SPSYSTEM

SP Native API 交易接口

说明书

文档更新日期 **2017/07/31**

内容目錄

1.介绍

SP交易系统API接口是一个基于C, C++的类库,通过扩展类库提供的接口来实现相关交易功能,其中包括订单处理,持仓查询,成交记录查询,价格订阅,市场成交记录查询,户口资料查询等.该类库下面包含以下文件:

Windows			
Example开发环境：Windows + Visual Studio 2010(VC++)			
文件名	版本	日期	备注
spapidllm32.dll spapidllm64.dll	V1.0, R8.75.3	20170829	动态连接
spapidll.h	V1.0, R8.75.3	20170829	接口头文件
Libeay32.dll, libssl32.dll, ssleay32.dll. 通信加密所依赖的库.			
Linux			
Example开发环境：CentOS 6 + G++			
libapiwrapper.so	R8.75.3	20170829	共享连接
ApiProxyWrapper.h请求方法	ApiProxyWrapperReply.h回调方法		
SPApiProxyDataType.h数据结构	CCTypes.h		

API R8.70 与之前版本的区别

淘汰函数(请求)			
API 循环	SPAPI_Poll	设置 API 后台循环	SPAPI_SetBackgroundPoll
按顺序取持仓	SPAPI_GetPos	根据 No 取成交	SPAPI_GetTradeByTradeNo
已订阅数量	SPAPI_GetPriceCount	按顺序取订阅价格	SPAPI_GetPrice
请求登录前订单	SPAPI_LoadOrderReport	请求登录前成交	SPAPI_LoadTradeReport
取货币兑换率数量	SPAPI_GetCcyRateCount	顺序取兑换率	SPAPI_GetCcyRate
淘汰函数(回调)			
Client 请求 Trade 后最后一个提示	SPAPI_RegisterLoadTradeEnd	AE 请求 Trade 后最后一个提示	SPAPI_RegisterLoadAETradeEnd
新增和修改过的函数			
设置返回字体	SPAPI_SetLanguageId	新增方法,参考:第二部分>1.3	
取登入状态	SPAPI_GetLoginStatus	返回值有变化.参考: 第二部分>1.8	
修改订单	SPAPI_ChangeOrderBy	旧不变,新增一个.参考: 第二部分>1.11 1.12	
删除订单	SPAPI_DeleteOrder	旧淘汰,新增二个.参考: 第二部分>1.16 1.17	
设置订单有效	SPAPI_ActivateOrder	旧淘汰,新增二个.参考: 第二部分>1.18 1.19	
设置订单无效	SPAPI_InactivateOrder	旧淘汰,新增二个.参考: 第二部分>1.20 1.21	
取订单	SPAPI_GetActiveOrder SPAPI_GetOrdersByArray	淘汰按顺序取,修改为一次性取. 参考 1.15-16	
取成交	SPAPI_GetAllTrades SPAPI_GetAllTradesByArray	淘汰按顺序取,修改为一次性取. 参考 1.27-28	
取货币信息	SPAPI_GetAllAccBal SPAPI_GetAllAccBalByArray	淘汰按顺序取,修改为一次性取. 参考 1.41-42	
取产品系列	SPAPI_GetInstrument SPAPI_GetInstrumentByArray	淘汰按顺序取,修改为一次性取. 参考 1.33-34	
取产品信息	SPAPI_GetProduct SPAPI_GetProductByArray	淘汰按顺序取,修改为一次性取. 参考 1.37-38	
用户状态回调	SPAPI_RegisterLoginStatusUpdate	淘汰旧的三个状态回调。修改成一个回调 参考: 第三部分>1.15	
	SPAPI_RegisterPServerLinkStatusUpdate		
	SPAPI_RegisterConnectionErrorUpdate		
登录成功后 AccInfo	SPAPI_RegisterLoginAccInfo	修改为：参考 第三部分>1.18	
Account 登录回调	SPAPI_RegisterAccountLoginReply	新增.：参考 第三部分>1.16	

Account 登出回调	SPAPI_RegisterAccountLogoutReply	新增.：参考 第三部分>1.17
Account 持仓回调	SPAPI_RegisterAccountPositionPush SPAPI_RegisterUpdatedAccountPositionPush	新增.：参考 第三部分>1.19 1.20
Account 账户回调	SPAPI_RegisterUpdatedAccountBalancePush	新增.：参考 第三部分>1.21
Account 成交回调	SPAPI_RegisterTradeReport SPAPI_RegisterLoadTradeReadyPush	新增.: 参考 第三部分>1.8 1.9

注：登录已经更改为两种模式 AE 与普通账户

1. AE 模式下:登录端口为 8081,需要 AE 开通 AE 模式,没开通登录时会自动登出.此模式下用户拿到的与回调的都是 AE 账户下的订单与成交数据.

2.普通账户模式下:登录端口为 8080,此模式下用户拿到的与回调的都是此账户的订单与成交数据.

2． 开发接口

1.1 SPAPI_Initialize 方法

该方法用于API初始化。

函数原型:

int SPAPI_Initialize()

返回值:

0, 代表成功.

1.2 SPAPI_Uninitialize 方法

该方法用于释放API.

函数原型:

int SPAPI_Uninitialize()

返回值:

0, 代表成功.
-1, 用户未登出
-2，释放异常

1.3 SPAPI_SetLanguageId 方法

设定服务器返回信息的字体.此方法在 SPAPI_Initialize 前使用,如不使用,返回字体默认为 0:英文。

函数原型:

int SPAPI_SetLanguageId(int langid)

参数:

langid:

0:英。 1:繁。 2:简

返回值:

0:表示请求成功

1.4 SPAPI_SetLoginInfo 方法

设定登录信息.

函数原型:

void SPAPI_SetLoginInfo(char *host, int port, char *license, char *app_id, char *user_id, char *password)

参数:

host: 服务器地址.
port: 连接端口.
license: 许可加密字符串.
app_id: 应用编号.
user_id: 用户名.
password: 用户密码.

1.5 SPAPI_Login 方法

发送登录请求.

函数原型:

int SPAPI_Login();

返回值:

0, 代表成功发送登录请求.

1.6 SPAPI_Logout 方法

发送登出请求.

函数原型:

int SPAPI_Logout(char *user_id)

参数:

user_id: 登入时 SetLoginInfo的user_id.

返回值:

0, 代表成功发送登出请求.

1.7 SPAPI_ChangePassword 方法

修改用户密码.

函数原型:

int SPAPI_ChangePassword(char *user_id, char *old_psw, char *new_psw)

参数:

user_id: 登入时用户账号
old_psw: 原始密码.
new_psw: 新的密码.

返回值:

0, 代表成功发送登出请求.

1.8 SPAPI_GetLoginStatus 方法

查询交易连接状态与行情连接状态.

函数原型:

```
int  SPAPI_GetLoginStatus(char *user_id short  host_id)
```

参数:

user_id: 登入时用户账号

host_id:

80,81, 表示交易连接.(用户登录状态)

83, 表示一般价格连接.

88, 表示一般资讯连接.

返回值:

1：没有登入，2：连接中，3：已连接，4：连接失败，5：已登出 6：API阻塞

1.9 SPAPI_AddOrder 方法

该方法用来添加订单.

函数原型:

```
int  SPAPI_AddOrder(SPApiOrder *order)
```

参数:

***order,** 表示一个指针订单结构.

订单结构:

```
typedef  struct
{
    double Price;               //价格
    double StopLevel;           //止损价格
    double UpLevel;             //上限水平
    double UpPrice;             //上限价格
    double DownLevel;           //下限水平
    double DownPrice;           //下限价格
    bigint ExtOrderNo;          //外部指示
    int32_t IntOrderNo;             //用户订单编号
    int32_t Qty;                    //剩下数量
    int32_t TradedQty;              //已成交数量
    int32_t TotalQty;           //订单全部数量
    uint32_t ValidTime;             //有效时间
    uint32_t SchedTime;             //预订发送时间
    uint32_t TimeStamp;             //服务器接收订单时间
    u_long OrderOptions;        //0=默认,1=T+1（参考第四部分：1.2）
    STR16 AccNo;                //用户帐号
    STR16 ProdCode;             //合约代号
    STR16 Initiator;            //下单用户
    STR16 Ref;                  //参考
```

```
    STR16 Ref2;                 //参考
    STR16 GatewayCode;          //网关
    STR40 ClOrderId;            //用户自定义参考
    char BuySell;               //买卖方向（参考第四部分：1.1）
    char StopType;              //止损类型
    char OpenClose;             //开平仓
    tinyint CondType;           //订单条件类型
    tinyint OrderType;          //订单类型
    tinyint ValidType;          //订单有效类型
    tinyint Status;             //状态
    tinyint DecInPrice;         //合约小数位
    tinyint OrderAction;
    uint32_t updateTime;
    int32_t updateSeqNo;
} SPApiOrder;
```

返回值:

0， 表示成功.
-1，用户未登入
-2，无记录
-3，价格输入有误
-4，止损/限价,触发价格不正确
-5，增强止损,止损价格不正确
-6，增强止损,追踪市场价格不正确
-7，双向限价,止损价格不正确
-8，开仓/平仓,触发价格不正确
-9，开仓/平仓,止赚价格不正确
-10，开仓/平仓,止损价格不正确

1.10 SPAPI_AddInactiveOrder 方法

该方法用来添加订单.

函数原型:

```
int  SPAPI_AddInactiveOrder(SPApiOrder *order)
```

参数:

***order,** 表示一个指针订单结构.

返回值：0：表示请求成功。
-1，用户未登入
-2，无记录
-3，价格输入有误
-4，止损/限价,触发价格不正确
-5，增强止损,止损价格不正确

-6，增强止损,追踪市场价格不正确
-7，双向限价,止损价格不正确
-8，开仓/平仓,触发价格不正确
-9，开仓/平仓,止赚价格不正确
-10，开仓/平仓,止损价格不正确

1.11 SPAPI_ChangeOrder 方法

该方法用来修改工作中的订单.

函数原型:

```
int  SPAPI_ChangeOrder (char *user_id, SPApiOrder *order, double org_price, long org_qty);
```

参数:

user_id: 登入时用户帐号

***order,** 表示一个指针订单结构.

org_price, 之前订单的价格,(注意:修改订单价格时,要将修改后的价格给结构体中Price,而之前的价格要给参数org_price.)

org_qty, 之前订单中的数量.

返回值: 0，表示成功.

-1，用户未登入
-2，无记录
-3，价格输入有误

1.12 SPAPI_ChangeOrderBy 方法

该方法根据订单编号来修改工作中的订单.

