Crystal Image through
Imaging Innovation

PIXELPLUS

SURROUND VIEW MONITORING SYSTEM

PI5008K OS Adaptation Layer

User Guide

Rev 0.2

Last Update : 2018.11.17

6th Floor, 105, Gwanggyo-ro, Yeongtong-gu,
Suwong-si, Gyeonggi-do, 16229, Korea
Tel : +82-31-888-5300, FAX : +82-31-888-5399

Contents

1. OS Adaptation Layers

1.1. Introduction

PI5008K SDK is based on FreeRTOS 8.0.0. Some FreeRTOS APIs are wrapped and others are not. Wrapped OS APIs are explained in this guide.

1.2. OS Wrapper API

1.2.1. OSAL_register_isr

Prototype	RESULT APIENTRY OSAL_register_isr(sint32 IN vector, sys_os_isr_t IN isr, sys_os_isr_t* OUT old)
Description	Use this function to register interrupt handler
Argument	vector : target interrupt vector isr : interrupt handler old : old interrupt handler
Return value	eSuccess
Example	sys_os_isr_t *old = NULL OSAL_register_isr(IRQ_VPU0_VECTOR, ISR_VPU, old);

1.2.2. OSAL_register_idle_task

This function is not used

1.2.3. OSAL_system_panic;

Prototype	OSAL_system_panic(uint32 IN err);
Description	Output error number through UART and wait forever
Argument	err : Error number
Return value	None
Remark	

1.2.4. OSAL_global_int_ctl

This function is not used.

1.2.5. OSAL_current

This function is not used

1.2.6. OSAL_ms2ticks

Prototype	OSAL_ms2ticks(uint32 IN timeout);
Description	Convert time to tick number
Argument	timeout : time (unit ms)
Return value	Number of tick interrupt
Remark	

1.2.7. OSAL_tick2ms

Prototype	OSAL_tick2ms(uint32 IN tick)
Description	Convert tick number to time
Argument	Number of tick interrupt
Return value	Time which corresponds to the number of tick number
Remark	

1.2.8. OSAL_sleep

Prototype	OSAL_sleep(uint32 IN ms)
Description	Places the task that calls OSAL_sleep() into the Blocked state for a fixed time.
Argument	ms : Time to be blocked
Return value	None
Remark	Refer to vTaskDelay

1.2.9. OSAL_delay

Prototype	OSAL_delay(uint32 IN ms, bool IN isFromISR)
Description	Wait for a fixed time without entering blocking state.
Argument	ms : Time to be waited isFromISR : show whether to be called from interrupt handler
Return value	None
Remark	Task that calls OSAL_delay is not blocked

1.2.10. OSAL_malloc

Prototype	void * OSAL_malloc(uint32 IN size);
Description	Allocates requested size of bytes and returns a pointer first byte of allocated space
Argument	Size : Memory size to be allocated
Return value	a pointer first byte of allocated space
Remark	SVM SDK uses Heap_4.c

1.2.11. OSAL_free

Prototype	void APIENTRY OSAL_free(void* IN ptr);
Description	deallocate the previously allocated space
paramter	ptr : pointer to be deallocated.
Return value	None
Remark	SVM SDK uses Heap_4.c.

1.2.12. OSAL_get_pid_current

Prototype	void* APIENTRY OSAL_get_pid_current(void);
Description	Returns the handle of the task that is in the Running state
Argument	None
Return value	The handle of the task that called OSAL_get_pid_current ().
Remark	Refer to xTaskGetCurrentTaskHandle().

1.2.13. OSAL_create_semaphore

Prototype	OSAL_create_semaphore(sys_os_semaphore_t* OUT sem, uint32 IN num, CONST void* IN param);
Description	1. Creates a counting semaphore 2. If creation succeed, assigns name ‘queue’ to the sema, and adds it to the queue registry.
Argument	sem : a handle by which the semaphore can be referenced. num : The count value assigned to the semaphore when it is created. param : Maximum count value that can be reached. If param is 0, 65535 is assigned.
Return value	eERROR_FAILURE: : Returned if the semaphore cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the semaphore data structures eSuccess
Remark	Used only inside OSAL_create_mutex() Refer to xSemaphoreCreateCounting()

1.2.14. OSAL_destroy_semaphore

Prototype	OSAL_destroy_semaphore(sys_os_semaphore_t *sem);
Description	Deletes a semaphore that was previously created
Parameter	sem : The handle of the semaphore being deleted.
Return value	eERROR_INVALID_ARGUMENT : The handle of the sema is nulll eSuccess
Remark	Used only inside OSAL_destroy_mutex(); Refer to vSemaphoreDelete()/vQueueDelete().

