...)`

Traces a given formatted message.

Prints the formatted message into a local buffer and places it to the trace queue (for the trace daemon to print it).

Parameters

<i>addTimeStamp</i>	: Flag to indicate whether to add time-stamp or not.
<i>traceFormat</i>	: Formatter string.
...	: Optional parameters.

Returns

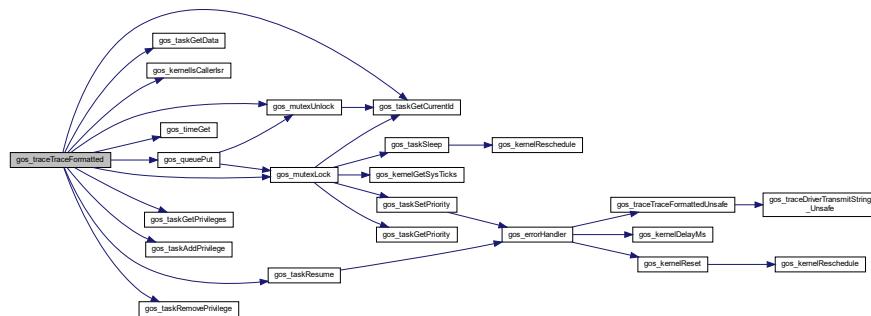
Result of formatted tracing.

Return values

<i>GOS_SUCCESS</i>	: Formatted tracing successful.
<i>GOS_ERROR</i>	: Queue put error.

Definition at line 273 of file gos_trace.c.

Here is the call graph for this function:



4.39.2.4 `gos_result_t gos_traceTraceFormattedUnsafe (GOS_CONST char_t * traceFormat, ...)`

Traces a given formatted message.

Prints the formatted message into a local buffer and transmits it via the trace port.

Parameters

<i>traceFormat</i>	: Formatter string.
...	: Optional parameters.

Returns

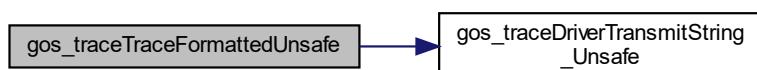
Result of formatted tracing.

Return values

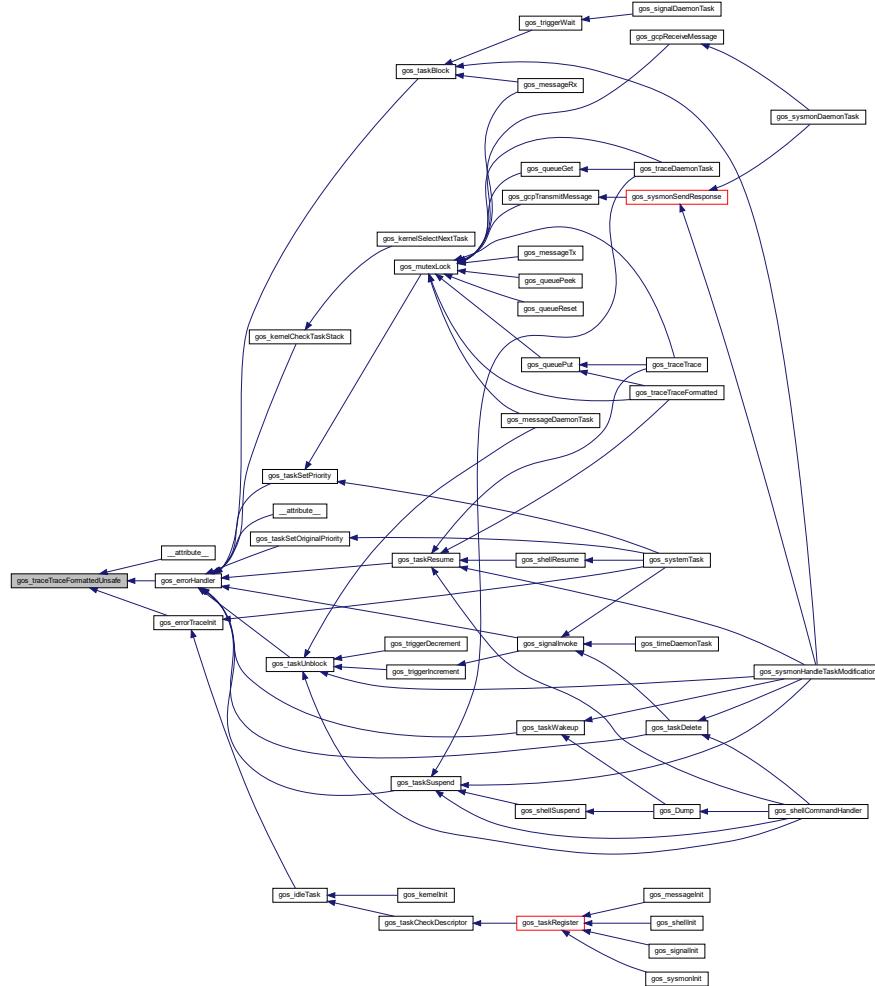
<i>GOS_SUCCESS</i>	: Message traced successfully.
<i>GOS_ERROR</i>	: Transmit error.

Definition at line 372 of file gos_trace.c.

Here is the call graph for this function:



Here is the caller graph for this function:

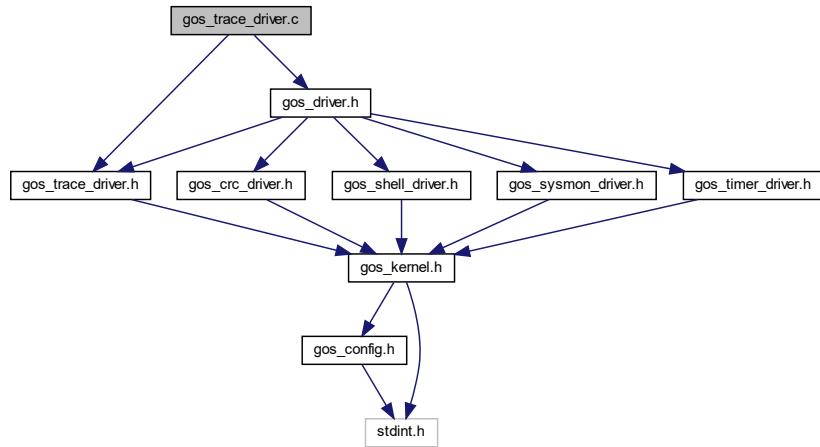


4.40 gos_trace_driver.c File Reference

GOS trace driver source.

```
#include <gos_trace_driver.h>
#include <gos_driver.h>
```

Include dependency graph for gos_trace_driver.c:



Functions

- `gos_result_t gos_traceDriverTransmitString (char_t *pString)`
Trace driver transmit string function skeleton.
- `gos_result_t gos_traceDriverTransmitString_Unsafe (char_t *pString)`
Trace driver unsafe transmit string function skeleton.

Variables

- `GOS_EXTERN gos_driver_functions_t driverFunctions`

4.40.1 Detailed Description

GOS trace driver source.

Author

Ahmed Gazar

Date

2023-01-13

Version

1.2

For a more detailed description of this driver, please refer to [gos_trace_driver.h](#)

Definition in file [gos_trace_driver.c](#).

4.40.2 Function Documentation

4.40.2.1 gos_result_t gos_traceDriverTransmitString (`char_t * pString`)

Trace driver transmit string function skeleton.

If registered, it calls the custom transmit function.

Parameters

<i>pString</i>	: String to trace.
----------------	--------------------

Returns

Result of string transmission.

Return values

<i>GOS_SUCCESS</i>	: According to user implementation.
<i>GOS_ERROR</i>	: According to user implementation / function not registered.

Definition at line 65 of file gos_trace_driver.c.

Here is the caller graph for this function:



4.40.2.2 `gos_result_t gos_traceDriverTransmitString_Unsafe(char_t * pString)`

Trace driver unsafe transmit string function skeleton.

If registered, it calls the custom unsafe transmit function.

Parameters

<i>pString</i>	: String to trace.
----------------	--------------------

Returns

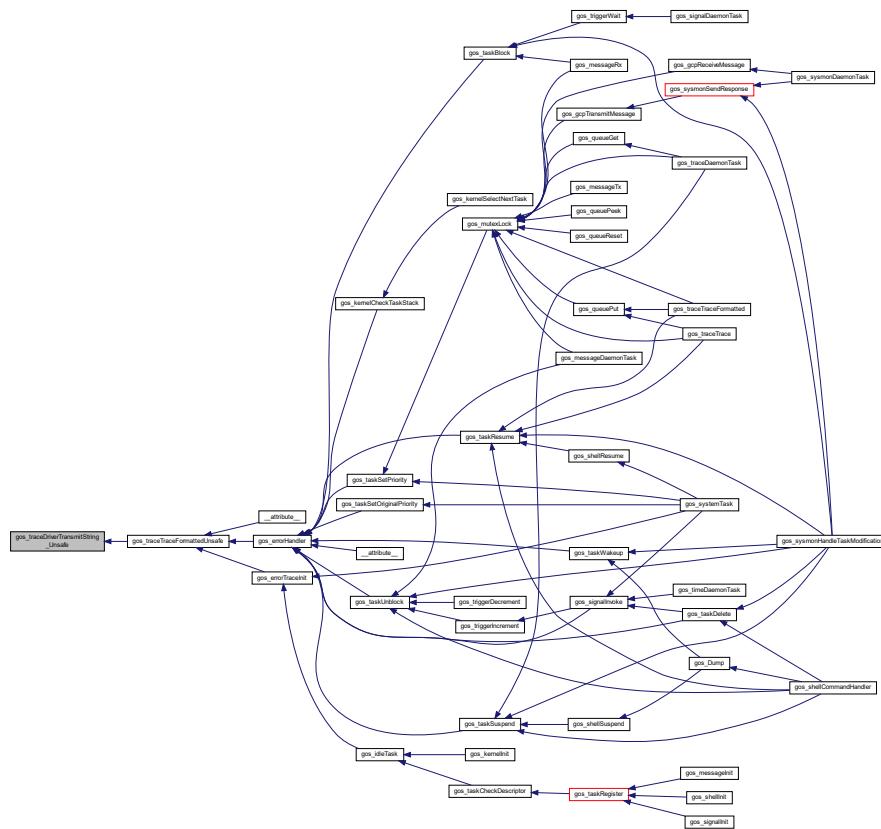
Result of string transmission.

Return values

<i>GOS_SUCCESS</i>	: According to user implementation.
<i>GOS_ERROR</i>	: According to user implementation / function not registered.

Definition at line 90 of file gos_trace_driver.c.

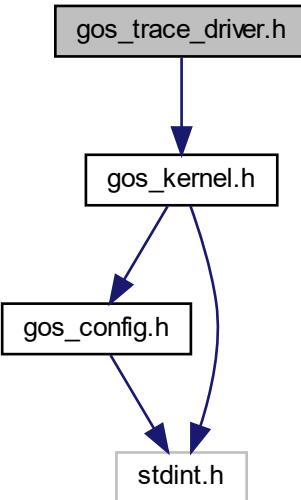
Here is the caller graph for this function:



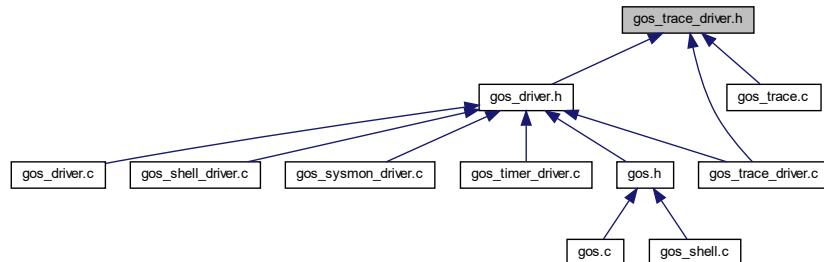
4.41 gos_trace_driver.h File Reference

GOS trace driver header.

```
#include "gos_kernel.h"
Include dependency graph for gos_trace_driver.h:
```



This graph shows which files directly or indirectly include this file:



Typedefs

- `typedef gos_result_t(* gos_traceDriverTransmitString_t)(char_t *)`
- `typedef gos_result_t(* gos_traceDriverTransmitString_Unsafe_t)(char_t *)`

Functions

- `gos_result_t gos_traceDriverTransmitString (char_t *pString)`
Trace driver transmit string function skeleton.
- `gos_result_t gos_traceDriverTransmitString_Unsafe (char_t *pString)`
Trace driver unsafe transmit string function skeleton.

4.41.1 Detailed Description

GOS trace driver header.

Author

Ahmed Gazar

Date

2023-01-13

Version

1.2

This driver provides a skeleton for the driver for the trace service.

Definition in file [gos_trace_driver.h](#).

4.41.2 Typedef Documentation

4.41.2.1 `typedef gos_result_t(* gos_traceDriverTransmitString_t)(char_t *)`

Trace driver transmit string function type.

Definition at line 63 of file gos_trace_driver.h.

4.41.2.2 `typedef gos_result_t(* gos_traceDriverTransmitString_Unsafe_t)(char_t *)`

Trace driver unsafe transmit string function type.

Definition at line 68 of file gos_trace_driver.h.

4.41.3 Function Documentation

4.41.3.1 `gos_result_t gos_traceDriverTransmitString(char_t * pString)`

Trace driver transmit string function skeleton.

If registered, it calls the custom transmit function.

Parameters

<code>pString</code>	: String to trace.
----------------------	--------------------

Returns

Result of string transmission.

Return values

<code>GOS_SUCCESS</code>	: According to user implementation.
<code>GOS_ERROR</code>	: According to user implementation / function not registered.

Definition at line 65 of file gos_trace_driver.c.

