8. API

8.1 TMOS API

8.1.1 指令

```
1.  bStatus_t TMOS_TimerInit( pfnGetSysClock fnGetClock )
```

TMOS 时钟初始化。

参数	描述
pfnGetSysClock	0：选择 RTC 作为系统时钟 其他有效值：其他时钟获取接口，如 SYS_GetSysTickCnt()
返回	0：SUCCESS 1：FAILURE

```
1.  tmosTasklD TMOS_ProcessEventRegister( pTaskEventHandIerFn eventCb )
```

注册事件回调函数，一般用于注册任务时首先执行。

参数	描述
eventCb	TMOS 任务回调函数
返回	分配的 ID 值，0xFF 表示无效

```
1.  bStatus_t tmos_set_event( tmosTaskID taskID, tmosEvents event )
```

立即启动 taskID 任务中对应的 event 事件，调用一次执行一次。

参数	描述
taskID	tmos 分配的任务 ID
event	任务中的事件
返回	0：成功

```
1.  bStatus_t tmos_start_task( tmosTaskID taskID, tmosEvents event, tmosTimer
time )
```

延迟 time*625μs 后启动 taskID 任务中对应的 event 事件，调用一次执行一次。

参数	描述
taskID	tmos 分配的任务 ID
event	任务中的事件
time	延迟的时间
返回	0：成功

```
1. bStatus_t tmos_stop_event( tmosTaskID taskID, tmosEvents event )
```

停止一个 event 事件，调用此函数后，该事件将不会生效。

参数	描述
taskID	tmos 分配的任务 ID
event	任务中的事件
返回	0：成功

```
1. bStatus_t tmos_clear_event( tmosTaskID taskID, tmosEvents event )
```

清理一个已经超时的 event 事件，注意不能在它自己的 event 函数内执行。

参数	描述
taskID	tmos 分配的任务 ID
event	任务中的事件
返回	0：成功

```
1. bStatus_t tmos_start_reload_task( tmosTaskID taskID, tmosEvents event,
tmosTimer time )
```

延迟 time*625μs 执行 event 事件，调用一次循环执行，除非运行 tmos_stop_task 关掉。

参数	描述
taskID	tmos 分配的任务 ID
event	任务中的事件
time	延迟的时间
返回	0：成功

```
1. tmosTimer tmos_get_task_timer( tmosTaskID taskID, tmosEvents event )
```

获取事件距离到期事件的滴答数。

参数	描述
taskID	tmos 分配的任务 ID
event	任务中的事件
返回	!0：事件距离到期的滴答数 0：事件未找到

```
1. uint32_t TMOS_GetSystemClock( void )
```

返回 tmos 系统运行时长,单位为 625μs，如 1600=1s。

参数	描述

返回	TMOS 运行时长

```
1. void TMOS_SystemProcess( void )
```

tmos 的系统处理函数,需要在主函数中不断运行。

```
1. bStatus_t tmos_msg_send( tmosTaskID taskID, uint8_t *msg_ptr )
```

发送消息到某个任务，当调用此函数时，对应任务的消息事件 event 会立即置 1 生效。

参数	描述
taskID	tmos 分配的任务 ID
msg_ptr	消息指针
返回	SUCCESS：成功 INVALID_TASK：任务 ID 无效 INVALID_MSG_POINTER：消息指针无效

```
1. uint8_t  *tmos_msg_receive( tmosTaskID taskID )
```

接收消息。

参数	描述
taskID	tmos 分配的任务 ID
返回	接收到的消息或者无消息待接收（NULL）

```
1. uint8_t  *tmos_msg_allocate( uint16_t len )
```

为消息申请内存空间。

参数	描述
len	消息的长度
返回	申请到的缓冲区指针 NULL：申请失败

```
1. bStatus_t tmos_msg_deallocate( uint8_t *msg_ptr )
```

释放消息占用的内存空间。

参数	描述
msg_ptr	消息指针
返回	0：成功

```
1. uint8_t tmos_snv_read( uint8_t id, uint8_t len, void *pBuf )
```

从 NV 读取数据。

注：NV 区的读写操作尽量在 TMOS 系统运行之前调用。

参数	描述
id	有效的 NV 项目 ID
len	读取数据的长度
pBuf	要读取的数据指针
返回	SUCCESS NV_OPER_FAILED：失败

```
1. void TMOS_TimerIRQHandler( void )
```

TMOS 定时器中断函数。

以下函数较 C 语言库函数更加节省内存空间

```
1. uint32_t tmos_rand( void  )
```

生成伪随机数。

参数	描述
返回	伪随机数

```
1. bool tmos_memcmp( const void *src1, const void *src2, uint32_t len )
```

把存储区 src1 和存储区 src2 的前 len 个字节进行比较。

参数	描述
src1	内存块指针
src2	内存块指针
len	要被比较的字节数
返回	1：相同 0：不同

```
1. bool tmos_isbufset( uint8_t *buf, uint8_t val, uint32_t len )
```

比较给定数据是否都是给定数值。

参数	描述
Buf	缓冲区地址
val	数值
len	数据的长度
返回	1：相同 0：不同

```
1. uint32_t tmos_strlen( char *pString )
```

计算字符串 pString 的长度，直到空结束字符，但不包括空结束字符。

参数	描述
pString	要计算长度的字符串
返回	字符串的长度

```
1. void tmos_memset( void * pDst, uint8_t Value, uint32_t len )
```

复制字符 Value 到参数 pDst 所指向的字符串的前 len 个字符。

参数	描述
pDst	要填充的内存块
Value	要被设置的值
len	要被设置为该值的字符数
返回	指向存储区 pDst 的指针

```
1. void tmos_memcpy( void *dst, const void *src, uint32_t len )
```

