Weline Windows SDK 开发手册

Weline.io

关于本手册

版权声明

信息更新

本文档仅用于为NDA用户提供信息，并且随时可由Weline.io更改或撤回。

免责条款

根据适用法律的许可范围，Weline.io按“原样”提供本文档而不承担任何形式的担保，包括（但不限于）任何隐含的适销性、特殊目的适用性或无侵害性。在任何情况下，Weline.io都不会对最终用户或任何第三方因根据说明文档使用SDK/后台接口造成的任何直接或间接损失或损坏负责，即使Weline.io明确得知这些损失或损坏，这些损坏包括（但不限于）利润损失、业务中断、信誉或数据丢失。

阅读对象

本文的读者对象为和 Weline.io 签署了 NDA（保密协议）的第三方厂商的研发人员。

密级

本文的密级为秘密，禁止向非 NDA 公司或个人开放此文件。

目录

修订记录

日期	修订版本	描述
2017/11/01	1.0.0	初稿完成
2017/11/03	1.0.1	补充状态信息
2017/11/05	1.0.2	修改 login 返回值
2017/11/05	1.0.3	增加 free 函数
2017/11/09	1.0.4	修正一些小细节
2018/03/13	1.0.5	增加 baseinfo, remove user, cancel login, read_events
2018/03/21	1.0.6	SDK 整合成安装包
2019/04/30	1.0.7	增加 5.15 节 cmapi_tunnle_check 接口
2020/08/14	1.0.8	1、调整 cmapi_login 接口以支持多种方式登录，方便接入方按需使用 2、5.1 小节新增部分 DR 定义和 AUTH_METHOD 定义
2020/10/18	1.0.9	1. 登录状态增加 *CS_AUTHENTICATED* 表示与后台身份认证已经完成 2. 设备变化事件通知 dtype 字段变成 device_type

1. 概要

此文档描述了第三方厂商集成 Weline.io 时，在 MS Windows 平台上所使用的 Weline.io SDK，其目的是让第三方厂商的研发人员能快速将 Weline.io 功能嵌入到第三方应用程序中。

此文档适合公司内部和签署了 NDA 的第三方研发人员阅读。

2. 名词缩写

缩写	全称	说明
MWSDK	Weline.io Windows SDK	Weline.io 提供的 Windows SDK

3. 系统需求

使用 Weline.io SDK 的 应用程序必须以 Administrator 以上的身份运行，可考虑使用系统服务的方式。

运行的目标系统要求 Windows Vista 以上，Windows XP 及以下的系统不提供技术支持。

4. 准备工作

在使用 Weline.io SDK 之前，请联系 Weline.io Inc. 获取 SDK 使用授权，取得授权后，将得到 partnerId(合作伙伴 ID)，appId(应用 ID)。

随本文档一起分发的 Weline.io SDK 包，包括以下内容：

- cmapi.h 为 SDK 的头文件，定义了 SDK 中的函数原型

- MWSDKInstaller.exe 为 SDK 安装包，其中 r00 为 SDK 版本号，安装方法请见下一小节
- cmapi.lib 为链接时所需要的 library 文件

4.1 安装 SDK

运行 Weline.io SDK 之前，必须先安装 Weline.io SDK，安装方法如下：

```
MWSDKInstaller.exe /S /D=c:\xxx
```

其中，

/S (大写 S)：静默安装，不显示安装界面，此参数可选

/D (大写 D)：安装目录，此参数必须指定，且只能是最后一个参数，并且路径不能带引号。

此命令行安装程序将返回安装执行结果：

返回值	说明
0	安装成功
3	安装成功，安装后需要重启系统
4	安装失败
5	安装前需要重启系统

4.2 部署相关

- 开放主程序 UDP 通讯

  调用 SDK 的主程序需要在本地防火墙上开放 UDP 通讯，指令如下（应该在软件安装包中执行）：

  netsh advfirewall firewall add rule name="$name" description="$desc" dir=in profile=any protocol=udp program="$program" action=allow

  在制作安装包时，需要根据实际情况替换指令中$变量部分

  1) name: 防火墙规则名称，可自行定义
  2) desc: 规则描述，自行定义
  3) program: 主程序的全路径

  卸载时应该执行如下指令：

  netsh advfirewall firewall delete rule name="$name"

