CoLink API

0.10.0

制作者 Doxygen 1.8.14

Contents

Chapter 1

模块索引

1.1 模块

这里列出了所有模块:

Chapter 2

结构体索引

2.1 结构体

这里列出了所有结构体，并附带简要说明:

Chapter 3

文件索引

3.1 文件列表

这里列出了所有文档化的文件，并附带简要说明:

Chapter 4

模块说明

4.1 ColinkCjsonType

宏定义

- #define **C_NULL** (0)
- #define **C_FALSE** (1)
- #define **C_TRUE** (2)
- #define **C_NUMBER** (3)
- #define **C_STRING** (4)
- #define **C_ARRAY** (5)
- #define **C_OBJECT** (6)

4.1.1 详细描述

4.2 ColinkCjsonParseErrCode

宏定义

- #define **C_PARSE_OK** (0)
- #define **C_PARSE_NODATA** (1)
- #define **C_PARSE_EXPECT_VALUE** (2)
- #define **C_PARSE_INVALID_VALUE** (3)
- #define **C_PARSE_ROOT_NOT_SINGULAR** (4)
- #define **C_PARSE_NUMBER_TOO_BIG** (5)
- #define **C_PARSE_MISS_QUOTATION_MARK** (6)
- #define **C_PARSE_INVALID_STRING_ESCAPE** (7)
- #define **C_PARSE_INVALID_STRING_CHAR** (8)
- #define **C_PARSE_INVALID_UNICODE_HEX** (9)
- #define **C_PARSE_INVALID_UNICODE_SURROGATE** (10)
- #define **C_PARSE_MISS_COMMA_OR_SQUARE_BRACKET** (11)
- #define **C_PARSE_MISS_KEY** (12)
- #define **C_PARSE_MISS_COLON** (13)
- #define **C_PARSE_MISS_COMMA_OR_CURLY_BRACKET** (14)

4.2.1 详细描述

4.3 ColinkInitErrorCode

宏定义

- #define COLINK_INIT_NO_ERROR (0)
  无错误
- #define COLINK_INIT_FAILED (-1)
  初始化失败
- #define COLINK_INIT_ARG_INVALID (-2)
  无效的参数

4.3.1 详细描述

4.4 ColinkErrorCode

宏定义

- #define COLINK_NO_ERROR (0)
  无错误
- #define COLINK_ARG_INVALID (-2)
  无效的参数
- #define COLINK_JSON_INVALID (-3)
  无效的*JSON*格式
- #define COLINK_JSON_CREATE_ERR (-4)
  创建*JSON*对象错误
- #define COLINK_DATA_SEND_ERROR (-5)
  发送数据出错
- #define COLINK_DEV_TYPE_ERROR (-6)
  设备类型错误

4.4.1 详细描述

4.5 ColinkProcessErrorCode

宏定义

- #define COLINK_PROCESS_NO_ERROR (0)
  无错误
- #define COLINK_PROCESS_INIT_INVALID (-12)
  *colinkInit*未初始化成功
- #define COLINK_PROCESS_TIMEOUT (-13)
  长时间未被调用*colinkProcess*
- #define COLINK_PROCESS_MEMORY_ERROR (-14)
  内存分配错误

4.5.1 详细描述

4.6 ColinkTcpErrorCode

宏定义

- #define COLINK_TCP_NO_ERROR (0)
  无错误
- #define COLINK_TCP_ARG_INVALID (-2)
  无效的参数
- #define COLINK_TCP_CREATE_CONNECT_ERR (-21)
  创建*TCP*连接错误
- #define COLINK_TCP_SEND_ERR (-23)
  *TCP*发送失败
- #define COLINK_TCP_READ_ERR (-25)
  *TCP*读取失败
- #define COLINK_TCP_CONNECT_TIMEOUT (-26)
  *TCP*连接超时
- #define COLINK_TCP_CONNECT_ERR (-27)
  *TCP*连接失败
- #define COLINK_TCP_CONNECTING (-28)
  *TCP*连接中
- #define COLINK_TCP_READ_INCOMPLETED (-29)
  *TCP*读包不完整
- #define COLINK_TCP_CREATE_SERVER_ERR (-31)
  创建*TCP*服务错误
- #define COLINK_TCP_SERVER_WAIT_CONNECT (-32)
  *tcp*等待客户端连接
- #define COLINK_TCP_DNS_PARSING (-35)
  *DNS*解析中
- #define COLINK_TCP_DNS_PARSE_ERR (-36)
  *DNS*解析失败
- #define COLINK_TCP_DNS_PARSE_TIMEOUT (-37)
  *DNS*解析超时

4.6.1 详细描述

4.7 ColinkReqResultCode

宏定义

- #define COLINK_REQ_RESULT_NO_ERROR (0)
  操作成功
- #define COLINK_REQ_RESULT_NUMBER_ERROR (-414)
  子设备数量超过限制

4.7.1 详细描述

4.8 ColinkOtaResCode

宏定义

- #define COLINK_OTA_NO_ERROR (0)
  *OTA*升级成功
- #define COLINK_OTA_DOWNLOAD_ERROR (-404)
  *OTA*文件下载失败
- #define COLINK_OTA_MODEL_ERROR (-406)
  设备型号不正确
- #define COLINK_OTA_DIGEST_ERROR (-409)
  固件校验失败

4.8.1 详细描述

4.9 ColinkLinkErrorCode

宏定义

- #define COLINK_LINK_NO_ERROR (0)
  无错误
- #define COLINK_LINK_ARG_INVALID (-2)
  无效的参数
- #define COLINK_LINK_OPERATION_ERROR (-50)
  流程操作不对

4.9.1 详细描述

4.10 ColinkGatewayErrorCode

宏定义

- #define COLINK_GATEWAY_NO_ERROR (0)
  无错误
- #define COLINK_GATEWAY_OPERATE_FAILED (-1)
  操作失败
- #define COLINK_GATEWAY_ARG_INVALID (-2)
  无效的参数
- #define COLINK_GATEWAY_NET_ERROR (-100)
  网络异常

4.10.1 详细描述

4.11 ColinkSubDevResultCode

宏定义

- #define COLINK_SUB_DEV_NO_ERROR (0)
  操作成功
- #define COLINK_SUB_DEV_NUMBER_ERROR (-414)
  子设备数量超过限制

4.11.1 详细描述

4.12 CoLinkLinkState

宏定义

- #define COLINK_LINK_INIT_INVALID (0)
  未复位配网初始状态
- #define COLINK_LINK_RES_DEV_INFO_OK (1)
  响应设备信息成功
- #define COLINK_LINK_RES_DEVICEID_ERROR (2)
  响应设备信息失败
- #define COLINK_LINK_GET_INFO_OK (3)
  获取配网信息成功
- #define COLINK_LINK_GET_INFO_ERROR (4)
  获取配网信息失败

4.12.1 详细描述

4.13 ColinkDevType

宏定义

- #define COLINK_SINGLE (0)
  普通设备
- #define COLINK_GATEWAY (1)
  网关设备

4.13.1 详细描述

4.14 ColinkDevStatus

宏定义

- #define DEVICE_OFFLINE (0)
  设备离线
- #define DEVICE_ONLINE (1)
  设备在线
- #define DEVICE_UNREGISTERED (2)
  设备未注册

4.14.1 详细描述

4.15 ColinkTimerType

宏定义

- #define **COLINK_SINGLE_TIMER** (0)
- #define **COLINK_REPEAT_TIMER** (1)

4.15.1 详细描述

4.16 ColinkTimerStatus

宏定义

- #define **COLINK_TIMER_CANCEL** (0)
- #define **COLINK_TIMER_RUNNING** (1)
- #define **COLINK_TIMER_TIMEOUT** (2)
- #define **COLINK_TIMER_NOT_EXIST** (3)

4.16.1 详细描述

4.17 ColinkTimerErrorCode

宏定义

- #define **COLINK_TIMER_NO_ERR** (0)
- #define **COLINK_TIMER_OPERATE_FAILED** (1)
- #define **COLINK_TIMER_NOT_FIND** (2)

4.17.1 详细描述

Chapter 5

结构体说明

5.1 cjson_member结构体 参考

表示json对象数据的结构类型

```
#include <colink_cjson.h>
```

成员变量

- char * k
  表示对象中的*key*
- uint32_t klen
  表示对象中的*key*的长度
- cjson_value v
  表示对象中*value*
- cjson_member * next
  指向对象中的子对象

5.1.1 详细描述

表示json对象数据的结构类型

该结构体的文档由以下文件生成:

- include/colink_cjson.h

5.2 cjson_value结构体 参考

表示json数据的结构类型

```
#include <colink_cjson.h>
```

成员变量

- union {
  struct {
  cjson_member * **m**
  uint32_t **size**
  } o
  表示一个对象
  struct {
  cjson_value * **e**
  uint32_t **size**
  } a
  表示一个数组
  struct {
  char * **s**
  uint32_t **len**
  } s
  表示一个字符串
  double n
  表示一个整形或浮点行数据
  } **u**

- int32_t type
  表示*true*、*false*、*NULL*类型
- cjson_value * next
  指向子对象的指针

5.2.1 详细描述

表示json数据的结构类型

该结构体的文档由以下文件生成:

- include/colink_cjson.h

5.3 ColinkDev结构体 参考

设备信息的结构体。

```
#include <colink_profile.h>
```

成员变量

- char deviceid [11]
  设备*ID*
- char apikey [37]
  设备密钥
- char model [11]
  设备型号

- char distor_domain [32]
  分配服务器域名，由*APP*下发给设备
- uint16_t distor_port
  分配服务器的端口号，由*APP*下发给设备
- char version [12]
  固件版本
- bool ssl_enable
  是否使能*SSL*
- int32_t dev_type
  设备类型，参见*ColinkDevType*
- uint32_t server_res_timeout
  服务器响应超时
- int32_t **__pad** [2]

5.3.1 详细描述

设备信息的结构体。

参见

ColinkDevType

该结构体的文档由以下文件生成:

- include/colink_profile.h

5.4 ColinkEvent结构体 参考

设备事件回调的结构体。

```
#include <colink_profile.h>
```

成员变量

- void(∗ colinkRecvUpdateCb)(char ∗data)
  接收*update*字段的回调函数。
- void(∗ colinkNotifyDevStatusCb)(int32_t status)
  设备状态发生改变时的回调函数。
- void(∗ colinkUpgradeRequestCb)(char ∗new_ver, ColinkOtaInfo file_list[ ], uint8_t file_num, char ∗sequence)
  升级通知的回调函数。
- void(∗ colinkSendUpdateCb)(int32_t error_code)
  发送*update*字段数据的回调函数。
- void(∗ colinkSendUTCRequestCb)(int32_t error_code, char utc_str[ ])
  向服务器请求*UTC*时间的回调函数。
- void(∗ colinkSendQueryCb)(int32_t error_code, char ∗params)
  向服务器发起查询申请的回调函数。
- void(∗ colinkRecvResetDispatchRequestCb)(char ∗dispatchUrl, uint16_t dispatchPort)
  服务器请求重置分配服务器
- void(∗ colinkRecvResetDispatchRegionRequestCb)(char ∗region)
  服务器请求重置分配服务器区域

5.4.1 详细描述

设备事件回调的结构体。

5.4.2 结构体成员变量说明

5.4.2.1 colinkNotifyDevStatusCb

```
void(* colinkNotifyDevStatusCb) (int32_t status)
```