Here is the caller graph for this function:



4.41.3.2 `gos_result_t gos_traceDriverTransmitString_Unsafe(char_t * pString)`

Trace driver unsafe transmit string function skeleton.

If registered, it calls the custom unsafe transmit function.

Parameters

<code>pString</code>	: String to trace.
----------------------	--------------------

Returns

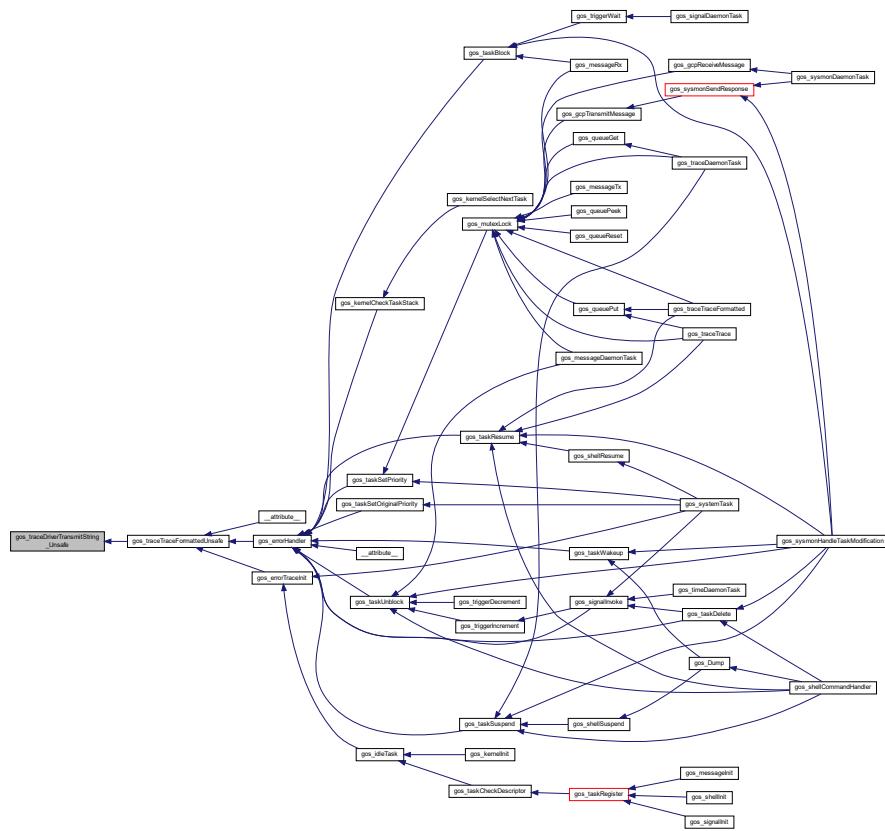
Result of string transmission.

Return values

<code>GOS_SUCCESS</code>	: According to user implementation.
<code>GOS_ERROR</code>	: According to user implementation / function not registered.

Definition at line 90 of file `gos_trace_driver.c`.

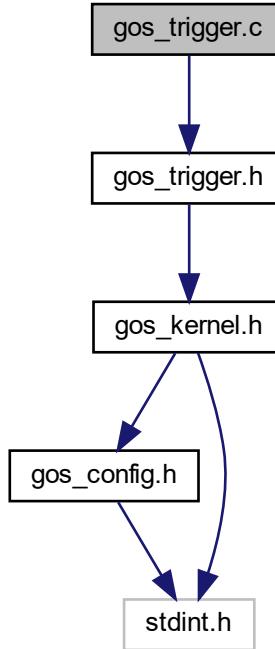
Here is the caller graph for this function:



4.42 gos_trigger.c File Reference

GOS trigger service source.

```
#include <gos_trigger.h>
Include dependency graph for gos_trigger.c:
```



Functions

- [`gos_result_t gos_triggerInit \(gos_trigger_t *pTrigger\)`](#)
Initializes the trigger instance.
- [`GOS_INLINE gos_result_t gos_triggerReset \(gos_trigger_t *pTrigger\)`](#)
Resets the trigger counter of the given trigger instance.
- [`GOS_INLINE gos_result_t gos_triggerWait \(gos_trigger_t *pTrigger, u32_t value, u32_t timeout\)`](#)
Waits for the trigger instance to reach the given trigger value.
- [`GOS_INLINE gos_result_t gos_triggerIncrement \(gos_trigger_t *pTrigger\)`](#)
Increments the trigger value of the given trigger.
- [`GOS_INLINE gos_result_t gos_triggerDecrement \(gos_trigger_t *pTrigger\)`](#)
Decrements the trigger value of the given trigger.

4.42.1 Detailed Description

GOS trigger service source.

Author

Ahmed Gazar

Date

2023-11-15

Version

2.9

For a more detailed description of this service, please refer to [gos_trigger.h](#)

Definition in file [gos_trigger.c](#).

4.42.2 Function Documentation

4.42.2.1 GOS_INLINE gos_result_t gos_triggerDecrement (*gos_trigger_t * pTrigger*)

Decrements the trigger value of the given trigger.

Decrements the trigger value of the given trigger.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
-----------------	------------------------------------

Returns

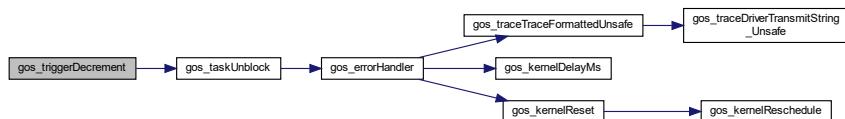
Result of trigger decrementing.

Return values

<i>GOS_SUCCESS</i>	: Decrementing successful.
<i>GOS_ERROR</i>	: Trigger is NULL pointer or value is already zero.

Definition at line 244 of file [gos_trigger.c](#).

Here is the call graph for this function:



4.42.2.2 GOS_INLINE gos_result_t gos_triggerIncrement (*gos_trigger_t * pTrigger*)

Increments the trigger value of the given trigger.

Increments the trigger value of the given trigger.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
-----------------	------------------------------------

Returns

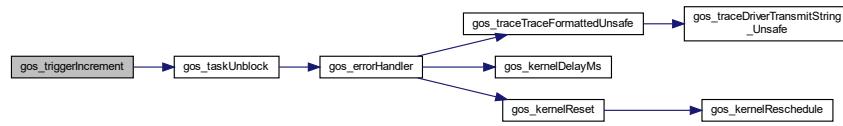
Result of trigger incrementing.

Return values

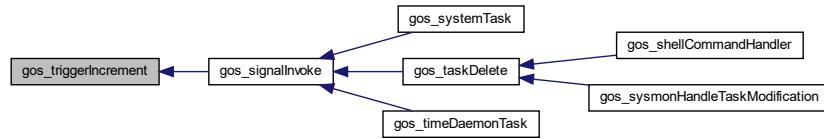
<i>GOS_SUCCESS</i>	: Incrementing successful.
<i>GOS_ERROR</i>	: Trigger is NULL pointer.

Definition at line 197 of file gos_trigger.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.42.2.3 `gos_result_t gos_triggerInit(gos_trigger_t * pTrigger)`

Initializes the trigger instance.

Calls the initializer for the trigger mutex.

Parameters

<i>pTrigger</i>	: Pointer to the trigger to be initialized.
-----------------	---

Returns

Result of trigger initializing.

Return values

<i>GOS_SUCCESS</i>	: Trigger initialized successfully.
<i>GOS_ERROR</i>	: Trigger descriptor or trigger mutex is NULL pointer.

Definition at line 76 of file gos_trigger.c.

Here is the caller graph for this function:

**4.42.2.4 GOS_INLINE gos_result_t gos_triggerReset(gos_trigger_t * pTrigger)**

Resets the trigger counter of the given trigger instance.

Sets the trigger counter of the given trigger instance to zero.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
-----------------	------------------------------------

Returns

Result of trigger resetting.

Return values

<i>GOS_SUCCESS</i>	: Trigger resetting successful.
<i>GOS_ERROR</i>	: Trigger is NULL pointer.

Definition at line 105 of file gos_trigger.c.

Here is the caller graph for this function:

**4.42.2.5 GOS_INLINE gos_result_t gos_triggerWait(gos_trigger_t * pTrigger, u32_t value, u32_t timeout)**

Waits for the trigger instance to reach the given trigger value.

Increases the trigger waiter number, and waits in a non-blocking way until the desired trigger value is reached or the timeout is reached.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
<i>value</i>	: The desired trigger value.
<i>timeout</i>	: Timeout value.

Returns

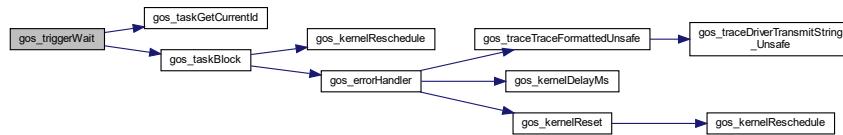
Result of trigger waiting.

Return values

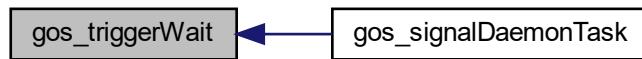
<i>GOS_SUCCESS</i>	: Trigger value reached.
<i>GOS_ERROR</i>	: Trigger value was not reached within the timeout value.

Definition at line 138 of file gos_trigger.c.

Here is the call graph for this function:



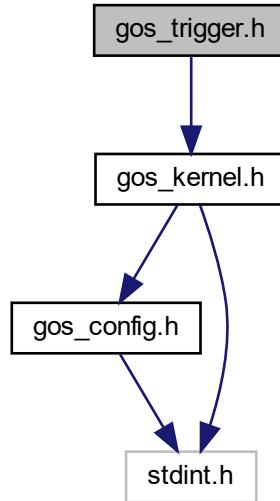
Here is the caller graph for this function:



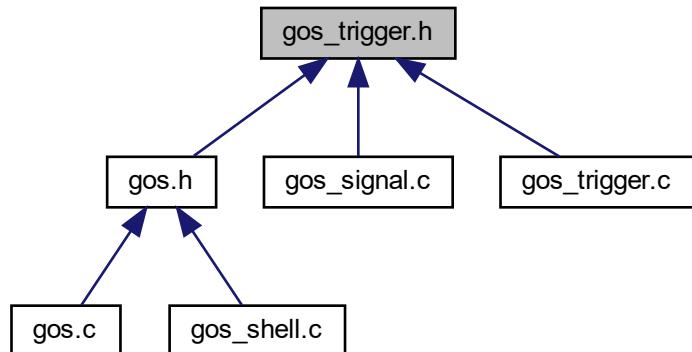
4.43 gos_trigger.h File Reference

GOS trigger service header.

```
#include <gos_kernel.h>
Include dependency graph for gos_trigger.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `gos_trigger_t`

Macros

- #define `GOS_TRIGGER_ENDLESS_TMO` (0xFFFFFFFF)
- #define `GOS_TRIGGER_NO_TMO` (0x00000000)

Functions

- `gos_result_t gos_triggerInit (gos_trigger_t *pTrigger)`
Initializes the trigger instance.
- `gos_result_t gos_triggerReset (gos_trigger_t *pTrigger)`
Resets the trigger counter of the given trigger instance.
- `gos_result_t gos_triggerWait (gos_trigger_t *pTrigger, u32_t value, u32_t timeout)`
Waits for the trigger instance to reach the given trigger value.
- `gos_result_t gos_triggerIncrement (gos_trigger_t *pTrigger)`
Increments the trigger value of the given trigger.
- `gos_result_t gos_triggerDecrement (gos_trigger_t *pTrigger)`
Decrements the trigger value of the given trigger.

4.43.1 Detailed Description

GOS trigger service header.

Author

Ahmed Gazar

Date

2023-11-15

Version

2.4

Trigger service is a way of synchronizing tasks. A trigger instance works as a counter. A trigger can be incremented, and it can be waited on. When waiting for a trigger, the caller must specify a desired trigger value to be reached. Until the value is reached, the caller will wait in a non-blocking way. Once the value is reached, the caller returns.

Definition in file [gos_trigger.h](#).

4.43.2 Macro Definition Documentation

4.43.2.1 `#define GOS_TRIGGER_ENDLESS_TMO (0xFFFFFFFF)`

Mutex endless timeout.

Definition at line 72 of file [gos_trigger.h](#).

4.43.2.2 `#define GOS_TRIGGER_NO_TMO (0x00000000)`

Mutex no timeout.

Definition at line 77 of file [gos_trigger.h](#).

4.43.3 Function Documentation

4.43.3.1 `gos_result_t gos_triggerDecrement (gos_trigger_t * pTrigger)`

Decrements the trigger value of the given trigger.

Decrements the trigger value of the given trigger.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
-----------------	------------------------------------

Returns

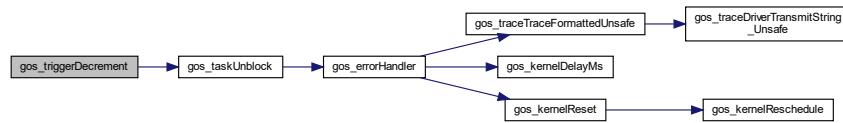
Result of trigger decrementing.

Return values

<i>GOS_SUCCESS</i>	: Decrementing successful.
<i>GOS_ERROR</i>	: Trigger is NULL pointer or value is already zero.

Definition at line 244 of file gos_trigger.c.

Here is the call graph for this function:

**4.43.3.2 gos_result_t gos_triggerIncrement(gos_trigger_t * pTrigger)**

Increments the trigger value of the given trigger.

Increments the trigger value of the given trigger.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
-----------------	------------------------------------

Returns

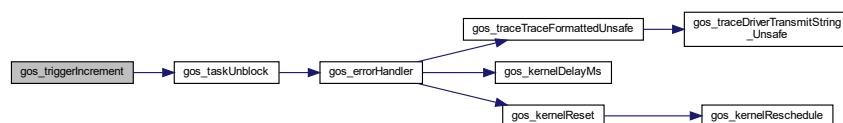
Result of trigger incrementing.

Return values

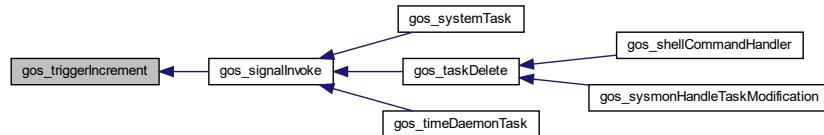
<i>GOS_SUCCESS</i>	: Incrementing successful.
<i>GOS_ERROR</i>	: Trigger is NULL pointer.

Definition at line 197 of file gos_trigger.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.43.3.3 gos_result_t gos_triggerInit(gos_trigger_t * pTrigger)

Initializes the trigger instance.

Calls the initializer for the trigger mutex.

Parameters

<i>pTrigger</i>	: Pointer to the trigger to be initialized.
-----------------	---

Returns

Result of trigger initializing.

Return values

<i>GOS_SUCCESS</i>	: Trigger initialized successfully.
<i>GOS_ERROR</i>	: Trigger descriptor or trigger mutex is NULL pointer.

Definition at line 76 of file gos_trigger.c.

Here is the caller graph for this function:



4.43.3.4 gos_result_t gos_triggerReset(gos_trigger_t * pTrigger)

Resets the trigger counter of the given trigger instance.

Sets the trigger counter of the given trigger instance to zero.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
-----------------	------------------------------------

Returns

Result of trigger resetting.

Return values

<i>GOS_SUCCESS</i>	: Trigger resetting successful.
<i>GOS_ERROR</i>	: Trigger is NULL pointer.

Definition at line 105 of file gos_trigger.c.

Here is the caller graph for this function:



4.43.3.5 `gos_result_t gos_triggerWait(gos_trigger_t * pTrigger, u32_t value, u32_t timeout)`

Waits for the trigger instance to reach the given trigger value.

Increases the trigger waiter number, and waits in a non-blocking way until the desired trigger value is reached or the timeout is reached.

Parameters

<i>pTrigger</i>	: Pointer to the trigger instance.
<i>value</i>	: The desired trigger value.
<i>timeout</i>	: Timeout value.

Returns

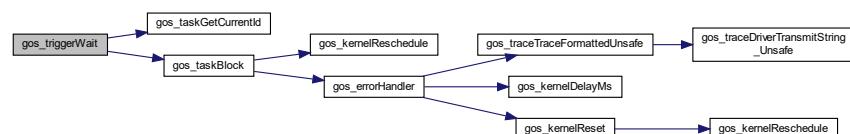
Result of trigger waiting.

Return values

<i>GOS_SUCCESS</i>	: Trigger value reached.
<i>GOS_ERROR</i>	: Trigger value was not reached within the timeout value.

Definition at line 138 of file gos_trigger.c.

Here is the call graph for this function:



Here is the caller graph for this function:



Index

__attribute__
gos.c, 16
gos_error.c, 47
gos_kernel.h, 104, 105
gos_sysmon.c, 223
gos_time.h, 290

ARM_CORTEX_M4
gos_bootloader_config.h, 24
gos_config.h, 32
gos_minimal_example_app_config.h, 148
gos_sysmon_app_config.h, 239

actualCommand
gos_shell.c, 195

atomicCntr
gos_kernel.c, 89

BINARY_PATTERN
gos_kernel.c, 73

CFG_GCP_CHANNELS_MAX_NUMBER
gos_bootloader_config.h, 24
gos_config.h, 32
gos_minimal_example_app_config.h, 148
gos_sysmon_app_config.h, 239

CFG_IDLE_TASK_STACK_SIZE
gos_bootloader_config.h, 24
gos_config.h, 32
gos_minimal_example_app_config.h, 148
gos_sysmon_app_config.h, 239

CFG_MESSAGE_MAX_ADDRESSEES
gos_bootloader_config.h, 25
gos_config.h, 33
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 239

CFG_MESSAGE_MAX_LENGTH
gos_bootloader_config.h, 25
gos_config.h, 33
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 239

CFG_MESSAGE_MAX_NUMBER
gos_bootloader_config.h, 25
gos_config.h, 33
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 239

CFG_MESSAGE_MAX_WAITER_IDS
gos_bootloader_config.h, 25
gos_config.h, 33
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 239

CFG_MESSAGE_MAX_WAITERS
gos_bootloader_config.h, 25
gos_config.h, 33
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 240

CFG_PROC_IDLE_PRIO
gos_bootloader_config.h, 25
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 240

CFG_PROC_MAX_NAME_LENGTH
gos_bootloader_config.h, 25
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 240

CFG_PROC_MAX_NUMBER
gos_bootloader_config.h, 25
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 240

CFG_PROC_MAX_PRIO_LEVELS
gos_bootloader_config.h, 25
gos_minimal_example_app_config.h, 149
gos_sysmon_app_config.h, 240

CFG_PROC_USE_SERVICE
gos_bootloader_config.h, 26
gos_minimal_example_app_config.h, 150
gos_sysmon_app_config.h, 240

CFG_QUEUE_MAX_ELEMENTS
gos_bootloader_config.h, 26
gos_config.h, 33
gos_minimal_example_app_config.h, 150
gos_sysmon_app_config.h, 240

CFG_QUEUE_MAX_LENGTH
gos_bootloader_config.h, 26
gos_config.h, 33
gos_minimal_example_app_config.h, 150
gos_sysmon_app_config.h, 240

CFG_QUEUE_MAX_NAME_LENGTH
gos_bootloader_config.h, 26
gos_config.h, 33
gos_minimal_example_app_config.h, 150
gos_sysmon_app_config.h, 240

CFG_QUEUE_MAX_NUMBER
gos_bootloader_config.h, 26
gos_config.h, 33
gos_minimal_example_app_config.h, 150
gos_sysmon_app_config.h, 241

CFG_QUEUE_USE_NAME
gos_bootloader_config.h, 26
gos_config.h, 34
gos_minimal_example_app_config.h, 150

gos_sysmon_app_config.h, 241
CFG_RESET_ON_ERROR
 gos_bootloader_config.h, 26
 gos_config.h, 34
 gos_minimal_example_app_config.h, 150
 gos_sysmon_app_config.h, 241
CFG_RESET_ON_ERROR_DELAY_MS
 gos_bootloader_config.h, 26
 gos_config.h, 34
 gos_minimal_example_app_config.h, 150
 gos_sysmon_app_config.h, 241
CFG_SCHED_COOPERATIVE
 gos_bootloader_config.h, 26
 gos_config.h, 34
 gos_minimal_example_app_config.h, 150
 gos_sysmon_app_config.h, 241
CFG_SHELL_COMMAND_BUFFER_SIZE
 gos_bootloader_config.h, 27
 gos_config.h, 34
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 241
CFG_SHELL_MAX_COMMAND_LENGTH
 gos_bootloader_config.h, 27
 gos_config.h, 34
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 241
CFG_SHELL_MAX_COMMAND_NUMBER
 gos_bootloader_config.h, 27
 gos_config.h, 34
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 241
CFG_SHELL_MAX_PARAMS_LENGTH
 gos_bootloader_config.h, 27
 gos_config.h, 34
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 241
CFG_SHELL_USE_SERVICE
 gos_bootloader_config.h, 27
 gos_config.h, 34
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 242
CFG_SIGNAL_MAX_NUMBER
 gos_bootloader_config.h, 27
 gos_config.h, 35
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 242
CFG_SIGNAL_MAX_SUBSCRIBERS
 gos_bootloader_config.h, 27
 gos_config.h, 35
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 242
CFG_SYSMON_GCP_CHANNEL_NUM
 gos_bootloader_config.h, 27
 gos_config.h, 35
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 242
CFG_SYSMON_MAX_USER_MESSAGES
 gos_bootloader_config.h, 27
 gos_config.h, 35
 gos_minimal_example_app_config.h, 151
 gos_sysmon_app_config.h, 242
gos_config.h, 35
gos_minimal_example_app_config.h, 151
gos_sysmon_app_config.h, 242