函数原型:

```
int  SPAPI_ChangeOrderBy (char *user_id, char *acc_no, long accOrderNo, double org_price, long org_qty, double newPrice, long newQty);
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

org_price: 之前订单的价格,

org_qty: 之前订单中的数量.

NewPrice: 新价格。

NewQty: 新订单数量(注：暂不支持数量修改，所以newQty要与org_qty一样)

返回值:

0，表示成功.
-1，用户未登入
-2，无记录
-3，价格输入有误

1.13 SPAPI_GetOrderByOrderNo 方法

根据订单编号查询该订单信息.

函数原型:

int SPAPI_GetOrderByOrderNo(char *user_id, char *acc_no, long int_order_no, SPApiOrder *order)

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

int_order_no, 输入要查询的订单编号.(输入)

*order, 表示返回的订单信息.(返回)

返回值:

0, 表示请求成功.

1.14 SPAPI_GetOrderCount 方法

查询订单数量.

函数原型:

int SPAPI_GetOrderCount(char *user_id, char* acc_no)

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

返回值:

订单数量.

1.15 SPAPI_GetActiveOrder 方法

该方法用来获取全部订单信息.

(注:单线程API分LoadOrder+GetOrder=全部订单,而最新多线程此方法一个即取全部,没使用过单线程API客户请忽略些注意)

函数原型:

int SPAPI_GetActiveOrder(char *user_id, char *acc_no, vector<SPApiOrder>& apiOrderList)

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户的账号(Client用户user_id等于acc_no)

apiOrderList: 订单信息列表。

返回值:

0, 表示请求成功.

1.16 SPAPI_GetOrdersByArray 方法

该方法用来获取全部订单信息.和上SPAPI_GetActiveOrder功能相同,使用C malloc array来获取全部订单信息，使用方法可参考Example

函数原型:

int SPAPI_GetOrdersByArray(char *user_id, char *acc_no,SPApiOrder* apiOrderList)

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户的账号(Client用户user_id等于acc_no)

apiOrderList: 订单信息列表。

返回值:

0, 表示请求成功.

1.17 SPAPI_DeleteOrderBy 方法

该方法根据订单编号与合约名来删除订单.

函数原型:

int SPAPI_DeleteOrderBy(char *user_id, char *acc_no, long accOrderNo, char* productCode, char* clOrderId)

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

accOrderNo：订单编号

productCode:订单产品号(合约名)

clOrderid:

返回值:

0, 表示成功.

1.18 SPAPI_DeleteAllOrders 方法

该方法用来一次删除全部订单.

函数原型:

int SPAPI_DeleteAllOrders(char *user_id, char *acc_no)

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

返回值:

0:表示成功.

1.19 SPAPI_ActivateOrderBy 方法

该方法根据订单编号来设置有效订单.

函数原型:

```
int SPAPI_ActiveateOrderBy(char *user_id, char *acc_no, long accOrderNo);
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

accOrderNo: 订单编号

返回值:

0，表示成功.

1.20 SPAPI_ActivateAllOrderOrders 方法

该方法一次将全部订单设置为有效订单.

函数原型:

```
int SPAPI_ActiveateAllOrders(char *user_id, char *acc_no);
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

返回值:

0，表示成功.

1.21 SPAPI_InactivateOrderBy 方法

该方法根据订单编号设置无效订单.

函数原型:

```
int  SPAPI_InactivateOrderBy(char *user_id, char *acc_no, long accOrderNo)
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

accOrderNo: 订单编号

返回值:

0，表示成功.

1.22 SPAPI_InactivateAllOrders 方法

该方法一次将全部订单设置为无效效订单.

函数原型:

```
int SPAPI_InactiveateAllOrders(char *user_id, char *acc_no);
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

返回值:

0，表示成功.

1.23 SPAPI_SendMarketMakingOrder 方法

造市商下单。可 Bid(买)/Ask(沽)同时下单,但只提供简单的限价盘。

函数原型:

```
int SPAPI_SendMarketMakingOrder(SPApiMMOrder *mmorder)
```

参数：

mmorder：造市订单结构参数。

造市订单结构

```
typedef struct
{
    bigint BidExtOrderNo;    //Bid(买)单外部指示
    bigint AskExtOrderNo;    //Ask(沽)单外部指示
    long BidAccOrderNo;      //Bid(买)单编号
    long AskAccOrderNo;      //Ask(沽)单编号
    long BidPrice;           //Bid(买)单价格
    long AskPrice;           //Ask(沽)单价格
    long BidQty;             //Bid(买)单数量
    long AskQty;             //Ask(沽)单数量
    long SpecTime;           //预订发送时间 //箇璹祇瀌  丁
    u_long OrderOptions;     //0=默认,1=T+1
    STR16 ProdCode;          //合约代号 //  [    腹
    STR16 AccNo;             //用户帐号 //ノめ明腹
    STR40 ClOrderId;         //用户自定义参考
     STR40 OrigClOrderId;    //旧用户自定义参考
    tinyint OrderType;       //订单类型 //璹虫摸□
    tinyint ValidType;       //订单有效类型 //璹虫 T   摸□
    tinyint DecInPrice;      //合约小数位 //  [    计
} SPApiMMOrder;
```

返回值:

0，表示成功.

注：

(AskAccOrderNo 且 AskExtOrderNo) 或 (BidAccOrderNo 且 BidAccOrderNo)

1：如果填上工作中订单相对应编号将修改前订单。

2：填 0 表示下新单。

3：如果填上工作中订单相对应编号，且 Qty=0 就是删除此订单。

1.24 SPAPI_GetPosCount 方法

该方法用来获取持仓数量.

函数原型:

```
int SPAPI_GetPosCount(char *user_id)
```

参数:

user_id: 登入时用户帐号

返回值:

持仓数量.

注: 此方法如果是AE登入需要AccountLogin一个客户才能取客户数据。

1.25 SPAPI_GetAllPos 方法

该方法用来获取全部持仓信息.(基于 C++)

函数原型:

```
int SPAPI_GetAllPos(char *user_id, vector<SPApiPos>& apiPosList)
```

参数:

user_id: 登入时用户帐号

apiPosList: 全部持仓信息

注: 此方法如果是 AE 登入需要 AccountLogin 一个客户才能取客户数据。

1.26 SPAPI_GetAllPosByArray 方法

该方法用来获取全部持仓信息.和上 SPAPI_GetAllPos 功能相同, 使用 C malloc array 来获取全部成交信息, 使用方法可参考 Example.

函数原型:

```
int  SPAPI_GetAllPosByArray(char *user_id, SPApiPos* apiPosList);
```

参数:

user_id: 登入时用户帐号

apiPosList: 返回的持仓信息列表(全部持仓).

返回值:

0, 表示请求成功.

1.27 SPAPI_ GetPosByProduct 方法

该方法通过合约代号查询该合约的持仓信息.

函数原型:

```
int  SPAPI_GetPosByProduct(char *user_id, char *prod_code,
SPApiPos *pos)
```

参数:

user_id: 登入时用户帐号
prod_code, 输入要查询的合约代号.
*pos, 返回的持仓信息.

注：此方法如果是AE登入需要AccountLogin一个客户才能取客户数据。

持仓结构体:

```
typedef  struct
{
    int32_t Qty;                //上日仓位
    int32_t DepQty;             //存储仓位
    int32_t LongQty;          //今日长仓
    int32_t ShortQty;         //今日短仓
    double TotalAmt;          //上日成交
    double DepTotalAmt;       //上日存储持仓总数(存储数量价格总和)
    double LongTotalAmt;      //今日长仓总数(长仓数量价格总和)
    double ShortTotalAmt;     //今日短仓总数(短仓数量价格总和)
    double PLBaseCcy;         //盈亏(基本货币)
    double PL;                //盈亏
    double ExchangeRate;      //汇率
    STR16 AccNo;              //用户帐号
    STR16 ProdCode;           //合约代码
    char LongShort;           //上日持仓长短方向
    tinyint DecInPrice;       //小数点
    } SPApiPos;
```

返回值:

0,表示请求成功.

上日持仓	LongShort上日仓位方向B或S	将 **B** 当做 **(+)** 将 **S** 当做 **(-)**
	Qty上日仓位数量	那 **QTY** 等于 （**LongShort**）**Qty**
	TotalAmt上日持仓总额	总额是所有 **Qty** 的价格总和,所以 **TotalAmt** 除以 **Qty** 得到平均价
今长仓	LongQty今日长仓总数量(买:B)	只要是买入 LongQty 将会加大，不会减少。
	LongTotalAmt今日长仓价总和	所有长仓成交价的总和,
	长仓均价	均价等于 LongTotalAmt/LongQty
今短仓	ShortQty今日长仓总数量(沽:S)	只要是买入 ShortQty 将会加大，不会减少。
	ShortTotalAmt今日短仓价总和	所有短仓成交价的总和
	短仓均价	均价等于 **ShortTotalAmt/ShortQty**

今日净仓	数量	今日长仓数量 － 今日短仓数量 ＝ 今日净仓数量		
	均价	今日净仓数量 ＞ 0	(今日长仓价总价－今日短仓总价)÷今日净仓数量	
		今日净仓数量 ＜ 0	(今日短仓价总价－今日长仓总价)÷ABS(今日净仓数量) [取正]	
		今日净仓数量 ＝ 0	均价也为 0.	
净仓	数量	上日持仓数量 ＋ 今日净仓数量 ＝ 净仓数量		
	均价	上日持：**B** 上日持数量+今日长数量=总长数 上日持总价+今日长总价=总长价	总长数≥短仓数	(总长价－短总价)÷净仓数量＝净仓均价
			总长数＜短仓数	(短总价－总长价)÷净仓数量＝净仓均价
		上日持：**S** 上日持数量+今日短数量=总短数 上日持总价+今日短总价=总短价	长仓数≥总短数	(长总价－总短价)÷净仓数量＝净仓均价
			长仓数＜总短数	(总短价－长总价)÷净仓数量＝净仓均价

盈亏(PL)	持仓中“盈亏”计算 (称:总盈亏)	净仓数量＞ 0	(市价－净仓均价)×净仓数量×合约值
		净仓数量＜ 0	(净仓均价－市价)×净仓数量×合约值
		净仓数量＝0	均价有显示此为平仓盈亏点数 平仓盈亏点数×合约值 ＝ 平仓盈亏价
	持仓均价 (高级交易指示，点击交易指示)	根据今日净仓方向计算均价.例今净仓为5	
		方向:买(B)5	以上日持仓均价加最后成交5条买(B)的成交价得到持仓均价。
		方向:沽(S)5	以上日持仓均价加最后成交5条沽(S)的成交价得到持仓均价。
	持仓盈亏 (高级交易指示，点击交易指示)	根据净仓数量与“持仓均价”计算	
		净仓:买(B)	(市价-持仓均价)×数量×合约值
		净仓:沽(S)	(持仓均价-市价)×数量×合约值
	平仓盈亏 （点击交易指示）	总盈亏 － 持仓盈亏 ＝ 平仓盈亏	

盈亏(基本货币)PLBaseCcy	通过 PL 与 ExchangeRate 得到的基本货币的盈亏
ExchangeRate兑换基本货币的兑换率	
产品交易货币计算的盈亏 (持仓盈亏+平仓盈亏)	

1.28 SPAPI_GetTradeCount 方法

查询已成交数量.

函数原型:

```
int  SPAPI_GetTradeCount(char *user_id, char *acc_no)
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

返回值:

成交数量.

1.29 SPAPI_GetAllTrades 方法

该方法用来获取已成交信息.

