# 消息系统接口文档

## 初始化接口

|  |  |
| --- | --- |
| **void** **init\_msg\_system**(); | |
| 参数 | 无 |
| 返回值 | 无 |
| 说明 | 应用程序调用该接口来初始化消息系统，该接口无输入参数，也无返回值，如果初始化成功，则程序会继续执行，如果该调用失败，应用程序将退出。 |

## 异步消息发送接口

|  |  |
| --- | --- |
| error\_no\_t **send\_msg**(uint8 dst\_grp\_id,  uint16 dst\_app\_id,  uint32 msg\_id,  uint8 msg\_priority,  **void**\* msg\_content,  uint32 msg\_len); | |
| 参数 | dst\_grp\_id 消息目的地的组ID；  dst\_app\_id 消息目的地的应用程序ID  msg\_id 消息ID  msg\_priority 消息的优先级  msg\_content 指向要发送的消息内容的指针  msg\_len 消息内容的长度 |
| 返回值 | 成功时返回 SUCCESS\_EC  失败时返回类型为error\_no\_t的错误码，错误码定义在error\_code.h中 |
| 说明 | 应用程序调用该接口来发送消息。 |

## 消息接收接口

|  |  |
| --- | --- |
| **void** **receive\_msg**(message\_t\*\* msg); | |
| 参数 | msg 指向接收到的消息的指针的指针，保存消息的空间由该接口分配，由用户释放 |
| 返回值 | 无返回值。该接口为阻塞接口，仅在收到一个消息后返回，返回后参数msg会保存指向接收到的消息的指针。 |
| 说明 | 应用程序调用该接口来接收消息。该函数仅在接收到消息后返回，且返回后参数msg仅含有指向一个接收到的消息的指针的指针。该接口会申请保存消息的空间，用户可直接通过 (\*msg)->body来获取消息内容。当用户不再需要该消息时，需要调用接口 **void** **free\_msg\_buff**(message\_t\*\* msg)来释放该消息。 |

## 同步消息发送接口

|  |  |
| --- | --- |
| error\_no\_t **send\_sync\_msg**(uint8 dst\_grp\_id,  uint16 dst\_app\_id,  uint32 msg\_id,  uint8 msg\_priority,  **void**\* msg\_content,  uint32 msg\_len,  uint32 time\_out,  **void**\* rsp\_buf,  uint32 rsp\_buf\_size); | |
| 参数 | dst\_grp\_id 消息目的地的组ID；  dst\_app\_id 消息目的地的应用程序ID  msg\_id 消息ID  msg\_priority 消息的优先级  msg\_content 指向要发送的消息内容的指针  msg\_len 消息内容的长度  time\_out 超时时间，单位为毫秒，如果超过该时间未收到对端的响应消息，则函数会返回，错误码中包含错误原因。  rsp\_buf 保存响应消息的缓存，由用户提供该缓存空间  rsp\_buf\_size 保存返回消息的缓存空间的大小 |
| 返回值 | 成功时返回 SUCCESS\_EC  失败时返回类型为error\_no\_t的错误码 |
| 说明 | 应用程序调用该接口来发送同步消息。该接口需收到远端的响应才返回，或者是等到超时后返回。 |

## 同步消息响应接口

|  |  |
| --- | --- |
| error\_no\_t **send\_sync\_msg\_rsp**(message\_t\* syn\_req\_msg,  **void**\* rsp\_content,  uint32 rsp\_content\_size); | |
| 参数 | syn\_req\_msg 接收到的同步消息的指针；  rsp\_content 指向要发送的响应消息的指针  rsp\_content\_size 要发送的响应消息的大小 |
| 返回值 | 成功时返回 SUCCESS\_EC  失败时返回类型为error\_no\_t的错误码 |
| 说明 | 应用程序调用该接口来发送同步消息的响应消息。 |

## 示例程序

### 6.1 发送消息示例

|  |
| --- |
| int main(int argc, char\*\* argv) {  if(argc != 2) {  printf("Usage: %s dest\_group\_id dest\_app\_id \n", argv[0]);  exit(0);  }  uint8 dst\_grp\_id = (uint8)atoi(argv[1]);  uint16 dst\_app\_id = (uint16)atoi(argv[2]);  int msg\_content = 1;  int rsp = 0;  int8 msg\_priority = 1;  uint32 time\_out = 10000;  init\_sys\_info();  init\_msg\_system();  int ret = send\_sync\_msg(dst\_grp\_id,  dst\_app\_id,  MSG\_ID\_TEST\_SYNC\_MSG,  msg\_priority,  &msg\_content,  sizeof(msg\_content),  time\_out,  &rsp,  sizeof(rsp));  if(SUCCESS\_EC == ret) {  LOG(INFO,"Received sync message response and the rsp is %d",rsp);  } else {  LOG(ERROR, "Remote party didn't response the the sync message request, error code is %d", ret);  }  ret = send\_msg(dst\_grp\_id,  dst\_app\_id,  MSG\_ID\_TEST\_ASYNC\_MSG,  msg\_priority,  &msg\_content,  sizeof(msg\_content));  if(SUCCESS\_EC == ret) {  LOG(INFO,"Message send successfully");  } else {  LOG(ERROR, "Send message failed, error code is %d", ret);  }  return 0;  } |

### 6.2 接收消息示例

|  |
| --- |
| int main() {  init\_sys\_info();  init\_msg\_system();  message\_t\* msg = NULL;  int ret = 1;  while(1) {  receive\_msg(&msg);  switch(MSG\_ID(msg)) {  case MSG\_ID\_TEST\_SYNC\_MSG:  LOG(INFO, "Received a sync message, begin to send rsp msg...");  send\_sync\_msg\_rsp(msg, &ret, sizeof(ret));  break;  case MSG\_ID\_TEST\_ASYNC\_MSG:  LOG(INFO, "Received an async message.");  /\*Access the message content by the pointer of msg->body \*/  break;  default:  LOG(INFO, "Message id %d is received but not processed", MSG\_ID(msg));  print\_msg\_header(msg);  break;  }  free\_msg\_buff(&msg);  }  return 0;  } |

## 限制

1. 对于消息接收端，仅使用receive\_msg 接口来接收消息，需要根据消息号来区分是否为同步消息，如果是同步消息，则需要使用send\_sync\_msg\_rsp来发送回应消息。消息接收端在调用receive\_msg并使用完该消息后需要调用free\_msg\_buff来释放消息，否则会造出内存泄露。
2. receive\_msg 接口为阻塞接口，返回时传入的指针会指向一个消息队列中最早接收到的消息。
3. 消息优先级暂时还未支持。
4. 同步消息接口可工作于多线程环境中。消息的接收统一在receive\_msg接口中，即发送到该程序的不同线程的消息都会接收到该接口中，如果需要每个线程都支持receive\_msg， 则需进一步开发。下一步计划实现该功能。