从存储区 src 复制 len 个字节到存储区 dst。

参数	描述
dst	用于存储复制内容的目标数组，类型强制转换为 void* 指针
src	要复制的数据源，类型强制转换为 void* 指针
len	要被复制的字节数
返回	指向目标存储区 dst 的指针

8.2 GAP API

8.2.1 指令

```
1.  bStatus_t GAP_SetParamValue( uint16_t paramID, uint16_t paramValue )
```

设置 GAP 参数值。使用此功能更改默认 GAP 参数值。

参数	描述
paramID	参数的 ID,参考 8.2.2
paramValue	新的参数值
返回	SUCCESS 或 INVALIDPARAMETER（无效参数 ID）

```
1.  uint16 GAP_GetParamValue( uint16_t paramID )
```

获取 GAP 参数值。

参数	描述
paramID	参数的 ID,参考 8.1.2
返回	GAP 的参数值；若参数 ID 无效返回 0xFFFF

8.2.2 配置参数

以下为常用的参数 ID，详细的参数 ID 请参考 CH58xBLE.LIB.h。

参数 ID	描述
TGAP_GEN_DISC_ADV_MIN	通用广播模式的广播时长，单位:0.625ms（默认值：0）
TGAP_LIM_ADV_TIMEOUT	限时可发现广播模式的广播时长，单位:1s（默认值：180）
TGAP_DISC_ADV_INT_MIN	最小广播间隔，单位:0.625ms（默认值：160）
TGAP_DISC_ADV_INT_MAX	最大广播间隔，单位:0.625ms（默认值：160）
TGAP_DISC_SCAN	扫描时长，单位:0.625ms（默认值：16384）
TGAP_DISC_SCAN_INT	扫描间隔，单位:0.625ms（默认值：16）
TGAP_DISC_SCAN_WIND	扫描窗口，单位:0.625ms（默认值：16）
TGAP_CONN_EST_SCAN_INT	建立连接的扫描间隔，单位:0.625ms（默认值：16）
TGAP_CONN_EST_SCAN_WIND	建立连接的扫描窗口，单位:0.625ms（默认值：16）
TGAP_CONN_EST_INT_MIN	建立连接的最小连接间隔，单位:1.25ms（默认值：80）
TGAP_CONN_EST_INT_MAX	建立连接的最大连接间隔，单位:1.25ms（默认值：80）
TGAP_CONN_EST_SUPERV_TIMEOUT	建立连接的连接管理超时时间，单位:10ms（默认值：2000）
TGAP_CONN_EST_LATENCY	建立连接的从设备延迟（默认值：0）

8.2.3 事件

本节介绍了 GAP 层相关的事件，可以在 CH58xBLE_LIB.h 文件中找到相关声明。其中一些事件是直接传递给应用程序，一些是由 GAPRole 和 GAPBondMgr 处理。

无论是传递给哪一层，它们都将作为带标头的 GAP_MSG_EVENT 传递：

```
1.  typedef struct
2.  {
3.    tmos_event_hdr_t  hdr;           //!< GAP_MSG_EVENT and status
4.    uint8_t opcode;                    //!< GAP type of command. Ref: @ref GAP
5.  _MSG_EVENT_DEFINES
6.  } gapEventHdr_t;
```

以下为常用事件名称以及事件传递消息的格式。详细请参考 CH58xBLE_LIB.h。

- GAP_DEVICE_INIT_DONE_EVENT：当设备初始化完成置此事件。

```
1.  typedef struct
2.  {
```

```
3.    tmos_event_hdr_t hdr;              //!< GAP_MSG_EVENT and status
4.    uint8_t opcode;                       //!< GAP_DEVICE_INIT_DONE_EVENT
5.    uint8_t devAddr[B_ADDR_LEN];          //!< Device's BD_ADDR
6.    uint16_t dataPktLen;                  //!< HC_LE_Data_Packet_Length
7.    uint8_t numDataPkts;                  //!< HC_Total_Num_LE_Data_Packets
8.  } gapDeviceInitDoneEvent_t;
```

- GAP_DEVICE_DISCOVERY_EVENT：设备发现过程完成时置此事件。

```
1.  typedef struct
2.  {
3.    tmos_event_hdr_t hdr; //!< GAP_MSG_EVENT and status
4.    uint8_t opcode;          //!< GAP_DEVICE_DISCOVERY_EVENT
5.    uint8_t numDevs;         //!< Number of devices found during scan
6.    gapDevRec_t *pDevList; //!< array of device records
7.  } gapDevDiscEvent_t;
```

- GAP_END_DISCOVERABLE_DONE_EVENT：当广播结束时置此事件。

```
1.  typedef struct
2.  {
3.   tmos_event_hdr_t hdr; //!< GAP_MSG_EVENT and status
4.   uint8_t opcode;          //!< GAP_END_DISCOVERABLE_DONE_EVENT
5.  } gapEndDiscoverableRspEvent_t;
```

- GAP_LINK_ESTABLISHED_EVENT：建立连接后置此事件。

```
1.  typedef struct
2.  {
3.    tmos_event_hdr_t hdr;     //!< GAP_MSG_EVENT and status
4.    uint8_t opcode;              //!< GAP_LINK_ESTABLISHED_EVENT
5.    uint8_t devAddrType;         //!< Device address type: @ref GAP_ADDR_TYPE_
6.  DEFINES
7.    uint8_t devAddr[B_ADDR_LEN]; //!< Device address of link
8.    uint16_t connectionHandle;   //!< Connection Handle from controller used t
9.  o ref the device
10.   uint8_t connRole;            //!< Connection formed as Master or Slave
11.   uint16_t connInterval;       //!< Connection Interval
12.   uint16_t connLatency;        //!< Connection Latency
13.   uint16_t connTimeout;        //!< Connection Timeout
14.   uint8_t clockAccuracy;       //!< Clock Accuracy
15. } gapEstLinkReqEvent_t;
```