(注:单线程API分LoadTrader+GetTrader=全部成交,而最新多线程此方法一个即取全部,没使用过单线程API客户请忽略些注意)

函数原型:

```
int  SPAPI_GetAllTrades(char *user_id, char *acc_no,
vector<SPApiTrade>& apiTradeList);
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

apiTradeList: 返回的成交信息列表(全部成交).

成交结构:

```
typedef  struct
{
        double RecNo;                   //成交记录
        double Price;                   //成交价格
        double AvgPrice;                //成交价格
        bigint TradeNo;                 //成交编号
        bigint ExtOrderNo;              //外部指示
        int32_t IntOrderNo;               //用户订单编号
        int32_t Qty;                      //成交数量
        uint32_t TradeDate;               //成交日期
        uint32_t TradeTime;               //成交时间
        STR16 AccNo;                    //用户帐号
        STR16 ProdCode;                 //合约代码
        STR16 Initiator;                //下单用户
        STR16 Ref;                      //参考
        STR16 Ref2;                     //参考
        STR16 GatewayCode;              //网关
```

```
    STR40 ClOrderId;          //用户自定义参考
    char BuySell;             //买卖方向（参考第四部分：1.1）
    char OpenClose;           //开平仓
    tinyint Status;           //状态
    tinyint DecInPrice;       //小数位
    double OrderPrice;
    STR40 TradeRef;
    int32_t TotalQty;
    int32_t RemainingQty;
    int32_t TradedQty;
    double AvgTradedPrice;
} SPApiTrade;
```

返回值:

0, 表示请求成功.

1.30 SPAPI_GetAllTradesByArray 方法

该方法用来获取已成交信息.和上 SPAPI_GetAllTrades 功能相同，使用 C malloc array 来获取全部成交信息，使用方法可参考 Example.

函数原型:

```
int  SPAPI_GetAllTradesByArray(char *user_id, char *acc_no, SPApiTrade* apiTradeList);
```

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

apiTradeList: 返回的成交信息列表(全部成交).

返回值:

0, 表示请求成功.

1.31 SPAPI_SubscribePrice 方法

该方法用来订阅/取消市场数据(行情).

函数原型:

```
int  SPAPI_SubscribePrice(char *user_id, char *prod_code, int mode);
```

参数:

user_id: 登入时用户帐号

prod_code, 要订阅的合约代码.

mode, 订阅的类型.

0:取消市场数据.

1:订阅市场数据.

返回值:

0, 表示请求成功.

注：过多订阅不使用产品的行情，浪费电脑资源，建议只订阅所需要的。

1.32 SPAPI_GetPriceByCode 方法

该方法通过合约代码获取价格信息.

函数原型:

```
int  SPAPI_GetPriceByCode(char *user_id, char *prod_code, SPApiPrice *price);
```

参数:

user_id: 登入时用户帐号

prod_code, 合约代码.

***price,** 价格资讯信息.

返回值:

0, 代表请求成功.

价格资讯结构体:

```
typedef  struct
{
double Bid[SP_MAX_DEPTH];       //买入价
int32_t BidQty[SP_MAX_DEPTH];       //买入量
int32_t BidTicket[SP_MAX_DEPTH];    //买指令数量
double Ask[SP_MAX_DEPTH];       //卖出价
int32_t AskQty[SP_MAX_DEPTH];       //卖出量
int32_t AskTicket[SP_MAX_DEPTH];    //卖指令数量
double Last[SP_MAX_LAST];       //成交价
int32_t LastQty[SP_MAX_LAST];       //成交数量
uint32_t LastTime[SP_MAX_LAST];     //成交时间
double Equil;                   //平衡价
double Open;                    //开盘价
double High;                    //最高价
double Low;                     //最低价
double Close;                   //收盘价
uint32_t CloseDate;                 //收市日期
double TurnoverVol;             //总成交量
double TurnoverAmt;             //总成交额
int32_t OpenInt;                    //未平仓
STR16 ProdCode;                 //合约代码
STR40 ProdName;                 //合约名称
char DecInPrice;                //合约小数位
int32_t ExstateNo;                  //港期市场状态
int32_t TradeStateNo;                //市场状态
bool Suspend;                  //是否停牌
int32_t ExpiryYMD;                  //产品到期日期
```

```
int32_t ContractYMD;                    //合约到期日期
int32_t Timestamp;                      //行情更新时间
} SPApiPrice;
```

1.33 SPAPI_LoadInstrumentList 方法

加载所有合约系列信息.

函数原型:

int SPAPI_LoadInstrumentList()

返回值:

小于 0:表示请求失败.

0:表示请求本地数据成功

大于 0:表示请求对应的 RequestId,请求返回值等余回调返回值.

1.34 SPAPI_GetInstrumentCount 方法

该方法用来获取市场产品系列的数量.

函数原型:

int SPAPI_GetInstrumentCount()

返回值:

市场产品系列的数量.

1.35 SPAPI_GetInstrument 方法

该方法通过产品系列名获取产品系列的信息.

函数原型:

int SPAPI_GetInstrument(vector<SPApiInstrument>& apiInstList)

参数:

apiInstList, 产品系列信息列表(全部已经加载的系列).

产品系列结构:

```
typedef  struct
{
        double Margin;
        double ContractSize;
        STR16 MarketCode;     //市场代码
        STR16 InstCode;       //产品系列代码
        STR40 InstName;       //英文名称
        STR40 InstName1;      //繁体名称
        STR40 InstName2;      //简体名称
        STR4 Ccy;             //产品系列的交易币种
        char DecInPrice;      //产品系列的小数位
        char InstType;        //产品系列的类型
```

```
} SPApiInstrument;
```

返回值:

0, 表示请求成功.

1.36 SPAPI_GetInstrumentByArray 方法

该方法通过产品系列名获取产品系列的信息.和上 SPAPI_GetInstrument 功能相同，使用 C malloc array 来获取全部 Instrument 信息，使用方法可参考 Example.

函数原型:

```
int  SPAPI_GetInstrumentByArray(SPApiInstrument* apiInstList)
```

参数:

apiInstList, 产品系列信息列表(全部已经加载的系列).

返回值:

0, 表示请求成功.

1.37 SPAPI_GetInstrumentByCode 方法

该方法通过产品代码获取该产品的产品系列信息.

函数原型:

```
int  SPAPI_GetInstrumentByCode)(char *inst_code, SPApiInstrument
*inst);
```

参数:

inst_code, 要查询产品系列代码.

***inst,** 产品系列信息.

1.38 SPAPI_GetProductCount 方法

该方法用来获取全部合约的数量.

函数原型:

```
int  SPAPI_GetProductCount()
```

返回值:

市场合约的数量.

1.39 SPAPI_GetProduct 方法

该方法获取已加载的产品合约信息.

函数原型:

```
int SPAPI_GetProduct(vector<SPApiProduct>& apiProdList)
```

参数:

apiProdList, 全部产品信息列表(已经加载的全部产品信息).

产品信息结构:

```
typedef  struct
{
    STR16 ProdCode;       //产品代码
    char ProdType;        //产品类型
    STR40 ProdName;       //产品英文名称
    STR16 Underlying;     //关联的期货合约
    STR16 InstCode;       //产品系列名称
    uint32_t ExpiryDate;     //产品到期时间
    char CallPut;         //期权方向认购与认沽
    int32_t Strike;          //期权行使价
    int32_t LotSize;         //手数
    STR40 ProdName1;      //产品繁体名称
    STR40 ProdName2;      //产品简体名称
    char OptStyle;        //期权类型
    int32_t TickSize;     //产品价格最小变动位数
}SPApiProduct;
```

返回值:

0, 表示请求成功.

1.40 SPAPI_GetProductByArray 方法

该方法获取已加载的产品合约信息.和上 SPAPI_GetProduct 功能相同，使用 C malloc array 来获取已加载的 Product 信息，使用方法可参考 Example.

函数原型:

```
int SPAPI_GetProductByArray(SPApiProduct* apiProdList)
```

参数:

apiProdList, 全部产品信息列表(已经加载的全部产品信息).

返回值:

0, 表示请求成功.

1.41 SPAPI_GetProductByCode 方法

该方法通过合约代码获取该产品的合约信息

函数原型:

```
int SPAPI_GetProductByCode(char *prod_code, SPApiProduct *prod)
```

参数:

prod_code, 要查询的合约代码.

***prod,** 返回的合约产品信息.

返回值:

0, 代表请求成功.

1.42 SPAPI_GetAccBalCount 方法

该方法用来获取现金结余的数量.

函数原型:

```
int  SPAPI_GetAccBalCount(char* user_id)
```

参数:

user_id: 登入时用户帐号

返回值:

返回一个整型的帐户现金结余数量.

注: 此方法如果是AE登入需要AccountLogin一个客户才能取客户数据

1.43 SPAPI_GetAllAccBal 方法

该方法一次全部获取账户现金结余信息.

函数原型:

```
int SPAPI_GetAllAccBal(char *user_id, vector<SPApiAccBal>&
apiAccBalList)
```

参数:

user_id: 登入时用户帐号

apiAccBalList, 全部现金结余信息.

帐户现金结余结构:

```
typedef  struct
{
        double CashBf;          //上日结余
        double TodayCash;       //今日存取
        double NotYetValue;     //未交收
        double Unpresented;     //未兑现
        double TodayOut;        //提取要求
        STR4 Ccy;               //货币
} SPApiAccBal;
```

现金结余 = CashBf + TodayCash + NotYetValue

参考兑换率: 请参考GetCcyRate。

现金(基本货币) 现金结余 * 兑换率

返回值:

0, 表示请求成功。

注: 此方法如果是AE登入需要AccountLogin一个客户才能取客户数据。

1.44 SPAPI_GetAllAccBalByArray 方法

该方法一次全部获取账户现金结余信息. 和上 SPAPI_GetAllAccBal 功能相同, 使用 C

malloc array 来获取已 AccountLogin 的 Bal 信息，使用方法可参考 Example.

函数原型:

int SPAPI_GetAllAccBal(char *user_id, SPApiAccBal* apiAccBalList)

参数:

user_id: 登入时用户帐号

apiAccBalList, 全部现金结余信息.

返回值:

0, 表示请求成功。

注：此方法如果是 AE 登入需要 AccountLogin 一个客户才能取客户数据。

1.45 SPAPI_GetAccBalByCurrency 方法

该方法通过币种获取现金结余信息.

函数原型:

int SPAPI_GetAccBalByCurrency(char *user_id, char *ccy, SPApiAccBal *acc_bal)

参数:

user_id: 登入时用户帐号

ccy, 要查询的币种.

*acc_bal, 现金结余信息.

返回值:

0, 表示请求成功.

1.46 SPAPI_SubscribeTicker 方法

该方法用来订阅/取消指定合约的市场成交信息.

函数原型:

int SPAPI_SubscribeTicker(char *user_id, char *prod_code, int mode)

参数:

user_id: 登入时用户帐号

prod_code, 合约代码.

mode, 0:取消市场成交记录.

1:订阅市场成交记录.

返回值:

0, 表示请求成功.

1.47 SPAPI_SubscribeQuoteRequest 方法

该方法用来订阅/取消指定合约的要求报价行情.

函数原型:

int SPAPI_SubscribeQuoteRequest(char *user_id, char *prod_code, int mode)

参数:

user_id: 登入时用户帐号
prod_code, 合约代码.
mode, 0:取消此合约的要求报价行情.
1:订阅此合约的要求报价行情.

返回值:

0, 表示请求成功.

1.48 SPAPI_SubscribeAllQuoteRequest 方法

该方法用来一次性订阅/取消所有合约要求报价行情.

函数原型:

```
int  SPAPI_SubscribeAllQuoteRequest(char *user_id, int mode)
```

参数:

user_id: 登入时用户帐号
mode, 0:取消全部合约的要求报价行情.
1:订阅全部合约的要求报价行情.

返回值:

0, 表示请求成功.

1.49 SPAPI_GetAccInfo 方法

该方法用来获取帐户信息

函数原型:

```
int SPAPI_GetAccInfo(char *user_id, SPApiAccInfo *acc_info)
```

参数:

user_id: 登入时用户帐号
acc_info, 账户信息.
账户信息结构:

```
typedef  struct
{
double NAV;                    //资产净值
double BuyingPower;            //购买力
double CashBal;                //现金结余
double MarginCall;             //追收金额
double CommodityPL;            //商品盈亏
double LockupAmt;              //冻结金额
double CreditLimit;            //信贷限额
double MaxMargin;              //最高保证金
double MaxLoanLimit;           //最高借贷上限
double TradingLimit;           //信用交易额
double RawMargin;              //原始保证金
double IMargin;                //基本保证金
double MMargin;                //维持保证金
```