1.2.15. OSAL_pend_semaphore

Prototype	OSAL_pend_semaphore(sys_os_semaphore_t* OUT sem, uint32 IN timeout);
Description	Takes’ (or obtains) a semaphore that has previously been created

Parameter	sem : The handle of the semaphore being taken timeout : The maximum amount of time the task should remain in the blocked state to wait for the semaphore to become available, if the semaphore is not available immediately. If timeout is SYS_OS_SUSPEND, the task waits indefinitely.
Return value	eERROR_FAILURE: : Returned if the call did not successfully obtain the semaphore eSuccess
Remark	Used only inside OSAL_wait_for_mutex() Refer to xSemaphoreTake()

1.2.16. OSAL_post_semaphore

Prototype	RESULT APIENTRY OSAL_post_semaphore(sys_os_semaphore_t* OUT sem);
Description	Gives' (or releases) a semaphore that has previously been created and has also been successfully 'taken'.
Parameter	sem : The handle of the semaphore being given.
Return value	eERROR_FAILURE: The semaphore 'post' operation was not successful. eSuccess
Remark	Used only inside OSAL_release_mutex(); Refer to xSemaphoreGive()

1.2.17. OSAL_post_semaphore_in_isr

Prototype	RESULT APIENTRY OSAL_post_semaphore_in_isr(sys_os_semaphore_t* OUT sem);
Description	A version of OSAL_post_semaphore that can be used in an ISR.
Parameter	sem : The handle of the semaphore being given.
Return value	eERROR_FAILURE: The semaphore 'post' operation was not successful eSuccess
Remark	Refer to xSemaphoreGiveISR()

1.2.18. OSAL_post_queue_from_lisr

This API is not used.

1.2.19. OSAL_create_mutex

Prototype	RESULT APIENTRY OSAL_create_mutex(sys_os_mutex_t* OUT mutex)
Description	Create counting semaphore with max and initial value is 1.
Parameter	mutex : pointer to the mutex structure including semaphore handle.
Return value	eERROR_FAILURE: : Returned if the semaphore cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the semaphore data structures. eSuccess
Remark	Refer to xSemaphoreCreateCounting()

1.2.20. OSAL_destroy_mutex

Prototype	RESULT APIENTRY OSAL_destroy_mutex(sys_os_mutex_t* IN mutex))
Description	Deletes a mutex that was previously created
Parameter	mutex : The handle of the semaphore being deleted.
Return value	eERROR_INVALID_ARGUMENT : The handle of the mutex is null. eSuccess
Remark	Refer to vSemaphoreDelete()/vQueueDelete()

1.2.21. OSAL_wait_for_mutex

Prototype	OSAL_wait_for_mutex(sys_os_mutex_t* OUT mutex)
Description	Takes’(or obtains) a mutex that has previously been created
Parameter	mutex : The handle of the mutex being taken
Return value	eERROR_FAILURE: : Returned if the call did not successfully obtain the mutex

	eSuccess
Remark	This function waits indefinitely until mutex is taken

1.2.22. OSAL_release_mutex

Prototype	OSAL_release_mutex(sys_os_mutex_t* OUT mutex);
Description	Gives’(or releases) a mutex that has previously been created.
Parameter	mutex : The handle of the mutex being given.
Return value	eERROR_FAILURE: The mutex ‘release’ operation was not successful. eSuccess
Remark	Refer to OSAL_post_semaphore();

1.2.23. OSAL_create_queue

This API is not used.

Prototype	RESULT APIENTRY OSAL_create_queue(sys_os_queue_t* OUT queue);
Description	Creates a new queue and returns a handle by which the queue can be referenced.
Parameter	queue->size : The handle of the mutex being given. oueue->obj : handle of message queue
Return value	eERROR_FAILURE: The queue cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the queue data structures and storage area. eSuccess
Remark	Not used

1.2.24. OSAL_destroy_queue

This API is not used

Prototype	RESULT APIENTRY OSAL_destroy_queue(sys_os_queue_t* OUT queue)

Description	Deletes a queue that was previously created.
Parameter	queue : structure to hold various information for message queue
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eSuccess
Remark	Not used

1.2.25. OSAL_pend_queue

This API is not used.