5. API 说明

5.1 常量定义

- *连接状态*

```
enum CONNECTION_STATUS {
   CNS_UNKNOWN = 0,       // 未知状态
   CNS_PREPARE = 1,       // 准备就绪
   CNS_CONNECTING = 2,    // 正在连接
   CNS_CONNECTED = 3,     // 已连接
   CNS_DISCONNECTED = 4,  // 已断开连接
    CNS_AUTHENTICATED = 5,  //身份认证完成
   CNS_UNACTIVE = 6,              //设备未激活（Windows 未使用）
   CNS_ACTIVING = 7,              // 设备激活中（Windows 未使用）
   CNS_DISCONNECTING = 8,         // 正在断开连接
   CNS_WAIT_RECONNECTING = 9, //正在等待重连(不关心的可以等效于状态 2)
}
```

- *事件类型*

```
enum EVENT_CLASS {
      EC_NULL = 0,
      EC_STATUS_CHANGE = 1,   //连接状态发生改变
      EC_DEVICE_CHANGE = 2,   // 设备发送改变
      EC_SWITCHNET_COMPLETE = 3, // 切换网络事件
}
```

- *连接断开原因*

```
enum _DISCONNECT_REASON {
      DR_UNSET = 0,
      DR_BY_USER,              // disconnected by user
      DR_MISVERSION,           // communication version mismatched
      DR_NETWORK_TIMEOUT,    // network timeout
```

```
    DR_MISSING_INFO,        // request user info
    DR_INVALID_USER,        // invalid user or user inactive or user locked
    DR_INVALID_PASS,        // invalid password
    DR_DEVICE_DELETED,   // device deleted
    DR_DEVICE_ONLINE,   // other device which have same device id on line already
    DR_DEVICE_DISABLED,   // device disabled
    DR_MAX_DEVICE,          // max device reached
    DR_NO_NETWORK,          // no network can enter
    DR_KO_USER_REMOVED,   // kick out cos user be removed from network
    DR_KO_DEVICE_REMOVED, // kick out cos device be removed from network
    DR_KO_DEVICE_DELETED, // device deleted
    DR_TUN_DEVICE,          // tunnel device can not be open
    DR_DTYPE_CHANGED,
    DR_NO_GATEWAY ,         //no gateway can use
    DR_USER_RELOGIN,        //user relogins
    DR_ALGO_TYPE_CHANGED,     //user algo type changed
    DR_AUX_AUTH_DISMATCH,
    DR_INVALID_AUTHORIZATION,
    DR_PROTOCOL,                //通讯协议格式错误
    DR_NETWORK_BROKEN,          //网络中断
    DR_KICKOUT_BY_AS,           //被后台踢下线
    DR_INVALID_SMS,             //无效的短信口令
    DR_INVALID_TICKET,          //无效的令牌，令牌过期
    DR_TRY_TOO_MANY_TIMES,    //尝试次数过于频繁
    DR_INVALID_DEVICE_CLASS,    //无效的设备类型
    DR_CALL_THIRD_API_FAIL,
    DR_INVALID_CODE,
    DR_WAKE_UP,             //机器从睡眠中唤醒
 } DISCONNECT_REASON;

typedef enum
```

```
{
    AUTH_METHOD_NONE = 0x0,
    AUTH_METHOD_ACCPWD = 0x1,   //静态账号密码登录
    AUTH_METHOD_SMS = 0x2,      //短信口令登录
    AUTH_METHOD_RECONNECT_BY_TICKET=0x3, //令牌登录
    AUTH_METHOD_THIRD_CODE = 0x4,   //windows 平台未使用
    AUTH_METHOD_PHONE_TOKEN = 0x65, //windows 平台未使用
} AUTH_METHOD;
```

5.2 初始化

在调用 SDK 内其他 API 之前，必须调用一次初始化函数，以便对 SDK 进行初始化工作。

函数原型

```
bool cmapi_init ( const char *pertnerId, const char *appId, const u_int deviceClass
, bool dns_divert = true);
```

参数说明

- *partnerId, appId, deviceClass: 向 Weline.io 申请的合作伙伴授权信息*
- *dns_divert: 是否开启 dns 重定向，默认开启，不需要虚拟域名解析的可以关闭*