```
    double TodayTrans;          //交易金额
    double LoanLimit;           //证券可按总值
    double TotalFee;            //费用总额
    double LoanToMR             //借贷/可按值%
    double LoanToMV             //借贷/市值%
    STR16 AccName;              //名称
    STR4 BaseCcy;               //基本币种
    STR16 MarginClass;          //保证金类别
    STR16 TradeClass;           //交易额别
    STR16 ClientId;             //客户
    STR16 AEId;                 // 经纪
    char AccType;               //户口类别
    char CtrlLevel;             //控制级数
    char Active;                // 生效
    char MarginPeriod;          //时段
} SPApiAccInfo;
```

注:GetAccInfo时如果想得到正确的参数!例如盈亏,需要先订阅持仓产品的合约得到报价才能得到正确数据!

返回值:

0, 表示请求成功.

1.50 SPAPI_GetDllVersion 方法

查询DLL版本信息.

函数原型:

```
int  SPAPI_GetDllVersion(char *dll_ver_no, char *dll_rel_no, char *dll_suffix)
```

参数:

dll_ver_no, DLL的版本信息.

dll_rel_no, 发布版本号

dll_suffix, 更新时间.

返回值:

0, 表示成功.

1.51 SPAPI_LoadProductInfoListByMarket 方法

根据交易所加载该交易所下的合约信息.

函数原型:

```
int  SPAPI_LoadProductInfoListByMarket(char *market_code)
```

参数:

market_code, 交易所代码.(多个加载:HFIK,CME.....)

返回值:

小于 0:表示请求失败.

0:表示请求本地数据成功

大于 0:表示请求对应的 RequestId,请求返回值等余回调返回值.

1.52 SPAPI_LoadProductInfoListByCode 方法

根据产品系列代码加载该系列下的合约信息.

函数原型:

```
int  SPAPI_LoadProductInfoListByCode(char *inst_code)
```

参数:

inst_code, 产品系列代码.

返回值:

小于 0:表示请求失败.

0:表示请求本地数据成功

大于 0:表示请求对应的 RequestId,请求返回值等余回调返回值.

注:产品以数万条计,建议只请求需要的产品,避免浪费电脑资源,不建议使用循环方法调用此方法,inst_code 可以多产品请求,例如:inst_code 等于(HSI,HHI.....)再一次请求。

1.53 SPAPI_SetApiLogPath 方法

设置日志的存放位置.

函数原型:

```
int  SPAPI_SetApiLogPath(char *path)
```

参数:

path, 路径.

返回值:

0, 表示成功.

1.54 SPAPI_GetCcyRateByCcy 方法

获取兑换率。

函数原型:

```
int  SPAPI_GetCcyRateByCcy(char *user_id, char *ccy, double &rate)
```

参数:

user_id: 登入时用户帐号

ccy,想要获取兑换率的货币名

rate,得到指定货币的兑换率

返回值:

0, 表示成功.

-1, 表示失败.

-2，没用 Access Login

1.55 SPAPI_AccountLogin 方法

该方法只针对 AE,当 AE 登录后可选择性登录账户

函数原型:

```
int  SPAPI_AccountLogin(char *user_id, char *acc_no)
```

参数:

user_id: 登入时用户帐号(即AE登入时账号)

acc_no, 指定客户账户名.

返回值:

0, 表示成功.

-1, 表示失败. 不是 AE 调用此方法返回-1.

1.56 SPAPI_AccountLogout 方法

该方法只针对 AE,当 AE 选择账户后可用它来释放账户.

函数原型:

```
int  SPAPI_AccountLogout(char *user_id, char *acc_no)
```

参数:

user_id: 登入时用户帐号(即AE登入时账号)

acc_no, **指定客户账户名.**

返回值:

0, 表示成功.

-1, 表示失败. 不是 AE 调用此方法返回-1.

1.57 SPAPI_SendAccControl 方法

户口控制，该方法只针对 AE。

函数原型:

```
int SPAPI_SetAccControl
                (chart *user_id, char *acc_no , char ctrl_mask,char
```

ctrl_level)

参数:

user_id: 登入时用户帐号

acc_no: AE登入后AccountLogin客户账号(Client用户user_id等于acc_no)

ctrl_mask,

```
#define CTRLMASK_CTRL_LEVEL     1  //户口控制:级别
#define CTRLMASK_KICKOUT        2  //户口控制:踢走
```

当ctrl_mask为1设置户口级数，为2时踢出用户登录状态。

ctrl_level,

0:"级别0 - 正常客户使用",
1:"级别1 - 暂停客户交易",
2:"级别2 - 暂停客户登入及交易",
3:"级别3 - 冻结户口",
4:"级别4 - 只限客户交易",
当ctrl_mask为1，ctrl_level输入0-4的级别,
当ctrl_mask为2，ctrl_level输入0就可以了。

返回值:

0, 表示请求成功.

3 接口回调方法

1.58

1.1 SPAPI_RegisterLoginReply 方法

注册一个登录回调方法.

函数原型:

```
void SPAPI_RegisterLoginReply(LoginReplyAddr  addr)
```

参数:

LoginReplyAddr addr,

LoginReplyAddr 参数原型:

```
void (SPDLLCALL *LoginReplyAddr)(char* user_id, long ret_code,
char *ret_msg);
```

LoginReplyAddr 参数:

user_id: 用户登入户口

ret_code:返回一个长整型编号.0:表示登录成功.如果登录失败也会有相应的错误编号.请查看错误信息列 表.

ret_msg:返回的登录信息.如果登录成功返回的是一个空字符串.如果登录失败会返回相应的错误提示.

1.2 SPAPI_RegisterPswChangeReply 方法

注册一个修改密码的回调方法.

函数原型:

```
void SPAPI_RegisterPswChangeReply(PswChangeReplyAddr addr)
```

参数:

PswChangeReplyAddr addr

PswChangeReplyAddr 参数原型:

```
void (SPDLLCALL *PswChangeReplyAddr)
                        (long ret_code, char *ret_msg);
```

PswChangeReplyAddr 参数:

ret_code:返回一个长整型编号.0:表示密码修改成功.如果修改失败也会有相应的错误编号.请查看错误信息列表.

ret_msg:返回的密码修改信息.如果密码成功返回的是一个

空字符串.如果修改失败会返回相应的错误提示.

1.3 SPAPI_RegisterOrderRequestFailed 方法

注册一个订单请求失败回调方法.

函数原型:

```
void SPAPI_RegisterOrderRequestFailed (
                            ApiOrderRequestFailedAdd addr)
```

参数:

ApiOrderRequestFailedAdd addr

ApiOrderRequestFailedAdd 参数原型:

```
void (SPDLLCALL *ApiOrderRequestFailedAdd)(tinyint action,
SPApiOrder *order, long err_code, char *err_msg);
```

ApiOrderRequestFailedAdd 参数:

action: 订单操作编号

*order: 订单信息

err_code: 错误编码

*err_msg: 错误信息

1.4 SPAPI_RegisterOrderBeforeSendReport 方法

注册一个订单请求后服务器接收前的发送中回调方法.请求仍在客户端.

函数原型:

```
void SPAPI_RegisterOrderBeforeSendReport(
                                    ApiOrderBeforeSendReportAddr addr);
```

参数:

ApiOrderBeforeSendReportAddr addr

ApiOrderBeforeSendReportAddr **参数原型:**

```
void (SPDLLCALL *ApiOrderBeforeSendReportAddr)(SPApiOrder *order);
```

ApiOrderRequestFailedAdd 参数:

*order: 订单信息.

1.5 SPAPI_RegisterMMOrderRequestFailed 方法

注册一个造市商订单请求失败回调方法.

函数原型:

```
void SPAPI_RegisterMMOrderRequestFailed (
                            ApiMMOrderRequestFailedAdd addr)
```

参数:

ApiMMOrderRequestFailedAdd addr

ApiMMOrderRequestFailedAdd 参数原型:

```
void (SPDLLCALL *ApiMMOrderRequestFailedAdd)(SPApiMMOrder *mm_order,
long err_code, char *err_msg);
```

ApiMMOrderRequestFailedAdd 参数:

***mm_order:** 订单信息
err_code: 错误编码
***err_msg:** 错误信息

1.6 SPAPI_RegisterMMOrderBeforeSendReport 方法

注册一个造市商订单请求后服务器接收前的发送中回调方法.请求仍在客户端

函数原型:

```
void SPAPI_RegisterMMOrderBeforeSendReport(
                                    ApiMMOrderBeforeSendReportAddr addr);
```

参数:

ApiMMOrderBeforeSendReportAddr addr

ApiMMOrderBeforeSendReportAddr 参数原型:

```
void (SPDLLCALL *ApiMMOrderBeforeSendReportAddr)(SPApiMMOrder *mm_order);
```

ApiMMOrderRequestFailedAdd 参数:

***mm_order:** 订单信息.

1.7 SPAPI_RegisterQuoteRequestReceivedReport 方法

注册一个要求报价行情信息回调方法。

函数原型:

```
void SPAPI_RegisterQuoteRequestReceivedReport(
                                        ApiQuoteRequestReceivedAddr addr);
```

参数:

ApiQuoteRequestReceivedAddr addr

ApiQuoteRequestReceivedAddr 参数原型:

```
void (SPDLLCALL *ApiQuoteRequestReceivedAddr)(char *product_code, char
buy_sell, long qty);
```

ApiQuoteRequestReceivedAddr 参数

product_code: 合约名
buy_sell:要求报价方向
0：双向报价。 B：买。 S：沽
qty:要求报价数量.

1.8 SPAPI_RegisterTradeReport 方法

注册一个成交记录更新(登入后)回调的方法.

函数原型:

```
void SPAPI_RegisterTradeReport(
```

ApiTradeReportAddr addr)

TradeUpdateAddr addr 参数原型:

void (SPDLLCALL *ApiTradeReportAddr)(long rec_no, SPApiTrade *trade);

ApiTradeReportAddr参数:

rec_no, 成交记录在服务器中的记录编号.

*trade, 已成交的订单信息.

1.9 SPAPI _RegisterLoadTradeReadyPush 方法

注册一个普通客户(Client Mode)登入后,成交信息(登入前的已存的成交)回调方法.

函数原型:

void SPAPI_RegisterLoadTradeReadyPush(ApiLoadTradeReadyPushAddr addr)

ApiLoadTradeReadyPushAddr addr 参数原型

void (SPDLLCALL *ApiLoadTradeReadyPushAddr)(long rec_no, SPApiTrade *trade);

ApiLoadTradeReadyPushAddr 参数:

rec_no, 成交记录在服务器中的记录编号.

*trade, 已成交的订单信息.

1.10 SPAPI_RegisterApiPriceUpdate 方法

注册行情更新回调的方法.

函数原型:

void SPAPI_RegisterApiPriceUpdate(
ApiPriceUpdateAddr addr)

ApiPriceUpdateAddr addr 参数原型:

void (SPDLLCALL *ApiPriceUpdateAddr)(
SPApiPrice *price)

ApiPriceUpdateAddr参数:

price, 合约更新的信息.

1.11 SPAPI_RegisterTickerUpdate 方法

注册一个市场成交记录的回调方法.

函数原型:

void SPAPI_RegisterTickerUpdate(
ApiTickerUpdateAddr addr)

ApiTickerUpdateAddr addr 参数原型:

void (SPDLLCALL *ApiTickerUpdateAddr)(
SPApiTicker *ticker);

ApiTickerUpdateAddr 参数:

ticker, 新的市场成交记录信息.

SPApiTicker结构:

```
typedef  struct
{
        double Price;           //价格
        int32_t Qty;              //成交量
        uint32_t TickerTime;       //时间
        int32_t DealSrc;          //来源
        STR16 ProdCode;         //合约代码
        char DecInPrice;        //小数位
} SPApiTicker;
```

注：long DealSrc;交易来源 - 说明是什么类型的交易，暫時只有香港期貨交易所提供這方面的額外資訊。请参考“第四部分”“1.9来源指令.

1.12 SPAPI_RegisterOrderReport 方法

注册一个订单报告回调方法.

函数原型:

```
void SPAPI_RegisterOrderReport)(
                                ApiOrderReportAddr addr);
```

ApiOrderReportAddr addr 参数原型:

```
void (SPDLLCALL *ApiOrderReportAddr)(long rec_no, SPApiOrder *order);
```

ApiOrderReportAddr 参数:

rec_no, 订单在服务器中的记录编号

*order, 订单信息

1.13 SPAPI_RegisterInstrumentListReply 方法

注册一个产品系列信息回调方法.

函数原型:

```
void SPAPI_RegisterInstrumentListReply(
                        InstrumentListReplyAddr  addr);
```

InstrumentListReplyAddr addr 参数原型:

```
void (SPDLLCALL *InstrumentListReplyAddr)(long req_id, bool is_ready
                                          char *ret_msg);
```

InstrumentListReplyAddr 参数:

req_id, 请求时返回值对应此回调Req_id。

is_ready, 是否加载成功:

true:产品系列信息加载成功

false:产品系列信息加载没有完成

ret_msg, 返回的提示信息.

1.14 SPAPI_RegisterBusinessDateReply 方法

登录成功后返回一个交易日期。

函数原型:

SPAPI_RegisterBusinessDateReply(BusinessDateReplyAddr addr)

BusinessDateReplyAddr addr 参数原型:

void (SPDLLCALL *BusinessDateReplyAddr)(long business_date);

BusinessDateReplyAddr 参数:

business_date:返回一个 Unix 时间戳

1.15 SPAPI_RegisterConnectingReply 方法

注册一个连接状态的回调方法.

函数原型:

void SPAPI_RegisterConnectingReply (ConnectedReplyAddr addr);

ConnectedReplyAddr addr 参数原型:

void (SPDLLCALL *ConnectedReplyAddr)(long host_type, long con_status);

ConnectedReplyAddr 参数:

host_type：返回发生改变的行情状态服务器的 ID.

80,81： 表示交易连接

83：表示一般价格连接.

88：表示一般资讯连接.

con_status：

1：连接中，2：已连接，3：连接错误， 4：连接失败

1.16 SPAPI_RegisterAccountLoginReply 方法

注册一个普通客户(Client Mode)登入的回调方法.

函数原型:

void SPAPI_RegisterAccountLoginReply(AccountLoginReplyAddr addr);

AccountLoginReplyAddr addr 参数原型:

void (SPDLLCALL *AccountLoginReplyAddr)(char *accNo, long ret_code, char* ret_msg);

AccountLoginReplyAddr 参数:

accNo： 客户账号

ret_code:返回一个长整型编号.0:表示登录成功.如果登录 失败也会有相应的错误编号.请查看错误信息列表.

ret_msg:返回的登录信息.如果登录成功返回的是一个空字符串.如果登录失败会返回相应的错误提示.

注：此方法如果是 AE Mode 登入需要 AccountLogin 客户才能回调，如果客户直接登入(client Mode)会直接回调

1.17 SPAPI_RegisterAccountLogoutReply 方法

注册一个普通客户(Client Mode)登出的回调方法.

函数原型:

```
void SPAPI_RegisterAccountLogoutReply)(AccountLogoutReplyAddr addr);
```

AccountLogoutReplyAddr addr 参数原型:

```
void (SPDLLCALL *AccountLogoutReplyAddr)(char* accNo, long ret_code, char* ret_msg);
```

AccountLogoutReplyAddr 参数:

accNo: 客户账号

ret_code:返回一个长整型编号.0:表示登录成功.如果登录 失败也会有相应的错误编号.请查看错误信息列表.

ret_msg:返回的登录信息.如果登录成功返回的是一个空字符串.如果登录失败会返回相应的错误提示.

注：此方法如果是 AE Mode 登入需要 AccountLogin 客户才能回调，如果客户直接登入(client Mode)会直接回调

1.18 SPAPI_RegisterAccountInfoPush 方法

注册一个普通客户(Client Mode)登入后户口信息回调方法.

函数原型:

```
void SPAPI_RegisterAccountInfoPush(AccountInfoPushAddr addr);
```

AccountInfoPushAddr addr 参数原型:

```
void (SPDLLCALL *AccountInfoPushAddr)(SPApiAccInfo *acc_info);
```

AccountInfoPushAddr 参数:

***acc_info:** 客户信息

注：此方法如果是 AE Mode 登入需要 AccountLogin 客户才能回调，如果客户直接登入(client Mode)会直接回调

1.19 SPAPI_RegisterAccountPositionPush 方法

注册一个普通客户(Client Mode)登入后持仓信息(登入前的已存在持仓)回调方法.

函数原型:

```
void SPAPI_RegisterAccountPositionPush(AccountPositionPushAddr addr);
```

AccountPositionPushAddr addr 参数原型:

```
void (SPDLLCALL *AccountPositionPushAddr)(SPApiPos *pos);
```

AccountPositionPushAddr 参数:

***pos:** 持仓信息

注：此方法如果是 AE Mode 登入需要 AccountLogin 客户才能回调，如果客户直接登入(client Mode)会直接回调

1.20 SPAPI_RegisterUpdatedAccountPositionPush 方法

注册一个普通客户(Client Mode)登入后持仓信息(登入后新的持仓信息)回调方法.

函数原型:

```
void SPAPI_RegisterUpdatedAccountPositionPush(UpdatedAccountPositionPushAddr addr);
```

UpdatedAccountPositionPushAddr addr 参数原型:

```
void (SPDLLCALL *UpdatedAccountPositionPushAddr)(SPApiPos *pos);
```

UpdatedAccountPositionPushAddr 参数:

***pos：**持仓信息

注：此方法如果是 AE Mode 登入需要 AccountLogin 客户才能回调，如果客户直接登入(client Mode)会直接回调

1.21 SPAPI_RegisterUpdatedAccountBalancePush 方法

注册一个户口账户发生变更时回调方法

函数原型:

```
void SPAPI_RegisterUpdatedAccountBalancePush(UpdatedAccountBalancePushAddr addr);
```

UpdatedAccountBalancePushAddr addr 参数原型:

```
void (SPDLLCALL *UpdatedAccountBalancePushAddr)(SPApiAccBal *acc_bal);
```

UpdatedAccountBalancePushAddr 参数:

***acc_bal:** 账户信息

1.22 SPAPI_RegisterProductListByCodeReply 方法

注册一个根据产品系列名返回合约信息的回调方法.

函数原型:

```
void SPAPI_RegisterProductListByCodeReply(
                                ProductListByCodeReplyAddr  addr);
```

ProductListByCodeReplyAddr addr 参数原型:

```
void (SPDLLCALL *ProductListByCodeReplyAddr)(long req_id, char
*inst_code, bool is_ready, char *ret_msg);
```

ProductListByCodeReplyAddr 参数:

req_id, 请求时返回值对应此回调Req_id。

inst_code, 产品系列代码

is_ready, 是否加载成功:

true:产品合约信息加载成功

false:产品合约信息加载没有完成

ret_msg, 返回的提示信息.

注：当 is_ready=true 时 inst_code=空，说明请求的系列没有 Product 信息.

1.23 SPAPI_RegisterAccountControlReply 方法

注册一个户口控制回调方法
函数原型:

```
void SPAPI_RegisterAccountControlReply(AccountControlReplyAddr addr);
```

AccountControlReplyAddr addr 参数原型:

```
void (SPDLLCALL *AccountControlReplyAddr)
                                        (long ret_code, char *ret_msg);
```

AccountControlReplyAddr 参数:

ret_code：0 成功, 失败 code。
ret_msg: 成功为空, 失败原因

4.字符对照表

1.1 关于请求函数 Return 返回值说明

return	
0	请求成功
-1	登入时 UserID 不一致或此 UserID 未登入
-2	AE Mode 时未 AccountLogin Client,取得 AccInfo 信息为空
-3	获取数据不存在或空
-5	DLL 异常

1.2 买卖动作

BuySell	
买入	‘B’
卖出	‘S’

1.3 设定 T+1 夜市

SPApiOrder > OrderOptions	
非 T+1	0
T+1	1

此设定针对期货

如果该合约支持收市后交易,此合约收市后下单将它设置为：OrderOptions = 1.

1.4 止损/限价触发类型

SPApiOrder > StopType.	
非止损/限价触发	0
限价止损	'L'
升市触发	'U'
跌市触发	'D'

1.5 竞价指定价格

SPApiOrder > Price	
竞价标准价格	((int32_t)0x7fffffff)

1.6 指令类型

SPApiOrder > OrderType	
限价指令类型	0
竞价指令类型	2
市场价指令类型	6

主要应用于 SPApiOrder 结构体中的<char OrderType>.
当 OrderType = 0 时, 指令为限价指令, 可再设置为多种条件指令类型.
当 OrderType = 2 时, 指令为竞价指令, 此时价格设置可参照:<第四部分-1.3>.
当 OrderType = 6 时, 指令为市价指令, 此时价格可设置为 0, 实际价格将按市价设定.

1.7 指令条件类型

SPApiOrder > CondType	
普通指令	0
止损指令	1
指定时间发送指令	3
双向限价指令	4
追踪止损指令	6
其他参数只作内部用途	

1.8 指令有效期

SPApiOrder > ValidType	
当天有效	0
成交并取消	1
成交或取消	2
直到有效期	3
直到自定时间	4

1.9 发送指令动作

action	
新增指令	1
更改指令	2
删除指令	3

1.10 指令状态

SPApiOrder > Status	
发送中	0
工作中	1
无效	2
待定	3
新增中	4
更改中	5
删除中	6
无效中	7
部分成交且工作中	8
已成交	9
已删除	10
等待批准	18
成交已覆盘	20
删除已覆盘	21
同步异常中	24

部分成交已删除	28
部分成交并删除已覆盘	29
交易所無效	30

1.11 时段

SPApiAccInfo > MarginPeriod	
错误 ???	-2
混合	-1
NULL	0
隔夜	1
即日	2
段落暂停	3

1.12 户口控制级数类型

SPApiAccInfo > CtrlLevel	
正常	0
客户交易已暂停	1
客户登入与交易已暂停	2
冻结户口	3

1.13 Ticker 来源指令

SPApiTicker > DealSrc	
auto matching	1
matched outside exchange, different participants	3
crossing	5
standard combo	7
auction	20
tailor made combination	36
combo match with out-right	43

来源资讯由香港交易所提供及更新，如有任何疑问，请向交易所查询.

1.14 TradeStateNo 产品市场状态

SPApiPrice > TradeStateNo		
0	UNDEF	--
1	pre	开市前
2	poa	对盘前
3	opa	对盘中
4	pause	暂停
5	open	开市
6	close	收市
7	premkt	开市前活动
8	stop	停止
已过期	SPApiPrice.ExpiryYMD < 当天. 说明产品已经过期	
已停牌	SPApiPrice.Suspend= True. 说明产品已经停牌	

1.15 ExStateNo 港期市场状态

SPApiPrice > ExstateNo 此状态只针对港期，是港期的附加状态		
0	UNDEF	--
1	opa	对盘中
2	clsoe	收市
3	open	开市
4	pre	开市前
5	poa	对盘前
6	pause	暂停
7	premkt	开市前活动
8	CL start	CL 起始
9	CL close	CL 结束
10	Aht Close	夜市收市
11	Aht clrinfo	夜市更新资讯
12	Aht inact T	夜市 T 盘无效
13	Aht nextday	夜市下一交易日
14	Aht open	夜市开市
15	Aht open PL	夜市开市(PL)
16	Aht premkt	夜市开始前活动
17	Open PL	开市(PL)
18	Clsoe today	午市收市
19	Open DPL	开市(DPL)
20	failover	故障转换
21	Clsoe today E	午市收市(E)
22	Aht close E	夜市收市(E)

1.16 期权类型

SPApiProduct > OptStyle	
其它	空

欧式	‘E’
美式	‘A’

1.17 期权方向

SPApiProduct > CallPut	
空	‘-’
认购	‘C’
认沽	‘P’

1.18 ProdType 产品类型

SPApiProduct > ProdType	
0	相关产品
1	期货
2	期权(基于期货)
3	差价
4	股票
5	期权(期于现货或股票)
6	认股证
7	一篮子认股证
8	债券
9	信托
10	现货
11,12,13	保留字段
其它	“--”

1.19 开仓平仓指令

Open&Close	UpLevel	说明
Combo Open (Stop)	**1**	止损(点)
Combo Open (Stop)	**11**	止损(价)
Combo Open (Trail)	**6**	止赚(点)
Combo Open (Trail)	**16**	止赚(价)
Combo Open (OCO)	**4**	双向限价(点)
Combo Open (OCO)	**14**	双向限价(价)

说明:

(点) 以最小变动位数 TickSize 为单位的递增与递减.

(价) 以当前价格进行递增或递减,最小变动位数为 TickSize.

例如: ESZ5 最新价:2022.25 TickSize:0.25

(点) 0.25 , 0.5, 0.75 以 TickSize 为单位的递增与递减.

(价) 2022.25 , 2022.5, 2022.75 以价格进行递增或递减,最小变动位数为 TickSize.

可参考 下一 **Add Order 参数表（1.13）**

1.20 Add Order 参考表

-Add order-		
Field Name		**Specification**
AccNo		Account Number
Initiator		User Id
ProdCode		Product Code
BuySell		Buy='B',Sell='S'（参考第四部分：1.1）
Qty		Qty
Ref		Reference
ClOrderId		Client Order Id
OrderOptions		T+1(1=set T+1)（参考第四部分：1.2）
OpenClose		Open and Close Type('\0'=default,'O'=Open)
CondType		Order Cond Type(0=Normal,1=Stoplosses,3=Sched,4=OCO,6=TrailStop, 8=ComboOpen,9=ComboClose,11=StopPrice,14=OCOStopPrice, 16=TrailStopPrice)
Normal order (CondType=0)	CondType	CondType=0(Normal)
	OrderType	OrderType(0=Limit,2=AO,6=Master Price)
	pirce	OrderType=0: Input Price OrderType=2: 2147483647(AO) OrderType=6: Master Price

<table>
<tr><td rowspan="5"></td><td>ValidType
(OrderType=0
)</td><td rowspan="5">Valid Type(0=Today,1=Fak,2=Fok,3=GTC,4=GTD)
Stop Type('L'=Loss,'U'=Stop Up,'D'=Stop Down)

Valid Type=(0,3,4)=Stop/TriggerLmt(Stop Limit,UpTriggerLmt,DnTriggerLmt,Stop Market)
CondType=1(Stop/TriggerLmt)
<u>Stop Limit</u>
StopType='L'
CondType=1

<u>UpTriggerLmt</u>
StopType='U'
CondType=1

<u>DnTriggerLmt</u>
StopType='D'
CondType=1

<u>Stop Market(</u>需经纪行给相应权限<u>)</u>
StopType='L'
Price=0
OrderType=6
CondType=1

StopLevel = Input(for ValidType=0,3,4)
StopLevel = 0(for ValidType=1,2)
ValidTime=Specific Time(unix time) for ValidType=4</td></tr>
<tr><td>StopType</td></tr>
<tr><td>StopLevel</td></tr>
<tr><td>ValidTime</td></tr>
<tr><td></td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td rowspan="9">Trailing Stop
(CondType=6)</td><td>CondType</td><td>CondType=6(Trailing Stop)</td></tr>
<tr><td>ValidType</td><td>ValidType = 0(ToDay)</td></tr>
<tr><td>StopType</td><td>StopType='L'(Loss)</td></tr>
<tr><td>BuySell</td><td rowspan="8"><u>Buy Trailing Stop</u>
BuySell = 'B'
StopLevel = stop loss price
UpLevel = Market Price
UpPrice = original trigger price
Price = StopLevel+toler
DownLevel = Trailing stop, step
accumulated step = UpPrice - StopLevel</td></tr>
<tr><td>StopLevel</td></tr>
<tr><td>Price</td></tr>
<tr><td>UpLevel</td></tr>
<tr><td>DpPrice</td></tr>
<tr><td>DownLevel</td></tr>
<tr><td>DownPrice</td></tr>
</table>

		Sell Trailing Stop BuySell = 'S' StopLevel = stop loss price DownLevel = Market Price DownPrice = original trigger price Price = StopLevel - toler UpLevel = Trailing stop, step accumulated step = StopLevel - DownPrice
OCO (CondType=4)	CondType	CondType=4(OCO)
	ValidType	ValidType = 0(ToDay)
	Price	Price = input price
	BuySell UpLevel UpPrice DownLevel DownPrice	***Buy OCO*** BuySell = 'B' UpLevel = stop limit UpPrice = stop limit + toler ***Sell OCO*** BuySell = 'S' DownLevel = stop limit DownPrice = soop limit - toler
Bull & Bear (CondType=0)	CondType	CondType = 0(Normal)
	ValidType	ValidType = 0(ToDay)
	Price	Price = input price
	BuySell UpLevel UpPrice DownLevel DownPrice	**Bull** BuySell = 'B' UpLevel = Price + profit UpPrice = Price + profit DownLevel = Price – loss DownPrice = price – loss – loss_tol **Bear** BuySell = 'S' DownLevel = Price - profit DownPrice = Price - profit UpLevel = Price + loss UpPrice = price + loss + loss_tol

Time to send (CondType=0)	CondType	CondType = 3(Sched)
	ValidType	ValidType = 0(ToDay)
	Price	Price = input price
	SchedTime	SchedTime=Specific Time(unix time)
Open & Close (CondType=0)	CondType	CondType=8(open)
	OrderType	OrderType=0(Limit)
	OpenClose	OpenClose='\0'(default)
	ValidType	ValidType=0(ToDay)
	StopType StopLevel	**Trigger(UpTriggerLmt And DnTriggerLmt)** **UpTriggerLmt** StopType = 'U'(stop up) StopLevel = input stop level **DnTriggerLmt** StopType = 'D'(stop Down) StopLevel = input stop level **Don't Trigger** StopType = 0 StopLevel = 0
	Price	**Price(inputPrice And MarketPrice)** **Input Price** Price = input price **Market Pirce**(for Don't Trigger And Close Don't OCO(pirces) And Stop(prices)) **Buy** Price = MarketPrice + open_toler **Sell** Price = MarketPrice - open_toler

	UpLevel UpPrice DnLevel DnPrice	Close [OCO(points),OCO(prices),Stop(points),Stop(prices)] **OCO(points)** UpLevel = 4(ref:ContType) UpPrice = profit DnLevel = loss DnPrice = loss toler **OCO(prices)** UpLevel = 14(ref:ContType) UpPrice = profit DnLevel = loss DnPrice = loss toler **Stop(points)** UpLevel = Step[True:6,False:1] UpPrice = Step[True:Step,False:0] DnLevel = loss DnPrice = loss toler **Stop(prices)** UpLevel =Step[True:16,False:11] UpPrice = Step[True:Step,False:0] DnLevel = loss DnPrice = loss toler [UpLevel=1,4,6,11,14,16. ref:CondType]
	SchedTime	SchedTime=Specific Time(unix time)

1.21 错误指令代码表

指令	中文	English
-35		Blocking
-48		Address In Use
-49		Address Not Available
-50		Network Is Down
-51		Network Is Unreachable
-52		Connection Reset
-53		Connection Abort
-54		Remote Disconnected
-55		Too Many Connection

-57		Socket Is Not Connected
-60		Connection Timeout
-61		Connection Refused
-64		Host Is Down
-65		Host Is Unreachable
-5001		SSL System Error
-5002		SSL Blocking (Read)
-5003		SSL Blocking (Write)
-5004		SSL Blocking (X509 Lookup)
-5005		SSL General Error
-5006		SSL Disconnected
-5007		SSL Blocking (Connect)
-10260001		Invalid Socket
