TMOS 使用说明

V1.0

TMOS 系统时钟单位为 625us，以 RTC 为基准得到所有需要系统的时间。

任务管理 – 多任务管理方式实际上只有一个任务在运行，但是可以使用任务调度的策略将多个任务进行调度，每个任务占用一定的时间，所有的任务通过时间分片的方式处理。

extern bStatus_t tmos_set_event(tmosTaskID taskID, tmosEvents event);
此函数将建立一个在 taskID 层生效的，名为 event 的任务，并立即生效

```
196 void Peripheral_Init( )
197 {
198   Peripheral_TaskID = TMOS_ProcessEventRegister( Peripheral_ProcessEvent );
199
200   // Setup the GAP Peripheral Role Profile
201   {
214   // Set the GAP Characteristics
215   GGS_SetParameter( GGS_DEVICE_NAME_ATT, GAP_DEVICE_NAME_LEN, attDeviceName );
216
217   // Set advertising interval
218   {
225   // Setup the GAP Bond Manager
226   {
239   // Initialize GATT attributes
240   GGS_AddService( GATT_ALL_SERVICES );            // GAP
241   GATTServApp_AddService( GATT_ALL_SERVICES );    // GATT attributes
242   DevInfo_AddService();                           // Device Information Service
243   SimpleProfile_AddService( GATT_ALL_SERVICES );  // Simple GATT Profile
244
245   // Setup the SimpleProfile Characteristic Values
246   {
260   // Init Connection Item
261   peripheralInitConnItem( &peripheralConnList );
262
263   // Register callback with SimpleGATTprofile
264   SimpleProfile_RegisterAppCBs( &Peripheral_SimpleProfileCBs );
265
266   // Setup a delayed profile startup
267   tmos_set_event( Peripheral_TaskID, SBP_START_DEVICE_EVT );
268 }
```

应用层taskID

建立的任务事件名

```
300 uint16 Peripheral_ProcessEvent( uint8 task_id, uint16 events )
301 {
302
303 //  VOID task_id; // TMOS required parameter that isn't used in this function
304
305   if ( events & SYS_EVENT_MSG ){
317   if ( events & SBP_START_DEVICE_EVT ){
318   // Start the Device
319     GAPRole_PeripheralStartDevice( Peripheral_TaskID, &Peripheral_BondMgrCBs, &Peri
320     return ( events ^ SBP_START_DEVICE_EVT );
321   }
322
323   if ( events & SBP_PERIODIC_EVT )
324   {
334   if ( events & SBP_PARAM_UPDATE_EVT )
335   {
347   if ( events & SBP_READ_RSSI_EVT )
348   {
354   // Discard unknown events
355   return 0;
356 }
```

创建事件时填的事件名

创建的事件在生效后，程序就会运行此处该事件对应的任务处理代码。

extern bStatus_t tmos_start_task(tmosTaskID taskID, tmosEvents event, tmosTimer time);