CFG_SYSMON_USE_SERVICE
 gos_bootloader_config.h, 28
 gos_config.h, 35
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 242
CFG_SYSTEM_TASK_STACK_SIZE
 gos_bootloader_config.h, 28
 gos_config.h, 35
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 242
CFG_TARGET_CPU
 gos_bootloader_config.h, 28
 gos_config.h, 35
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 242
CFG_TASK_MAX_NAME_LENGTH
 gos_bootloader_config.h, 28
 gos_config.h, 35
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 242
CFG_TASK_MAX_NUMBER
 gos_bootloader_config.h, 28
 gos_config.h, 35
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 243
CFG_TASK_MAX_STACK_SIZE
 gos_bootloader_config.h, 28
 gos_config.h, 36
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 243
CFG_TASK_MESSAGE_DAEMON_PRIO
 gos_bootloader_config.h, 28
 gos_config.h, 36
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 243
CFG_TASK_MESSAGE_DAEMON_STACK
 gos_bootloader_config.h, 28
 gos_config.h, 36
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 243
CFG_TASK_MIN_STACK_SIZE
 gos_bootloader_config.h, 28
 gos_config.h, 36
 gos_minimal_example_app_config.h, 152
 gos_sysmon_app_config.h, 243
CFG_TASK_PROC_DAEMON_PRIO
 gos_bootloader_config.h, 29
 gos_minimal_example_app_config.h, 153
 gos_sysmon_app_config.h, 243
CFG_TASK_PROC_DAEMON_STACK
 gos_bootloader_config.h, 29
 gos_minimal_example_app_config.h, 153
 gos_sysmon_app_config.h, 243
CFG_TASK_SHELL_DAEMON_PRIO
 gos_bootloader_config.h, 29

gos_config.h, 36
gos_minimal_example_app_config.h, 153
gos_sysmon_app_config.h, 243
CFG_TASK_SHELL_DAEMON_STACK
gos_bootloader_config.h, 29
gos_config.h, 36
gos_minimal_example_app_config.h, 153
gos_sysmon_app_config.h, 243
CFG_TASK_SIGNAL_DAEMON_PRIO
gos_bootloader_config.h, 29
gos_config.h, 36
gos_minimal_example_app_config.h, 153
gos_sysmon_app_config.h, 244
CFG_TASK_SIGNAL_DAEMON_STACK
gos_bootloader_config.h, 29
gos_config.h, 36
gos_minimal_example_app_config.h, 153
gos_sysmon_app_config.h, 244
CFG_TASK_SYS_PRIO
gos_bootloader_config.h, 29
gos_config.h, 36
gos_minimal_example_app_config.h, 153
gos_sysmon_app_config.h, 244
CFG_TASK_SYSMON_DAEMON_PRIO
gos_bootloader_config.h, 29
gos_config.h, 37
gos_minimal_example_app_config.h, 153
gos_sysmon_app_config.h, 244
CFG_TASK_SYSMON_DAEMON_STACK
gos_bootloader_config.h, 29
gos_config.h, 37
gos_minimal_example_app_config.h, 153
gos_sysmon_app_config.h, 244
CFG_TASK_TIME_DAEMON_PRIO
gos_bootloader_config.h, 30
gos_config.h, 37
gos_minimal_example_app_config.h, 154
gos_sysmon_app_config.h, 244
CFG_TASK_TIME_DAEMON_STACK
gos_bootloader_config.h, 30
gos_config.h, 37
gos_minimal_example_app_config.h, 154
gos_sysmon_app_config.h, 244
CFG_TASK_TRACE_DAEMON_PRIO
gos_bootloader_config.h, 30
gos_config.h, 37
gos_minimal_example_app_config.h, 154
gos_sysmon_app_config.h, 244
CFG_TASK_TRACE_DAEMON_STACK
gos_bootloader_config.h, 30
gos_config.h, 37
gos_minimal_example_app_config.h, 154
gos_sysmon_app_config.h, 244
CFG_TRACE_MAX_LENGTH
gos_bootloader_config.h, 30
gos_config.h, 37
gos_minimal_example_app_config.h, 154
gos_sysmon_app_config.h, 245

CFG_USE_PRIO_INHERITANCE
gos_bootloader_config.h, 30
gos_config.h, 37
gos_minimal_example_app_config.h, 154
gos_sysmon_app_config.h, 245
CONFIG_DUMP_SEPARATOR
gos_kernel.c, 73
CRC_INITIAL_VALUE
gos_crc_driver.c, 39
CRC_POLYNOMIAL_VALUE
gos_crc_driver.c, 39
channelFunctions
gos_gcp.c, 65
commandBuffer
gos_shell.c, 195
commandBufferIndex
gos_shell.c, 196
commandParams
gos_shell.c, 196
cpuMessage
gos_sysmon.c, 232
cpuUseLimit
gos_kernel.c, 89
currentTaskIndex
gos_kernel.c, 89

DUMP_SEPARATOR
gos_queue.c, 171
dayLookupTable
gos_time.c, 286
dumpRequired
gos.c, 19

ERROR_BUFFER_SIZE
gos_error.c, 46
errorBuffer
gos_error.c, 50

formattedBuffer
gos_shell_driver.c, 205
gos_trace.c, 306

GCP_ACK_CRC_ERROR
gos_gcp.c, 58
GCP_ACK_INVALID
gos_gcp.c, 58
GCP_ACK_OK
gos_gcp.c, 58
GCP_ACK_PV_ERROR
gos_gcp.c, 58
GCP_ACK_REQ
gos_gcp.c, 58
GCP_ACK_RESEND
gos_gcp.c, 58
GCP_ACK_SIZE_ERROR
gos_gcp.c, 58
GOS_BUSY
gos_kernel.h, 103
GOS_ERROR

gos_kernel.h, 103
GOS_ERROR_LEVEL_OS_FATAL
 gos_error.h, 52
GOS_ERROR_LEVEL_OS_WARNING
 gos_error.h, 52
GOS_ERROR_LEVEL_USER_FATAL
 gos_error.h, 52
GOS_ERROR_LEVEL_USER_WARNING
 gos_error.h, 52
GOS_FALSE
 gos_kernel.h, 102
GOS_MUTEX_LOCKED
 gos_mutex.h, 161
GOS_MUTEX_UNLOCKED
 gos_mutex.h, 161
GOS_PRIVILEGED
 gos_kernel.h, 103
GOS_SUCCESS
 gos_kernel.h, 103
GOS_SYSMON_MSG_CPU_USAGE_GET
 gos_sysmon.c, 221
GOS_SYSMON_MSG_CPU_USAGE_GET_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_CPU_USAGE_GET_PV
 gos_sysmon.c, 222
GOS_SYSMON_MSG_CPU_USAGE_GET_RESP
 gos_sysmon.c, 221
GOS_SYSMON_MSG_CPU_USAGE_GET_RESP_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_CPU_USAGE_GET_RESP_PV
 gos_sysmon.c, 222
GOS_SYSMON_MSG_INV_PAYLOAD_CRC
 gos_sysmon.c, 223
GOS_SYSMON_MSG_INV_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_NUM_OF_MESSAGES
 gos_sysmon.c, 222
GOS_SYSMON_MSG_PING
 gos_sysmon.c, 221
GOS_SYSMON_MSG_PING_ACK_PV
 gos_sysmon.c, 222
GOS_SYSMON_MSG_PING_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_PING_PV
 gos_sysmon.c, 222
GOS_SYSMON_MSG_PING_RESP
 gos_sysmon.c, 221
GOS_SYSMON_MSG_PING_RESP_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_RES_ERROR
 gos_sysmon.c, 223
GOS_SYSMON_MSG_RES_OK
 gos_sysmon.c, 223
GOS_SYSMON_MSG_RESET_REQ
 gos_sysmon.c, 222
GOS_SYSMON_MSG_RESET_REQ_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_RESET_REQ_PV
 gos_sysmon.c, 223
gos_sysmon.c, 223
GOS_SYSMON_MSG_SYSRUNTIME_GET
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSRUNTIME_GET_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSRUNTIME_GET_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_SYSRUNTIME_GET_RESP
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSRUNTIME_GET_RESP_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSRUNTIME_GET_RESP_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_SYSTIME_SET
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSTIME_SET_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSTIME_SET_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_SYSTIME_SET_RESP
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSTIME_SET_RESP_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_SYSTIME_SET_RESP_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_TASK_GET_DATA
 gos_sysmon.c, 221
GOS_SYSMON_MSG_TASK_GET_DATA_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_GET_DATA_PV
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_GET_DATA_RESP
 gos_sysmon.c, 221
GOS_SYSMON_MSG_TASK_GET_DATA_RESP_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_GET_DATA_RESP_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_TASK_GET_VAR_DATA
 gos_sysmon.c, 221
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP
 gos_sysmon.c, 221
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_GET_VAR_DATA_RESP_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_TASK_MODIFY_STATE
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_MODIFY_STATE_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_MODIFY_STATE_PV

gos_sysmon.c, 223
GOS_SYSMON_MSG_TASK_MODIFY_STATE_RE←
 SP
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_MODIFY_STATE_RE←
 SP_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_TASK_MODIFY_STATE_RE←
 SP_PV
 gos_sysmon.c, 223
GOS_SYSMON_MSG_UNKNOWN
 gos_sysmon.c, 221
GOS_SYSMON_MSG_UNKNOWN_ID
 gos_sysmon.c, 222
GOS_SYSMON_MSG_UNKNOWN_PV
 gos_sysmon.c, 222
GOS_SYSMON_TASK_MOD_TYPE_BLOCK
 gos_sysmon.c, 223
GOS_SYSMON_TASK_MOD_TYPE_DELETE
 gos_sysmon.c, 223
GOS_SYSMON_TASK_MOD_TYPE_RESUME
 gos_sysmon.c, 223
GOS_SYSMON_TASK_MOD_TYPE_SUSPEND
 gos_sysmon.c, 223
GOS_SYSMON_TASK_MOD_TYPE_UNBLOCK
 gos_sysmon.c, 223
GOS_SYSMON_TASK_MOD_TYPE_WAKEUP
 gos_sysmon.c, 223
GOS_TASK_BLOCKED
 gos_kernel.h, 104
GOS_TASK_PRIVILEGE_KERNEL
 gos_kernel.h, 103
GOS_TASK_PRIVILEGE_SUPERVISOR
 gos_kernel.h, 103
GOS_TASK_PRIVILEGE_USER
 gos_kernel.h, 103
GOS_TASK_PRIVILEGED_USER
 gos_kernel.h, 103
GOS_TASK_READY
 gos_kernel.h, 104
GOS_TASK_SLEEPING
 gos_kernel.h, 104
GOS_TASK_SUSPENDED
 gos_kernel.h, 104
GOS_TASK_ZOMBIE
 gos_kernel.h, 104
GOS_TIME_APRIIL
 gos_time.h, 290
GOS_TIME_AUGUST
 gos_time.h, 290
GOS_TIME_DAY_ELAPSED_SENDER_ID
 gos_time.h, 289
GOS_TIME_DECEMBER
 gos_time.h, 290
GOS_TIME_EARLIER
 gos_time.h, 289
GOS_TIME_EQUAL
 gos_time.h, 289
GOS_TIME_FEBRUARY
 gos_time.h, 290
GOS_TIME_HOUR_ELAPSED_SENDER_ID
 gos_time.h, 289
GOS_TIME_JANUARY
 gos_time.h, 290
GOS_TIME_JULY
 gos_time.h, 290
GOS_TIME_JUNE
 gos_time.h, 290
GOS_TIME_LATER
 gos_time.h, 289
GOS_TIME_MARCH
 gos_time.h, 290
GOS_TIME_MAY
 gos_time.h, 290
GOS_TIME_MINUTE_ELAPSED_SENDER_ID
 gos_time.h, 289
GOS_TIME_MONTH_ELAPSED_SENDER_ID
 gos_time.h, 289
GOS_TIME_NOVEMBER
 gos_time.h, 290
GOS_TIME_NUMBER_OF_MONTHS
 gos_time.h, 290
GOS_TIME_OCTOBER
 gos_time.h, 290
GOS_TIME_SECOND_ELAPSED_SENDER_ID
 gos_time.h, 289
GOS_TIME_SEPTEMBER
 gos_time.h, 290
GOS_TIME_YEAR_ELAPSED_SENDER_ID
 gos_time.h, 289
GOS_TRUE
 gos_kernel.h, 102
GOS_UNPRIVILEGED
 gos_kernel.h, 103
GCP_PROTOCOL_VERSION_MAJOR