设备状态发生改变时的回调函数。

描述:

当设备状态发生改变时会产生回调函数。

参数

status	[IN] 设备状态。参见ColinkDevStatus。

参见

ColinkDevStatus

5.4.2.2 colinkRecvResetDispatchRegionRequestCb

```
void(* colinkRecvResetDispatchRegionRequestCb) (char *region)
```

服务器请求重置分配服务器区域

描述:

当分配服务器不可用时，长连接服务器会下发该请求，请求重置分配服务器区域。

参数

region	[IN] 新的分配服务器区域

返回值

无	

5.4.2.3 colinkRecvResetDispatchRequestCb

```
void(* colinkRecvResetDispatchRequestCb) (char *dispatchUrl, uint16_t dispatchPort)
```

服务器请求重置分配服务器

描述:

当分配服务器不可用时，长连接服务器会下发该请求，请求重置分配服务器。

参数

dispatchUrl	[IN] 新的分配服务器链接
dispatchPort	[IN] 新的分配服务器端口

返回值

无	

5.4.2.4 colinkRecvUpdateCb

```
void(* colinkRecvUpdateCb) (char *data)
```

接收update字段的回调函数。

描述:

当APP需要改变设备状态时，通过此回调函数获取数据。

参数

data	[IN] 收到的字符串数据，以Json格式表示。

5.4.2.5 colinkSendQueryCb

```
void(* colinkSendQueryCb) (int32_t error_code, char *params)
```

向服务器发起查询申请的回调函数。

描述:

当设备调用colinkSendQuery发送查询请求后， 通过此回调函数获取请求结果。

参数

error_code	[IN] 收到服务器响应的错误码。参见ColinkReqResultCode。
params	[IN] 查询返回的结果。

参见

ColinkReqResultCode

5.4.2.6 colinkSendUpdateCb

```
void(* colinkSendUpdateCb) (int32_t error_code)
```

发送update字段数据的回调函数。

描述:

当发送update字段数据后需要调用colinkSendUpdateCb获取服务器应答。

参数

new_ver	[IN] 新固件的版本号。

5.4.2.7 colinkSendUTCRequestCb

```
void(* colinkSendUTCRequestCb) (int32_t error_code, char utc_str[])
```

向服务器请求UTC时间的回调函数。

描述:

当设备调用colinkSendUTCRequest发送获取UTC时间后， 通过此回调函数获取请求结果。

参数

error_code	[IN] 收到服务器响应的错误码。参见ColinkReqResultCode。
utc_str	[IN] UTC时间字符串。

参见

ColinkReqResultCode

5.4.2.8 colinkUpgradeRequestCb

```
void(* colinkUpgradeRequestCb) (char *new_ver, ColinkOtaInfo file_list[], uint8_t file_num,
char *sequence)
```

升级通知的回调函数。

描述:

当APP发出升级通知后，通过此回调函数获取升级信息。 当升级完成后需要调用colinkUpgradeRes通知服务器升级完成。

参数

new_ver	[IN] 新固件的版本号。
file_list	[IN] OTA固件信息列表。
file_num	[IN] OTA固件的数量。
sequence	[IN] 唯一标识字符串，需要保存下来， 在通过colinkUpgradeRes通知服务器时传入

该结构体的文档由以下文件生成:

- include/colink_profile.h

5.5 ColinkGatewayEvent结构体 参考

成员变量

- void(* colinkSubDevRecvReqCb)(ColinkSubDevice *sub_dev, char *data, char req_sequence[])
  接收请求数据的回调函数。
- void(* colinkSubDevRecvResCb)(ColinkSubDevice *sub_dev, int32_t error_code)
  接收服务器响应的回调函数。
- void(* colinkAddSubDevResultCb)(ColinkSubDevice sub_dev[], int32_t error_code[], uint16_t num)
  注册新子设备时服务器响应的回调函数。
- void(* colinkDelSubDevResultCb)(ColinkSubDevice sub_dev[], int32_t error_code[], uint16_t num)
  删除子设备时服务器响应的回调函数。
- void(* colinkOnlineSubDevResultCb)(ColinkSubDevice sub_dev[], int32_t error_code[], uint16_t num)
  上报子设备在线时服务器响应的回调函数。
- void(* colinkOfflineSubDevResultCb)(ColinkSubDevice sub_dev[], int32_t error_code[], uint16_t num)
  上报子设备离线时服务器响应的回调函数。
- void(* colinkServerDelSubDevCb)(ColinkSubDevice *sub_dev)
  服务器发送删除子设备指令，通知网关处理的回调函数。
- void(* colinkReportSubDevStateCb)(int32_t error_code)
  网关设备批量上报子设备状态时服务器响应的回调函数。
- void(* colinkRecvReportSubDevStateCb)(void)
  网关设备收到服务器发来的批量上报子设备状态请求时的回调函数。

5.5.1 结构体成员变量说明

5.5.1.1 colinkAddSubDevResultCb

```
void(* colinkAddSubDevResultCb) (ColinkSubDevice sub_dev[], int32_t error_code[], uint16_↩
t num)
```

注册新子设备时服务器响应的回调函数。

描述:

当调用colinkAddSubDev注册多个新子设备，随后产生此回调函数来获取服务器响应的结果。 子设备可以通过此回调判断是否注册成功。

参数

sub_dev	[IN] 子设备对应的数组列表。
error_code	[IN] 收到服务器响应的错误码列表。参见ColinkSubDevResultCode。
num	[IN] 子设备数量。

参见

ColinkSubDevResultCode

5.5.1.2 colinkDelSubDevResultCb

```
void(* colinkDelSubDevResultCb) (ColinkSubDevice sub_dev[], int32_t error_code[], uint16_↩
t num)
```

删除子设备时服务器响应的回调函数。

描述:

当调用colinkDelSubDev删除多个子设备，随后产生此回调函数来获取服务器响应的结果。 子设备可以通过此回调判断是否删除成功。

参数

sub_dev	[IN] 子设备对应的数组列表。
error_code	[IN] 收到服务器响应的错误码列表。参见ColinkSubDevResultCode。
num	[IN] 子设备数量。

参见

ColinkSubDevResultCode

5.5.1.3 colinkOfflineSubDevResultCb

```
void(* colinkOfflineSubDevResultCb) (ColinkSubDevice sub_dev[], int32_t error_code[], uint16↩
_t num)
```

上报子设备离线时服务器响应的回调函数。

描述:

当调用colinkOfflineSubDev上报多个子设备离线，随后产生此回调函数来获取服务器响应的结果。 子设备可以通过此回调判断是否删除成功。

参数

sub_dev	[IN] 子设备对应的数组列表。
error_code	[IN] 收到服务器响应的错误码列表。参见ColinkSubDevResultCode。
num	[IN] 子设备数量。

参见

ColinkSubDevResultCode

5.5.1.4 colinkOnlineSubDevResultCb

```
void(* colinkOnlineSubDevResultCb) (ColinkSubDevice sub_dev[], int32_t error_code[], uint16_t
num)
```

上报子设备在线时服务器响应的回调函数。

描述:

当调用colinkOnlineSubDev上报多个子设备在线，随后产生此回调函数来获取服务器响应的结果。 子设备可以通过此回调判断是否删除成功。

参数

sub_dev	[IN] 子设备对应的数组列表。
error_code	[IN] 收到服务器响应的错误码列表。参见ColinkSubDevResultCode。
num	[IN] 子设备数量。

参见

ColinkSubDevResultCode

5.5.1.5 colinkRecvReportSubDevStateCb

```
void(* colinkRecvReportSubDevStateCb) (void)
```

网关设备收到服务器发来的批量上报子设备状态请求时的回调函数。

描述:

当服务器需要网关设备批量上报子设备状态时，会产生此回调，需将本地子设备状态上报给服务器。

参数

无	

5.5.1.6 colinkReportSubDevStateCb

```
void(* colinkReportSubDevStateCb) (int32_t error_code)
```

网关设备批量上报子设备状态时服务器响应的回调函数。

描述:

当调用colinkGatewayReportSubDevState批量上报子设备状态后，随后产生此回调来获取服务器响应结果。

参数

error_code	[IN] 收到服务器响应的错误码。参见ColinkSubDevResultCode。

参见

ColinkSubDevResultCode

5.5.1.7 colinkServerDelSubDevCb

```
void(* colinkServerDelSubDevCb) (ColinkSubDevice *sub_dev)
```

服务器发送删除子设备指令，通知网关处理的回调函数。

描述:

已注册当前网关的子设备连接其它网关或者app端删除子设备的时候，服务器会发送删除子设备指令，通过回调函数来得知哪些子设备已被删除。

参数

sub_dev	[IN] 通知删除的子设备。

5.5.1.8 colinkSubDevRecvReqCb

```
void(* colinkSubDevRecvReqCb) (ColinkSubDevice *sub_dev, char *data, char req_sequence[])
```

接收请求数据的回调函数。

描述:

当APP需要改变设备状态时，子设备通过此回调函数获取数据。 需要调用colinkSubDevSendRes来响应子设备是否操作成功。

参数

sub_dev	[IN] 子设备对应的指针。
data	[IN] 收到请求的数据。
req_sequence	[IN] 收到的sequence。

5.5.1.9 colinkSubDevRecvResCb

```
void(* colinkSubDevRecvResCb) (ColinkSubDevice *sub_dev, int32_t error_code)
```

接收服务器响应的回调函数。

描述:

当调用colinkSubDevSendReq发送数据，随后产生此回调函数来获取服务器响应的结果。 子设备可以通过此回调判断数据是否发送成功。

参数

sub_dev	[IN] 子设备对应的指针。
error_code	[IN] 收到服务器响应的错误码。参见ColinkSubDevResultCode。

参见

ColinkSubDevResultCode

该结构体的文档由以下文件生成:

- include/colink_gateway_profile.h

5.6 ColinkLinkInfo结构体 参考

配网信息结构体。

```
#include <colink_link.h>
```

成员变量

- char ssid [32]
  路由器*ssid*
- char password [64]
  路由器密码
- char distor_domain [32]
  分配服务器域名
- uint16_t distor_port
  分配服务器端口

5.6.1 详细描述

配网信息结构体。

该结构体的文档由以下文件生成:

- include/colink_link.h

5.7 ColinkOtaInfo结构体 参考

OTA固件信息的结构体。

```
#include <colink_profile.h>
```

成员变量

- char name [20]
  固件名称
- char download_url [150]
  下载链接
- char digest [65]
  固件的*SHA256*值

5.7.1 详细描述

OTA固件信息的结构体。

该结构体的文档由以下文件生成:

- include/colink_profile.h

5.8 ColinkSubDevAddr结构体 参考

子设备地址的结构体。

`#include <colink_gateway_profile.h>`

成员变量

- uint8_t mac [8]
  子设备*MAC*地址,仅支持*6*字节和*8*字节*mac*
- uint32_t port
  子设备端口

5.8.1 详细描述

子设备地址的结构体。

该结构体的文档由以下文件生成:

- include/colink_gateway_profile.h

5.9 ColinkSubDevice结构体 参考

子设备的结构体。

`#include <colink_gateway_profile.h>`

成员变量

- ColinkSubDevAddr addr
  子设备地址
- uint32_t uiid
  子设备*uiid*
- bool onlineState
  子设备在线状态

5.9.1 详细描述

子设备的结构体。

该结构体的文档由以下文件生成:

- include/colink_gateway_profile.h

5.10 ColinkSubDeviceList结构体 参考

子设备列表的结构体。

`#include <colink_gateway_profile.h>`

成员变量

- ColinkSubDevice dev
  子设备
- char deviceid [11]
  子设备*id*

5.10.1 详细描述

子设备列表的结构体。

该结构体的文档由以下文件生成:

- include/colink_gateway_profile.h

Chapter 6

文件说明

6.1 include/colink_cjson.h 文件参考

Show profile of colink_cJSON.

```
#include <stddef.h>
#include "stdint.h"
```

结构体

- struct cjson_value
  表示*json*数据的结构类型
- struct cjson_member
  表示*json*对象数据的结构类型

宏定义

- #define **C_NULL** (0)
- #define **C_FALSE** (1)
- #define **C_TRUE** (2)
- #define **C_NUMBER** (3)
- #define **C_STRING** (4)
- #define **C_ARRAY** (5)
- #define **C_OBJECT** (6)
- #define **C_PARSE_OK** (0)
- #define **C_PARSE_NODATA** (1)
- #define **C_PARSE_EXPECT_VALUE** (2)
- #define **C_PARSE_INVALID_VALUE** (3)
- #define **C_PARSE_ROOT_NOT_SINGULAR** (4)
- #define **C_PARSE_NUMBER_TOO_BIG** (5)
- #define **C_PARSE_MISS_QUOTATION_MARK** (6)
- #define **C_PARSE_INVALID_STRING_ESCAPE** (7)
- #define **C_PARSE_INVALID_STRING_CHAR** (8)
- #define **C_PARSE_INVALID_UNICODE_HEX** (9)
- #define **C_PARSE_INVALID_UNICODE_SURROGATE** (10)
- #define **C_PARSE_MISS_COMMA_OR_SQUARE_BRACKET** (11)
- #define **C_PARSE_MISS_KEY** (12)
- #define **C_PARSE_MISS_COLON** (13)
- #define **C_PARSE_MISS_COMMA_OR_CURLY_BRACKET** (14)

类型定义

- typedef struct cjson_value cjson_value
- typedef struct cjson_member **cjson_member**

函数

- int32_t cjson_parse (cjson_value *v, const char *json)
  解析*json*字符串。
- char * cjson_stringify (const cjson_value *v, uint32_t *length)
  生成*json*字符串。
- void cjson_free (cjson_value **v)
  释放*json*数据资源。
- cjson_value * cjson_create_object (void)
  创建一个*json*对象。
- cjson_value * cjson_create_array (void)
  创建一个*json*数组对象。
- void cjson_object_additem (cjson_value *object, const char *key, cjson_value *val)
  往对象中添加成员。
- void cjson_array_additem (cjson_value *object, cjson_value *val)
  往数组对象中添加成员。
- cjson_value * cjson_create_boolean (int32_t b)
  创建一个*json*布尔值对象。
- cjson_value * cjson_create_number (double b)
  创建一个*json*整形数或浮点数对象。
- cjson_value * cjson_create_string (const char *string, uint32_t len)
  创建一个*json*字符串对象。
- void cjson_set_boolean (cjson_value *v, int32_t b)
  修改布尔值对象数值。
- void cjson_set_number (cjson_value *v, double n)
  修改数值对象数值。
- void cjson_set_string (cjson_value *v, const char *s, uint32_t len)
  修改字符串对象数值。
- int32_t cjson_get_type (const cjson_value *v)
  获取*json*对象类型。
- int32_t cjson_get_boolean (const cjson_value *v)
  获取*json*尔布对象数据。
- double cjson_get_number (const cjson_value *v)
  获取*json*数值对象数据。
- const char * cjson_get_string (const cjson_value *v)
  获取*json*字符串对象数据。
- uint32_t cjson_get_string_length (const cjson_value *v)
  获取*json*字符串对象中字符串的长度。
- uint32_t cjson_get_array_size (const cjson_value *v)
  获取*json*数组对象中数组的长度。
- cjson_value * cjson_get_array_element (const cjson_value *v, uint32_t index)
  根据索引值，获取*json*数组对象中某个成员的对象。
- uint32_t cjson_get_object_size (const cjson_value *v)
  获取*json*对象中子对象的个数。
- const char * cjson_get_object_key (const cjson_value *v, uint32_t index)

根据索引值，获取*json*对象中某个子对象的*key*值。

- uint32_t cjson_get_object_key_length (const cjson_value *v, uint32_t index)
  根据索引值，获取*json*对象中某个子对象的*key*值 的长度。
- cjson_value * cjson_get_object_value (const cjson_value *v, uint32_t index)
  根据索引值，获取*json*对象中某个子对象的*value*值。
- cjson_value * cjson_get_value_Bykey (const cjson_value *v, const char *str)
  根据*key*值，获取*json*对象中某个子对象的对象。

6.1.1 详细描述

Show profile of colink_cJSON.

It shows the detail of colink_cJSON struct and API.

作者

Huang Xiaoming

日期

2018.09.13

版权所有

6.1.2 类型定义说明

6.1.2.1 cjson_value

```
typedef struct cjson_value cjson_value
```

json对象结构

```
This json library encapsulate all data types(including C_NULL,C_FALSE,C_TRUE,C_NUMBER,C_STRING,C_ARRAY,
      C_OBJECT) int32_to a cjson_value.
struct cjson_value:   root_object--------C_NULL
                                        |----C_FALSE/C_TRUE
                                        |----C_NUMBER
                                        |----C_STRING
                                        |----C_ARRAY-------C_NULL
                                                      |----C_FALSE/C_TRUE
                                                      |----C_NUMBER
                                                      |----C_STRING
                                                      |----C_ARRAY------ ......
                                                      |----C_OBJECT----- ......


                                        |----C_OBJECT------C_NULL
                                                      |----C_FALSE/C_TRUE
                                                      |----C_NUMBER
                                                      |----C_STRING
                                                      |----C_ARRAY------ ......
                                                      |----C_OBJECT----- ......
```

example

```
void cjson_test()
{
    char *buf = NULL;
    char test_buf[512] = {"{\"action\":\"update\",\"deviceid\":\"100016a929\",\"apikey\":\"
      fbfaa76f-b05d-4298-b918-09e2e30e2373\",\"userAgent\":\"device\",\"ts\":0,\"params\":{\"rssi\":-44,\"power\":\"0\",\"swit
      :\"off\"},\"from\":\"device\"}"};

    uint32_t len = 0;
    cjson_value *test = NULL;
    cjson_value *test_array = NULL;
    cjson_value *test_object = NULL;

    cjson_value *parse = NULL;
    cjson_value *val = NULL;
    cjson_value *num = NULL;
    int32_t ret;

    int32_t i = 0,j = 0;

    //cjson create data test.....................................................

    test = cjson_create_object();
    PLATFORM_DEBUG(DEBUG_JSON,TAG, "cjson_create_object ok");
    cjson_object_additem(test, (const char *)"1111",
      cjson_create_string((const char *)"2222", (uint32_t)5));
    PLATFORM_DEBUG(DEBUG_JSON,TAG, "cjson_object_additem ok");

    cjson_object_additem(test, (const char *)"aaaa",
      cjson_create_string((const char *)"bbbb", (uint32_t)4));
    PLATFORM_DEBUG(DEBUG_JSON,TAG, "cjson_object_additem ok");

    cjson_object_additem(test, (const char *)"5555",
      cjson_create_string((const char *)"6666", (uint32_t)4));
    PLATFORM_DEBUG(DEBUG_JSON,TAG, "cjson_object_additem ok");

    test_array = cjson_create_array();
    cjson_array_additem(test_array, cjson_create_string((const char
      *)"7777", (uint32_t)4));
    cjson_array_additem(test_array, cjson_create_string((const char
      *)"8888", (uint32_t)4));
    cjson_array_additem(test_array, cjson_create_number(-23244343.232
      43232));
    cjson_array_additem(test_array, cjson_create_boolean(0));
    cjson_array_additem(test_array, cjson_create_boolean(1));

    cjson_object_additem(test, (const char *)"array", test_array);

    test_object = cjson_create_object();
    cjson_object_additem(test_object, (const char *)"xxxx",
      cjson_create_string((const char *)"yyyy", (uint32_t)4));
    PLATFORM_DEBUG(DEBUG_JSON,TAG, "cjson_object_additem ok");

    cjson_object_additem(test, (const char *)"object", test_object);
```

```
    buf = cjson_stringify(test, &len);
    PLATFORM_DEBUG(DEBUG_JSON,TAG, "cjson_stringify ok");

    PLATFORM_DEBUG(DEBUG_JSON,TAG, "buf:%s", buf);

    cFree(buf);

    cjson_free(&test);

    //cjson parse data test.....................................................................

    parse = cjson_create_object();
    ret = cjson_parse(parse,test_buf);
    if (ret == C_PARSE_OK)
    {
        PLATFORM_DEBUG(DEBUG_JSON,TAG,"cjson parse ok");
        len = cjson_get_object_size(parse);
        for (i=0;i<len;i++)
        {
            //find object info By index
            buf = (char*)cjson_get_object_key((const
      cjson_value*)parse,(uint32_t)i);
            PLATFORM_DEBUG(DEBUG_JSON,TAG,"parse key[%d]:%s",i,buf);
            val = cjson_get_object_value(parse,i);
            if (val->type == C_STRING)
            {
                PLATFORM_DEBUG(DEBUG_JSON,TAG,"parse value[%d]:%s",i,val->u.s.s);
            }
            else if (val->type == C_ARRAY)
            {
                PLATFORM_DEBUG(DEBUG_JSON,TAG,"parse array value bein");
                for (j=0;j<val->u.a.size;j++)
                {
                    test_array = cjson_get_array_element(val,j);
                    if (test_array->type == C_STRING)
                    {
                        ;
                    }
                    else if (test_array->type == C_NUMBER)
                    {
                        ;
                    }
                    else if (test_array->type == C_TRUE || test_array->type == C_FALSE)
                    {
                        ;
                    }
                }
            }

            //find object info By key
            val = cjson_get_value_Bykey(parse,(const char*)"action");
            if (val != NULL)
            {
                PLATFORM_DEBUG(DEBUG_JSON,TAG,"action:%s",val->u.s.s);
            }
        }
        buf = cjson_stringify(parse,&len);
        PLATFORM_DEBUG(DEBUG_JSON,TAG,"buf:%s",buf);
        cFree(buf);
    }
    cjson_free(&parse);

#endif
}
```

6.1.3 函数说明

6.1.3.1 cjson_array_additem()

```
void cjson_array_additem (
            cjson_value * object,
            cjson_value * val )
```

往数组对象中添加成员。

描述:

往根数组对象中添加成员。

参数

object	[IN] json根对象数据对象指针，NOT NULL。
val	[IN] 新的数组成员指针，NOT NULL。

返回值

无。	

6.1.3.2 cjson_create_array()

```
cjson_value* cjson_create_array (
            void  )
```

创建一个json数组对象。

描述:

创建一个json数组对象。

参数

无。	

返回值

NOT	NULL 创建成功。
NULL	创建失败。

6.1.3.3 cjson_create_boolean()

```
cjson_value* cjson_create_boolean (
            int32_t b )
```

创建一个json布尔值对象。

描述:

创建一个json布尔值对象。

参数

b	[IN] json数据布尔值，true or false。

返回值

NOT	NULL 创建成功。
NULL	创建失败。

6.1.3.4 cjson_create_number()

```
cjson_value* cjson_create_number (
            double b )
```

创建一个json整形数或浮点数对象。

描述:

创建一个json数据对象。

参数

b	[IN] json整形数或浮点数。

返回值

NOT	NULL 创建成功。
NULL	创建失败。

6.1.3.5 cjson_create_object()

```
cjson_value* cjson_create_object (
            void  )
```

创建一个json对象。

描述:

创建一个json对象。

参数

无。	

返回值

NOT	NULL 创建成功。
NULL	创建失败。

6.1.3.6 cjson_create_string()

```
cjson_value* cjson_create_string (
            const char * string,
            uint32_t len )
```

创建一个json字符串对象。

描述:

创建一个json字符串对象。

参数

string	[IN] json字符串。
len	[IN] json字符串长度。

返回值

NOT	NULL 创建成功。
NULL	创建失败。

6.1.3.7 cjson_free()

```
void cjson_free (
            cjson_value ** v )
```

释放json数据资源。

描述:

释放json数据对象资源。

参数

v	[IN] json数据对象指针的指针，NOT NULL。

返回值

无。	

6.1.3.8 cjson_get_array_element()

```
cjson_value* cjson_get_array_element (
            const cjson_value * v,
            uint32_t index )
```

根据索引值，获取json数组对象中某个成员的对象。

描述:

获取某个json数组成员的对象。

参数

v	[IN] json对象指针。
index	[IN] 索引值，从0开始。

返回值

返回成员对象。	

6.1.3.9 cjson_get_array_size()

```
uint32_t cjson_get_array_size (
            const cjson_value * v )
```

获取json数组对象中数组的长度。

描述:

获取json数组对象中数组的长度。

参数

v	[IN] json对象指针。

返回值

返回长度值。	

6.1.3.10 cjson_get_boolean()

```
int32_t cjson_get_boolean (
            const cjson_value * v )
```

获取json尔布对象数据。

描述:

获取json布尔对象数据。

参数

v	[IN] json对象指针。

返回值

true	or false。

6.1.3.11 cjson_get_number()

```
double cjson_get_number (
            const cjson_value * v )
```

获取json数值对象数据。

描述:

获取json数值对象数据。

参数

v	[IN] json对象指针。

返回值

返回实际数值。	

6.1.3.12 cjson_get_object_key()

```
const char* cjson_get_object_key (
            const cjson_value * v,
            uint32_t index )
```

根据索引值，获取json对象中某个子对象的key值。

描述:

获取某个json对象成员的key值。

参数

v	[IN] json对象指针。
index	[IN] 索引值，从0开始。

返回值

返回*json*对象的*key*值。	

6.1.3.13 cjson_get_object_key_length()

```
uint32_t cjson_get_object_key_length (
            const cjson_value * v,
            uint32_t index )
```

根据索引值，获取json对象中某个子对象的key值 的长度。

描述:

获取某个json对象成员的key值长度。

参数

v	[IN] json对象指针。
index	[IN] 索引值，从0开始。

返回值

返回*json*对象的*key*值长度。	

6.1.3.14 cjson_get_object_size()

```
uint32_t cjson_get_object_size (
            const cjson_value * v )
```

获取json对象中子对象的个数。

描述:

获取json对象中子对象的个数。

参数

v	[IN] json对象指针。

返回值

返回个数。	

6.1.3.15 cjson_get_object_value()

```
cjson_value* cjson_get_object_value (
            const cjson_value * v,
            uint32_t index )
```

根据索引值，获取json对象中某个子对象的value值。

描述:

获取某个json对象成员的对象。

参数

v	[IN] json对象指针。
index	[IN] 索引值，从0开始。

返回值

返回*json*对象的对象指针。	

6.1.3.16 cjson_get_string()

```
const char* cjson_get_string (
            const cjson_value * v )
```

获取json字符串对象数据。

描述:

获取json字符串对象数据。

参数

v	[IN] json对象指针。

返回值

返回实际字符串。	

6.1.3.17 cjson_get_string_length()

```
uint32_t cjson_get_string_length (
            const cjson_value * v )
```

获取json字符串对象中字符串的长度。

描述:

获取json字符串对象中字符串的长度。

参数

v	[IN] json对象指针。

返回值

返回长度值。	

6.1.3.18 cjson_get_type()

```
int32_t cjson_get_type (
            const cjson_value * v )
```

获取json对象类型。

描述:

获取json对象类型。

参数

v	[IN] json对象指针。

返回值

ColinkCjsonType	json定义的数据类型

参见

ColinkCjsonType

6.1.3.19 cjson_get_value_Bykey()

```
cjson_value* cjson_get_value_Bykey (
            const cjson_value * v,
            const char * str )
```

根据key值，获取json对象中某个子对象的对象。

描述:

获取某个json对象成员的对象。

参数

v	[IN] json对象指针。
index	[IN] 索引值，从0开始。

返回值

返回*json*对象的对象指针。	

6.1.3.20 cjson_object_additem()

```
void cjson_object_additem (
            cjson_value * object,
            const char * key,
            cjson_value * val )
```

往对象中添加成员。

描述:

往根对象中添加成员。

参数

object	[IN] json根对象数据对象指针，NOT NULL。
key	[IN] 添加成员的key值，NOT NULL。
val	[IN] 添加成员的value值，NOT NULL。

返回值

无。	

6.1.3.21 cjson_parse()

```
int32_t cjson_parse (
            cjson_value * v,
            const char * json )
```

解析json字符串。

描述:

解析json字符串,解析成功后传入v数据对象中，数据处理完后，v对象需要调用cjson_free释放资源。

参数

v	[IN] json数据对象指针，NOT NULL。
json	[IN] json字符串，NOT NULL。

返回值

ColinkCjsonParseErrCode	json解析返回值

参见

ColinkCjsonParseErrCode

6.1.3.22 cjson_set_boolean()

```
void cjson_set_boolean (
            cjson_value * v,
            int32_t b )
```

修改布尔值对象数值。

描述:

修改对象布尔数值。

参数

v	[IN] json对象指针。
b	[IN] ture or false。

返回值

无。	

6.1.3.23 cjson_set_number()

```
void cjson_set_number (
            cjson_value * v,
            double n )
```

修改数值对象数值。

描述:

修改对象数值。

参数

v	[IN] json对象指针。
n	[IN] 整形或浮点形。

返回值

无。	

6.1.3.24 cjson_set_string()

```
void cjson_set_string (
            cjson_value * v,
            const char * s,
            uint32_t len )
```

修改字符串对象数值。

描述:

修改对象字符串。

参数

v	[IN] json对象指针。
s	[IN] 字符串。
len	[IN] 字符串长度。

返回值

无。	

6.1.3.25 cjson_stringify()

```
char* cjson_stringify (
            const cjson_value * v,
            uint32_t * length )
```

生成json字符串。

描述:

由v数据对象生成对应的json字符串。

参数

v	[IN] json数据对象指针，NOT NULL。
length	[OUT] 输出json字符串的长度，若不需要，可写NULL。

返回值

NOT NULL	解析成功。
NULL	解析失败。

6.2 include/colink_error.h 文件参考

Show profile of colink error code.

宏定义

- #define COLINK_INIT_NO_ERROR (0)
  无错误
- #define COLINK_INIT_FAILED (-1)
  初始化失败
- #define COLINK_INIT_ARG_INVALID (-2)
  无效的参数
- #define COLINK_NO_ERROR (0)
  无错误
- #define COLINK_ARG_INVALID (-2)
  无效的参数
- #define COLINK_JSON_INVALID (-3)
  无效的*JSON*格式
- #define COLINK_JSON_CREATE_ERR (-4)
  创建*JSON*对象错误
- #define COLINK_DATA_SEND_ERROR (-5)
  发送数据出错
- #define COLINK_DEV_TYPE_ERROR (-6)
  设备类型错误
- #define COLINK_PROCESS_NO_ERROR (0)
  无错误
- #define COLINK_PROCESS_INIT_INVALID (-12)
  *colinkInit*未初始化成功
- #define COLINK_PROCESS_TIMEOUT (-13)
  长时间未被调用*colinkProcess*
- #define COLINK_PROCESS_MEMORY_ERROR (-14)
  内存分配错误
- #define COLINK_TCP_NO_ERROR (0)
  无错误
- #define COLINK_TCP_ARG_INVALID (-2)
  无效的参数

- #define COLINK_TCP_CREATE_CONNECT_ERR (-21)
  创建*TCP*连接错误
- #define COLINK_TCP_SEND_ERR (-23)
  *TCP*发送失败
- #define COLINK_TCP_READ_ERR (-25)
  *TCP*读取失败
- #define COLINK_TCP_CONNECT_TIMEOUT (-26)
  *TCP*连接超时
- #define COLINK_TCP_CONNECT_ERR (-27)
  *TCP*连接失败
- #define COLINK_TCP_CONNECTING (-28)
  *TCP*连接中
- #define COLINK_TCP_READ_INCOMPLETED (-29)
  *TCP*读包不完整
- #define COLINK_TCP_CREATE_SERVER_ERR (-31)
  创建*TCP*服务错误
- #define COLINK_TCP_SERVER_WAIT_CONNECT (-32)
  *tcp*等待客户端连接
- #define COLINK_TCP_DNS_PARSING (-35)
  *DNS*解析中
- #define COLINK_TCP_DNS_PARSE_ERR (-36)
  *DNS*解析失败
- #define COLINK_TCP_DNS_PARSE_TIMEOUT (-37)
  *DNS*解析超时
- #define COLINK_REQ_RESULT_NO_ERROR (0)
  操作成功
- #define COLINK_REQ_RESULT_NUMBER_ERROR (-414)
  子设备数量超过限制
- #define COLINK_OTA_NO_ERROR (0)
  *OTA*升级成功
- #define COLINK_OTA_DOWNLOAD_ERROR (-404)
  *OTA*文件下载失败
- #define COLINK_OTA_MODEL_ERROR (-406)
  设备型号不正确
- #define COLINK_OTA_DIGEST_ERROR (-409)
  固件校验失败
- #define COLINK_LINK_NO_ERROR (0)
  无错误
- #define COLINK_LINK_ARG_INVALID (-2)
  无效的参数
- #define COLINK_LINK_OPERATION_ERROR (-50)
  流程操作不对

6.2.1 详细描述

Show profile of colink error code.