Prototype	RESULT APIENTRY OSAL_pend_queue(sys_os_queue_t* OUT queue);
Description	Receive (read) an item from a queue.
Parameter	queue : structure to hold various information for message queue queue->timeout: The maximum amount of time the task should remain in the Blocked state to wait for data to become available on the queue. If timeout is SYS_OS_SUSPEND, the task will wait indefinitely (without timing out)
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eERROR_FAILURE : Returned if data cannot be read from the queue because the queue is already empty. eSuccess
Remark	Not used

1.2.26. OSAL_post_queue

This API is not used.

Prototype	RESULT APIENTRY OSAL_post_queue(sys_os_queue_t* OUT queue);
Description	Sends (writes) an item to the front or the back of a queue
Parameter	queue : structure to hold various information for message queue queue->msg: data to be copied into the queue queue->timeout: The maximum amount of time the task should remain in the Blocked state to wait for space to become available on the queue.

Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eERROR_FAILURE : Returned if data cannot be read from the queue because the queue is already empty. eSuccess
Remark	Not used

1.2.27. OSAL_EVENTGROUP_CMD_Create

Prototype	EventGroupHandle_t * OSAL_EVENTGROUP_CMD_Create(void)
Description	Creates a new event group and returns a handle by which the created event group can be referenced.
Parameter	NULL
Return value	NULL The event group could not be created because there was insufficient FreeRTOS heap available. Any other value The event group was created and the value returned is the handle of the created event group
Remark	Refer to xEventGroupCreate()

1.2.28. OSAL_EVENTGROUP_CMD_Destroy

Prototype	RESULT APIENTRY OSAL_EVENTGROUP_CMD_Destroy(EventGroupHandle_t *phEventGroup)
Description	Delete an event group that was previously created
Parameter	phEventGroup : The event group to be deleted.
Return value	eERROR_INVALID_ARGUMENT: Input event group is null. eSuccess
Remark	Refer to vEventGroupDelete()

1.2.29. OSAL_EVENTGROUP_CMD_WaitBits

Prototype	EventBits_t APIENTRY OSAL_EVENTGROUP_CMD_WaitBits(EventGroupHandle_t

	*phEventGroup, const EventBits_t eventBits, const BaseType_t xClearOnExit, const BaseType_t xWaitForAllBits, const int timeOut);
Description	Read bits within an RTOS event group, optionally entering the Blocked state (with a timeout) to wait for a bit or group of bits to become set.
Parameter	phEventGroup : The event group in which the bits are being tested. eventBits : bitwise value that indicates the bit or bits to test inside the event group. xClearOnExit: If xClearOnExit is set to 1 then any bits set in the value passed as the uxBitsToWaitFor parameter will be cleared in the event group before xEventGroupWaitBits() returns if xEventGroupWaitBits() returns for any reason other than a timeout. xWaitForAllBits : xWaitForAllBits is used to create either a logical AND test (where all bits must be set) or a logical OR test (where one or more bits must be set) timeOut: The maximum amount of time (specified in ms) to wait for one/all (depending on the xWaitForAllBits value) of the bits specified by uxBitsToWaitFor to become set. If timeOut is SYS_OS_SUSPEND, the task calls this API waits indefinitely.
Return value	eERROR_INVALID_ARGUMENT: Input event group is null. eSuccess
Remark	Refer to xEventGroupWaitBits()

1.2.30. OSAL_EVENTGROUP_CMD_SetBits

Prototype	EventBits_t APIENTRY OSAL_EVENTGROUP_CMD_SetBits(EventGroupHandle_t *phEventGroup, const EventBits_t eventBits);
Description	Sets bits (flags) within an RTOS event group
Parameter	phEventGroup : The event group in which the bits are being tested. eventBits : A bitwise value that indicates the bit or bits to set in the event group
Return value	eERROR_INVALID_ARGUMENT: Input event group is null. Other value : The value of the bits in the event group at the time the call to OSAL_EVENTGROUP_CMD_SetBits() returned

Remark	Refer to xEventGroupSetBits()