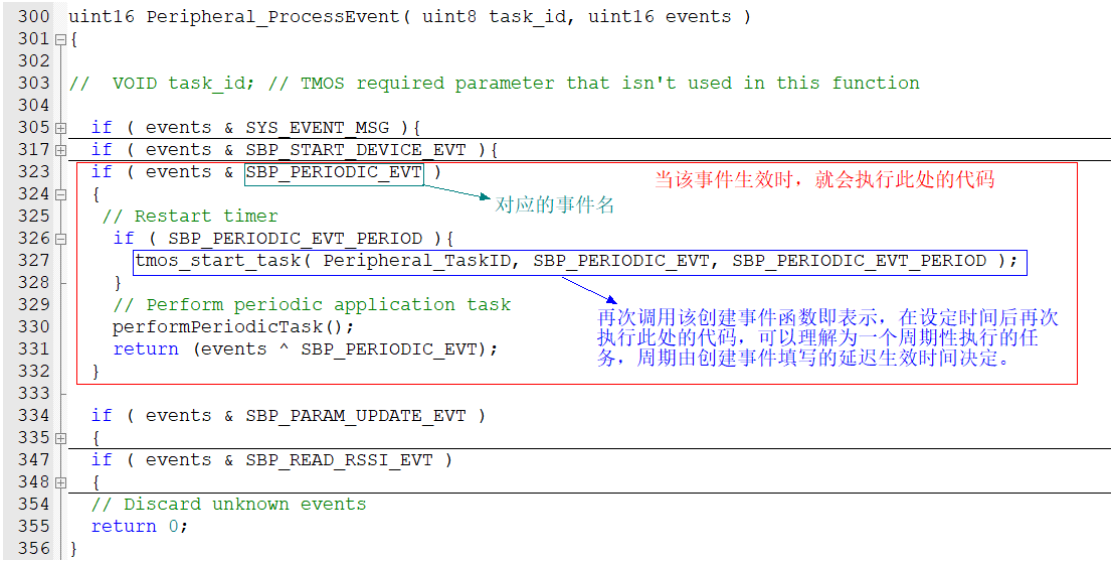
此函数将建立一个在 taskID 层生效的，名为 event 的任务，并延迟 time*625us 后生效。

```
384 static void Peripheral_LinkEstablished( gapRoleEvent_t * pEvent )
385 {
386   gapEstLinkReqEvent_t *event = (gapEstLinkReqEvent_t *) pEvent;
387
388   // See if already connected
389   if( peripheralConnList.connHandle != GAP_CONNHANDLE_INIT )
390   {
394   else
395   {
396     peripheralConnList.connHandle = event->connectionHandle;
397     peripheralConnList.connInterval = event->connInterval;
398     peripheralConnList.connSlaveLatency = event->connLatency;
399     peripheralConnList.connTimeout = event->connTimeout;
400
401     // Set timer for periodic event
402     tmos_start_task( Peripheral_TaskID, SBP_PERIODIC_EVT, SBP_PERIODIC_EVT_PERIOD );
403
404     // Set timer for param update event
405     tmos_start_task( Peripheral_TaskID, SBP_PARAM_UPDATE_EVT, SBP_PARAM_UPDATE_DELAY );
406
407     // Start read rssi
408     tmos_start_task( Peripheral_TaskID, SBP_READ_RSSI_EVT, SBP_READ_RSSI_EVT_PERIOD );
409
410     printf("Conn %x - Int %x \n", event->connectionHandle, event->connInterval);
411   }
412 }
```

应用层taskID

创建的事件名

延迟生效的延迟时间，单位为0.625ms，若填写1600，则表示该事件将在1秒后生效并执行对应事件的处理函数。



extern bStatus_t tmos_stop_task(tmosTaskID taskID, tmosEvents event);

此函数将停止一个会在 taskID 层生效的，名为 event 的任务，调用此函数后，该事件任务将不会生效。

任务调度函数使用注意事项：

1、 禁止在中断中调用
2、 建议不要在单个任务中执行超过连接间隔一半时长的任务，否则将影响蓝牙通讯
3、 同理，在中断中建议不要执行超过连接间隔一半时长的任务，否则将影响蓝牙通讯
4、 在事件生效执行的代码中调用延时执行函数时，延时时间以当前事件生效时间点为基准偏移，所以对调用延时执行函数在生效执行的代码中摆放的位置没有要求。
5、 任务存在优先级，根据在 xxx_ProcessEvent 函数中判断的先后顺序决定，同时生效的任务，先执行先判断，后执行后判断。注意，执行完先判断的事件任务后，要等到任务调度系统轮巡一遍后，才会执行后判断的事件任务。
6、 事件名按位定义，每一层 taskID 最多包含 1 个消息事件和 15 个任务事件（共 16 位）

消息管理 — 消息是一个带有数据的事件，用于协议栈各层之间传递数据，支持同时添加多个消息。

extern u8 *tmos_msg_allocate(u16 len);

申请内存函数，发送消息之前需要先给消息申请内存空间。如果返回为 NULL，则申请失败。

extern bStatus_t tmos_msg_send(tmosTaskID taskID, u8 *msg_ptr);