-10260002	无效的登入端口	Invalid Entry Port
-10260003	没有权限	No User Right
-10260004	密码错误	Wrong Password
-10260005		User is challenging
-11050005	已登入	User already login
-11050002	登入已满	User login full
-11150005	已登入	User already login
-11650002	登入已满	User login full
-11230001	用户或密码错误	No Such User or Wrong Password
-11460001	没有权限:在服务器地址上	No User Right: On Server Address
-11460002	没有权限:在服务器端口上	No User Right: On Server Port
-11460003	没有权限:在客户端地址上	No User Right: On Client Address
-11460004	没有权限:在客户端端口上	No User Right: On Client Port
-11460005	没有权限:在登入端口上	No User Right: On Entry Point
-11460006	密码已过期,请更新	Password Expired
-11460007	多次密码错误,请与经纪联络	Too Many Password Err., Call Broker
-11460008	用户或密码错误	No Such User or Wrong Password
-11460009	转移登入	Redirect Login
-11460010	只能在公司内部的网路登入	Login from in-house network only

-11460011	未知的许可证密钥	Unknown license key
-11460012	未知的应用编号	Unknown application id
-11460013	在其他装置上已登入	User already login in another device
-11860001	版本太旧,请升级	Version Too Old, Please Upgrade First
-13330001	客户不存在	No Such Client
-13530000	户口未启动	Account Is Inactive
-13530001	户口已冻结	Account is frozen
-13530002	此户口之客户已被暂停	The client of this account is suspended
-15260003	没有权限作交易指示	No User Right To Access Order
-15260004	用户已停用	User Suspended
-15260005	密码错误	Wrong Password
-15260006	系统已停止	System Closed
-15260007	系统已暂停	System Suspended
-15260008	买卖繁忙	Order Request Busy
-15260009	此产品暂时未能提供电子交易，请联络阁下经纪	Product currently not available for e-trading, please contact your AE
-15260010	只限客户作交易指示	Client Trade Only
-15260011	用户在公司以外的网路登入，只能查看户口	User can view only when outside of In-house network
-15260012	没有权限作批核指示	No User Right To Approve Order
-15260013	没有权限作手动交易	No User Right For Manual Dealing
-15260014	没有权限作拒绝指示	No User Right To Reject Order
-15260015	没有权限作放弃指示	No User Right To Abort Order
-15260016	没有权限作成交入账	No User Right For Trade Booking
-15260017	不准许更改为相同密码	Change to same password not allowed
-15260018	新密码不能与最近的旧密码相同	Password cannot same as old one
-15260019	系统价格连结中断，请联络阁下经纪	System price feed not linked, please contact your AE
-15270001	价格不正确	Invalid Price
-15270002	数量不正确	Invalid Quantity
-15270003	买卖不正确	Invalid Buy/Sell
-15270004	产品不正确	Invalid Product
-15270005	有效期不正确	Invalid Validity

-15270006	日期时间不正确	Invalid Specific Time
-15270007	指示种类不正确	Invalid Order Type
-15270008	止损价不正确	Invalid Stop Price
-15270009	指示不提供	Order Not Supported
-15270010	指示已满	Order Is Full
-15270011	价格不符	Price Not Matched
-15270012	价格过高	Price Over Limit
-15270013	价格过低	Price Under Limit
-15270014	持仓已满	Position Is Full
-15270015	此指示不能无效	Inactive Order Not Allowed For This Type
-15270016	指示已经有效	Order Is Already Active
-15270017	指示已经无效	Order Is Already Inactive
-15270018	小数点不符	Decimal Point Not Matched
-15270019	户口不能买卖此产品	Account cannot trade this product
-15270020	户口不正确	Invalid Account
-15270021	读取户口错误	Get Account Error
-15270022	户口类型不正确	Invalid Acc Code Type
-15270023	获取新的请求编号错误	Get New Request No. Error
-15270024	获取新的订单编号错误	Get New Order No. Error
-15270025	获取用户市场信息错误	Get Acc Market Error
-15270026	获取订单错误	Get Order Error
-15270027	更新指示错误	Update Order Error
-15270028	此时段不允许这动作	Action Not Allowed In This Status
-15270029	此指示不允许更改	Change Not Allowed For This Order
-15270030	这动作未能提供	Action Not Supported
-15270031	当条件被触发时,指示价格偏离太远	Order Outside Price Limit When Condition Triggered
-15270032	产品已停止买卖	Product Is Stopped Trading
-15270033	指示未能执行,请等待回覆	Order not executed, please wait for reply
-15270034	指示已成交，警告:持仓可能不正确，请即联络经纪	Order Already Traded, Warning: check position with broker
-15270035	产品不允许沽出	Product Not Allowed To Sell

-15270036	产品不允许买入	Product Not Allowed To Buy
-15270037	产品只允许平仓	Product Must Be Closed Only
-15270038	预定时间已过了，指示不接受	Schedule Time Is Under Current Time, Order Not Accepted
-15270039	止损水平会被即时触发，指示不接受	Stop Level Will Be Triggered Immediately, Order Not Accepted
-15270040	OCO 止损水平可能先被触发，指示不接受	OCO Stop Level Will Be Triggered First, Order Not Accepted
-15270041	超出批核金额上限	Over Approval Limit
-15270042	已过了最后交易日	Over Last Trading Date
-15270043	超出单项指示上限	Over Single Order Limit
-15270044		Get Position Error
-15270045	交易指示批核请求进行中,请等候或先删除现有之请求	Order approval requesting, please wait or delete the existing request first
-15270046	不允许期权交易	Option Trading Not Allowed
-15270047	不允许沽空期权	Option Short Sell Not Allowed
-15270048	不允许沽空证券	Securities Short Sell Not Allowed
-15270049	预定时间超出本交易日，指示不接受	Schedule Time over current trade day, order not accepted
-15270050	超出允许的最大数量	Over the allowed quantity
-15270051	成交并/或取消未能提供	Fill and/or Kill Not Supported
-15270052	指示不允许双边开仓	Order not allowed to open on both sides
-15270053	指示不允许超出平仓数量	Order not allowed to over close
-15270054	指示不允许增加数量	Add Quantity Not Allowed
-15270055	超出持仓上限	Over Position Limit
-15270056	超出用户每日指示上限	Over user daily order limit
-15270057	不允许股票以外之证券交易	Non-Stock Product Not Allowed
-15270058	差价不正确	Invalid Tick Size
-15270099	请求批核	Approval Request
-15280000	保证金不足	Insufficient Margin
-15280001	信用查核错误	Credit Check Error
-15288001	此产品未提供保证金	Margin is not set for this product
-15289000	购买力不足	Insufficient Buying Power

-15289001	不准沽空	Short Sell Not Allowed
-15289002	购买力不足	Insufficient Buying Power
-15289003	购买力不足	Insufficient Buying Power
-15289004	超出最高保证金	Over Maximum Margin
-15230000	指示正在处理中	Order Requesting
-15999996	过期指示已被删除	The Expired Order Is Deleted
-15999997	系统增加指示	Order Added By Gateway
-15999998	系统更改指示	Order Changed By Gateway
-15999999	系统删除指示	Order Deleted By Gateway
-90000012	GW 错误:指示错误，请联络阁下经纪	GW: Transaction Failed, please contact your AE
-90002008	GW 错误:未连线，请联络阁下经纪	GW: Not Connected, please contact your AE
-90002012	GW 错误:连线错误，请联络阁下经纪	GW: Connection Error, please contact your AE
-90002014	GW 错误:段落错误，请联络阁下经纪	GW: Session Aborted, please contact your AE
-90009010	GW 错误:登入错误，请联络阁下经纪	GW: Login Error, please contact your AE
-90009011	GW 错误:产品不正确	GW: Invalid Product
-90009012	GW 错误:有效期不正确	GW: Invalid Validity
-90009013	GW 错误:买卖不正确	GW: Invalid Buy/Sell
-90009014	GW 错误:指示种类不正确	GW: Invalid Order Type
-90009015	GW 错误:价格不正确	GW: Invalid Price
-90009016	GW 错误:数量不正确	GW: Invalid Quantity
-90009021	GW 错误:[成交并取消]没有成交	GW: No Matched Order For FaK
-90009022	GW 错误:[成交或取消]没有成交	GW: No Matched Order For FoK
-90009023	GW 错误:指示没有放出市场	GW: No Order Placed
-90009024	GW 错误:未开市	GW: Market Not Open
-90009025	GW 错误:价格不准许	GW: Price Not Allowed
-90009026	GW 错误:不准许竞价盘	GW: Auction Order Not Allowed
-90009027	GW 错误:更改数量错误	GW: Change Quantity Failed
-90009028	GW 错误:指示不存在	GW: Order Not Exists
-90009029	GW 错误:不能更改价格及数量	GW: Price and Quantity Cannot Be Changed
-90009030	GW 错误:价格超出范围	GW: Price Out Of Range
-90009031	GW 错误:更改错误导致指示已被删除	GW: Order Deleted Due To Unrecoverable Change Error

-90009032	GW 错误:不正确结果	GW: Incorrect Result
-90009033	GW 错误:此时段不准许限价盘	GW: Limit Order Not Allowed At This Trading State
-90009034	GW 错误:数量过高	GW: Quantity Too High
-90009035	GW 错误:此时段不准许更改价格	GW: Price Change Not Allowed At This Trading State
-90009036	GW 错误:市场繁忙	GW: Market Is Busy
-90009037	GW 错误:有效期不提供	GW: Validity Not Supported
-90009038	GW 错误:指示种类不提供	GW: Order Type Not Supported
-90009039	GW 错误:此要求不提供	GW: Request Not Supported
-90009040	GW 错误:已暂停接受指示	GW: Gateway Is Inactive
-90009041	GW 错误:不准许所选有效期	GW: Given Time Validity Not Allowed
-90009042	GW 错误:买卖差价不正确	GW: Invalid Tick Size
-90009043	GW: 指示状态不确定	GW: Order Status Uncertain
-90009044	GW: 此时段不接受非 T+1 指示	GW: Non T+1 Order Not Allowed At This Trading State
-92000201	GW 错误:指示被拒绝	GW: Order Rejected
-92000202	GW 错误:指示被取消	GW: Order Cancelled
-90009101	GW 错误:已收市	GW: Market Closed
-90009102	GW 错误:已暂停市	GW: Market Paused
-90009103	GW 错误:此时段不准许更改指示	GW: Change Order Not Allowed At This Trading State
-90009104	GW 错误:此时段不准许删除指示	GW: Delete Order Not Allowed At This Trading State
-90009105	GW 错误:此时段不准许交易	GW: Transaction Not Allowed At This Trading State
-90009106	GW 错误:此指示种类不准许更改指示	GW: Change Not Allowed For This OrderType
-16160001	新密码太短	New Password Too Short
-16160002	密码必须为字母或数字	New Password Must Be Alpha-Numeric
-17260003	没有权限:现金存取	No User Right: Cash Change
-17260004	密码错误	Wrong Password
-17260010	金额不正确	Invalid Amount
-17260011	买卖不正确	Invalid Buy Sell
-17260012	货币不正确	Invalid Currency

-17260013	现金不足	Not Enough Cash
-17260014	计算购买力错误	Get Buying Power Error