- GAP_LINK_TERMINATED_EVENT：连接断开后置此事件。

```
typedef struct
{
  tmos_event_hdr_t hdr;   //!< GAP_MSG_EVENT and status
  uint8_t opcode;            //!< GAP_LINK_TERMINATED_EVENT
  uint16_t connectionHandle; //!< connection Handle
  uint8_t reason;            //!< termination reason from LL
  uint8_t connRole;
} gapTerminateLinkEvent_t;
```

- GAP_LINK_PARAM_UPDATE_EVENT：接收到参数更新事件后置此事件。

```
typedef struct
{
  tmos_event_hdr_t hdr;     //!< GAP_MSG_EVENT and status
  uint8_t opcode;             //!< GAP_LINK_PARAM_UPDATE_EVENT
  uint8_t status;             //!< bStatus_t
  uint16_t connectionHandle;  //!< Connection handle of the update
  uint16_t connInterval;      //!< Requested connection interval
  uint16_t connLatency;       //!< Requested connection latency
  uint16_t connTimeout;       //!< Requested connection timeout
} gapLinkUpdateEvent_t;
```

- GAP_DEVICE_INFO_EVENT：在发现设备期间发现设备置此事件。

```
typedef struct
{
  tmos_event_hdr_t hdr;    //!< GAP_MSG_EVENT and status
  uint8_t opcode;             //!< GAP_DEVICE_INFO_EVENT
  uint8_t eventType;          //!< Advertisement Type: @ref GAP_ADVERTISEMEN
T_REPORT_TYPE_DEFINES
  uint8_t addrType;           //!< address type: @ref GAP_ADDR_TYPE_DEFINES
  uint8_t addr[B_ADDR_LEN];   //!< Address of the advertisement or SCAN_RSP
  int8_t  rssi;               //!< Advertisement or SCAN_RSP RSSI
  uint8_t dataLen;            //!< Length (in bytes) of the data field (evtD
ata)
  uint8_t *pEvtData;          //!< Data field of advertisement or SCAN_RSP
} gapDeviceInfoEvent_t;
```

8.3 GAPRole API

8.3.1 GAPRole Common Role API

8.3.1.1 指令

```
1.  bStatus_t GAPRole_SetParameter( uint16_t param, uint16_t len, void *pValue )
```

设置 GAP 角色参数。

参数	描述
param	配置参数 ID，详见 8.2.1.2 节
len	写入的数据长度
pValue	指向设置参数值的指针。该指针取决于参数 ID，并将被强制转换成合适的数据类型。
返回	SUCCESS INVALIDPARAMETER：参数无效 bleInvalidRange：参数长度无效 blePending：上次参数更新未结束 bleIncorrectMode：模式错误

```
1.  bStatus_t GAPRole_GetParameter( uint16_t param, void *pValue )
```

获取 GAP 角色参数。

参数	描述
param	配置参数 ID，详见 8.2.1.2 节
pValue	指向获取参数的位置的指针。该指针取决于参数 ID，并将被强制转换成合适的数据类型。
返回	SUCCESS INVALIDPARAMETER：参数无效

```
1.  bStatus_t GAPRole_TerminateLink( uint16_t connHandle )
```

断开当前 connHandle 指定的连接。

参数	描述
connHandle	连接句柄
返回	SUCCESS bleIncorrectMode：模式错误

```
1.  bStatus_t GAPRole_ReadRssiCmd( uint16_t connHandle )
```

读取当前 connHandle 指定连接的 RSSI 值。

参数	描述
connHandle	连接句柄
返回	SUCCESS

	0x02:无有效连接

8.3.1.2 常用可配置参数

参数	读/写	大小	描述
GAPROLE_BD_ADDR	只读	uint8	设备地址
GAPRPLE_ADVERT_ENABLE	可读可写	uint8	使能或关闭广播，默认使能
GAPROLE_ADVERT_DATA	可读可写	≤240	广播数据，默认全 0。
GAPROLE_SCAN_RSP_DATA	可读可写	≤240	扫描应答数据，默认全 0
GAPROLE_ADV_EVENT_TYPE	可读可写	uint8	广播类型,默认可连接非定向广播
GAPROLE_MIN_CONN_INTERVAL	可读可写	uint16	最小连接间隔，范围：1.5ms~4s，默认 8.5ms。
GAPROLE_MAX_CONN_INTERVAL	可读可写	uint16	最大连接间隔，范围：1.5ms~4s，默认 8.5ms。

8.3.1.3 回调函数

```
/**
 * Callback when the device has read a new RSSI value during a connection.
 */
typedef void (*gapRolesRssiRead_t)(uint16_t connHandle, int8_t newRSSI )
```

此函数为读取 RSSI 的回调函数，其指针指向应用程序，以便 GAPRole 可以将事件返回给应用程序。传递方式如下：

```
// GAP Role Callbacks
static gapCentralRoleCB_t centralRoleCB =
{
  centralRssiCB,       // RSSI callback
  centralEventCB,       // Event callback
  centralHciMTUChangeCB       // MTU change callback
};
```

8.3.2 GAPRolePeripheral Role API

8.3.2.1 指令

```
bStatus_t GAPRole_PeripheralInit( void )
```

蓝牙从机 GAPRole 任务初始化。

参数	描述
返回	SUCCESS bleInvalidRange：参数超出范围

```
1.  bStatus_t GAPRole_PeripheralStartDevice( uint8_t taskid, gapBondCBs_t *pCB,g
    apRolesCBs_t *pAppCallbacks )
```