1.2.31. OSAL_EVENTGROUP_CMD_SetBitsFromISR

Prototype	EventBits_t APIENTRY OSAL_EVENTGROUP_CMD_SetBitsFromISR(EventGroupHandle_t *phEventGroup, const EventBits_t eventBits, BaseType_t *p);
Description	Set bits (flags) within an event group. A version of OSAL_EVENTGROUP_CMD_SetBits that can be called from an interrupt service routine (ISR).
Parameter	phEventGroup : The event group in which the bits are being tested. eventBits : A bitwise value that indicates the bit or bits to set in the event group. eventBits : A bitwise value that indicates the bit or bits to set in the event group p: If OSAL_EVENTGROUP_CMD_SetBitsFromISR sets this value to pdTRUE, then a context switch should be performed before the interrupt is exited. This will ensure that the interrupt returns directly to the highest priority Ready state task.
Return value	eERROR_INVALID_ARGUMENT: Input event group is null. Other value : The value of the bits in the event group at the time the call to xEventGroupSetBits() returned.
Remark	Refer to xEventGroupSetBitsFromISR()

1.2.32. OSAL_QUEUE_CMD_Create

Prototype	QueueHandle_t * APIENTRY OSAL_QUEUE_CMD_Create(const int maxQueueCnt, const uint32 sizeQueue);
Description	Creates a new queue and returns a handle by which the queue can be referenced.
Parameter	maxQueueCnt : The maximum number of items that the queue being created can hold at any one time. sizeQueue : The size, in bytes, of each data item that can be stored in the queue

Return value	NULL	The queue cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the queue data structures and storage area.
	Any other value	The queue was created successfully. The returned value is a handle by which the created queue can be referenced.
Remark	Refer to xQueueCreate()	

1.2.33. OSAL_QUEUE_CMD_Destroy

Prototype	RESULT APIENTRY OSAL_QUEUE_CMD_Destroy(QueueHandle_t *phQueue);
Description	Deletes a queue that was previously created
Parameter	phQueue: The handle of the queue being deleted
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eSuccess
Remark	Refer to vQueueDelete()

1.2.34. OSAL_QUEUE_CMD_GetCount

Prototype	int APIENTRY OSAL_QUEUE_CMD_GetCount(QueueHandle_t *phQueue);
Description	Returns the number of items that are currently held in a queue
Parameter	phQueue: The handle of the queue being deleted
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. Else The number of items that are held in the queue being queried at the time that OSAL_QUEUE_CMD_GetCount is called
Remark	Refer to uxQueueMessagesWaiting()

1.2.35. OSAL_QUEUE_CMD_Receive

Prototype	RESULT APIENTRY OSAL_QUEUE_CMD_Receive(QueueHandle_t *phQueue, void *pvParameters, const int timeOut);
Description	Receive (read) an item from a queue.
Parameter	phQueue: The handle of the queue from which the data is being received (read). pvParameters: A pointer to the memory into which the received data will be copied timeOut: The maximum amount of time the task should remain in the Blocked state to wait for data to become available on the queue, should the queue already be empty. If xTicksToWait is zero, then OSAL_QUEUE_CMD_Receive will return immediately if the queue is already empty. The block time is specified in ms. Setting timeOUT to portMAX_DELAY will cause the task to wait indefinitely (without timing out) provided INCLUDE_vTaskSuspend is set to 1
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eERROR_FAILURE: Returned if data cannot be read from the queue because the queue is already empty eSuccess
Remark	Refer to xQueueReceiv()

1.2.36. OSAL_QUEUE_CMD_Send

Prototype	RESULT APIENTRY OSAL_QUEUE_CMD_Send(QueueHandle_t *phQueue, void *pvParameters, const int timeOut);
Description	Sends (writes) an item to the back of a queue.
Parameter	phQueue: The handle of the queue from which the data is being received (read). pvParameters: A pointer to the data to be copied into the queue. timeOut: The maximum amount of time the task should remain in the Blocked state to wait for data to become available on the queue, should

	the queue already be empty. If xTicksToWait is zero, then OSAL_QUEUE_CMD_Receive will return immediately if the queue is already empty. The block time is specified in ms Setting timeOUT to SYS_OS_SUSPEND will cause the task to wait indefinitely (without timing out) provided INCLUDE_vTaskSuspend is set to 1.
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eERROR_FAILURE: Returned if data could not be written to the queue because the queue was already full. eSuccess
Remark	