作者

Wu Jiale

日期

2018.09.06

6.3 include/colink_gateway_profile.h 文件参考

Show profile of colinkgateway.

```
#include "colink_typedef.h"
```

结构体

- struct ColinkSubDevAddr
  子设备地址的结构体。
- struct ColinkSubDevice
  子设备的结构体。
- struct ColinkSubDeviceList
  子设备列表的结构体。
- struct ColinkGatewayEvent

宏定义

- #define COLINK_GATEWAY_NO_ERROR (0)
  无错误
- #define COLINK_GATEWAY_OPERATE_FAILED (-1)
  操作失败
- #define COLINK_GATEWAY_ARG_INVALID (-2)
  无效的参数
- #define COLINK_GATEWAY_NET_ERROR (-100)
  网络异常
- #define COLINK_SUB_DEV_NO_ERROR (0)
  操作成功
- #define COLINK_SUB_DEV_NUMBER_ERROR (-414)
  子设备数量超过限制

函数

- int32_t colinkGatewayInit (const char *device_id, ColinkGatewayEvent *event)
  初始化网关功能。
- const ColinkSubDevAddr * colinkGetAddrByDeviceid (char deviceid[11])
  通过*deviceid*获取子设备的地址。
- const char * colinkGetDeviceidByAddr (ColinkSubDevAddr *sub_dev_addr)
  通过子设备地址获取*deviceid*。
- void colinkGatewayDelAllSubDev (void)
  清除子设备列表。
- int32_t colinkAddSubDev (ColinkSubDevice dev_list[ ], uint16_t dev_num)
  注册新子设备。
- int32_t colinkDelSubDev (ColinkSubDevice dev_list[ ], uint16_t dev_num)
  删除子设备。
- int32_t colinkOnlineSubDev (ColinkSubDevice dev_list[ ], uint16_t dev_num)
  上报子设备在线。
- int32_t colinkOfflineSubDev (ColinkSubDevice dev_list[ ], uint16_t dev_num)
  上报子设备离线。
- int32_t colinkSubDevSendRes (ColinkSubDevice *sub_dev, int32_t error_code, char req_sequence[ ])
  子设备响应服务器的错误码。
- int32_t colinkSubDevSendReq (ColinkSubDevice *sub_dev, char *data)
  子设备发送请求的数据。
- uint16_t colinkGatewayAddSubDev (ColinkSubDeviceList *list, uint16_t num)
  添加多个子设备到子设备列表。
- uint16_t colinkGatewayGetSubDevNum (void)
  获取当前子设备数量。
- uint16_t colinkGatewayGetSubDevList (ColinkSubDeviceList *list, uint16_t index, uint16_t num)
  获取当前子设备信息列表。
- uint16_t colinkGatewayReportSubDevState (ColinkSubDevice dev_list[ ], uint16_t dev_num, bool end_report)
  批量上报子设备状态。

6.3.1 详细描述

Show profile of colinkgateway.

It shows the detail of colinkgateway struct and API.

作者

Wu Jiale

日期

2018.09.06

6.3.2 函数说明

6.3.2.1 colinkAddSubDev()

```
int32_t colinkAddSubDev (
            ColinkSubDevice dev_list[],
            uint16_t dev_num )
```

注册新子设备。

描述:

注册多个新子设备。

参数

dev_list	[IN] 子设备数组的指针。
dev_num	[IN] 子设备数组的数量。

返回值

ColinkGatewayErrorCode	子设备操作错误码。

参见

ColinkGatewayErrorCode

警告

colinkAddSubDev,colinkDelSubDev,colinkOnlineSubDev,colinkOfflineSubDev属于同类操作，不支持并发，请在某个操作的回调结束后再去进行调用同类的操作！！！

6.3.2.2 colinkDelSubDev()

```
int32_t colinkDelSubDev (
            ColinkSubDevice dev_list[],
            uint16_t dev_num )
```

删除子设备。

描述:

删除多个子设备。

参数

dev_list	[IN] 子设备数组的指针。
dev_num	[IN] 子设备数组的数量。

返回值

ColinkGatewayErrorCode	子设备操作错误码。

参见

ColinkGatewayErrorCode

警告

colinkAddSubDev,colinkDelSubDev,colinkOnlineSubDev,colinkOfflineSubDev属于同类操作，不支持并发，请在某个操作的回调结束后再去进行调用同类的操作！！！

6.3.2.3 colinkGatewayAddSubDev()

```
uint16_t colinkGatewayAddSubDev (
            ColinkSubDeviceList * list,
            uint16_t num )
```

添加多个子设备到子设备列表。

描述:

添加多个子设备到本地子设备列表，用于重启时添加保存在flash中的子设备列表，需 在初始化（colinkInit）之后调用。

参数

list	[IN] 欲添加进本地子设备列表的子设备列表。
num	[IN] 子设备数量。

返回值

0	全部添加成功。
大于*0*	已添加成功的子设备数量。

6.3.2.4 colinkGatewayDelAllSubDev()

```
void colinkGatewayDelAllSubDev (
            void  )
```

清除子设备列表。

描述:

清除网关本地子设备列表。

参数

无	

返回值

无	

6.3.2.5 colinkGatewayGetSubDevList()

```
uint16_t colinkGatewayGetSubDevList (
            ColinkSubDeviceList * list,
            uint16_t index,
            uint16_t num )
```

获取当前子设备信息列表。

描述:

获取当前网关设备上的子设备信息列表。

参数

list	[IN/OUT] 子设备信息列表缓冲区，由开发者根据实际需求分配空间。
index	[IN] 子设备信息i列表读取起始位置
num	[IN] 获取的子设备信息数量

返回值

成功获取的子设备信息数量。	

6.3.2.6 colinkGatewayGetSubDevNum()

```
uint16_t colinkGatewayGetSubDevNum (
            void  )
```

获取当前子设备数量。

描述:

获取当前网关设备上子设备的数量。

参数

无	

返回值

当前子设备数量。	

6.3.2.7 colinkGatewayInit()

```
int32_t colinkGatewayInit (
            const char * device_id,
            ColinkGatewayEvent * event )
```

初始化网关功能。

描述:

初始化网关功能。

参数

device_id	设备id
event	网关设备事件回调

返回值

ColinkGatewayErrorCode	

6.3.2.8 colinkGatewayReportSubDevState()

```
uint16_t colinkGatewayReportSubDevState (
            ColinkSubDevice dev_list[],
            uint16_t dev_num,
            bool end_report )
```

批量上报子设备状态。

描述:

在网关设备掉线重连后，上报当前子设备的状态。

参数

dev_list	[IN] 子设备数组的指针。
dev_num	[IN] 子设备数量。
end_report	[IN] 是否上报完成（是否为最后一次上报）。

返回值

成功	上报成功的子设备数量。 失败 0

6.3.2.9 colinkGetAddrByDeviceid()

```
const ColinkSubDevAddr* colinkGetAddrByDeviceid (
            char deviceid[11] )
```

通过deviceid获取子设备的地址。

描述:

通过deviceid获取子设备的地址。

参数

deviceid	[IN] deviceid字符串。

返回值

非*NULL*	获取成功。
NULL	获取失败。

6.3.2.10 colinkGetDeviceidByAddr()

```
const char* colinkGetDeviceidByAddr (
            ColinkSubDevAddr * sub_dev_addr )
```

通过子设备地址获取deviceid。

描述:

通过子设备地址获取deviceid。

参数

sub_dev_addr	[IN] 子设备地址的指针。

返回值

非*NULL*	获取成功。
NULL	获取失败。

6.3.2.11 colinkOfflineSubDev()

```
int32_t colinkOfflineSubDev (
            ColinkSubDevice dev_list[],
            uint16_t dev_num )
```

上报子设备离线。

描述:

上报多个子设备离线。

参数

dev_list	[IN] 子设备数组的指针。
dev_num	[IN] 子设备数组的数量。

返回值

ColinkGatewayErrorCode	子设备操作错误码。

参见

ColinkGatewayErrorCode

警告

colinkAddSubDev,colinkDelSubDev,colinkOnlineSubDev,colinkOfflineSubDev属于同类操作，不支持并发，请在某个操作的回调结束后再去进行调用同类的操作！！！

6.3.2.12 colinkOnlineSubDev()

```
int32_t colinkOnlineSubDev (
            ColinkSubDevice dev_list[],
            uint16_t dev_num )
```

上报子设备在线。

描述:

上报多个子设备在线。

参数

dev_list	[IN] 子设备数组的指针。
dev_num	[IN] 子设备数组的数量。

返回值

ColinkGatewayErrorCode	子设备操作错误码。

参见

ColinkGatewayErrorCode

警告

colinkAddSubDev,colinkDelSubDev,colinkOnlineSubDev,colinkOfflineSubDev属于同类操作， 不支持并发，请在某个操作的回调结束后再去进行调用同类的操作！！！

6.3.2.13 colinkSubDevSendReq()

```
int32_t colinkSubDevSendReq (
            ColinkSubDevice * sub_dev,
            char * data )
```

子设备发送请求的数据。

描述:

子设备发送请求的数据。之后会产生colinkSubDevRecvResCb回调函数来获取 服务器响应的错误码。每调用此函数需要等收到服务器响应才能再次调用。

参数

sub_dev	[IN] 子设备的指针。
data	[IN] 发送请求的数据。

返回值

ColinkErrorCode	colink错误码。

参见

ColinkErrorCode

6.3.2.14 colinkSubDevSendRes()

```
int32_t colinkSubDevSendRes (
            ColinkSubDevice * sub_dev,
            int32_t error_code,
            char req_sequence[] )
```

子设备响应服务器的错误码。

描述:

子设备响应服务器的错误码。当收到colinkSubDevRecvReqCb回调函数后 需要调用此函数响应给服务器。

参数

sub_dev	[IN] 子设备的指针。
error_code	[IN] 发送响应的错误码。参见ColinkSubDevResultCode。
req_sequence	[IN] 传入colinkSubDevRecvReqCb获取的req_sequence。

返回值

ColinkErrorCode	colink错误码。

参见

ColinkSubDevResultCode
ColinkErrorCode

6.4 include/colink_link.h 文件参考

Show profile of distributing network.

```
#include <stdint.h>
#include "colink_typedef.h"
```

结构体

- struct ColinkLinkInfo
  配网信息结构体。

宏定义

- #define COLINK_LINK_INIT_INVALID (0)
  未复位配网初始状态
- #define COLINK_LINK_RES_DEV_INFO_OK (1)
  响应设备信息成功
- #define COLINK_LINK_RES_DEVICEID_ERROR (2)
  响应设备信息失败
- #define COLINK_LINK_GET_INFO_OK (3)
  获取配网信息成功
- #define COLINK_LINK_GET_INFO_ERROR (4)
  获取配网信息失败

函数

- int32_t colinkLinkInit (char *deviceid, char *apikey)
  初始化配网。
- void colinkLinkReset (void)
  复位配网阶段到初始状态。
- int32_t colinkLinkParse (uint8_t *in, uint16_t in_len, uint8_t *out, uint16_t out_buf_len, uint16_t *out_len)
  用在与*APP*交互阶段，负责解析并输出响应*APP*的数据。
- int32_t colinkLinkGetInfo (ColinkLinkInfo *info)
  配网成功后获取分配服务器信息。

6.4.1 详细描述

Show profile of distributing network.

It shows the detail of distributing network struct, enmu and API.