蓝牙从机设备初始化。

参数	描述
taskid	tmos 分配的任务 ID
pCB	绑定回调函数，包括密钥回调，配对状态回调
pAppCallbacks	GAPRole 回调函数，包括设备的状态回调，RSSI 回调，参数更新回调
返回	SUCCESS bleAlreadyInRequestedMode:设备已经初始化过

```
1.  bStatus_t GAPRole_PeripheralConnParamUpdateReq( uint16_t connHandle,
2.                                                  uint16_t minConnInterval,
3.                                                  uint16_t maxConnInterval,
4.                                                  uint16_t latency,
5.                                                  uint16_t connTimeout,
6.                                                  uint8_t taskId)
```

蓝牙从机连接参数更新。

注：与 GAPRole_UpdateLink()不同，此为从机与主机协商连接参数，而 GAPRole_UpdateLink()是主机直接配置连接参数。

参数	描述
connHandle	连接句柄
minConnInterval	最小连接间隔
maxConnInterval	最大连接间隔
latency	从设备延迟事件数
connTimeout	连接超时
taskID	toms 分配的任务 ID
返回	SUCCESS：参数上传成功 BleNotConnected：无连接所以参数无法更新 bleInvalidRange：参数错误

8.3.2.2 回调函数

```
1.  typedef struct
2.  {
3.      gapRolesStateNotify_t pfnStateChange; //!< Whenever the device changes
4.  state
5.      gapRolesRssiRead_t pfnRssiRead; //!< When a valid RSSI is read from
6.  controller
7.      gapRolesParamUpdateCB_t pfnParamUpdate; //!< When the connection
8.  parameteres are updated
9.  } gapRolesCBs_t;
```

从机状态回调函数：

```
1.  /**
2.      * Callback when the device has been started.  Callback event to
3.      * the Notify of a state change.
4.   */
5.  void (*gapRolesStateNotify_t )( gapRole_States_t newState,
    gapRoleEvent_t *pEvent);
```

其中，状态分为以下几种：

- GAPROLE_INIT //等待启动
- GAPROLE_STARTED //初始化完成但是未未广播
- GAPROLE_ADVERTISING //正在广播
- GAPROLE_WAITING //设备启动了但是未广播，此时正在等待再次广播
- GAPROLE_CONNECTED //连接状态
- GAPROLE_CONNECTED_ADV //连接状态且在广播
- GAPROLE_ERROR//无效状态，若为此状态表明错误

从机参数更新回调函数：

```
1.  /**
2.      * Callback when the connection parameteres are updated.
3.      */
4.  typedef void (*gapRolesParamUpdateCB_t)( uint16_t connHandle,
5.                                          uint16_t connInterval,
6.                                          uint16_t connSlaveLatency,
7.                                          uint16_t connTimeout );
```

参数更新成功调用此回调函数。

8.3.3 GAPRole Central Role API

8.3.3.1 指令

```
1.  bStatus_t GAPRole_CentralInit( void )
```

主机 GAPRole 任务初始化。

参数	描述
返回	SUCCESS bleInvalidRange：参数超出范围

```
1.  bStatus_t GAPRole_CentralStartDevice( uint8_t taskid, gapBondCBs_t *pCB, gap
    CentralRoleCB_t *pAppCallbacks )
```

以主机角色启动设备。此函数通常在系统启动期间调用一次。

参数	描述
taskid	tmos 分配的任务 ID
pCB	绑定回调函数，包括密钥回调，配对状态回调
pAppCallbacks	GAPRole 回调函数，包括设备的状态回调，RSSI 回调，参数更新回调
返回	SUCCESS bleAlreadyInRequestedMode：设备已经启动

```
1.  bStatus_t GAPRole_CentralStartDiscovery( uint8_t mode, uint8_t activeScan, u
    int8_t whiteList )
```

主机扫描参数配置。

参数	描述
mode	扫描模式，分为： DEVDISC_MODE_NONDISCOVERABLE：不作设置 DEVDISC_MODE_GENERAL：扫描通用可发现设备 DEVDISC_MODE_LIMITED：扫描有限的可发现设备 DEVDISC_MODE_ALL：扫描所有
activeScan	TRUE 为使能扫描
whiteList	TRUE 为只扫描白名单设备
返回	SUCCESS

```
1.  bStatus_t GAPRole_CentralCancelDiscovery( void )
```

主机停止扫描。

参数	描述
返回	SUCCESS bleInvalidTaskID：没有任务正在扫描 bleIncorrectMode：不在扫描模式

```
1.  bStatus_t GAPRole_CentralEstablishLink( uint8_t highDutyCycle, uint8_t white
    List, uint8_t addrTypePeer, uint8_t *peerAddr )
```

与对端设备进行连接。

参数	描述
highDutyCycle	TURE 使能高占空比扫描
whiteList	TURE 使用白名单
addrTypePeer	对端设备的地址类型，包括： ADDRTYPE_PUBLIC：BD_ADDR ADDRTYPE_STATIC：静态地址

	ADDRTYPE_PRIVATE_NONRESOLVE：不可解析私有地址 ADDRTYPE_PRIVATE_RESOLVE：可解析的私有地址
peerAddr	对端设备地址
返回	SUCCESS：成功连接 bleIncorrectMode：无效的配置文件 bleNotReady：正在进行扫描 bleAlreadyInRequestedMode：暂时无法处理 bleNoResources：连接过多