 gos_gcp.c, 57
GCP_PROTOCOL_VERSION_MINOR
 gos_gcp.c, 57
GLOBAL_STACK
 gos_kernel.h, 97
GOS_ASM
 gos_kernel.h, 97
GOS_ATOMIC_ENTER
 gos_kernel.h, 97
GOS_ATOMIC_EXIT
 gos_kernel.h, 98
GOS_CFG_OVERCONFIG
 gos_bootloader_config.h, 30
 gos_minimal_example_app_config.h, 154
 gos_sysmon_app_config.h, 245
GOS_CONCAT_RESULT
 gos_kernel.h, 98
GOS_CONST
 gos_kernel.h, 99
GOS_DEFAULT_QUEUE_ID
 gos_queue.h, 180

GOS_DEFAULT_TASK_ID
 gos_kernel.h, 99

GOS_DISABLE_SCHED
 gos_kernel.h, 99

GOS_ENABLE_SCHED
 gos_kernel.h, 99

GOS_EXTERN
 gos_kernel.h, 99

GOS_INLINE
 gos_kernel.h, 99

GOS_INVALID_QUEUE_ID
 gos_queue.h, 180

GOS_INVALID_TASK_ID
 gos_kernel.h, 99

GOS_ISR_ENTER
 gos_kernel.h, 100

GOS_ISR_EXIT
 gos_kernel.h, 100

GOS_MESSAGE_DAEMON_POLL_TIME_MS
 gos_message.c, 139

GOS_MESSAGE_ENDLESS_TMO
 gos_message.h, 144

GOS_MESSAGE_INVALID_ID
 gos_message.h, 144

GOS_MUTEX_ENDLESS_TMO
 gos_mutex.h, 161

GOS_MUTEX_NO_TMO
 gos_mutex.h, 161

GOS_NAKED
 gos_kernel.h, 100

GOS_NOP
 gos_kernel.h, 100

GOS_PRIV_RESERVED_3
 gos_kernel.h, 100

GOS_PRIV_RESERVED_4
 gos_kernel.h, 100

GOS_PRIV_RESERVED_5
 gos_kernel.h, 100

GOS_PRIV_SIGNALING
 gos_kernel.h, 101

GOS_PRIV_TASK_MANIPULATE
 gos_kernel.h, 101

GOS_PRIV_TASK_PRIO_CHANGE
 gos_kernel.h, 101

GOS_PRIV_TRACE
 gos_kernel.h, 101

GOS_SHELL_DAEMON_POLL_TIME_MS
 gos_shell.c, 189

GOS_SHELL_DISPLAY_TEXT
 gos_shell.c, 189

GOS_STATIC
 gos_kernel.h, 101

GOS_STATIC_INLINE
 gos_kernel.h, 101

GOS_SYS_TASK_SLEEP_TIME
 gos.c, 16

GOS_TASK_IDLE_PRIO
 gos_kernel.h, 101

GOS_TASK_MAX_BLOCK_TIME_MS
 gos_kernel.h, 101

GOS_TASK_MAX_PRIO_LEVELS
 gos_kernel.h, 101

GOS_TRACE_MUTEX_TMO_MS
 gos_trace.c, 302

GOS_TRACE_QUEUE_TMO_MS
 gos_trace.c, 302

GOS_TRACE_TIMESTAMP_FORMAT
 gos_trace.c, 302

GOS_TRACE_TIMESTAMP_LENGTH
 gos_trace.c, 302

GOS_TRIGGER_ENDLESS_TMO
 gos_trigger.h, 328

GOS_TRIGGER_NO_TMO
 gos_trigger.h, 328

GOS_UNUSED
 gos_kernel.h, 102

GOS_VERSION_MAJOR
 gos.h, 21

GOS_VERSION_MINOR
 gos.h, 21

gcpRxMutexes
 gos_gcp.c, 65

gcpTxMutexes
 gos_gcp.c, 65

gos.c, 15
 __attribute__, 16
 dumpRequired, 19
 GOS_SYS_TASK_SLEEP_TIME, 16
 gos_Dump, 17
 gos_Start, 18
 gos_initFunc_t, 16
 gos_systemTask, 18
 initError, 19
 initializers, 19
 systemTaskDesc, 19
 systemTaskId, 20
 gos.h, 20
 GOS_VERSION_MAJOR, 21
 GOS_VERSION_MINOR, 21
 gos_Dump, 22
 gos_error.h
 GOS_ERROR_LEVEL_OS_FATAL, 52
 GOS_ERROR_LEVEL_OS_WARNING, 52
 GOS_ERROR_LEVEL_USER_FATAL, 52
 GOS_ERROR_LEVEL_USER_WARNING, 52
 gos_gcp.c
 GCP_ACK_CRC_ERROR, 58
 GCP_ACK_INVALID, 58
 GCP_ACK_OK, 58
 GCP_ACK_PV_ERROR, 58
 GCP_ACK_REQ, 58
 GCP_ACK_RESEND, 58
 GCP_ACK_SIZE_ERROR, 58
 gos_kernel.h
 GOS_BUSY, 103
 GOS_ERROR, 103

GOS_FALSE, 102
GOS_PRIVILEGED, 103
GOS_SUCCESS, 103
GOS_TASK_BLOCKED, 104
GOS_TASK_PRIVILEGE_KERNEL, 103
GOS_TASK_PRIVILEGE_SUPERVISOR, 103
GOS_TASK_PRIVILEGE_USER, 103
GOS_TASK_PRIVILEGED_USER, 103
GOS_TASK_READY, 104
GOS_TASK_SLEEPING, 104
GOS_TASK_SUSPENDED, 104
GOS_TASK_ZOMBIE, 104
GOS_TRUE, 102
GOS_UNPRIVILEGED, 103

gos_mutex.h
 GOS_MUTEX_LOCKED, 161
 GOS_MUTEX_UNLOCKED, 161

gos_sysmon.c
 GOS_SYSMON_MSG_CPU_USAGE_GET, 221
 GOS_SYSMON_MSG_CPU_USAGE_GET_ID,
 222
 GOS_SYSMON_MSG_CPU_USAGE_GET_PV,
 222
 GOS_SYSMON_MSG_CPU_USAGE_GET_RE←
 SP, 221
 GOS_SYSMON_MSG_CPU_USAGE_GET_RE←
 SP_ID, 222
 GOS_SYSMON_MSG_CPU_USAGE_GET_RE←
 SP_PV, 222
 GOS_SYSMON_MSG_INV_PAYLOAD_CRC, 223
 GOS_SYSMON_MSG_INV_PV, 223
 GOS_SYSMON_MSG_NUM_OF_MESSAGES,
 222
 GOS_SYSMON_MSG_PING, 221
 GOS_SYSMON_MSG_PING_ACK_PV, 222
 GOS_SYSMON_MSG_PING_ID, 222
 GOS_SYSMON_MSG_PING_PV, 222
 GOS_SYSMON_MSG_PING_RESP, 221
 GOS_SYSMON_MSG_PING_RESP_ID, 222
 GOS_SYSMON_MSG_RES_ERROR, 223
 GOS_SYSMON_MSG_RES_OK, 223
 GOS_SYSMON_MSG_RESET_REQ, 222
 GOS_SYSMON_MSG_RESET_REQ_ID, 222
 GOS_SYSMON_MSG_RESET_REQ_PV, 223
 GOS_SYSMON_MSG_SYSRUNTIME_GET, 222
 GOS_SYSMON_MSG_SYSRUNTIME_GET_ID,
 222
 GOS_SYSMON_MSG_SYSRUNTIME_GET_PV,
 223
 GOS_SYSMON_MSG_SYSRUNTIME_GET_RE←
 SP, 222
 GOS_SYSMON_MSG_SYSRUNTIME_GET_RE←
 SP_ID, 222
 GOS_SYSMON_MSG_SYSRUNTIME_GET_RE←
 SP_PV, 223
 GOS_SYSMON_MSG_SYSTIME_SET, 222
 GOS_SYSMON_MSG_SYSTIME_SET_ID, 222
 GOS_SYSMON_MSG_SYSTIME_SET_PV, 223

gos_time.h
 GOS_TIME_APRIIL, 290
 GOS_TIME_AUGUST, 290

GOS_TIME_DAY_ELAPSED_SENDER_ID, 289
 GOS_TIME_DECEMBER, 290
 GOS_TIME_EARLIER, 289
 GOS_TIME_EQUAL, 289
 GOS_TIME_FEBRUARY, 290
 GOS_TIME_HOUR_ELAPSED_SENDER_ID, 289
 GOS_TIME_JANUARY, 290
 GOS_TIME_JULY, 290
 GOS_TIME_JUNE, 290
 GOS_TIME_LATER, 289
 GOS_TIME_MARCH, 290
 GOS_TIME_MAY, 290
 GOS_TIME_MINUTE_ELAPSED_SENDER_ID, 289
 GOS_TIME_MONTH_ELAPSED_SENDER_ID, 289
 GOS_TIME_NOVEMBER, 290
 GOS_TIME_NUMBER_OF_MONTHS, 290
 GOS_TIME_OCTOBER, 290
 GOS_TIME_SECOND_ELAPSED_SENDER_ID, 289
 GOS_TIME_SEPTEMBER, 290
 GOS_TIME_YEAR_ELAPSED_SENDER_ID, 289
gos_Dump
 gos.c, 17
 gos.h, 22
gos_Start
 gos.c, 18
gos_boolValue_t
 gos_kernel.h, 102
gos_bootloader_config.h, 22
 ARM_CORTEX_M4, 24
 CFG_GCP_CHANNELS_MAX_NUMBER, 24
 CFG_IDLE_TASK_STACK_SIZE, 24
 CFG_MESSAGE_MAX_ADDRESSEES, 25
 CFG_MESSAGE_MAX_LENGTH, 25
 CFG_MESSAGE_MAX_NUMBER, 25
 CFG_MESSAGE_MAX_WAITER_IDS, 25
 CFG_MESSAGE_MAX_WAITERS, 25
 CFG_PROC_IDLE_PRIO, 25
 CFG_PROC_MAX_NAME_LENGTH, 25
 CFG_PROC_MAX_NUMBER, 25
 CFG_PROC_MAX_PRIO_LEVELS, 25
 CFG_PROC_USE_SERVICE, 26
 CFG_QUEUE_MAX_ELEMENTS, 26
 CFG_QUEUE_MAX_LENGTH, 26
 CFG_QUEUE_MAX_NAME_LENGTH, 26
 CFG_QUEUE_MAX_NUMBER, 26
 CFG_QUEUE_USE_NAME, 26
 CFG_RESET_ON_ERROR, 26
 CFG_RESET_ON_ERROR_DELAY_MS, 26
 CFG_SCHED_COOPERATIVE, 26
 CFG_SHELL_COMMAND_BUFFER_SIZE, 27
 CFG_SHELL_MAX_COMMAND_LENGTH, 27
 CFG_SHELL_MAX_COMMAND_NUMBER, 27
 CFG_SHELL_MAX_PARAMS_LENGTH, 27
 CFG_SHELL_USE_SERVICE, 27
 CFG_SIGNAL_MAX_NUMBER, 27
 CFG_SIGNAL_MAX_SUBSCRIBERS, 27
 CFG_SYSMON_GCP_CHANNEL_NUM, 27
 CFG_SYSMON_MAX_USER_MESSAGES, 27
 CFG_SYSMON_USE_SERVICE, 28
 CFG_SYSTEM_TASK_STACK_SIZE, 28
 CFG_TARGET_CPU, 28
 CFG_TASK_MAX_NAME_LENGTH, 28
 CFG_TASK_MAX_NUMBER, 28
 CFG_TASK_MAX_STACK_SIZE, 28
 CFG_TASK_MESSAGE_DAEMON_PRIO, 28
 CFG_TASK_MESSAGE_DAEMON_STACK, 28
 CFG_TASK_MIN_STACK_SIZE, 28
 CFG_TASK_PROC_DAEMON_PRIO, 29
 CFG_TASK_PROC_DAEMON_STACK, 29
 CFG_TASK_SHELL_DAEMON_PRIO, 29
 CFG_TASK_SHELL_DAEMON_STACK, 29
 CFG_TASK_SIGNAL_DAEMON_PRIO, 29
 CFG_TASK_SIGNAL_DAEMON_STACK, 29
 CFG_TASK_SYS_PRIO, 29
 CFG_TASK_SYSMON_DAEMON_PRIO, 29
 CFG_TASK_SYSMON_DAEMON_STACK, 29
 CFG_TASK_TIME_DAEMON_PRIO, 30
 CFG_TASK_TIME_DAEMON_STACK, 30
 CFG_TASK_TRACE_DAEMON_PRIO, 30
 CFG_TASK_TRACE_DAEMON_STACK, 30
 CFG_TRACE_MAX_LENGTH, 30
 CFG_USE_PRIO_INHERITANCE, 30
 GOS_CFG_OVERCONFIG, 30
gos_config.h, 30
 ARM_CORTEX_M4, 32
 CFG_GCP_CHANNELS_MAX_NUMBER, 32
 CFG_IDLE_TASK_STACK_SIZE, 32
 CFG_MESSAGE_MAX_ADDRESSEES, 33
 CFG_MESSAGE_MAX_LENGTH, 33
 CFG_MESSAGE_MAX_NUMBER, 33
 CFG_MESSAGE_MAX_WAITER_IDS, 33
 CFG_MESSAGE_MAX_WAITERS, 33
 CFG_QUEUE_MAX_ELEMENTS, 33
 CFG_QUEUE_MAX_LENGTH, 33
 CFG_QUEUE_MAX_NAME_LENGTH, 33
 CFG_QUEUE_MAX_NUMBER, 33
 CFG_QUEUE_USE_NAME, 34
 CFG_RESET_ON_ERROR, 34
 CFG_RESET_ON_ERROR_DELAY_MS, 34
 CFG_SCHED_COOPERATIVE, 34
 CFG_SHELL_COMMAND_BUFFER_SIZE, 34
 CFG_SHELL_MAX_COMMAND_LENGTH, 34
 CFG_SHELL_MAX_COMMAND_NUMBER, 34
 CFG_SHELL_MAX_PARAMS_LENGTH, 34
 CFG_SHELL_USE_SERVICE, 34
 CFG_SIGNAL_MAX_NUMBER, 35
 CFG_SIGNAL_MAX_SUBSCRIBERS, 35
 CFG_SYSMON_GCP_CHANNEL_NUM, 35
 CFG_SYSMON_MAX_USER_MESSAGES, 35
 CFG_SYSMON_USE_SERVICE, 35
 CFG_SYSTEM_TASK_STACK_SIZE, 35
 CFG_TARGET_CPU, 35
 CFG_TASK_MAX_NAME_LENGTH, 35

CFG_TASK_MAX_NUMBER, 35
CFG_TASK_MAX_STACK_SIZE, 36
CFG_TASK_MESSAGE_DAEMON_PRIO, 36
CFG_TASK_MESSAGE_DAEMON_STACK, 36
CFG_TASK_MIN_STACK_SIZE, 36
CFG_TASK_SHELL_DAEMON_PRIO, 36
CFG_TASK_SHELL_DAEMON_STACK, 36
CFG_TASK_SIGNAL_DAEMON_PRIO, 36
CFG_TASK_SIGNAL_DAEMON_STACK, 36
CFG_TASK_SYS_PRIO, 36
CFG_TASK_SYSMON_DAEMON_PRIO, 37
CFG_TASK_SYSMON_DAEMON_STACK, 37
CFG_TASK_TIME_DAEMON_PRIO, 37
CFG_TASK_TIME_DAEMON_STACK, 37
CFG_TASK_TRACE_DAEMON_PRIO, 37
CFG_TASK_TRACE_DAEMON_STACK, 37
CFG_TRACE_MAX_LENGTH, 37
CFG_USE_PRIO_INHERITANCE, 37

gos_crc_driver.c, 38
 CRC_INITIAL_VALUE, 39
 CRC_POLYNOMIAL_VALUE, 39
 gos_crcDriverGetCrc, 39

gos_crc_driver.h, 40
 gos_crcDriverGetCrc, 41

gos_crcDriverGetCrc
 gos_crc_driver.c, 39
 gos_crc_driver.h, 41

gos_driver.c, 41
 gos_driverInit, 42

gos_driver.h, 43
 gos_driverInit, 44

gos_driver_functions_t, 5

gos_driverInit
 gos_driver.c, 42
 gos_driver.h, 44

gos_error.c, 44
 __attribute__, 47
 ERROR_BUFFER_SIZE, 46
 errorBuffer, 50
 gos_errorHandler, 47
 gos_errorTraceInit, 48
 gos_traceResultToString, 49
 RESULT_STRING_ERROR, 46
 RESULT_STRING_SUCCESS, 46
 RESULT_STRING_UNKNOWN, 46
 SEPARATOR_LINE, 46

gos_error.h, 50
 gos_errorHandler, 52
 gos_errorLevel_t, 52
 gos_errorTraceInit, 54
 gos_printStartupLogo, 55

gos_errorHandler
 gos_error.c, 47
 gos_error.h, 52

gos_errorLevel_t
 gos_error.h, 52

gos_errorTraceInit
 gos_error.c, 48

gos_error.h, 54
gos_gcp.c, 55
 channelFunctions, 65
 GCP_PROTOCOL_VERSION_MAJOR, 57
 GCP_PROTOCOL_VERSION_MINOR, 57
 gcpRxMutexes, 65
 gcpTxMutexes, 65
 gos_gcpAck_t, 58
 gos_gcpInit, 58
 gos_gcpReceiveMessage, 58
 gos_gcpReceiveMessageInternal, 59
 gos_gcpRegisterPhysicalDriver, 60
 gos_gcpTransmitMessage, 61