作者

Wu Jiale

日期

2018.09.06

版权所有

6.4.2 函数说明

6.4.2.1 colinkLinkGetInfo()

```
int32_t colinkLinkGetInfo (
            ColinkLinkInfo * info )
```

配网成功后获取分配服务器信息。

描述:

当colinkLinkParse返回COLINK_LINK_GET_INFO_OK后，直接调用 此函数获取分配服务器信息。否则返回COLINK_LINK_OPERATION_ERROR错误类型。

参数

info	[OUT] 输出配对结果信息的指针。

返回值

ColinkLinkErrorCode	colink配网错误码。

参见

ColinkLinkErrorCode

6.4.2.2 colinkLinkInit()

```
int32_t colinkLinkInit (
            char * deviceid,
            char * apikey )
```

初始化配网。

描述:

在colinkLinkReset之前调用初始化。

参数

deviceid	[IN] 设备ID。
apikey	[IN] 设备密钥。

返回值

ColinkLinkErrorCode	colink配网错误码。

参见

ColinkLinkErrorCode

6.4.2.3 colinkLinkParse()

```
int32_t colinkLinkParse (
            uint8_t * in,
```

```
            uint16_t in_len,
            uint8_t * out,
            uint16_t out_buf_len,
            uint16_t * out_len )
```

用在与APP交互阶段，负责解析并输出响应APP的数据。

描述:

在每次进入配网流程前需要调用colinkLinkReset。每收到来自APP的数据，就调用一次 colinkLink↩Parse，并将输出数据发送给APP。正常流程会依次返回结果 COLINK_LINK_RES_DEV_INFO_OK和↩COLINK_LINK_GET_INFO_OK，当返回 COLINK_LINK_GET_INFO_OK时即可结束配网流程。

参数

in	[IN] 输入数据的指针。
in_len	[IN] 输入数据的长度。
out	[IN] 输出数据的指针。
out_buf_len	[IN] 输出数据缓冲区的长度。
out_len	[IN] 输出数据的的长度。

返回值

CoLinkLinkState	配网结果枚举类型。

参见

CoLinkLinkState

6.4.2.4 colinkLinkReset()

```
void colinkLinkReset (
            void  )
```

复位配网阶段到初始状态。

描述:

每次进入配网阶段前都需要调用colinkLinkReset， 使配网进入初始状态。

参数

无。	

6.5 include/colink_profile.h 文件参考

Show profile of colink.

```
#include "colink_typedef.h"
#include "colink_gateway_profile.h"
```

结构体

- struct ColinkDev
  设备信息的结构体。
- struct ColinkOtaInfo
  *OTA*固件信息的结构体。
- struct ColinkEvent
  设备事件回调的结构体。

宏定义

- #define COLINK_SINGLE (0)
  普通设备
- #define COLINK_GATEWAY (1)
  网关设备
- #define DEVICE_OFFLINE (0)
  设备离线
- #define DEVICE_ONLINE (1)
  设备在线
- #define DEVICE_UNREGISTERED (2)
  设备未注册

函数

- int32_t colinkInit (ColinkDev *dev_data, ColinkEvent *reg_event)
  初始化*colink Device*。
- bool colinkDeInit (void)
  反初始化*colink Device*。
- char * colinkGetUserApiKey (void)
  获取用户*ApiKey*。
- int32_t colinkProcess (void)
  *colink*事件处理。
- int32_t colinkSendUpdate (char *data)
  发送*update*字段的数据。
- int32_t colinkGetDevStatus (void)
  获取设备的状态。
- int32_t colinkUpgradeRes (int32_t error_code, char *seqeuence)
  响应升级的结果。
- int32_t colinkSendUTCRequest (void)
  向服务器发送获取*UTC*时间请求。
- int32_t colinkSendQuery (char *params)
  发送*query*字段的数据。
- const char * colinkGetVersion (void)
  获取*colink*版本号。

6.5.1 详细描述

Show profile of colink.

It shows the detail of colink struct, enmu and API.

作者

Wu Jiale

日期

2018.09.06

6.5.2 函数说明

6.5.2.1 colinkDeInit()

```
bool colinkDeInit (
            void  )
```

反初始化colink Device。

描述:

反初始化colink，释放当前colink占用的资源。 用于使用过程中，设备需要重新配网或者出现问题需要重新初始化colink的状况下，须先 把colinkProcess进程结束，然后调用该接口释放colink占用的资源。

参数

无。	

返回值

true	反初始化成功。
false	反初始化失败。

6.5.2.2 colinkGetDevStatus()

```
int32_t colinkGetDevStatus (
            void  )
```

获取设备的状态。

描述:

获取设备的离线上线状态，开发者通过返回值作相应处理。

参数

无。	

返回值

ColinkDevStatus	设备状态。

参见

ColinkDevStatus

6.5.2.3 colinkGetUserApiKey()

```
char* colinkGetUserApiKey (
            void  )
```

获取用户ApiKey。

描述:

获取服务器下发的用户ApiKey。

参数

无。	

返回值

服务器下发的用户*ApiKey*。	

6.5.2.4 colinkGetVersion()

```
const char* colinkGetVersion (
            void  )
```

获取colink版本号。

描述:

获取colink版本号。

参数

无。	

返回值

非*NULL*	成功获取colink版本。
NULL	获取版本失败。

6.5.2.5 colinkInit()

```
int32_t colinkInit (
            ColinkDev * dev_data,
            ColinkEvent * reg_event )
```

初始化colink Device。

描述:

初始化colink设备信息和注册回调事件。

参数

dev_data	[IN] 设备信息结构体指针。
reg_event	[IN] 注册回调事件结构体指针。

返回值

ColinkInitErrorCode	colink初始化错误码。

参见

ColinkInitErrorCode

6.5.2.6 colinkProcess()

```
int32_t colinkProcess (
            void  )
```

colink事件处理。

描述:

启动colink事件处理，必须在colinkInit初始化成功后调用！ 建议调用间隔最大25~50毫秒。若该函数长时间未被调用，则可能返回异常信息。 调用该接口的任务分配栈空间建议4096字节！！！

参数

无。	

返回值

ColinkProcessErrorCode	colink进程运行错误码。

参见

ColinkProcessErrorCode

6.5.2.7 colinkSendQuery()

```
int32_t colinkSendQuery (
            char * params )
```

发送query字段的数据。

描述:

当设备需要查询定时器等信息时，向服务器发送查询请求。

参数

params	[IN] 以Json格式表示的字符串数据。

返回值

ColinkErrorCode	colink错误码。

参见

ColinkErrorCode

6.5.2.8 colinkSendUpdate()

```
int32_t colinkSendUpdate (
            char * data )
```

发送update字段的数据。

描述:

当设备状态发生改变时主动发到云平台，云平台再转发到APP同步设备状态。

参数

data	[IN] 以Json格式表示的字符串数据。

返回值

ColinkErrorCode	colink错误码。

参见

ColinkErrorCode

6.5.2.9 colinkSendUTCRequest()

```
int32_t colinkSendUTCRequest (
            void  )
```

向服务器发送获取UTC时间请求。

描述:

当设备需要同步服务器UTC时间时，调用此接口发送请求， 发送请求成功后通过colinkSendUTC↩RequestCb获取请求结果。

参数

error_code	[IN] 响应错误码值。参考ColinkOtaResCode。

返回值

ColinkErrorCode	colink错误码。

参见

ColinkOtaResCode
ColinkErrorCode

6.5.2.10 colinkUpgradeRes()

```
int32_t colinkUpgradeRes (
            int32_t error_code,
            char * seqeuence )
```

响应升级的结果。

描述:

当收到升级通知后（即colinkUpgradeRequestCb）， 升级完成需要调用此接口来响应服务器是否升级成功的结果。

参数

error_code	[IN] 响应错误码值。参考ColinkOtaResCode。

返回值

ColinkErrorCode	colink错误码。

参见

ColinkOtaResCode
ColinkErrorCode

6.6 include/colink_socket.h 文件参考

Show profile of colink socket.

```
#include <stdint.h>
#include "colink_typedef.h"
```

函数

- int32_t colinkCreateTcpServer (uint16_t port)
  创建*TCP*服务
- int32_t colinkTcpServerGetState (int32_t fd)
  *tcp*服务状态
- int32_t colinkTcpConnect (const char *dst, uint16_t port)
  创建*tcp*连接
- int32_t colinkTcpState (int32_t fd)
  *tcp*连接状态
- void colinkTcpDisconnect (int32_t fd)
  断开*tcp*连接。
- int32_t colinkTcpSend (int32_t fd, const uint8_t *buf, uint16_t len)
  发送*tcp*数据
- int32_t colinkTcpRead (int32_t fd, uint8_t *buf, uint16_t len)
  读取*tcp*数据
- int32_t colinkTcpSslConnect (const char *dst, uint16_t port)
  创建加密*tcp*连接
- int32_t colinkTcpSslState (int32_t fd)
  加密*tcp*连接状态
- void colinkTcpSslDisconnect (int32_t fd)
  断开加密*tcp*连接。
- int32_t colinkTcpSslSend (int32_t fd, const uint8_t *buf, uint16_t len)
  发送加密*tcp*数据
- int32_t colinkTcpSslRead (int32_t fd, uint8_t *buf, uint16_t len)
  读取加密*tcp*数据
- int32_t colinkGethostbyname (const char *hostname, uint8_t len)
  获取远端主机*ip*地址。
- int32_t colinkGethostbynameState (char *ip)
  获取*DNS*解析状态。

6.6.1 详细描述

Show profile of colink socket.

It shows the detail of colink socket API.

作者

Wu Jiale

日期

2018.09.06

6.6.2 函数说明

6.6.2.1 colinkCreateTcpServer()

```
int32_t colinkCreateTcpServer (
            uint16_t port )
```

创建TCP服务

描述:

创建非阻塞模式tcp服务

参数

port	[IN] 接收方端口号。

返回值

COLINK_TCP_CREATE_SERVER_ERR	建立tcp服务出错
大于等于*0*	建立成功，返回tcp套接字，使用colinkTcpState判断是否有外部连接建立

6.6.2.2 colinkGethostbyname()

```
int32_t colinkGethostbyname (
            const char * hostname,
            uint8_t len )
```

获取远端主机ip地址。

描述:

通过DNS域名解析，获取远端主机ip地址，此接口实现为非阻塞。

参数

hostname	[IN] 远端主机名称。
num	[IN] 远端主机名称长度。

返回值

ColinkTcpErrorCode	colink tcp错误码。

参见

ColinkTcpErrorCode

6.6.2.3 colinkGethostbynameState()

```
int32_t colinkGethostbynameState (
            char * ip )
```

获取DNS解析状态。

描述:

获取DNS解析状态

参数

无。	

返回值

COLINK_TCP_DNS_PARSING	解析中
COLINK_TCP_NO_ERROR	解析成功

6.6.2.4 colinkTcpConnect()

```
int32_t colinkTcpConnect (
            const char * dst,
            uint16_t port )
```

创建tcp连接

描述:

创建非阻塞模式tcp连接

参数

dst	[IN] 接收方ip地址。
port	[IN] 接收方端口号。

返回值

COLINK_TCP_ARG_INVALID	dst为空。
COLINK_TCP_CREATE_CONNECT_ERR	创建连接失败。
大于*0*	连接成功,返回tcp套接字，然后使用colinkTcpState判断连接是否完全建立。

6.6.2.5 colinkTcpDisconnect()

```
void colinkTcpDisconnect (
            int32_t fd )
```

断开tcp连接。

描述:

断开tcp连接。

参数

fd	[IN] colinkTcpConnect创建的套接字。

返回值

无。	

6.6.2.6 colinkTcpRead()

```
int32_t colinkTcpRead (
            int32_t fd,
            uint8_t * buf,
            uint16_t len )
```

读取tcp数据

描述:

非阻塞读取tcp数据

参数

fd	[IN] tcp套接字。
buf	[IN] 指向存放接收数据缓冲区的指针。
len	[IN] 存放接收数据缓冲区的最大长度，范围为[0，512)。

返回值

ColinkTcpErrorCode	colink tcp错误码。

参见

ColinkTcpErrorCode

6.6.2.7 colinkTcpSend()

```
int32_t colinkTcpSend (
            int32_t fd,
            const uint8_t * buf,
            uint16_t len )
```