1.2.37. OSAL_QUEUE_CMD_ReceiveFromISR

Prototype	RESULT APIENTRY OSAL_QUEUE_CMD_ReceiveFromISR(QueueHandle_t *phQueue, void *pvParameters, BaseType_t *p);
Description	version of OSAL_QUEUE_CMD_Receive that can be called from an ISR. Unlike OSAL_QUEUE_CMD_Receive, OSAL_QUEUE_CMD_ReceiveFromISR does not permit a block time to be specified.
Parameter	phQueue: The handle of the queue from which the data is being received (read). pvParameters: A pointer to the memory into which the received data will be copied p: If OSAL_QUEUE_CMD_ReceiveFromISR sets this value to pdTRUE, then a context switch should be performed before the interrupt is exited. This will ensure that the interrupt returns directly to the highest priority Ready state task.
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eERROR_FAILURE: Returned if data cannot be read from the queue because the queue is already empty eSuccess
Remark	

1.2.38. OSAL_QUEUE_CMD_SendFromISR

Prototype	RESULT APIENTRY OSAL_QUEUE_CMD_SendFromISR(QueueHandle_t *phQueue, void *pvParameters, BaseType_t *p);
Description	Versions of the OSAL_QUEUE_CMD_Send API functions that can be called from an ISR. Unlike OSAL_QUEUE_CMD_Send, the ISR safe versions do not permit a block time to be specified.
Parameter	phQueue: The handle of the queue from which the data is being received (read). pvParameters: A pointer to the data to be copied into the queue. p: If OSAL_QUEUE_CMD_SendFromISR sets this value to pdTRUE, then a context switch should be performed before the interrupt is exited. This will ensure that the interrupt returns directly to the highest priority Ready state task.
Return value	eERROR_INVALID_ARGUMENT: Input queue is null. eERROR_FAILURE: Returned if data could not be written to the queue because the queue was already full eSuccess
Remark	

1.2.39. OSAL_create_thread

Prototype	RESULT APIENTRY OSAL_create_thread(sys_os_thread_t* OUT th);
Description	Creates a new instance of a task
Parameter	th->fn: Tasks are simply C functions that never exit and, as such, are normally implemented as an infinite loop. The fn parameter is simply a pointer to the function (in effect, just the function name) that implements the task. th->name : A descriptive name for the task th->stack_size: Each task has its own unique stack that is allocated by the kernel to the task when the task is created

	th->pvParameters: Task functions accept a parameter of type 'pointer to void' (void*). The value assigned to pvParameters will be the value passed into the task. th->prio: Defines the priority at which the task will execute. Priorities can be assigned from 0, which is the lowest priority, to (configMAX_PRIORITIES – 1), which is the highest priority. th->phTask : can be used to pass out a handle to the task being created. This handle can then be used to reference the task in API calls that, for example, change the task priority or delete the task.
Return value	eERROR_FAILURE: Indicates that the task could not be created eSuccess
Remark	Refer to xTaskCreate()

1.2.40. OSAL_destroy_thread

Prototype	RESULT APIENTRY OSAL_destroy_thread(sys_os_thread_t* OUT th)
Description	Deletes an instance of a task that was previously created
Parameter	th->phTask: The handle of the task being deleted.
Return value	None
Remark	Refer to vTaskDelete()

1.2.41. OSAL_create_bh

Not used

1.2.42. OSAL_destroy_bh

Not used

1.2.43. OSAL_register_bh

Not used

1.2.44. OSAL_raise_bh

Not used

1.2.45. OSAL_init_os

Not used

1.2.46. OSAL_start_os

Prototype	void APIENTRY OSAL_start_os(void);
Description	Starts the FreeRTOS scheduler running
Parameter	None
Return value	None
Remark	Refer to vTaskStartScheduler()

1.2.47. OSAL_get_start_os;

Not used

1.2.48. GetTickCount

Prototype	uint32 APIENTRY GetTickCount (void);
Description	The tick count is the total number of tick interrupts that have occurred since the scheduler was started. GetTickCount() returns the current tick count value.
Parameter	None
Return value	GetTickCount() always returns the tick count value at the time that GetTickCount() was called.
Remark	Refer to xTaskGetTickCount()

2. FreeRTOS APIs

SVM SDK also uses FreeRTOS original APIs.

For those APIs, please refer to FreeRTOS Reference Manual.

3. Revision Histo

Version	Date	Description
V0.1	20180510	
V0.2	20181116	