返回值

返回 true 时表示初始化成功，false 时表示失败，此时不应该再调用 SDK 内其他函数

5.3 获取基本信息

函数原型

```
char* cmapi_get_baseinfo ();
返回值为json 串：
{
   “version”: “x.x.x.x”,  // 当前版本号
   “users”: [ {“account”: “xx”}], // 曾成功登录过的账号
   “account”: “xxx”, // 最后一次登录的账号
   “uid”: “xxx”, // 用户 Id （唯一编号）
   “domain”: “xx”, // 本机虚拟域名
   “ad_domain”: “xx”, // 如果本机属于 AD 域，此为 AD 域名
   “vip”: “x.x.x.x”, // 虚拟 IP
   “gatewayVip”: “x.x.x.x”, // Gateway 虚拟域名
   “timestamp”: xxx, // 登录时间戳，秒 (from 1970)
   “ticket”: “xxx”, // 认证票据
   “netid”: “xxx”, // 当前网络 Id
   “snid”: “xx”, // 当前选中的节点 Id
   “dlt”: true | false // 是否开启 DLT 功能
   "api_gw_info": { "domain": "xxxx" } //第三方调用后台接口时使用的域名
}
```

说明：

返回的字符串在使用完后必须调用 cmapi_aux_free 来销毁

5.4 移除历史账号

```
函数原型
bool cmapi_remove_user (const char *account);
参数说明
● account: 登录账号，可通过 cmapi_get_baseinfo 中的 users 项获得
返回值
true 表示成功，false 表示失败
```

5.5 登录

函数原型

*bool cmapi_login (const char *account, const char *password, const u_int auth_method);*

参数说明

- *account: 登录账号，如果成功登录过，此值可传 NULL，表示使用上一次的账号登录*
- *password: 登录密码，如果成功登录过，此值可传 NULL，表示使用上一次的密码登录*
- *auth_method：登录方式，参见 5.1 常量定义小节 AUTH_METHOD*

返回值

true 表示成功，false 表示失败

说明：

1、登录过程是异步的，即调用此方法成功后，仅表示成功开始了登录过程，必须通过 EC_STATUS_CHANGE 事件来获知登录的结果，如果从 connecting 到 connected，表示登录成功，如果从 connecting 到 disconnected，表示登录失败，具体的失败原因通过调用 cmapi_get_status 中的 disconnect_reason 来获取

*2、由于短信口令密码是一次性的，所以新增 AUTH_METHOD 方式为 3 的令牌登录。用户首次使用短信口令登录成功后，后台会下发一个临时令牌，在令牌有效期内客户端可以使用 AUTH_METHOD 方式为 3 的令牌方式进行免密登录，登录成功后会更新令牌，如果长时间不登录，令牌过期(*DR_INVALID_TICKET*)将需要重新输入短信口令登录。*

5.6 登出

函数说明

void cmapi_logout ();

说明: 和登录过程一样，登出的过程也是异步操作，需要等待状态从 connected 转换到 disconnected 才完成整个登出操作

5.7 终止登录过程

函数说明

void cmapi_cancel_login ();

说明: 在 cmapi_login 调用成功后，状态转换成 CONNECTED 之前，都可以调用 cmapi_cancel_login 来终止登录过程。

5.8 获取当前状态

函数原型

*char * cmapi_get_status ();*

返回值为json 串：

```
{
  "status": xx, // CONNECTION_STATUS 类型
  "data": {
    "account": "xxx", // 当前账号, CNS_CONNECTING 和 CNS_DISCONNECTED 时有值
    "reason": xx, // DISCONNECT_REASON 类型，CNS_DISCONNECTED 时有值
    "network_status": {
      "rx_bytes": "xxx", // 接收到的字节数
      "rx_speed": xxx, // 当前接收数据的速率
      "tx_bytes": "xxx", // 发送的字节数
      "tx_speed": xxx, // 当前发送数据的速率
      "latency": xx  // 到网关的时延，单位 ms
    }
  }
}
```

说明：

返回的字符串在使用完后必须调用 cmapi_aux_free 来销毁

5.9 获取网络列表

```
函数说明
char*   cmapi_get_networks ();
返回值为json 串：
{
   “netid”: “xx”, //  当前网络 Id
   “data”: [
      {
         “id”: “xxx”, //  网络 Id
         “name”: “xx”, //  网络名称
         “owner”: “xx”, //  网络属主
      }
   ]
}
```