8.3.3.2 回调函数

这些回调函数的指针从应用程序传递到 GAPRole，以便 GAPRole 可以将事件返回给应用程序。它们按如下方式传递：

```
1.  typedef struct
2.  {
3.      gapRolesRssiRead_t rssiCB;  //!< RSSI callback.
4.      pfnGapCentralRoleEventCB_t eventCB;  //!< Event callback.
5.      pfnHciDataLenChangeEvCB_t ChangCB;  //!< Length Change Event Callback.
6.  } gapCentralRoleCB_t; // gapCentralRoleCB_t
```

主机 RSSI 回调函数：

```
1.  /**
2.   * Callback when the device has read a new RSSI value during a connection.
3.   */
4.  typedef void ( *gapRolesRssiRead_t )( uint16_t connHandle, int8_t newRSSI )
```

此函数将 RSSI 报告给应用程序。

主机事件回调函数：

```
1.  /**
2.      * Central Event Callback Function
3.      */
4.  typedef void ( *pfnGapCentralRoleEventCB_t ) ( gapRoleEvent_t *pEvent );
5.  //!< Pointer to event structure.
```

此回调用于将 GAP 状态更改事件传递给应用程序。
回调事件可参考 8.1.3 节。

MTU 交互回调函数：

```
typedef void (*pfnHciDataLenChangeEvCB_t)
(
  uint16_t connHandle,
  uint16_t maxTxOctets,
  uint16_t maxRxOctets
);
```

即与低功耗蓝牙交互的数据包大小。

8.4 GATT API

8.4.1 指令

8.4.1.1 从机指令

```
bStatus_t GATT_Indication( uint16_t connHandle, attHandleValueInd_t *pInd, uint8_t authenticated, uint8_t taskId )
```

服务器向客户端指示一个特征值并期望属性协议层确认已经成功收到指示。
需要注意的是，当返回失败时需要释放内存。

参数	描述
connHandle	连接句柄
pInd	指向要发送的指令
authenticated	是否需要经过身份验证的连接
taskId	tmos 分配的任务 ID

```
bStatus_t GATT_Notification( uint16_t connHandle, attHandleValueNoti_t *pNoti, uint8_t authenticated )
```

服务器向客户端通知特征值，但不期望任何属性协议层确认已经成功收到通知。
需要注意的是，当返回失败时需要释放内存。

参数	描述
connHandle	连接句柄
pInd	指向要通知的指令
authenticated	是否需要经过身份验证的连接

8.4.1.2 主机指令

```
bStatus_t GATT_ExchangeMTU( uint16_t connHandle, attExchangeMTUReq_t *pReq, uint8_t taskId )
```

当客户端支持的值大于属性协议默认 ATT_MTU 的值时，客户端使用此程序将 ATT_MTU 设置为两个设备均可支持的最大可能值。

参数	描述
connHandle	连接句柄
pReq	指向要发送的指令
taskID	通知的任务的 ID

```
1.  bStatus_t GATT_DiscAllPrimaryServices( uint16_t connHandle, uint8_t taskId)
```

发现服务器上的所有主服务。

参数	描述
connHandle	连接句柄
taskID	通知的任务的 ID

```
1.  bStatus_t GATT_DiscPrimaryServiceByUUID( uint16_t connHandle, uint8_t *pUUID
    , uint8_t len, uint8_t taskId )
```

当仅知道 UUID 时，客户端可以通过此函数发现服务器上的主服务。由于主服务在服务器上可能存在多个，所以被发现的主服务有 UUID 标识。

参数	描述
connHandle	连接句柄
pUUID	指向要查找的服务器的 UUID 的指针
len	值的长度
taskID	通知的任务的 ID

```
1.   bStatus_t GATT_FindIncludedServices( uint16_t connHandle, uint16_t startHan
    dle, uint16_t endHandle, uint8_t taskId )
```

客户端使用此函数在服务器上查找此服务。查找的服务由服务句柄范围标识。

参数	描述
connHandle	连接句柄
startHandle	起始句柄
endHandle	结束句柄
taskID	通知的任务的 ID

```
1.  bStatus_t GATT_DiscAllChars( uint16_t connHandle, uint16_t startHandle, uint
    16_t endHandle, uint8_t taskId )
```

当仅知道服务句柄范围时，客户端可使用此函数在服务器上查找所有特征声明。

参数	描述
connHandle	连接句柄
startHandle	起始句柄

endHandle	结束句柄
taskID	通知的任务的 ID

```
1. bStatus_t GATT_DiscCharsByUUID( uint16_t connHandle, attReadByTypeReq_t *pRe
   q, uint8_t taskId )
```

当服务句柄范围和特征 UUID 已知时，客户端可使用此函数在服务器上发现特征。

参数	描述
connHandle	连接句柄
pReq	指向要发送的请求的指针
taskID	通知的任务的 ID

```
1. bStatus_t GATT_DiscAllCharDescs( uint16_t connHandle, uint16_t startHandle,
   uint16_t endHandle, uint8_t taskId )
```

当知道特征的句柄范围时，客户端可使用此程序在特征定义中查找所有特征描述符 AttributeHandles 和 AttributeTypes。

参数	描述
connHandle	连接句柄
startHandle	起始句柄
endHandle	结束句柄
taskID	通知的任务的 ID

```
1. bStatus_t GATT_ReadCharValue( uint16_t connHandle, attReadReq_t *pReq, uint8
   _t taskId )
```

当客户端知道特征句柄后，可用此函数从服务器读取特征值。

参数	描述
connHandle	连接句柄
pReq	指向要发送的请求的指针
taskID	通知的任务的 ID

```
1. bStatus_t GATT_ReadUsingCharUUID( uint16_t connHandle, attReadByTypeReq_t *p
   Req, uint8_t taskId )
```