 gos_gcpTransmitMessageInternal, 62
 gos_gcpValidateData, 63
 gos_gcpValidateHeader, 64

gos_gcp.h, 65
 gos_gcpInit, 67
 gos_gcpReceiveFunction_t, 67
 gos_gcpReceiveMessage, 67
 gos_gcpRegisterPhysicalDriver, 68
 gos_gcpTransmitFunction_t, 67
 gos_gcpTransmitMessage, 69

gos_gcpAck_t
 gos_gcp.c, 58

gos_gcpChannelFunctions_t, 5

gos_gcpHeaderFrame_t, 6

gos_gcpInit
 gos_gcp.c, 58
 gos_gcp.h, 67

gos_gcpReceiveFunction_t
 gos_gcp.h, 67

gos_gcpReceiveMessage
 gos_gcp.c, 58
 gos_gcp.h, 67

gos_gcpReceiveMessageInternal
 gos_gcp.c, 59

gos_gcpRegisterPhysicalDriver
 gos_gcp.c, 60
 gos_gcp.h, 68

gos_gcpTransmitFunction_t
 gos_gcp.h, 67

gos_gcpTransmitMessage
 gos_gcp.c, 61
 gos_gcp.h, 69

gos_gcpTransmitMessageInternal
 gos_gcp.c, 62

gos_gcpValidateData
 gos_gcp.c, 63

gos_gcpValidateHeader
 gos_gcp.c, 64

gos_idleTask
 gos_kernel.c, 74
 gos_task.c, 253

gos_initFunc_t
 gos.c, 16

gos_initStruct_t, 6

gos_kernel.c, 70

atomicCntr, 89
 BINARY_PATTERN, 73
 CONFIG_DUMP_SEPARATOR, 73
 cpuUseLimit, 89
 currentTaskIndex, 89
 gos_idleTask, 74
 gos_kernelCalculateTaskCpuUsages, 75
 gos_kernelCheckTaskStack, 76
 gos_kernelDelayMs, 77
 gos_kernelDelayUs, 78
 gos_kernelDump, 79
 gos_kernelGetCpuUsage, 79
 gos_kernelGetCurrentPsp, 80
 gos_kernelGetMaxCpuLoad, 80
 gos_kernelGetSysTicks, 81
 gos_kernelGetTaskStateString, 81
 gos_kernellInit, 82
 gos_kernelsCallerlsr, 82
 gos_kernelPrivilegedModeSetRequired, 83
 gos_kernelProcessorReset, 83
 gos_kernelRegisterPrivilegedHook, 83
 gos_kernelRegisterSwapHook, 85
 gos_kernelRegisterSysTickHook, 85
 gos_kernelReschedule, 86
 gos_kernelReset, 86
 gos_kernelSaveCurrentPsp, 87
 gos_kernelSelectNextTask, 88
 gos_kernelSetMaxCpuLoad, 88
 gos_kernelStart, 89
 ICSR, 73
 inlsr, 90
 isKernelRunning, 90
 kernelDumpReadySignal, 90
 kernelDumpSignal, 90
 kernelPrivilegedHookFunction, 90
 kernelSwapHookFunction, 90
 kernelSysTickHookFunction, 90
 MAX_CPU_DUMP_SEPARATOR, 73
 monitoringTime, 90
 previousTick, 90
 primask, 91
 privilegedModeSetRequired, 91
 resetRequired, 91
 SHCSR, 73
 STACK_STATS_SEPARATOR, 74
 schedDisableCntr, 91
 sysTicks, 91
 sysTimerValue, 91
 TASK_DUMP_SEPARATOR, 74
 TO_BINARY, 74
 gos_kernel.h, 91
 __attribute__, 104, 105
 GLOBAL_STACK, 97
 GOS_ASM, 97
 GOS_ATOMIC_ENTER, 97
 GOS_ATOMIC_EXIT, 98
 GOS_CONCAT_RESULT, 98
 GOS_CONST, 99
 GOS_DEFAULT_TASK_ID, 99
 GOS_DISABLE_SCHED, 99
 GOS_ENABLE_SCHED, 99
 GOS_EXTERN, 99
 GOS_INLINE, 99
 GOS_INVALID_TASK_ID, 99
 GOS_ISR_ENTER, 100
 GOS_ISR_EXIT, 100
 GOS_NAKED, 100
 GOS_NOP, 100
 GOS_PRIV_RESERVED_3, 100
 GOS_PRIV_RESERVED_4, 100
 GOS_PRIV_RESERVED_5, 100
 GOS_PRIV_SIGNALING, 101
 GOS_PRIV_TASK_MANIPULATE, 101
 GOS_PRIV_TASK_PRIO_CHANGE, 101
 GOS_PRIV_TRACE, 101
 GOS_STATIC, 101
 GOS_STATIC_INLINE, 101
 GOS_TASK_IDLE_PRIO, 101
 GOS_TASK_MAX_BLOCK_TIME_MS, 101
 GOS_TASK_MAX_PRIO_LEVELS, 101
 GOS_UNUSED, 102
 gos_boolValue_t, 102
 gos_kernel_privilege_t, 102
 gos_kernelCalculateTaskCpuUsages, 105
 gos_kernelDelayMs, 106
 gos_kernelDelayUs, 107
 gos_kernelDump, 108
 gos_kernelGetCpuUsage, 108
 gos_kernelGetMaxCpuLoad, 109
 gos_kernelGetSysTicks, 109
 gos_kernellInit, 110
 gos_kernelsCallerlsr, 110
 gos_kernelPrivilegedModeSetRequired, 111
 gos_kernelRegisterIdleHook, 111
 gos_kernelRegisterPrivilegedHook, 112
 gos_kernelRegisterSwapHook, 112
 gos_kernelRegisterSysTickHook, 113
 gos_kernelReschedule, 113
 gos_kernelReset, 114
 gos_kernelSetMaxCpuLoad, 115
 gos_kernelStart, 116
 gos_kernelSubscribeDumpReadySignal, 116
 gos_result_t, 103
 gos_taskAddPrivilege, 118
 gos_taskBlock, 118
 gos_taskDelete, 119
 gos_taskGetCurrentId, 120
 gos_taskGetData, 121
 gos_taskGetDataByIndex, 122
 gos_taskGetId, 122
 gos_taskGetName, 123
 gos_taskGetNumber, 123
 gos_taskGetOriginalPriority, 124
 gos_taskGetPriority, 124
 gos_taskGetPrivileges, 125
 gos_taskPrivilegeLevel_t, 103

gos_taskRegister, 126
gos_taskRegisterTasks, 127
gos_taskRemovePrivilege, 128
gos_taskResume, 128
gos_taskSetOriginalPriority, 129
gos_taskSetPriority, 130
gos_taskSetPrivileges, 131
gos_taskSleep, 132
gos_taskState_t, 103
gos_taskSubscribeDeleteSignal, 133
gos_taskSuspend, 133
gos_taskUnblock, 134
gos_taskWakeUp, 135
gos_taskYield, 136
MAIN_STACK, 102
NULL, 102
RAM_SIZE, 102
RAM_START, 102
gos_kernel_privilege_t
 gos_kernel.h, 102
gos_kernelCalculateTaskCpuUsages
 gos_kernel.c, 75
 gos_kernel.h, 105
gos_kernelCheckTaskStack
 gos_kernel.c, 76
gos_kernelDelayMs
 gos_kernel.c, 77
 gos_kernel.h, 106
gos_kernelDelayUs
 gos_kernel.c, 78
 gos_kernel.h, 107
gos_kernelDump
 gos_kernel.c, 79
 gos_kernel.h, 108
gos_kernelGetCpuUsage
 gos_kernel.c, 79
 gos_kernel.h, 108
gos_kernelGetCurrentPsp
 gos_kernel.c, 80
gos_kernelGetMaxCpuLoad
 gos_kernel.c, 80
 gos_kernel.h, 109
gos_kernelGetSysTicks
 gos_kernel.c, 81
 gos_kernel.h, 109
gos_kernelGetTaskStateString
 gos_kernel.c, 81
gos_kernellInit
 gos_kernel.c, 82
 gos_kernel.h, 110
gos_kernelsCallerIsr
 gos_kernel.c, 82
 gos_kernel.h, 110
gos_kernelPrivilegedModeSetRequired
 gos_kernel.c, 83
 gos_kernel.h, 111
gos_kernelProcessorReset
 gos_kernel.c, 83
gos_kernelRegisterIdleHook
 gos_kernel.h, 111
gos_kernelRegisterPrivilegedHook
 gos_kernel.c, 83
 gos_kernel.h, 112
gos_kernelRegisterSwapHook
 gos_kernel.c, 85
 gos_kernel.h, 112
gos_kernelRegisterSysTickHook
 gos_kernel.c, 85
 gos_kernel.h, 113
gos_kernelReschedule
 gos_kernel.c, 86
 gos_kernel.h, 113
gos_kernelReset
 gos_kernel.c, 86
 gos_kernel.h, 114
gos_kernelSaveCurrentPsp
 gos_kernel.c, 87
gos_kernelSelectNextTask
 gos_kernel.c, 88
gos_kernelSetMaxCpuLoad
 gos_kernel.c, 88
 gos_kernel.h, 115
gos_kernelStart
 gos_kernel.c, 89
 gos_kernel.h, 116
gos_kernelSubscribeDumpReadySignal
 gos_kernel.h, 116
gos_message.c, 137
 GOS_MESSAGE_DAEMON_POLL_TIME_MS,
 139
 gos_messageDaemonTask, 139
 gos_messageInit, 139
 gos_messageRx, 140
 gos_messageTx, 140
 messageArray, 141
 messageDaemonTaskDesc, 141
 messageDaemonTaskId, 141
 messageMutex, 141
 messageWaiterArray, 142
 nextMessageIndex, 142
 nextWaiterIndex, 142
gos_message.h, 142
 GOS_MESSAGE_ENDLESS_TMO, 144
 GOS_MESSAGE_INVALID_ID, 144
 gos_messageId_t, 144
 gos_messageInit, 145
 gos_messageRx, 145
 gos_messageTimeout_t, 144
 gos_messageTx, 146
gos_message_t, 7
gos_messageDaemonTask
 gos_message.c, 139
gos_messageId_t
 gos_message.h, 144
gos_messageInit
 gos_message.c, 139

gos_message.h, 145
 gos_messageRx
 gos_message.c, 140
 gos_message.h, 145
 gos_messageTimeout_t
 gos_message.h, 144
 gos_messageTx
 gos_message.c, 140
 gos_message.h, 146
 gos_messageWaiterDesc_t, 7
 gos_minimal_example_app_config.h, 146
 ARM_CORTEX_M4, 148
 CFG_GCP_CHANNELS_MAX_NUMBER, 148
 CFG_IDLE_TASK_STACK_SIZE, 148
 CFG_MESSAGE_MAX_ADDRESSEES, 149
 CFG_MESSAGE_MAX_LENGTH, 149
 CFG_MESSAGE_MAX_NUMBER, 149
 CFG_MESSAGE_MAX_WAITER_IDS, 149
 CFG_MESSAGE_MAX_WAITERS, 149
 CFG_PROC_IDLE_PRIO, 149
 CFG_PROC_MAX_NAME_LENGTH, 149
 CFG_PROC_MAX_NUMBER, 149
 CFG_PROC_MAX_PRIO_LEVELS, 149
 CFG_PROC_USE_SERVICE, 150
 CFG_QUEUE_MAX_ELEMENTS, 150
 CFG_QUEUE_MAX_LENGTH, 150
 CFG_QUEUE_MAX_NAME_LENGTH, 150
 CFG_QUEUE_MAX_NUMBER, 150
 CFG_QUEUE_USE_NAME, 150
 CFG_RESET_ON_ERROR, 150
 CFG_RESET_ON_ERROR_DELAY_MS, 150
 CFG_SCHED_COOPERATIVE, 150
 CFG_SHELL_COMMAND_BUFFER_SIZE, 151
 CFG_SHELL_MAX_COMMAND_LENGTH, 151
 CFG_SHELL_MAX_COMMAND_NUMBER, 151
 CFG_SHELL_MAX_PARAMS_LENGTH, 151
 CFG_SHELL_USE_SERVICE, 151
 CFG_SIGNAL_MAX_NUMBER, 151
 CFG_SIGNAL_MAX_SUBSCRIBERS, 151
 CFG_SYSMON_GCP_CHANNEL_NUM, 151
 CFG_SYSMON_MAX_USER_MESSAGES, 151
 CFG_SYSMON_USE_SERVICE, 152
 CFG_SYSTEM_TASK_STACK_SIZE, 152
 CFG_TARGET_CPU, 152
 CFG_TASK_MAX_NAME_LENGTH, 152
 CFG_TASK_MAX_NUMBER, 152
 CFG_TASK_MAX_STACK_SIZE, 152
 CFG_TASK_MESSAGE_DAEMON_PRIO, 152
 CFG_TASK_MESSAGE_DAEMON_STACK, 152
 CFG_TASK_MIN_STACK_SIZE, 152
 CFG_TASK_PROC_DAEMON_PRIO, 153
 CFG_TASK_PROC_DAEMON_STACK, 153
 CFG_TASK_SHELL_DAEMON_PRIO, 153
 CFG_TASK_SHELL_DAEMON_STACK, 153
 CFG_TASK_SIGNAL_DAEMON_PRIO, 153
 CFG_TASK_SIGNAL_DAEMON_STACK, 153
 CFG_TASK_SYS_PRIO, 153
 CFG_TASK_SYSMON_DAEMON_PRIO, 153
 CFG_TASK_SYSMON_DAEMON_STACK, 153
 CFG_TASK_TIME_DAEMON_PRIO, 154
 CFG_TASK_TIME_DAEMON_STACK, 154
 CFG_TASK_TRACE_DAEMON_PRIO, 154
 CFG_TASK_TRACE_DAEMON_STACK, 154
 CFG_TRACE_MAX_LENGTH, 154
 CFG_USE_PRIO_INHERITANCE, 154
 GOS_CFG_OVERCONFIG, 154
 gos_mutex.c, 154
 gos_mutexInit, 156
 gos_mutexLock, 157
 gos_mutexUnlock, 158
 MUTEX_LOCK_SLEEP_MS, 156
 gos_mutex.h, 159
 GOS_MUTEX_ENDLESS_TMO, 161
 GOS_MUTEX_NO_TMO, 161
 gos_mutexInit, 161
 gos_mutexLock, 162
 gos_mutexState_t, 161
 gos_mutexUnlock, 163
 gos_mutex_t, 8
 gos_mutexInit
 gos_mutex.c, 156
 gos_mutex.h, 161
 gos_mutexLock
 gos_mutex.c, 157
 gos_mutex.h, 162
 gos_mutexState_t
 gos_mutex.h, 161
 gos_mutexUnlock
 gos_mutex.c, 158
 gos_mutex.h, 163
 gos_port.h, 164
 gos_ported_doContextSwitch, 166
 gos_ported_enableFaultHandlers, 166
 gos_ported_handleSVC, 167
 gos_ported_handleSVCMain, 167
 gos_ported_kernelStartInit, 167
 gos_ported_pendSVHandler, 168
 gos_ported_procReset, 168
 gos_ported_reschedule, 168
 gos_ported_svcHandler, 169
 gos_ported_svcHandlerMain, 169
 gos_ported_sysTickInterrupt, 169