说明: 此接口必须在 CONNECTED 状态才能调用

返回的字符串在使用完后必须调用 cmapi_aux_free 来销毁

5.10 获取设备列表

```
函数原型
char *cmapi_get_devices ()
返回值为json 串：
{
   “devices”: [
      {
         “id”: “xx”, //  设备 Id
         “owner”: “xxx”, //  设备属主
         “userid”: “xx”, //  设备属主用户 Id
         “domain”: “xxx”, //  设备域名
         “adDomain”: “xxx”, //  设备 AD 域名
         “os”: x, //  设备操作系统类型
```

```
        "vip": "x.x.x.x", //  虚拟 IP
        "ver": "xxx", //  软件版本号
        "pubip": "x.x.x.x", //  设备公网 IP
        "privip": "x.x.x.x", //  设备内网 IP
        "dlt": true|false, //  是否已建立 DLT 连接
      }
  ],
  "smartnodes": [
      {
        "id": "xxxx", //  节点 Id
        "domain": "xxx", //  设备域名
        "adDomain": "xxx", //  设备 AD 域名
        "os": x, //  设备操作系统类型
        "vip": "x.x.x.x", //  虚拟 IP
        "ver": "xxx", //  软件版本号
        "pubip": "x.x.x.x", //  设备公网 IP
        "privip": "x.x.x.x", //  设备内网 IP
        "dlt": true|false, //  是否已建立 DLT 连接
        "subnet": [   //  子网信息
          { "net": "x.x.x.x",   "mask": "x.x.x.x" }
        ]
      }
  ]
}
```

说明：

Smartnodes 信息中的 id 是节点 Id，此值用于选择节点时的参数

返回的字符串在使用完后必须调用 cmapi_aux_free 来销毁

5.11 节点选择

函数原型

```
bool cmapi_select_snode (const char *snId);
```

参数说明

snId: 节点 Id，在 cmapi_get_devices 和 EC_DEVICE_CHANGE 事件中的设备信息的设备 Id

5.12 切换网络

函数原型

*bool cmapi_switch_network (const char *netId);*

参数说明

netId: 要切换的网络 Id

返回值: true 为成功，false 为失败

说明：切换网络*过程是异步的，即调用此方法成功后，仅表示成功开始了切换过程，必须通过 EC_SWITCHNET_COMPLETE 事件来获知切换的结果，*

5.13 读取事件

函数原型

char cmapi_read_events (int timeout);*

参数说明

timeout:在超时时间内(单位毫秒)，如果没有事件，将返回 NULL

返回值为json 串(事件格式)：

```
{
  “event”: xx, // 事件类型，见 EVENT_CLASS 定义
  “timestamp”: {
    “seconds”: xx, // 秒
    “useconds”: xx // 毫秒
  },
  “message”: { … } // 事件内容，见下面各类型的定义
}
```

- *EC_STATUS_CHANGE*

  “messge”: {

"from": xx, //CONNECTION_STAT类型，前一个状态

"to": xx, // CONNECTION_STAT类型，当前状态

}

- *EC_DEVICE_CHANGE*

"message": {

"status": "online|offline", // 对方上线/下线

"device_type": xx, //设备类型

"id": "xxx", //设备 Id

"owner": "xxx", // 设备属主

"uid": "xxx", // 设备属主用户 Id

"domain": "xxx", // 设备域名

"adDomain": "xxx", // 设备 AD 域名(如果设备在 AD 域中)，设备属性

"name": "xxx", // 设备名称

"os": xxx, // 设备操作系统类型

"vip": "x.x.x.x", // 虚拟 IP

"ver": "x.x.x.x", // 软件版本

"pubip": "x.x.x.x", // 公网IP

}

- *EC_DLT_EVENT*

"message": {

"connected": true|false, //建立连接/拆除连接

"vip": "x.x.x.x", //对方虚拟IP

}

- *EC_SWITCHNET_COMPLETE*

"message": {

"result": x, // 0表示切换网络完成

"id": "xxx", // 网络Id (当result为0时有值)

}

说明：返回的字符串在使用完后必须调用 cmapi_aux_free 来销毁

5.14 辅助函数 – 释放内存

函数原型

*void cmapi_aux_free (char *str);*

参数说明

str: 通过调用 cmapi_get_status 和 cmapi_get_devices 等返回的字符串

5.15 辅助函数 – 检测虚拟隧道是否工作

函数原型

*bool cmapi_tunnle_check(u_int timeout_second, const char * dest_vip = NULL);*

参数说明

timeout_second: 指定超时时间，单位秒

dest_vip: 检测的目标虚地址，如果为空则检测与虚拟网关的隧道

说明:

该函数意在帮助通过虚拟网访问对端设备之前，确认到对端的隧道网络状态是否为可用

5.16 设置 DLT 状态

函数原型

bool cmapi_enable_dlt(bool enabled);

参数说明

enabled: true 开启 DLT，false 关闭 DLT

返回值: true 表示操作成功，false 表示无权限执行此操作

说明:

DTl 设置用于帮助设备之间点对点通讯，以便提升虚拟网隧道传输速度，DLT 设置默认为 TURE