当客户端只知道特征的 UUID 而不知道特征的句柄的时候，可使用此函数从服务器读取特征值。

参数	描述
connHandle	连接句柄
pReq	指向要发送的请求的指针

taskID	通知的任务的 ID

```
1.  bStatus_t GATT_ReadLongCharValue( uint16_t connHandle, attReadBlobReq_t *pRe
    q, uint8_t taskId )
```

读取服务器特征值，但特征值比单个读取响应协议中可发送的长度长。

参数	描述
connHandle	连接句柄
pReq	指向要发送的请求的指针
taskID	通知的任务的 ID

```
1.  bStatus_t GATT_ReadMultiCharValues( uint16_t connHandle, attReadMultiReq_t *
    pReq, uint8_t taskId )
```

从服务器读取多个特征值。

参数	描述
connHandle	连接句柄
pReq	指向要发送的请求的指针
taskID	通知的任务的 ID

```
1.  bStatus_t GATT_WriteNoRsp( uint16_t connHandle, attWriteReq_t *pReq )
```

当客户端已知特征句柄可向服务器写入特征而不需要确认写入是否成功。

参数	描述
connHandle	连接句柄
pReq	指向要发送的命令的指针

```
1.  bStatus_t GATT_SignedWriteNoRsp(uint16_t connHandle, attWriteReq_t *pReq)
```

当客户端知道特征句柄且 ATT 确认未加密时，可用此函数向服务器写入特征值。仅当 Characteristic Properties 认证位使能且服务器和客户端均建立绑定。

参数	描述
connHandle	连接句柄
pReq	指向要发送的命令的指针

```
1.  bStatus_t GATT_WriteCharValue( uint16_t connHandle, attWriteReq_t *pReq, uin
    t8_t taskId )
```

当客户端知道特征句柄，此函数可将特征值写入服务器。仅能写入特征值的第一个八位数据。此函数会返回写过程是否成功。

参数	描述
connHandle	连接句柄
pReq	指向要发送的请求的指针
taskID	通知的任务的 ID

```
1.  bStatus_t GATT_WriteLongCharDesc( uint16_t connHandle, attPrepareWriteReq_t
    *pReq, uint8_t taskId )
```

当客户端知道特征值句柄但是特征值长度大于单个写入请求属性协议中定义的长度时，可使用此函数。

参数	描述
connHandle	连接句柄
pReq	指向要发送的请求的指针
taskID	通知的任务的 ID

8.4.2 返回

- SUCCESS（0x00）：指令按预期执行。
- INVALIDPARAMETER（0x02）：无效的连接句柄或请求字段。
- MSG_BUFFER_NOT_AVAIL（0x04）：HCI 缓冲区不可用。请稍后重试。
- bleNotConnected（0x14）：设备未连接。
- blePending（0x17）：
  - 当返回到客户端功能时，服务器或 GATT 子过程正在进行中，有待处理的响应。
  - 返回服务器功能时，来自客户端的确认待处理。
- bleTimeout（0x16）：上一个事务超时。重新连接之前，无法发送 ATT 或 GATT 消息。
- bleMemAllocError（0x13）：发生内存分配错误
- bleLinkEncrypted（0x19）：链接已加密。不要在加密的链接上发送包含身份验证签名的 PDU。

8.4.3 事件

应用程序通过 TMOS 的消息（GATT_MSG_EVENT）接收协议栈的事件。

以下为常用事件名称以及事件传递消息的格式。详细请参考 CH58xBLE_LIB.h。

- ATT_ERROR_RSP：

```
1.  typedef struct
2.  {
3.       uint8_t reqOpcode; //!< Request that generated this error response
4.      uint16_t handle;   //!< Attribute handle that generated error response
5.       uint8_t errCode;   //!< Reason why the request has generated error resp
6.  onse
7.  } attErrorRsp_t;
```

- ATT_EXCHANGE_MTU_REQ：

```
typedef struct
{
    uint16_t clientRxMTU;  //!< Client receive MTU size
} attExchangeMTUReq_t;
```

- ATT_EXCHANGE_MTU_RSP：

```
typedef struct
{
    uint16_t serverRxMTU;  //!< Server receive MTU size
} attExchangeMTURsp_t;
```

- ATT_READ_REQ：

```
typedef struct
{
    uint16_t handle; //!< Handle of the attribute to be read (must be first
     field)
} attReadReq_t;
```

- ATT_READ_RSP：

```
typedef struct
{
    uint16_t len;    //!< Length of value
    uint8_t *pValue; //!< Value of the attribute with the handle given (0 t
o ATT_MTU_SIZE-1)
} attReadRsp_t;
```

- ATT_WRITE_REQ：

```
typedef struct
{
    uint16_t handle; //!< Handle of the attribute to be written (must be fi
rst field)
    uint16_t len;    //!< Length of value
    uint8_t *pValue; //!< Value of the attribute to be written (0 to ATT_MT
U_SIZE-3)
    uint8_t sig;     //!< Authentication Signature status (not included (0)
, valid (1), invalid (2))
    uint8_t cmd;     //!< Command Flag
```

```
} attWriteReq_t;
```

- ATT_WRITE_RSP:
- ATT_HANDLE_VALUE_NOTI:

```
typedef struct
{
    uint16_t handle; //!< Handle of the attribute that has been changed (mu
st be first field)
    uint16_t len;    //!< Length of value
    uint8_t *pValue; //!< Current value of the attribute (0 to ATT_MTU_SIZE
-3)
} attHandleValueNoti_t;
```

- ATT_HANDLE_VALUE_IND:

```
typedef struct
{
    uint16_t handle; //!< Handle of the attribute that has been changed (mu
st be first field)
    uint16_t len;    //!< Length of value
    uint8_t *pValue; //!< Current value of the attribute (0 to ATT_MTU_SIZE
-3)
} attHandleValueInd_t;
```