发送tcp数据

描述:

非阻塞发送tcp数据

参数

fd	[IN] tcp套接字。
buf	[IN] 指向待发送数据缓冲区的指针。
len	[IN] 待发送数据的长度，范围为[0，512)。

返回值

ColinkTcpErrorCode	colink tcp错误码。

参见

ColinkTcpErrorCode

6.6.2.8 colinkTcpServerGetState()

```
int32_t colinkTcpServerGetState (
            int32_t fd )
```

tcp服务状态

描述:

获取tcp服务状态

参数

fd	tcp套接字

返回值

COLINK_TCP_NO_ERROR	与客户端建立连接成功
COLINK_TCP_SERVER_WAIT_CONNECT	等待建立连接

6.6.2.9 colinkTcpSslConnect()

```
int32_t colinkTcpSslConnect (
```

```
            const char * dst,
            uint16_t port )
```

创建加密tcp连接

描述:

创建非阻塞模式加密tcp连接

参数

dst	[IN] 接收方ip地址。
port	[IN] 接收方端口号。

返回值

COLINK_TCP_ARG_INVALID	dst为空。
COLINK_TCP_CREATE_CONNECT_ERR	创建连接失败。
大于*0*	连接成功,返回tcp套接字，然后使用colinkTcpSslState判断连接是否完全建立。

6.6.2.10 colinkTcpSslDisconnect()

```
void colinkTcpSslDisconnect (
            int32_t fd )
```

断开加密tcp连接。

描述:

断开加密tcp连接。

参数

fd	[IN] colinkTcpSslConnect创建的套接字。

返回值

无。	

6.6.2.11 colinkTcpSslRead()

```
int32_t colinkTcpSslRead (
```

```
            int32_t fd,
            uint8_t * buf,
            uint16_t len )
```

读取加密tcp数据

描述:

非阻塞读取加密tcp数据

参数

fd	[IN] tcp套接字。
buf	[IN] 指向存放接收数据缓冲区的指针。
len	[IN] 存放接收数据缓冲区的最大长度，范围为[0，512)。

返回值

ColinkTcpErrorCode	colink tcp错误码。

参见

ColinkTcpErrorCode

6.6.2.12 colinkTcpSslSend()

```
int32_t colinkTcpSslSend (
            int32_t fd,
            const uint8_t * buf,
            uint16_t len )
```

发送加密tcp数据

描述:

非阻塞发送加密tcp数据

参数

fd	[IN] tcp套接字。
buf	[IN] 指向待发送数据缓冲区的指针。
len	[IN] 待发送数据的长度，范围为[0，512)。

返回值

ColinkTcpErrorCode	colink tcp错误码。

参见

ColinkTcpErrorCode

6.6.2.13 colinkTcpSslState()

```
int32_t colinkTcpSslState (
            int32_t fd )
```

加密tcp连接状态

描述:

查询加密tcp连接状态

参数

fd	[IN] tcp套接字。

返回值

ColinkTcpErrorCode	colink tcp错误码。

参见

ColinkTcpErrorCode

6.6.2.14 colinkTcpState()

```
int32_t colinkTcpState (
            int32_t fd )
```

tcp连接状态

描述:

查询tcp连接状态

参数

fd	[IN] tcp套接字。

返回值

ColinkTcpErrorCode	colink tcp错误码。

参见

ColinkTcpErrorCode

6.7 include/colink_sysadapter.h 文件参考

Show profile of colink system adapter.

```
#include <stdint.h>
#include "colink_error.h"
#include "colink_typedef.h"
```

函数

- int32_t colinkGettime (uint32_t *ms)
  获取系统运行时间。
- int32_t colinkNetworkState (int32_t *state)
  获取网络状态。
- uint32_t colinkRand (void)
  随机数发生器的初始化函数。
- bool colinkSha256 (char *in, char *checkout)
  计算*sha256*校验值函数
- uint32_t colinkStrlen (const char *src)
  计算给定字符串的长度。
- char * colinkStrcpy (char *dst, const char *src)
  复制字符串中的内容。
- char * colinkStrncpy (char *dst, const char *src, uint32_t len)
  复制字符串中的内容。
- char * colinkStrcat (char *dst, const char *src)
  连接字符串。
- char * colinkStrncat (char *dst, const char *src, uint32_t len)
  连接字符串。
- char * colinkStrstr (const char *s1, const char *s2)
  查找字符串第一次出现的位置。
- int32_t colinkStrcmp (const char *str1, const char *str2)
  比较两个字符串的大小。
- int32_t colinkStrncmp (const char *str1, const char *str2, uint32_t len)
  比较两个字符串的大小。

- char * colinkStrchr (char *str, int32_t ch)
  查找一个字符在一个字符串中首次出现的位置。
- char * colinkStrrchr (const char *str, char c)
  查找一个字符在一个字符串中末次出现的位置。
- int32_t colinkAtoi (const char *str)
  把字符串转换成整型数。
- int32_t colinkSprintf (char *buf, const char *format,...)
  字符串格式化函数。
- int32_t colinkSnprintf (char *buf, uint32_t size, const char *format,...)
  字符串格式化函数。
- int32_t colinkPrintf (const char *format,...)
  格式化输出函数。
- void * colinkMemset (void *dst, int32_t c, uint32_t len)
  在一段内存块中填充某个给定的值。
- void * colinkMemcpy (void *dst, const void *src, uint32_t len)
  复制内存中的内容。
- int32_t colinkMemcmp (const void *buf1, const void *buf2, uint32_t len)
  比较两块内存区域。
- void * colinkMalloc (uint32_t n)
  获取所需的内存空间。
- void * colinkRealloc (void *ptr, uint32_t n)
  调整一块内存空间大小。
- void colinkFree (void *pt)
  释放指针*pt*所占用的内存空间。
- unsigned short colinkHtons (unsigned short hs)
  将主机字节顺序转换为网络字节顺序。
- unsigned short colinkNtohs (unsigned short ns)
  将网络字节顺序转换为主机字节顺序。
- int32_t colinkSscanf (const char *s, const char *format,...)
  从一个字符串中读进与指定格式相符的数据。
- double colinkStrtod (const char *nptr, char **endptr)
  把字符串转换成双精度浮点数。
- int32_t colinkTolower (int32_t c)
  将字母转换为小写字母。
- char * colinkStrtok (char *s, const char *delim)
  分解字符串为一组字符串

6.7.1 详细描述

Show profile of colink system adapter.

It shows the detail of colink system adapter interface.

作者

Wu Jiale

日期

2018.09.06

6.7.2 函数说明

6.7.2.1 colinkAtoi()

```
int32_t colinkAtoi (
            const char * str )
```

把字符串转换成整型数。

描述:

如果第一个非空格字符存在，是数字或者正负号则开始做类型转换， 之后检测到非数字(包括结束符 \0) 字符时停止转换。

参数

str	[IN] 字符串的首地址

返回值

整型数	

6.7.2.2 colinkFree()

```
void colinkFree (
            void * pt )
```

释放指针pt所占用的内存空间。

描述:

根据用户提供的指针pt，释放其所占用的空间。指针必须指向合法内存。

参数

pt	[IN] 指针

返回值

无	

6.7.2.3 colinkGettime()

```
int32_t colinkGettime (
            uint32_t * ms )
```

获取系统运行时间。

描述:

获取系统运行时间，以毫秒为单位。

参数

ms	[OUT] 当前时间

返回值

COLINK_NO_ERROR	获取成功
-1	获取失败

6.7.2.4 colinkHtons()

```
unsigned short colinkHtons (
            unsigned short hs )
```

将主机字节顺序转换为网络字节顺序。

描述:

把用户提供的主机字节顺序的16位数转换为网络字节顺序表示的16位数。

参数

ns	[IN] 主机字节顺序表示的16位数

返回值

网络字节顺序表示的*16*位数	

6.7.2.5 colinkMalloc()

```
void* colinkMalloc (
            uint32_t n )
```

获取所需的内存空间。

描述:

分配所需的内存空间，并返回一个指向它的指针。指针必须指向合法内存。

参数

n	[IN] 需要分配内存大小。

返回值

NULL	分配成功
非*NULL*	分配失败

6.7.2.6 colinkMemcmp()

```
int32_t colinkMemcmp (
            const void * buf1,
            const void * buf2,
            uint32_t len )
```

比较两块内存区域。

描述:

根据用户提供的内存首地址及长度，比较两块内存的前len个字节。指针必须指向合法内存。

参数

buf1	[IN] 内存1的首地址
buf2	[IN] 内存2的首地址
len	[IN] 要比较的字节数

返回值

等于*0*	buf1等于buf2
小于*0*	buf1小于buf2
大于*0*	buf1大于buf2

6.7.2.7 colinkMemcpy()

```
void* colinkMemcpy (
            void * dst,
            const void * src,
            uint32_t len )
```

复制内存中的内容。

描述:

从源src所指的内存地址的起始位置开始复制len个字节到目标dst所指的内存地址的起始位置中。

参数

dst	[IN/OUT] 目标内存的起始位置
src	[IN] 源内存的起始位置
len	[IN] 要复制的字节数

6.7.2.8 colinkMemset()

```
void* colinkMemset (
            void * dst,
            int32_t c,
            uint32_t len )
```

在一段内存块中填充某个给定的值。

描述:

将dst中前len个字节用c替换并返回dst。

参数

dst	[IN/OUT] 目标的起始位置
c	[IN] 要填充的值
len	[IN] 要填充的值字节数

6.7.2.9 colinkNetworkState()

```
int32_t colinkNetworkState (
            int32_t * state )
```

获取网络状态。

描述:

获取网络是否可以与外网通信的状态。当设备连上路由器且分配IP， state为1.当未连接路由器或已连接路由器但未分配IP，state为0.