发送消息函数，参数为消息想要发送到哪一层的 taskID 以及消息指针。当调用此函数时，对应参数 taskID 层的消息事件将会立即置 1 生效。

extern u8 *tmos_msg_receive(tmosTaskID taskID);
接收消息函数，参数为想要接收哪一层的 taskID。

extern bStatus_t tmos_msg_deallocate(u8 *msg_ptr);
释放消息占用内存的函数，处理完消息后需要释放内存占用。

消息管理使用范例：

```
197 void Peripheral_Init( )
198 {
199   Peripheral_TaskID = TMOS_ProcessEventRegister( Peripheral_ProcessEvent );
200
201   // Setup the GAP Peripheral Role Profile
202   {
215   // Set the GAP Characteristics
216   GGS_SetParameter( GGS_DEVICE_NAME_ATT, GAP_DEVICE_NAME_LEN, attDeviceName );
217
218   // Set advertising interval
219   {
226   // Setup the GAP Bond Manager
227   {
240   // Initialize GATT attributes
241   GGS_AddService( GATT_ALL_SERVICES );            // GAP
242   GATTServApp_AddService( GATT_ALL_SERVICES );    // GATT attributes
243   DevInfo_AddService();                           // Device Information Service
244   SimpleProfile_AddService( GATT_ALL_SERVICES );  // Simple GATT Profile
245
246   // Setup the SimpleProfile Characteristic Values
247   {
261   // Init Connection Item
262   peripheralInitConnItem( &peripheralConnList );
263
264   // Register callback with SimpleGATTprofile
265   SimpleProfile_RegisterAppCBs( &Peripheral_SimpleProfileCBs );
266
267   // 注册按键消息传递的TaskID
268   HAL_KEY_RegisterForKeys( Peripheral_TaskID );
269
270   // Setup a delayed profile startup
271   tmos_set_event( Peripheral_TaskID, SBP_START_DEVICE_EVT );
272 }
```

在应用层调用按键的注册函数，将应用层的taskID传递到按键所在层

```
57  void HAL_KEY_RegisterForKeys( tmosTaskID id )
58  {
59    registeredKeysTaskID = id;
60  }
61
62  /*************************************************************************************************
72  void HalKeyConfig (uint8 interruptEnable, halKeyCBack_t cback)
73  {
92  /*************************************************************************
102 uint8 OnBoard_SendKeys( uint8 keys, uint8 state )
103 {
104   keyChange_t *msgPtr;
105
106   if ( registeredKeysTaskID != TASK_NO_TASK ){
107     // Send the address to the task
108     msgPtr = (keyChange_t *)tmos_msg_allocate( sizeof(keyChange_t) );
109     if ( msgPtr ){
110       msgPtr->hdr.event = KEY_CHANGE;
111       msgPtr->state = state;
112       msgPtr->keys = keys;
113       tmos_msg_send( registeredKeysTaskID, (uint8 *)msgPtr );
114     }
115     return ( SUCCESS );
116   }
117   else{
118     return ( FAILURE );
119   }
120 }
```

应用层调用的注册函数，保存传递进来的taskID值

分配要发送的消息内存，如果申请内存成功，再进行下面的赋值发送。

要发送的消息类型，下面处理消息时会用到

调用发送消息函数，参数为此前注册函数保存的应用层taskID，以及消息指针

```
304 uint16 Peripheral_ProcessEvent( uint8 task_id, uint16 events )
305 {
306
307 //  VOID task_id; // TMOS required parameter that isn't used in this function
308
309   if ( events & SYS_EVENT_MSG ){
310     uint8 *pMsg;
311
312     if ( (pMsg = tmos_msg_receive( Peripheral_TaskID )) != NULL ){
313       Peripheral_ProcessTMOSMsg( (tmos_event_hdr_t *)pMsg );
314       // Release the TMOS message
315       tmos_msg_deallocate( pMsg );
316     }
317     // return unprocessed events
318     return (events ^ SYS_EVENT_MSG);
319   }
320
321   if ( events & SBP_START_DEVICE_EVT ){
327   if ( events & SBP_PERIODIC_EVT )
328   {
338   if ( events & SBP_PARAM_UPDATE_EVT )
339   {
351   if ( events & SBP_READ_RSSI_EVT )
352   {
358   // Discard unknown events
359   return 0;
360 }
```

如果有消息传递到当前层，则当前层的消息事件置1，在消息事件中调用接收消息的函数获取消息并处理，参数为当前层的taskID

处理完消息后需要释放消息占用的内存

处理消息的函数注意传递的参数为所有消息通用的指针，具体会在函数中判断

```
371 static void Peripheral_ProcessTMOSMsg( tmos_event_hdr_t *pMsg )
372 {
373   switch ( pMsg->event ){
374     case KEY_CHANGE:
375     {
376       PRINT("Key changed \n");
377       break;
378     }
379     default:
380       break;
381   }
382 }
```

根据所有消息通用的格式，判断传递进来的消息类型

这里的 KEY_CHANGE 是在发送消息的时候定义的消息类型，判断不同的消息类型做相应处理