- ATT_HANDLE_VALUE_CFM:
  - Empty msg field

8.4.4 GATT 指令与相应的 ATT 事件

ATT 响应事件	GATT API 调用
ATT_EXCHANGE_MTU_RSP	GATT_ExchangeMTU
ATT_FIND_INFO_RSP	GATT_DiscAllCharDescs
ATT_FIND_BY_TYPE_VALUE_RSP	GATT_DiscPrimaryServiceByUUID
ATT_READ_BY_TYPE_RSP	GATT_PrepareWriteReq GATT_ExecuteWriteReq GATT_FindIncludedServices GATT_DiscAllChars GATT_DiscCharsByUUID GATT_ReadUsingCharUUID
ATT_READ_RSP	GATT_ReadCharValue GATT_ReadCharDesc

ATT_READ_BLOB_RSP	GATT_ReadLongCharValue GATT_ReadLongCharDesc
ATT_READ_MULTI_RSP	GATT_ReadMultiCharValues
ATT_READ_BY_GRP_TYPE_RSP	GATT_DiscAllPrimaryServices
ATT_WRITE_RSP	GATT_WriteCharValue GATT_WriteCharDesc
ATT_PREPARE_WRITE_RSP	GATT_WriteLongCharValue GATT_ReliableWrites GATT_WriteLongCharDesc
ATT_EXECUTE_WRITE_RSP	GATT_WriteLongCharValue GATT_ReliableWrites GATT_WriteLongCharDesc

8.4.5 ATT_ERROR_RSP 错误码

- ATT_ERR_INVALID_HANDLE（0x01）：给定的属性句柄值在此属性服务器上无效。
- ATT_ERR_READ_NOT_PERMITTED（0x02）：无法读取属性。
- ATT_ERR_WRITE_NOT_PERMITTED（0x03）：无法写入属性。
- ATT_ERR_INVALID_PDU（0x04）：属性 PDU 无效。
- ATT_ERR_INSUFFICIENT_AUTHEN（0x05）：该属性需要进行身份验证才能被读取或写入。
- ATT_ERR_UNSUPPORTED_REQ（0x06）：属性服务器不支持从属性客户端收到的请求。
- ATT_ERR_INVALID_OFFSET（0x07）：指定的偏移量超出了属性的末尾。
- ATT_ERR_INSUFFICIENT_AUTHOR（0x08）：该属性需要授权才能被读取或写入。
- ATT_ERR_PREPARE_QUEUE_FULL（0x09）：准备写入的队列过多。
- ATT_ERR_ATTR_NOT_FOUND（0x0A）：在给定的属性句柄范围内找不到属性。
- ATT_ERR_ATTR_NOT_LONG（0x0B）：无法使用读取 Blob 请求或准备写入请求来读取或写入属性。
- ATT_ERR_INSUFFICIENT_KEY_SIZE（0x0C）：用于加密此链接的加密密钥大小不足。
- ATT_ERR_INVALID_VALUE_SIZE（0x0D）：属性值长度对该操作无效。
- ATT_ERR_UNLIKELY（0x0E）：请求的属性请求遇到了一个不太可能发生的错误，并且未能按要求完成。
- ATT_ERR_INSUFFICIENT_ENCRYPT（0x0F）：该属性需要加密才能读取或写入。
- ATT_ERR_UNSUPPORTED_GRP_TYPE（0x10）：属性类型不是更高层规范定义的受支持的分组属性。
- ATT_ERR_INSUFFICIENT_RESOURCES（0x11）：资源不足，无法完成请求。

8.5 GATTServApp API

8.5.1 指令

```
1.  void GATTServApp_InitCharCfg( uint16_t connHandle, gattCharCfg_t *charCfgTbl
    )
```

初始化客户端特性配置表。

参数	描述

connHandle	连接句柄
charCfgTbl	客户端特征配置表
返回	无

```
1.  uint16_t GATTServApp_ReadCharCfg( uint16_t connHandle, gattCharCfg_t *charCf
    gTbl )
```

读取客户端的特征配置。

参数	描述
connHandle	连接句柄
charCfgTbl	客户端特征配置表
返回	属性值

```
1.  uint8_t GATTServApp_WriteCharCfg( uint16_t connHandle, gattCharCfg_t *charCf
    gTbl, uint16_t value )
```

向客户端写入特征配置。

参数	描述
connHandle	连接句柄
charCfgTbl	客户端特征配置表
value	新的值
返回	SUCCESS FAILURE

```
1.  bStatus_t GATTServApp_ProcessCCCWriteReq( uint16_t connHandle,
2.                                            gattAttribute_t *pAttr,
3.                                           uint8_t *pValue,
4.                                            uint16_t len,
5.                                            uint16_t offset,
6.                                           uint16_t validCfg );
```

处理客户端特征配置写入请求。

参数	描述
connHandle	连接句柄
pAttr	指向属性的指针
pvalue	指向写入的数据的指针
len	数据长度
offset	写入的第一个八位数据的偏移量
validCfg	有效配置
返回	SUCCESS FAILURE

8.6 GAPBondMgr API

8.6.1 指令

```
1.  bStatus_t GAPBondMgr_SetParameter( uint16_t param, uint8_t len, void *pValue
    )
```

设置绑定管理的参数。

参数	描述
param	配置参数
len	写入长度
pValue	指向写入数据的指针
返回	SUCCESS INVALIDPARAMETER：无效参数

```
1.  bStatus_t GAPBondMgr_GetParameter( uint16_t param, void *pValue )
```