参数

state	[OUT] 网络状态,取值范围为[0,1]

返回值

0	获取成功
非*0*	获取失败

6.7.2.10 colinkNtohs()

```
unsigned short colinkNtohs (
            unsigned short ns )
```

将网络字节顺序转换为主机字节顺序。

描述:

把用户提供的网络字节顺序的16位数转换为主机字节顺序表示的16位数。

参数

ns	[IN] 网络字节顺序表示的16位数

返回值

主机字节顺序表示的*16*位数	

6.7.2.11 colinkPrintf()

```
int32_t colinkPrintf (
            const char * format,
             ...  )
```

格式化输出函数。

描述:

格式化输出，一般用于向标准输出设备按规定格式输出信息。

参数

format	[IN] 格式控制信息
...	[IN] 可选参数，可以是任何类型的数据

返回值

小于*0*	输出失败
大于等于*0*	输出的长度

6.7.2.12 colinkRand()

```
uint32_t colinkRand (
            void  )
```

随机数发生器的初始化函数。

描述:

根据系统提供的种子值，产生随机数。

参数

无	

返回值

随机数	

6.7.2.13 colinkRealloc()

```
void* colinkRealloc (
            void * ptr,
            uint32_t n )
```

调整一块内存空间大小。

描述:

尝试重新调整之前调用 colinkMalloc 所分配的 ptr 所指向的内存块的大小。

参数

n	[IN] 需要分配内存大小。

返回值

NULL	重新分配成功。
非*NULL*	返回指针指向重新分配大小的内存。

6.7.2.14 colinkSha256()

```
bool colinkSha256 (
            char * in,
            char * checkout )
```

计算sha256校验值函数

描述:

根据传入的字符串，计算sha256校验值，将结果返回到checkout

参数

in	[IN] 传入的字符串
checkout	[OUT] 计算后的sha256校验值字符串

返回值

true	计算校验值成功
false	计算校验值失败

6.7.2.15 colinkSnprintf()

```
int32_t colinkSnprintf (
            char * buf,
            uint32_t size,
            const char * format,
             ...  )
```

字符串格式化函数。

描述:

把格式化的数据写入某个字符串缓冲区。

参数

buf	[IN/OUT] 指向将要写入的字符串的缓冲区
size	[IN] 要写入的字符的最大数目(含'\0'，超过size会被截断,末尾自动补'\0'
format	[IN] 格式控制信息
...	[IN] 可选参数，可以是任何类型的数据

返回值

小于*0*	写入失败
大于等于*0*	成功则返回欲写入的字符串长度

6.7.2.16 colinkSprintf()

```
int32_t colinkSprintf (
            char * buf,
            const char * format,
             ...  )
```

字符串格式化函数。

描述:

把格式化的数据写入某个字符串缓冲区。

参数

buf	[IN/OUT] 指向将要写入的字符串的缓冲区
format	[IN] 格式控制信息
...	[IN] 可选参数，可以是任何类型的数据

返回值

小于*0*	写入失败
大于等于*0*	写入的字符串长度

6.7.2.17 colinkSscanf()

```
int32_t colinkSscanf (
            const char * s,
            const char * format,
             ...  )
```

从一个字符串中读进与指定格式相符的数据。

描述:

从一个字符串中读进与指定格式相符的数据。

参数

buf	[IN] 检索的字符串
format	[IN] 格式化的字符串
...	[IN] 可选参数，可以是任何类型的数据

返回值

小于*0*	失败
大于*0*	成功填充的参数列表中的项数

6.7.2.18 colinkStrcat()

```
char* colinkStrcat (
            char * dst,
            const char * src )
```

连接字符串。

描述:

把src所指字符串添加到dst结尾处实现字符串的连接，连接过程覆盖dst结尾处的'/0'。

参数

dst	[IN] 目标字符串的指针
src	[IN] 源字符串的指针

返回值

dst	目标字符串的指针

6.7.2.19 colinkStrchr()

```
char* colinkStrchr (
            char * str,
            int32_t ch )
```

查找一个字符在一个字符串中首次出现的位置。

描述:

查找一个字符ch在另一个字符串str中末次出现的位置，并返回这个位置的地址。 如果未能找到指定字符，那么函数将返回NULL。

参数

str	[IN] 字符串的首地址
ch	[IN] 要查找的字符

返回值

Null,未能找到指定字符	
非*Null*,字符首次出现位置的地址	

6.7.2.20 colinkStrcmp()

```
int32_t colinkStrcmp (
            const char * str1,
            const char * str2 )
```

比较两个字符串的大小。

描述:

根据用户提供的字符串首地址及长度，比较两个字符串的大小。指针必须指向合法内存。

参数

str1	[IN] 字符串1的首地址
str2	[IN] 字符串2的首地址

返回值

等于*0*	str1等于str2
小于*0*	str1小于str2
大于*0*	str1大于str2

6.7.2.21 colinkStrcpy()

```
char* colinkStrcpy (
            char * dst,
            const char * src )
```

复制字符串中的内容。

描述:

把src所指向的字符串中以src地址开始复制到dst所指的数组中，并返回dst。

参数

dst	[IN] 目标字符数组
src	[IN] 源字符串的起始位置

返回值

dst	目标字符数组的首地址

6.7.2.22 colinkStrlen()

```
uint32_t colinkStrlen (
            const char * src )
```

计算给定字符串的长度。

描述:

计算给定字符串的（uint32_t型）长度，不包括'\0'在内。

参数

src	[IN] 字符串的首地址

返回值

字符串的长度	

6.7.2.23 colinkStrncat()

```
char* colinkStrncat (
            char * dst,
            const char * src,
            uint32_t len )
```

连接字符串。

描述:

把src所指字符串添加到dst结尾处实现字符串的连接，连接过程覆盖dst结尾处的'/0'。

参数

dst	[IN] 目标字符串的指针
src	[IN] 源字符串的指针
len	[IN] 字符串的长度

返回值

dst	目标字符串的指针

6.7.2.24 colinkStrncmp()

```
int32_t colinkStrncmp (
            const char * str1,
            const char * str2,
            uint32_t len )
```

比较两个字符串的大小。

描述:

根据用户提供的字符串首地址及长度，比较两个字符串的大小。指针必须指向合法内存。

参数

str1	[IN] 字符串1的首地址
str2	[IN] 字符串2的首地址
len	[IN] 字符串的长度

返回值

等于*0*	str1等于str2
小于*0*	str1小于str2
大于*0*	str1大于str2

6.7.2.25 colinkStrncpy()

```
char* colinkStrncpy (
            char * dst,
            const char * src,
            uint32_t len )
```

复制字符串中的内容。

描述:

把src所指向的字符串中以src地址开始的前len个字节复制到dst所指的数组中，并返回dst。

参数

dst	[IN] 目标字符数组
src	[IN] 源字符串的起始位置
len	[IN] 要复制的字节数

返回值

dst	目标字符数组的首地址

6.7.2.26 colinkStrrchr()

```
char* colinkStrrchr (
```

```
            const char * str,
            char c )
```

查找一个字符在一个字符串中末次出现的位置。

描述:

查找一个字符c在另一个字符串str中末次出现的位置，并返回这个位置的地址。 如果未能找到指定字符，那么函数将返回NULL。

参数

str	[IN] 字符串的首地址
c	[IN] 要查找的字符

返回值

Null	未能找到指定字符
非*Null*	字符末次出现位置的地址

6.7.2.27 colinkStrstr()

```
char* colinkStrstr (
            const char * s1,
            const char * s2 )
```

查找字符串第一次出现的位置。

描述:

在字符串s1中查找第一次出现字符串s2的位置，不包含终止符 '\0'。

参数

s1	[IN] 字符串1的首地址
s2	[IN] 字符串2的首地址

返回值

非*NULL*	找到第一次出现的s1字符串位置。
NULL	未找到。

6.7.2.28 colinkStrtod()

```
double colinkStrtod (
            const char * nptr,
            char ** endptr )
```

把字符串转换成双精度浮点数。

描述:

扫描参数nptr字符串，跳过前面的空白符，直到遇上数字或正负符号才开始做转换， 到出现非数字或字符串结束时('\0')才结束转换，并将结果返回。

参数

nptr	[IN] 字符串的首地址
endptr	[OUT] 可以为NULL，若不为NULL则会将遇到不合条件而终止的nptr中的字符指针由endptr传回

返回值

双精度浮点数	

6.7.2.29 colinkStrtok()

```
char* colinkStrtok (
            char * s,
            const char * delim )
```

分解字符串为一组字符串

描述:

分解字符串为一组字符串,详细参考strtok函数。

参数

s	[IN] 要分解的字符
delim	[IN] 分隔符字符

返回值

从*s*开头开始的一个被分割的串。	

6.7.2.30 colinkTolower()

```
int32_t colinkTolower (
            int32_t c )
```

将字母转换为小写字母。

描述:

将字符转换成小写字母,非字母字符不做出处理。

参数

c	[IN] 需要转换的字符

返回值

当*c*是大写字母字符返回对应小写字母字符，其它值不处理返回*c*。	

6.8 include/colink_typedef.h 文件参考

Show profile of colink typedef.

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

宏定义

- #define **bool** int32_t
- #define **true** (1)
- #define **false** (0)
- #define **TRUE** (1)
- #define **FALSE** (0)
- #define **NULL** (0)

6.8.1 详细描述

Show profile of colink typedef.

It shows the detail of colink typedef.

作者

Wu Jiale

日期

2018.09.06

6.9 include/colink_user_timer.h 文件参考

Show profile of colink user timer.

```
#include "colink_typedef.h"
```

宏定义

- #define **__COLINK_TIMER_DEF__**
- #define **COLINK_SINGLE_TIMER** (0)
- #define **COLINK_REPEAT_TIMER** (1)
- #define **COLINK_TIMER_CANCEL** (0)
- #define **COLINK_TIMER_RUNNING** (1)
- #define **COLINK_TIMER_TIMEOUT** (2)
- #define **COLINK_TIMER_NOT_EXIST** (3)
- #define **COLINK_TIMER_NO_ERR** (0)
- #define **COLINK_TIMER_OPERATE_FAILED** (1)
- #define **COLINK_TIMER_NOT_FIND** (2)

类型定义

- typedef int32_t **COLINK_TIMER**

函数

- bool colinkUserTimerInit (void)
  初始化*colink*定时器
- COLINK_TIMER colinkUserTimerAdd (int32_t timerType)
  新增定时器
- int32_t colinkUserTimerSet (COLINK_TIMER fd, uint32_t nms, void(∗colinkTimerTimeoutCb)(COLINK_TIMER))
  设置定时器
- int32_t colinkUserTimerDel (COLINK_TIMER fd)
  删除定时器
- int32_t colinkUserCheckTimer (COLINK_TIMER fd)
  查看定时器
- int32_t colinkUserTimerProcess (void)
  定时器进程

6.9.1 详细描述

Show profile of colink user timer.

It shows the detail of colink user timer.

作者

Huang Xunyan

日期

2018.12.01

版权所有

6.9.2 函数说明

6.9.2.1 colinkUserCheckTimer()

```
int32_t colinkUserCheckTimer (
            COLINK_TIMER fd )
```

查看定时器

描述

查看一个定时器定时状态

参数

COLINK_TIMER	定时器描述符

返回值

ColinkTimerStatus	

参见

ColinkTimerStatus

6.9.2.2 colinkUserTimerAdd()

```
COLINK_TIMER colinkUserTimerAdd (
            int32_t timerType )
```

新增定时器

描述

新增一个定时器

参数

ColinkTimerType	定时器类型

返回值

COLINK_TIMER	定时器描述符

参见

ColinkTimerType

6.9.2.3 colinkUserTimerDel()

```
int32_t colinkUserTimerDel (
            COLINK_TIMER fd )
```

删除定时器

描述

删除一个定时器

参数

COLINK_TIMER	定时器描述符,传入(-1)表示删除所有定时器

返回值

ColinkTimerErrorCode	

参见

ColinkTimerErrorCode

6.9.2.4 colinkUserTimerInit()

```
bool colinkUserTimerInit (
            void  )
```

初始化colink定时器

描述

初始化colink定时器

参数

NULL	无

返回值

true	init success
false	init failed

6.9.2.5 colinkUserTimerProcess()

```
int32_t colinkUserTimerProcess (
            void  )
```

定时器进程

描述

定时器进程，如果使用了定时器超时回调，需要一直循环调用该函数。

返回值

ColinkTimerErrorCode	

参见

ColinkTimerErrorCode

6.9.2.6 colinkUserTimerSet()

```
int32_t colinkUserTimerSet (
            COLINK_TIMER fd,
            uint32_t nms,
            void(*)(COLINK_TIMER) colinkTimerTimeoutCb )
```

设置定时器

描述

设置一个定时器

参数

fd	定时器描述符
nms	定时时间（ms）,0代表取消定时器
timeoutCb	超时回调函数,不需要填NULL

返回值

ColinkTimerErrorCode	

参见

ColinkTimerErrorCode

Index