 gos_ported_doContextSwitch
 gos_port.h, 166
 gos_ported_enableFaultHandlers
 gos_port.h, 166
 gos_ported_handleSVC
 gos_port.h, 167
 gos_ported_handleSVCMain
 gos_port.h, 167
 gos_ported_kernelStartInit
 gos_port.h, 167
 gos_ported_pendSVHandler
 gos_port.h, 168
 gos_ported_procReset
 gos_port.h, 168

gos_ported_reschedule
 gos_port.h, 168
gos_ported_svcHandler
 gos_port.h, 169
gos_ported_svcHandlerMain
 gos_port.h, 169
gos_ported_sysTickInterrupt
 gos_port.h, 169
gos_printStartupLogo
 gos_error.h, 55
gos_queue.c, 169
 DUMP_SEPARATOR, 171
 gos_queueCreate, 171
 gos_queueDump, 172
 gos_queueGet, 172
 gos_queueGetElementNumber, 173
 gos_queueGetName, 174
 gos_queueInit, 174
 gos_queuePeek, 174
 gos_queuePut, 175
 gos_queueRegisterEmptyHook, 176
 gos_queueRegisterFullHook, 176
 gos_queueReset, 177
 queueEmptyHook, 177
 queueFullHook, 177
 queueMutex, 177
 queues, 178
 readCounters, 178
 writeCounters, 178
gos_queue.h, 178
 GOS_DEFAULT_QUEUE_ID, 180
 GOS_INVALID_QUEUE_ID, 180
 gos_queueByte_t, 181
 gos_queueCreate, 181
 gos_queueDump, 182
 gos_queueEmptyHook, 181
 gos_queueFullHook, 181
 gos_queueGet, 182
 gos_queueGetElementNumber, 183
 gos_queueGetName, 184
 gos_queueInit, 184
 gos_queuePeek, 185
 gos_queuePut, 185
 gos_queueRegisterEmptyHook, 186
 gos_queueRegisterFullHook, 187
 gos_queueReset, 187
gos_queue_t, 9
gos_queueByte_t
 gos_queue.h, 181
gos_queueCreate
 gos_queue.c, 171
 gos_queue.h, 181
gos_queueDescriptor_t, 9
gos_queueDump
 gos_queue.c, 172
 gos_queue.h, 182
gos_queueElement_t, 10
gos_queueEmptyHook
 gos_queue.h, 181
 gos_queueFullHook
 gos_queue.h, 181
 gos_queueGet
 gos_queue.c, 172
 gos_queue.h, 182
 gos_queueGetElementNumber
 gos_queue.c, 173
 gos_queue.h, 183
 gos_queueGetName
 gos_queue.c, 174
 gos_queue.h, 184
 gos_queueInit
 gos_queue.c, 174
 gos_queue.h, 184
 gos_queuePeek
 gos_queue.c, 174
 gos_queue.h, 185
 gos_queuePut
 gos_queue.c, 175
 gos_queue.h, 185
 gos_queueRegisterEmptyHook
 gos_queue.c, 176
 gos_queue.h, 186
 gos_queueRegisterFullHook
 gos_queue.c, 176
 gos_queue.h, 187
 gos_queueReset
 gos_queue.c, 177
 gos_queue.h, 187
gos_result_t
 gos_kernel.h, 103
gos_runTimeAddMicroseconds
 gos_time.c, 278
 gos_time.h, 290
gos_runTimeAddMilliseconds
 gos_time.c, 279
 gos_time.h, 291
gos_runTimeAddSeconds
 gos_time.c, 279
 gos_time.h, 291
gos_runTimeGet
 gos_time.c, 280
 gos_time.h, 292
gos_shell.c, 188
 actualCommand, 195
 commandBuffer, 195
 commandBufferIndex, 196
 commandParams, 196
 GOS_SHELL_DAEMON_POLL_TIME_MS, 189
 GOS_SHELL_DISPLAY_TEXT, 189
 gos_shellCommandHandler, 190
 gos_shellDaemonTask, 190
 gos_shellEchoOff, 191
 gos_shellEchoOn, 191
 gos_shellInit, 191
 gos_shellRegisterCommand, 192
 gos_shellRegisterCommands, 192

gos_shellResume, 194
 gos_shellSuspend, 195
 shellCommand, 196
 shellCommands, 196
 shellDaemonTaskDesc, 196
 shellDaemonTaskId, 196
 useEcho, 196
 gos_shell.h, 197
 gos_shellEchoOff, 199
 gos_shellEchoOn, 199
 gos_shellFunction, 199
 gos_shellInit, 200
 gos_shellRegisterCommand, 200
 gos_shellRegisterCommands, 201
 gos_shellResume, 201
 gos_shellSuspend, 202
 gos_shell_driver.c, 203
 formattedBuffer, 205
 gos_shellDriverReceiveChar, 204
 gos_shellDriverTransmitString, 204
 gos_shell_driver.h, 205
 gos_shellDriverReceiveChar, 207
 gos_shellDriverReceiveChar_t, 207
 gos_shellDriverTransmitString, 208
 gos_shellDriverTransmitString_t, 207
 gos_shellCommand_t, 10
 gos_shellCommandHandler
 gos_shell.c, 190
 gos_shellDaemonTask
 gos_shell.c, 190
 gos_shellDriverReceiveChar
 gos_shell_driver.c, 204
 gos_shell_driver.h, 207
 gos_shellDriverReceiveChar_t
 gos_shell_driver.h, 207
 gos_shellDriverTransmitString
 gos_shell_driver.c, 204
 gos_shell_driver.h, 208
 gos_shellDriverTransmitString_t
 gos_shell_driver.h, 207
 gos_shellEchoOff
 gos_shell.c, 191
 gos_shell.h, 199
 gos_shellEchoOn
 gos_shell.c, 191
 gos_shell.h, 199
 gos_shellFunction
 gos_shell.h, 199
 gos_shellInit
 gos_shell.c, 191
 gos_shell.h, 200
 gos_shellRegisterCommand
 gos_shell.c, 192
 gos_shell.h, 200
 gos_shellRegisterCommands
 gos_shell.c, 192
 gos_shell.h, 201
 gos_shellResume
 gos_shell.c, 194
 gos_shell.h, 201
 gos_shellSuspend
 gos_shell.c, 195
 gos_shell.h, 202
 gos_signal.c, 208
 gos_signalCreate, 210
 gos_signalDaemonTask, 210
 gos_signallInit, 211
 gos_signallInvoke, 211
 gos_signalSubscribe, 212
 signalArray, 213
 signalDaemonTaskDescriptor, 213
 signallInvokeTrigger, 213
 gos_signal.h, 213
 gos_signalCreate, 215
 gos_signallInit, 216
 gos_signallInvoke, 216
 gos_signalSubscribe, 217
 gos_signalCreate
 gos_signal.c, 210
 gos_signal.h, 215
 gos_signalDaemonTask
 gos_signal.c, 210
 gos_signalDescriptor_t, 10
 gos_signallInit
 gos_signal.c, 211
 gos_signal.h, 216
 gos_signallInvoke
 gos_signal.c, 211
 gos_signal.h, 216
 gos_signallInvokeDescriptor, 11
 gos_signalSubscribe
 gos_signal.c, 212
 gos_signal.h, 217
 gos_sysmon.c, 218
 __attribute__, 223
 cpuMessage, 232
 gos_sysmonCheckMessage, 225
 gos_sysmonDaemonTask, 226
 gos_sysmonGetLutIndex, 226
 gos_sysmonHandleCpuUsageGet, 226
 gos_sysmonHandlePingRequest, 227
 gos_sysmonHandleResetRequest, 227
 gos_sysmonHandleSysRuntimeGet, 228
 gos_sysmonHandleSystimeSet, 228
 gos_sysmonHandleTaskDataGet, 229
 gos_sysmonHandleTaskModification, 229
 gos_sysmonHandleTaskVariableDataGet, 230
 gos_sysmonInit, 230
 gos_sysmonMessageEnum_t, 221
 gos_sysmonMessageHandler_t, 221
 gos_sysmonMessageId_t, 222
 gos_sysmonMessagePv_t, 222
 gos_sysmonMessageResult_t, 223
 gos_sysmonRegisterUserMessage, 231
 gos_sysmonSendResponse, 231
 gos_sysmonTaskModifyType_t, 223

pingMessage, 232
RECEIVE_BUFFER_SIZE, 221
receiveBuffer, 233
sysRuntimeGetResultMessage, 233
sysTimeSetMessage, 233
sysTimeSetResultMessage, 233
sysmonDaemonTaskDesc, 233
sysmonLut, 233
taskDataGetMsg, 233
taskDataMsg, 233
taskDesc, 234
taskModifyMessage, 234
taskModifyResultMessage, 234
taskVariableDataMsg, 234
userMessages, 234
gos_sysmon.h, 234
gos_sysmonInit, 236
gos_sysmonMessageReceivedCallback, 236
gos_sysmonRegisterUserMessage, 237
gos_sysmon_app_config.h, 237
ARM_CORTEX_M4, 239
CFG_GCP_CHANNELS_MAX_NUMBER, 239
CFG_IDLE_TASK_STACK_SIZE, 239
CFG_MESSAGE_MAX_ADDRESSEES, 239
CFG_MESSAGE_MAX_LENGTH, 239
CFG_MESSAGE_MAX_NUMBER, 239
CFG_MESSAGE_MAX_WAITER_IDS, 239
CFG_MESSAGE_MAX_WAITERS, 240
CFG_PROC_IDLE_PRIO, 240
CFG_PROC_MAX_NAME_LENGTH, 240
CFG_PROC_MAX_NUMBER, 240
CFG_PROC_MAX_PRIO_LEVELS, 240
CFG_PROC_USE_SERVICE, 240
CFG_QUEUE_MAX_ELEMENTS, 240
CFG_QUEUE_MAX_LENGTH, 240
CFG_QUEUE_MAX_NAME_LENGTH, 240
CFG_QUEUE_MAX_NUMBER, 241
CFG_QUEUE_USE_NAME, 241
CFG_RESET_ON_ERROR, 241
CFG_RESET_ON_ERROR_DELAY_MS, 241
CFG_SCHED_COOPERATIVE, 241
CFG_SHELL_COMMAND_BUFFER_SIZE, 241
CFG_SHELL_MAX_COMMAND_LENGTH, 241
CFG_SHELL_MAX_COMMAND_NUMBER, 241
CFG_SHELL_MAX_PARAMS_LENGTH, 241
CFG_SHELL_USE_SERVICE, 242
CFG_SIGNAL_MAX_NUMBER, 242
CFG_SIGNAL_MAX_SUBSCRIBERS, 242
CFG_SYSMON_GCP_CHANNEL_NUM, 242
CFG_SYSMON_MAX_USER_MESSAGES, 242
CFG_SYSMON_USE_SERVICE, 242
CFG_SYSTEM_TASK_STACK_SIZE, 242
CFG_TARGET_CPU, 242
CFG_TASK_MAX_NAME_LENGTH, 242
CFG_TASK_MAX_NUMBER, 243
CFG_TASK_MAX_STACK_SIZE, 243
CFG_TASK_MESSAGE_DAEMON_PRIO, 243
CFG_TASK_MESSAGE_DAEMON_STACK, 243
CFG_TASK_MIN_STACK_SIZE, 243
CFG_TASK_PROC_DAEMON_PRIO, 243
CFG_TASK_PROC_DAEMON_STACK, 243
CFG_TASK_SHELL_DAEMON_PRIO, 243
CFG_TASK_SHELL_DAEMON_STACK, 243
CFG_TASK_SIGNAL_DAEMON_PRIO, 244
CFG_TASK_SIGNAL_DAEMON_STACK, 244
CFG_TASK_SYS_PRIO, 244
CFG_TASK_SYSMON_DAEMON_PRIO, 244
CFG_TASK_SYSMON_DAEMON_STACK, 244
CFG_TASK_TIME_DAEMON_PRIO, 244
CFG_TASK_TIME_DAEMON_STACK, 244
CFG_TASK_TRACE_DAEMON_PRIO, 244
CFG_TASK_TRACE_DAEMON_STACK, 244
CFG_TRACE_MAX_LENGTH, 245
CFG_USE_PRIO_INHERITANCE, 245
GOS_CFG_OVERCONFIG, 245
gos_sysmon_driver.c, 245
gos_sysmonDriverReceive, 246
gos_sysmonDriverTransmit, 247
gos_sysmon_driver.h, 247
gos_sysmonDriverReceive, 249
gos_sysmonDriverReceive_t, 249
gos_sysmonDriverTransmit, 250
gos_sysmonDriverTransmit_t, 249
gos_sysmonCheckMessage
 gos_sysmon.c, 225
gos_sysmonDaemonTask
 gos_sysmon.c, 226
gos_sysmonDriverReceive
 gos_sysmon_driver.c, 246
 gos_sysmon_driver.h, 249
gos_sysmonDriverReceive_t
 gos_sysmon_driver.h, 249
gos_sysmonDriverTransmit
 gos_sysmon_driver.c, 247
 gos_sysmon_driver.h, 250
gos_sysmonDriverTransmit_t
 gos_sysmon_driver.h, 249
gos_sysmonGetLutIndex
 gos_sysmon.c, 226
gos_sysmonHandleCpuUsageGet
 gos_sysmon.c, 226
gos_sysmonHandlePingRequest
 gos_sysmon.c, 227
gos_sysmonHandleResetRequest
 gos_sysmon.c, 227
gos_sysmonHandleSysRuntimeGet
 gos_sysmon.c, 228
gos_sysmonHandleSystimeSet
 gos_sysmon.c, 228
gos_sysmonHandleTaskDataGet
 gos_sysmon.c, 229
gos_sysmonHandleTaskModification
 gos_sysmon.c, 229
gos_sysmonHandleTaskVariableDataGet
 gos_sysmon.c, 230
gos_sysmonInit

gos_sysmon.c, 230
gos_sysmon.h, 236
gos_sysmonLut_t, 11
gos_sysmonMessageEnum_t
gos_sysmon.c, 221
gos_sysmonMessageHandler_t
gos_sysmon.c, 221
gos_sysmonMessageId_t
gos_sysmon.c, 222
gos_sysmonMessagePv_t
gos_sysmon.c, 222
gos_sysmonMessageReceivedCallback
gos_sysmon.h, 236
gos_sysmonMessageResult_t
gos_sysmon.c, 223
gos_sysmonRegisterUserMessage
gos_sysmon.c, 231
gos_sysmon.h, 237