获取绑定管理的参数。

参数	描述
param	配置参数
pValue	指向读出数据的地址
返回	SUCCESS INVALIDPARAMETER：无效参数

```
1.  bStatus_t GAPBondMgr_PasscodeRsp( uint16_t connectionHandle, uint8_t status,
    uint32_t passcode )
```

响应密码请求。

参数	描述
connectionHandle	连接句柄
status	SUCCESS：密码可用 其他详见 SMP_PAIRING_FAILED_DEFINES
passcode	整数值密码
返回	SUCCESS：绑定记录已找到且更改 bleIncorrectMode：未发现连接

8.6.2 配置参数

常用配置参数如下表所示，详细的参数 ID 请参考 CH58xBLE.LIB.h。

参数 ID	读写	大小	描述
GAPBOND_PERI_PAIRI	可读	uint8	配对的方式，默认为：

NG_MODE	可写		GAPBOND_PAIRING_MODE_WAIT_FOR_REQ
GAPBOND_PERI_DEFAULT_PASSCODE		uint32	默认的中间人保护密钥，范围：0-999999，默认为 0。
GAPBOND_PERI_MITM_PROTECTION	可读可写	uint8	中间人（MITM）保护。默认为 0，关闭中间人保护。
GAPBOND_PERI_IO_CAPABILITIES	可读可写	uint8	I/O 能力，模默认为：GAPBOND_IO_CAP_DISPLAY_ONLY，即设备仅能现实。
GAPBOND_PERI_BONDING_ENABLED	可读可写	uint8	如启用，则在配对过程中请求绑定。默认为 0，不请求绑定。

8.7 RF PHY API

8.7.1 指令

```
1.  bStatus_t RF_RoleInit( void )
```

RF 协议栈初始化。

参数	描述
返回	SUCCESS：初始化成功

```
1.  bStatus_t RF_Config( rfConfig_t *pConfig )
```

RF 参数配置。

参数	描述
pConfig	指向配置参数的指针
返回	SUCCESS

```
1.  bStatus_t RF_Rx( uint8_t *txBuf, uint8_t txLen, uint8_t pktRxType, uint8_t
    pktTxType )
```

RF 接受数据函数：将 RF PHY 配置为接受状态，接收到数据后需重新配置。

参数	描述
txBuf	自动模式下，指向 RF 收到数据后返回的数据的指针
txLen	自动模式下，RF 收到数据后返回的数据的长度（0-251）
pkRxType	接受的数据包类型（0xFF：接受所有类型数据包）
pkTxType	自动模式下，RF 收到数据后返回的数据的数据包类型
返回	SUCCESS

```
1.  bStatus_t RF_Tx( uint8_t *txBuf, uint8_t txLen, uint8_t pktTxType, uint8_t p
    ktRxType )
```

RF 发送数据函数。

参数	描述
txBuf	指向 RF 发送数据的指针
txLen	RF 发送数据的数据长度（0-251）
pkTxType	发送的数据包的类型
pkRxType	自动模式下，RF 发送数据后接收数据的数据类型（0xFF：接受所有类型数据包）
返回	SUCCESS

```
1.  bStatus_t RF_Shut( void )
```

关闭 RF，停止发送或接收。

参数	描述
返回	SUCCESS

```
1.  uint8_t RF_FrequencyHoppingTx( uint8_t resendCount )
```

RF 发送端开启跳频

参数	描述
resendCount	发送 HOP_TX pdu 的最大计数（0 ：无限制）
返回	0：SUCCESS

```
1.  uint8_t RF_FrequencyHoppingRx( uint32_t timeoutMS );
```

RF 接收端开启跳频

参数	描述
timeoutMS	等待接收 HOP_TX pdu 的最长时间（Time = n * 1ms，0：无限制）
返回	0：SUCCESS 1：Failed 2：LLEMode error(需要处于自动模式)

```
1.  void RF_FrequencyHoppingShut( void )
```

关闭 RF 跳频功能

8.7.2 配置参数

RF 配置参数 rfConfig_t 描述如下：

参数	描述
LLEMode	LLE_MODE_BASIC：Basic 模式，发送或接受结束后进入空闲模式 LLE_MODE_AUTO：Auto 模式，在发送完成后自动切换至接收模式

	LLE_MODE_EX_CHANNEL：切换至 Frequency 配置频段 LLE_MODE_NON_RSSI：将接收数据的第一字节设置为包类型
Channel	RF 通信通道（0-39）
Frequency	RF 通信频率（2400000KHz-2483500KHz），建议不使用超过 24 次位翻转且连续的 0 或 1 不超过 6 个
AccessAddress	RF 通信地址
CRCInit	CRC 初始值
RFStatusCB	RF 状态回调函数
ChannelMap	频道图，每一位对应一个频道。位值为 1 代表频道有效，反之无效。频道按位递增且频道 0 对应第 0 位
Resv	保留
HeartPeriod	心跳包间隔，为 100ms 的整数倍
HopPeriod	跳跃周期（T=32n*RTC 时钟），默认为 8
HopIndex	数据信道选择算法中跳频信道间隔值，默认为 17
RxMaxlen	RF 模式下接收的最大数据长度，默认 251
RxMaxlen	RF 模式下传输的最大数据长度，默认 251

8.7.3 回调函数

```
1.  void RF_2G4StatusCallBack( uint8_t sta , uint8_t crc, uint8_t *rxBuf )
```

RF 状态回调函数，发送或接收完成均会进入此回调函数。
注：不可在此函数中直接调用 RF 接收或者发送 API，需要使用事件的方式调用。

参数	描述
sta	RF 收发状态
crc	数据包状态校验，每一位表征不同的状态： bit0：数据 CRC 校验错误； bit1：数据包类型错误；
rxBuf	指向接收到的数据的指针
返回	NULL