gos_sysmonSendResponse
gos_sysmon.c, 231
gos_sysmonTaskModifyType_t
gos_sysmon.c, 223
gos_sysmonUserMessageDescriptor_t, 12
gos_systemTask
gos.c, 18
gos_task.c, 250
gos_idleTask, 253
gos_taskAddPrivilege, 254
gos_taskBlock, 254
gos_taskCheckDescriptor, 255
gos_taskDelete, 257
gos_taskGetCurrentId, 258
gos_taskGetData, 259
gos_taskGetDataByIndex, 260
gos_taskGetId, 261
gos_taskGetName, 261
gos_taskGetNumber, 262
gos_taskGetOriginalPriority, 262
gos_taskGetPriority, 262
gos_taskGetPrivileges, 264
gos_taskRegister, 265
gos_taskRegisterTasks, 266
gos_taskRemovePrivilege, 267
gos_taskResume, 267
gos_taskSetOriginalPriority, 268
gos_taskSetPriority, 269
gos_taskSetPrivileges, 270
gos_taskSleep, 271
gos_taskSuspend, 272
gos_taskUnblock, 273
gos_taskWakeUp, 274
gos_taskYield, 275
kernelIdleHookFunction, 275
kernelTaskDeleteSignal, 275
taskDescriptors, 275
gos_taskAddPrivilege
gos_kernel.h, 118
gos_task.c, 254
gos_taskBlock
gos_kernel.h, 118
gos_task.c, 254
gos_taskCheckDescriptor
gos_task.c, 255
gos_taskDelete
gos_kernel.h, 119
gos_task.c, 257
gos_taskGetCurrentId
gos_kernel.h, 120
gos_task.c, 258
gos_taskGetData
gos_kernel.h, 121
gos_task.c, 259
gos_taskGetDataByIndex
gos_kernel.h, 122
gos_task.c, 260
gos_taskGetId
gos_kernel.h, 122
gos_task.c, 261
gos_taskGetName
gos_kernel.h, 123
gos_task.c, 261
gos_taskGetNumber
gos_kernel.h, 123
gos_task.c, 262
gos_taskGetOriginalPriority
gos_kernel.h, 124
gos_task.c, 262
gos_taskGetPriority
gos_kernel.h, 124
gos_task.c, 262
gos_taskGetPrivileges
gos_kernel.h, 125
gos_task.c, 264
gos_taskPrivilegeLevel_t
gos_kernel.h, 103
gos_taskRegister
gos_kernel.h, 126
gos_task.c, 265
gos_taskRegisterTasks
gos_kernel.h, 127
gos_task.c, 266
gos_taskRemovePrivilege
gos_kernel.h, 128
gos_task.c, 267
gos_taskResume
gos_kernel.h, 128
gos_task.c, 267
gos_taskSetOriginalPriority
gos_kernel.h, 129
gos_task.c, 268
gos_taskSetPriority
gos_kernel.h, 130
gos_task.c, 269
gos_taskSetPrivileges
gos_kernel.h, 131
gos_task.c, 270

gos_taskSleep
 gos_kernel.h, 132
 gos_task.c, 271
gos_taskState_t
 gos_kernel.h, 103
gos_taskSubscribeDeleteSignal
 gos_kernel.h, 133
gos_taskSuspend
 gos_kernel.h, 133
 gos_task.c, 272
gos_taskUnblock
 gos_kernel.h, 134
 gos_task.c, 273
gos_taskWakeUp
 gos_kernel.h, 135
 gos_task.c, 274
gos_taskYield
 gos_kernel.h, 136
 gos_task.c, 275
gos_time.c, 276
 dayLookupTable, 286
 gos_runTimeAddMicroseconds, 278
 gos_runTimeAddMilliseconds, 279
 gos_runTimeAddSeconds, 279
 gos_runTimeGet, 280
 gos_timeAddMilliseconds, 280
 gos_timeAddSeconds, 282
 gos_timeCompare, 282
 gos_timeDaemonTask, 283
 gos_timeGet, 283
 gos_timeIncreaseSystemTime, 284
 gos_timeInit, 284
 gos_timeSet, 285
 systemRunTime, 286
 systemTime, 286
 TIME_DEFAULT_DAY, 278
 TIME_DEFAULT_MONTH, 278
 TIME_DEFAULT_YEAR, 278
 TIME_SLEEP_TIME_MS, 278
timeDaemonTaskDesc, 286
timeDaemonTaskId, 286
timeSignalId, 287
gos_time.h, 287
 __attribute__, 290
 gos_runTimeAddMicroseconds, 290
 gos_runTimeAddMilliseconds, 291
 gos_runTimeAddSeconds, 291
 gos_runTimeGet, 292
 gos_timeAddMilliseconds, 292
 gos_timeAddSeconds, 294
 gos_timeCompare, 294
 gos_timeComprareResult_t, 289
 gos_timeElapsedSenderId_t, 289
 gos_timeGet, 295
 gos_timeIncreaseSystemTime, 295
 gos_timeInit, 297
 gos_timeMonthEnum_t, 289
 gos_timeSet, 297
gos_timeAddMilliseconds
 gos_time.c, 280
 gos_time.h, 292
gos_timeAddSeconds
 gos_time.c, 282
 gos_time.h, 294
gos_timeCompare
 gos_time.c, 282
 gos_time.h, 294
gos_timeComprareResult_t
 gos_time.h, 289
gos_timeDaemonTask
 gos_time.c, 283
gos_timeElapsedSenderId_t
 gos_time.h, 289
gos_timeGet
 gos_time.c, 283
 gos_time.h, 295
gos_timeIncreaseSystemTime
 gos_time.c, 284
 gos_time.h, 295
gos_timeInit
 gos_time.c, 284
 gos_time.h, 297
gos_timeMonthEnum_t
 gos_time.h, 289
gos_timeSet
 gos_time.c, 285
 gos_time.h, 297
gos_timer_driver.h, 298
 gos_timerDriverSysTimerGet, 300
 gos_timerDriverSysTimerGetVal_t, 300
gos_timerDriverSysTimerGet
 gos_timer_driver.h, 300
gos_timerDriverSysTimerGetVal_t
 gos_timer_driver.h, 300
gos_trace.c, 301
 formattedBuffer, 306
 GOS_TRACE_MUTEX_TMO_MS, 302
 GOS_TRACE_QUEUE_TMO_MS, 302
 GOS_TRACE_TIMESTAMP_FORMAT, 302
 GOS_TRACE_TIMESTAMP_LENGTH, 302
 gos_traceDaemonTask, 303
 gos_tracelnit, 303
 gos_traceTrace, 303
 gos_traceTraceFormatted, 304
 gos_traceTraceFormattedUnsafe, 305
 timeStampBuffer, 306
 traceDaemonTaskDesc, 306
 traceLine, 307
 traceMutex, 307
 traceQueue, 307
gos_trace.h, 307
 gos_tracelnit, 310
 gos_traceTrace, 310
 gos_traceTraceFormatted, 311
 gos_traceTraceFormattedUnsafe, 312
 gos_trace_driver.c, 313

gos_traceDriverTransmitString, 315
 gos_traceDriverTransmitString_Unsafe, 316
 gos_trace_driver.h, 317
 gos_traceDriverTransmitString, 319
 gos_traceDriverTransmitString_Unsafe, 320
 gos_traceDriverTransmitString_Unsafe_t, 319
 gos_traceDriverTransmitString_t, 319
 gos_traceDaemonTask
 gos_trace.c, 303
 gos_traceDriverTransmitString
 gos_trace_driver.c, 315
 gos_trace_driver.h, 319
 gos_traceDriverTransmitString_Unsafe
 gos_trace_driver.c, 316
 gos_trace_driver.h, 320
 gos_traceDriverTransmitString_Unsafe_t
 gos_trace_driver.h, 319
 gos_traceDriverTransmitString_t
 gos_trace_driver.h, 319
 gos_tracelInit
 gos_trace.c, 303
 gos_trace.h, 310
 gos_traceResultToString
 gos_error.c, 49
 gos_traceTrace
 gos_trace.c, 303
 gos_trace.h, 310
 gos_traceTraceFormatted
 gos_trace.c, 304
 gos_trace.h, 311
 gos_traceTraceFormattedUnsafe
 gos_trace.c, 305
 gos_trace.h, 312
 gos_trigger.c, 321
 gos_triggerDecrement, 323
 gos_triggerIncrement, 323
 gos_triggerInit, 324
 gos_triggerReset, 325
 gos_triggerWait, 325
 gos_trigger.h, 326
 GOS_TRIGGER_ENDLESS_TMO, 328
 GOS_TRIGGER_NO_TMO, 328
 gos_triggerDecrement, 328
 gos_triggerIncrement, 329
 gos_triggerInit, 330
 gos_triggerReset, 330
 gos_triggerWait, 331
 gos_trigger_t, 12
 gos_triggerDecrement
 gos_trigger.c, 323
 gos_trigger.h, 328
 gos_triggerIncrement
 gos_trigger.c, 323
 gos_trigger.h, 329
 gos_triggerInit
 gos_trigger.c, 324
 gos_trigger.h, 330
 gos_triggerReset

gos_trigger.c, 325
 gos_trigger.h, 330
 gos_triggerWait
 gos_trigger.c, 325
 gos_trigger.h, 331

ICSR
 gos_kernel.c, 73

inlSr
 gos_kernel.c, 90

initError
 gos.c, 19

initializers
 gos.c, 19

isKernelRunning
 gos_kernel.c, 90

kernelDumpReadySignal
 gos_kernel.c, 90

kernelDumpSignal
 gos_kernel.c, 90

kernelIdleHookFunction
 gos_task.c, 275

kernelPrivilegedHookFunction
 gos_kernel.c, 90

kernelSwapHookFunction
 gos_kernel.c, 90

kernelSysTickHookFunction
 gos_kernel.c, 90

kernelTaskDeleteSignal
 gos_task.c, 275

MAIN_STACK
 gos_kernel.h, 102

MAX_CPU_DUMP_SEPARATOR
 gos_kernel.c, 73

MUTEX_LOCK_SLEEP_MS
 gos_mutex.c, 156

messageArray
 gos_message.c, 141

messageDaemonTaskDesc
 gos_message.c, 141

messageDaemonTaskId
 gos_message.c, 141

messageMutex
 gos_message.c, 141

messageWaiterArray
 gos_message.c, 142

monitoringTime
 gos_kernel.c, 90

NULL
 gos_kernel.h, 102

nextMessageIndex
 gos_message.c, 142

nextWaiterIndex
 gos_message.c, 142

pingMessage

gos_sysmon.c, 232
previousTick
 gos_kernel.c, 90
primask
 gos_kernel.c, 91
privilegedModeSetRequired
 gos_kernel.c, 91

queueEmptyHook
 gos_queue.c, 177
queueFullHook
 gos_queue.c, 177
queueMutex
 gos_queue.c, 177
queues
 gos_queue.c, 178

RAM_SIZE
 gos_kernel.h, 102
RAM_START
 gos_kernel.h, 102
RECEIVE_BUFFER_SIZE
 gos_sysmon.c, 221
RESULT_STRING_ERROR
 gos_error.c, 46
RESULT_STRING_SUCCESS
 gos_error.c, 46
RESULT_STRING_UNKNOWN
 gos_error.c, 46
readCounters
 gos_queue.c, 178
receiveBuffer
 gos_sysmon.c, 233
resetRequired
 gos_kernel.c, 91

SEPARATOR_LINE
 gos_error.c, 46
SHCSR
 gos_kernel.c, 73
STACK_STATS_SEPARATOR
 gos_kernel.c, 74
schedDisableCntr
 gos_kernel.c, 91
shellCommand
 gos_shell.c, 196
shellCommands
 gos_shell.c, 196
shellDaemonTaskDesc
 gos_shell.c, 196
shellDaemonTaskId
 gos_shell.c, 196
signalArray
 gos_signal.c, 213
signalDaemonTaskDescriptor
 gos_signal.c, 213
signalInvokeTrigger
 gos_signal.c, 213
sysRuntimeGetResultMessage
 gos_sysmon.c, 233
sysTicks
 gos_kernel.c, 91
sysTimeSetMessage
 gos_sysmon.c, 233
sysTimeSetResultMessage
 gos_sysmon.c, 233
sysTimerValue
 gos_kernel.c, 91
sysmonDaemonTaskDesc
 gos_sysmon.c, 233
sysmonLut
 gos_sysmon.c, 233
systemRunTime
 gos_time.c, 286
systemTaskDesc
 gos.c, 19
systemTaskId
 gos.c, 20
systemTime
 gos_time.c, 286

TASK_DUMP_SEPARATOR
 gos_kernel.c, 74
TIME_DEFAULT_DAY
 gos_time.c, 278
TIME_DEFAULT_MONTH
 gos_time.c, 278
TIME_DEFAULT_YEAR
 gos_time.c, 278
TIME_SLEEP_TIME_MS
 gos_time.c, 278
TO_BINARY
 gos_kernel.c, 74
taskDataGetMsg
 gos_sysmon.c, 233
taskDataMsg
 gos_sysmon.c, 233
taskDesc
 gos_sysmon.c, 234
taskDescriptors
 gos_task.c, 275
taskModifyMessage
 gos_sysmon.c, 234
taskModifyResultMessage
 gos_sysmon.c, 234
taskVariableDataMsg
 gos_sysmon.c, 234
timeDaemonTaskDesc
 gos_time.c, 286
timeDaemonTaskId
 gos_time.c, 286
timeSignalId
 gos_time.c, 287
timeStampBuffer
 gos_trace.c, 306
traceDaemonTaskDesc
 gos_trace.c, 306
traceLine

gos_trace.c, 307
traceMutex
 gos_trace.c, 307
traceQueue
 gos_trace.c, 307

useEcho
 gos_shell.c, 196
userMessages
 gos_sysmon.c, 234

writeCounters
 gos_queue.c, 178