-17260015	购买力不足	Over Buying Power
-17260016	已超出现金存取的请求限额	Over Cash Request Limit
-17260017	已超出现金存取的批核限额	Over Cash Approval Limit
-17260018	现金存取请求进行中,请等候或先删除现有之请求	Cash Change Requesting, please wait or delete the existing request first
-18260003	没有权限:更改信贷限额	No User Right: On Credit Limit Change
-18260004	没有权限:更改保证金上限	No User Right: On Max Margin Change
-18260005	密码错误	Wrong Password
-18260006	没有权限:更改最高借贷上限	No User Right: On Max Loan Limit Change
-18260007	没有权限:更改信用交易额	No User Right: On Trading Limit Change
-18260010	信贷限额不正确	Invalid Credit Limit Amount
-18260011	保证金上限不正确	Invalid Max Margin Amount
-18260012	最高借贷上限不正确	Invalid Max Loan Limit Amount
-18260013	已超出准许可更改的上限	Over Allowed Changable Limit
-18260014	信用交易额不正确	Invalid Trading Limit Amount
-20260003	没有权限:户口控制	No User Right: On Account Control
-20260004	密码错误	Wrong Password
-20430004	客户已登出	Client Already Logged Out
-22150004	客户未登入	User Is Not Logged In
-23260001	不准许自我批核	Self Approval Not Allowed
-24260003	没有权限:股票/商品存取	No User Right: Stock/Commodity In/Out
-25660001	处理中有指示存在产品不正确	Invalid Product
-25660002	指示处理中导致不能平仓,请待处理完成后再试	Invalid Status
-25660003	处理中有指示删除失败	Delete Order Failed
-25660004	未能提供市价导致不能平仓,请待提供市价后再试	No Market Price To Close Position
-25660005	处理中有平仓指示失败	Close Position Failed
-25660006	平仓指示并没有进行	Close Position Not Requested

1.22 错误指令范围表

指令范围	中文	English
-1015xxxx	用户配置文件读取错误	Read User Profile Error
-1026xxxx	验证失败	Validation Failed
-1105xxxx	添加用户错误	Add User Error
-1115xxxx	检查用户配置文件错误	Check User Profile Error
-1123xxxx	获取用户配置文件错误	Get User Profile Error
-1133xxxx	获取用户权限错误	Get User Right Error
-1153xxxx	获取账户编号错误	Get Account No. Error
-1165xxxx	添加用户配置文件错误	Add User Profile Error
-1173xxxx	获取账户错误	Get Account Error
-1171xxxx	发送错误	Send Error
-1205xxxx	用户读取错误	Read User Error
-1215xxxx	读取用户配置文件错误	Read User Profile Error
-1225xxxx	删除用户配置文件错误	Delete User Profile Error
-1231xxxx	发送错误	Send Error
-1305xxxx	用户读取错误	Read User Error
-1315xxxx	读取用户配置文件错误	Read User Profile Error
-1326xxxx	账户登录验证失败	Account Login Validation Failed
-1333xxxx	获取客户端配置文件错误	Get Client Profile Error
-1343xxxx	获取客户端账户编号错误	Get Client Account No. Error
-1353xxxx		Get Client Access Error
-1365xxxx	修改用户配置文件错误	Change User Profile Error
-1373xxxx	获取账户错误	Get Account Error
-1371xxxx	发送错误	Send Error
-1383xxxx	获取账户信息错误	Get Account Information Error
-1405xxxx	用户读取错误	Read User Error
-1415xxxx	读取用户配置文件错误	Read User Profile Error
-1426xxxx	账户登出验证失败	Account Logout Validation Failed

-1435xxxx	修改用户配置文件错误	Change User Profile Error
-1441xxxx	发送错误	Send Error
-1505xxxx	用户读取错误	Read User Error
-1515xxxx	读取用户配置文件错误	Read User Profile Error
-1526xxxx	订单请求验证失败	Order Request Validation Failed
-1527xxxx	订单检查失败	Order Check Failed
-1528xxxx		Credit Check Error
-1523xxxx	订单请求	Order Requesting
-1533xxxx	获取账户错误	Get Account Error
-1543xxxx	添加请求错误	Add Request Error
-1553xxxx	获取账户日志错误	Get Account Log Error
-1561xxxx	发送错误	Send Error
-1570xxxx		Exchange Rate Error
-1583xxxx		Get Approval Data Error
-9000xxxx	OM:请求错误	OM: Request Error
-1616xxxx	密码验证失败	Password Validation Failed
-1715xxxx	用户配置文件读取错误	Read User Profile Error
-1726xxxx	现金修改验证失败	Cash Change Validation Failed
-1733xxxx	获取账户错误	Get Account Error
-1743xxxx	修改账户错误	Change Account Error
-1751xxxx	发送错误	Send Error
-1760xxxx		Exchange Rate Error
-1773xxxx		Get Approval Data Error
-1815xxxx	读取用户配置文件错误	Read User Profile Error
-1826xxxx	市场数据验证失败	Market Data Validation Failed
-1833xxxx	获取账户错误	Get Account Error
-1843xxxx	修改账户错误	Change Account Error
-1851xxxx	发送指令	Send Error
-1860xxxx		Exchange Rate Error
-2326xxxx		Approval Validation Failed
-2323xxxx		Get Approval Request Error
-2426xxxx		Stock/Commodity In/Out Validation

		Failed
-2515xxxx	加载账户错误	Load Account Error
-2525xxxx	添加请求错误	Add Request Error
-2535xxxx	释放账户错误	Release Account Error
-2545xxxx	删除请求错误	Delete Request Error
-2555xxxx	获取账户错误	Get Account Error
-2566xxxx	平仓验证错误	Close Pos Validation Error

5. 使用说明 Q&A

1:Q - 我是不是需要向我的經紀行取得 API 的 App id & 授權碼去做 程式買賣?

A – 對。

2:Q - 是不是所有經紀行都可以支援使用 API?

A – SP 系統支援這功能，但開放與否請向你的經紀商查詢.

3:Q - 我看到你們的產品列表中，支持大部分期貨品種，請問這些品種可以通過 API 來交易嗎?

A –所有 SP 支援的產品，都可以透過 API 進行交易。

4:Q - SP 的 API 庫是否支援自動連接功能?

A- API 可以自動重連 , 請參考我們的 C#例子.

5:Q -调用 DLL 流程

A: SPAPI_SetLanguageId(可使用或不使用) → SPAPI_Initialize (初始化) → SetLoginInfo → Login → 自由使用 → Logout → Uninitialize.

6:Q SP API 支援那些程式語言?

A: SP 原生 API 是一個多用途的函式庫，支援 NET, C, C++, Java, Phyton 等程式語言而無需理會用那一個編譯器

7: Q- 我在內地用 SP 的模擬服務器 demo.spsystem.info，斷綫頻密。

A - demo 戶口只爲測試用途，並不保證內地用戶的 IP, 故連接內地可能會不穩定。

在實盤情況時，一般經紀行會有綫去內地，故只要通知經紀行是由內地登入，實盤時使用 API 應不會有此情況。

8.Q- 持仓是如何計算?
A - 上日持仓 = 上日的净仓. (LongShort 方向, B 为长(正)，S 为短(负))
今日长仓=买。(此数量只增大) 今日短仓 = 沽。(此数量只增大)
今日净仓=今日长-今日短(长>短:今净为长，长<短:今净为短，长=短:今净为 0)

正负数举例:长 5,短 3,今净仓:2(长). 长 3,短 5,今净仓:-2(短). 长 5 短 5 今净仓:0
净仓=上日持仓+今日净仓。
存取：是指客户分别在两个经纪商有两个户口可以通过存取将一个户口的持仓转到另一个户口的持仓中。

9:Q- GetPrice 中的成交时間，為何有有些產品沒有數值 ？
A – 有關成交時間，只要交易所有提供的都会有。(例如香港股票有，因交易所有提供；而香港期货沒有，因交易所沒有提供)。建議每個成交正式使用 SubscribeTicker 去讀取。

10:Q – 我透過 SPAPI_GetAccInfo 這個接口獲取的 AccName 亂碼，為什麼？

A - 經紀行是用繁體中文 BIG5 碼, 但內地使用 GB 碼，因此你看見亂碼。只要稍作轉換，就會正確顯示，示範例子請參考最新的說明書。

11:Q – 關於接口查詢現金結餘，我只有一個賬戶，爲什麼還有多個結餘？

A - 因你有不同貨幣，故有多個結餘。

12:Q - 請問一下，你們的資金賬戶《現金結餘》中不同的貨幣累計的時候匯率是怎麼取的？每一天的匯率是不變的嗎？

A – 如何取匯率可參考說明書，呼叫功能 GetCurrencyRateByCurrency。匯率可隨時更新，由經紀商決定，因此你要問一問經紀商。

13:Q - 关于 Instrument 与 Product 的关系。

A: SPAPI_LoadInstrumentList 是取交易所下面的产品系列都有那一些,
SPAPI_LoadProductInfoListByCode 根据它的系列再取它产品的详细信息。

14:Q - 在 SP 的期货及期权的保证金是如何计算？

A：由交易所及經紀行釐定。

不过，大多采取国际标准即 SPAN（风险标准 Portfolio 分析）标准组合风险分析系统，SPAN 是一种保证金计算系统，能根据期货/期权的预期风险来计算保证金，它能灵活的根据不同的市场因素（如波幅、标的指数）来度量风险，并按整个投资组合来计算。

此外，SPAN 把市场分为 16 个不同市况，计算出不同的风险排列的盈亏价值，以根据不同组合的最大亏损确定保证金水平。同时，它还能考虑到期权交易中多仓和空仓期权风险不一样，因而更为准确的衡量期权保证金，并可以确定期货或期权等衍生品组合的所有风险，因此，可以大大提高交易者的资金利用率。

15:Q – 关于下竞价单

A：下竞价单与市价单时(模拟不支持)，在真实盘时需要交易所支持当前产品能下竞价单与市价单。例如香港交易就支持产品能下期货与股票竞价单与市价单。在交易所支持的情况下 SP 系统都会支持。

16:Q – 关于 SPApiPrice 中的 LastTime

A：LastTime 时间格式为 Unix 时间。他只包含了时间数据，并不包含日期。交易所支持 LastTime 会显示为 Unix 时间。如果不支持请忽略此参数。

17:Q 关于市场历史数据

A：SP API 并不支持获取历史市场数据，有关数据用户需自行找相关经纪行购买。

18:Q 关于不同版本 Visual Studio 打开 Example 问题

A：SPAPI VC Example 使用 Visual Studio 2010 开发如果客户使用更高版本的 Visual Studio 打开它。请设置 Project/属性/配置属性/常规->下面的“平台工具集”选择“v100”.

或者客户可以自己用高版本 Visual Studio 新建立一个 Project。再将.h 与 .cpp 代码拷贝过去生成新的 Project。

19:Q 为什么会有两个不同方法但功能相同!

A: 有一些方法是一次性获取全部 List 信息, 其分别是 C++的 vector 实现的，别一种是用 C 的 malloc array pointer 来实现，因为不同的客户的需求不同，客户只需针对适合自己的方法注册其中一个来使用即可.使用方法可参数 Example.

20:Q 有关循环使用 LoadLibrary 与 FreeLibrary 去加载 DLL 阻塞问题!

A：1,虽然 DLL 可以 LoadLibrary 与 FreeLibrary 多次，但建议使用一次就离开程序.

2,如果出现阻塞或崩溃，可以重新启动解决问题!

正常使用步骤可参数 Q&A 5 和 Example.

21:Q GetAccInfo()时，取出来的参数总是 0?

A:AccInfo 中有一些数据是根据行情计算得到(例如:CommodityPL)。

1:客户可以收到 client 用户持仓后订阅持仓产品行情！从而得到正确的参数值

2:客户也能根据行情自己计算自己想要的数据!

22:Q 当 AE 轮流切换 AccountLogin 后取 GetPos 或 Trade. 获取数据会出现混乱问题!

A: DLL 采用快速缓存技术,当客户 AccountLogin 后会有相应的持仓或成交有回调,说明当前户口已完成加载相应数据.再去 Get 相应数据才会正确!否则 Get 的是上一户口数据！ 例如 POS 与 Trade 都有两个回调,一个登入前数据 和一个登入后更新数据.

23:Q 合约信息里的 DecInPrice 是什么意思呢，为什么下单也需要填。

A: DecInPrice 是指该合约的最大小数位是多少, 下单时填写能避免客户输入错误的价格.

24:Q 原有的 SPAPI_RegisterLoginAccInfo 在 v8.743 里被移除 sample 里 acc_no 是从这个方法里获取的，现在从哪里拿？SPAPI_RegisterAccountInfoPush 方法里有 ClientID，可以用这个么。

A:原有 SPAPI_RegisterLoginAccInfo 使用性不大,所以已经淘汰,客户如果想要相应的 acc_no 可以使用 SPAPI_RegisterAccountInfoPush 回调中的 ClientID

25:Q IntOrderNo or TradeNo 是否第二天后会重设为 1?SP 系统 order id 应该用什么来识别.

A：每个用户的 IntOrderNo 是独一无二的,即使是在第二天
ext_order_no & trade_no 每个网关或交换是独一无二的,因为他们是来自网关/交换

26:Q 关于 SPApiOrder 中的 OpenClose 怎么去填写?

A:无论开平仓(期货平仓,等于相反方向开)OpenClose 是会直接传給 gateway/excahnge,包括所有港期及外期,而港期(股票期权特别需要)会传给 clearing system.如沒有特别要求,可以填'\0'(OC_DEFAULT),如果乱给值,system 或者会当作 Open,不过最好都是填'\0'

27:Q 关于 PriceUpdate-LastQty 与 TickerUpdate-Qty 的区别

A：PriceUpdate - Last/LastQty 只记录最近 5~10 个成交

- Server side 有记录,每次 request 都有最近 5~10 个成交

TickerUpdate - Qty 发送即时成交

Server side 沒有记录,只在 client side 自行记录,每次 request 都要重新记录

28：Q 有时候 API 成交推送信息会滞后，市场价格都到了，产品还没有成交。

A:这是服务器/订单网关响应时间问题. Api 只实时转换 Server 消息。客户可以自行检查网络延时

如果客户为 AE Mode(经纪,多用户户口)需要注意以下几点

1.首先要向证券或经纪开通 AE 账户(例如开通 AE 账户名为 AE_01),再向 AE_01 关联多个普通子账户(例如在 AE_01 下开通 1000,1001,1002.....)。

2.AE 登录与普通账户登录不同， port 要用 8081,而普通账户则用 8080.填写好登录信息后登录。

3.AE 登录成功后与 Client 登录成功后的区别:当 Client 登录成功可以直接对“下单,持仓,用户信息等”进行操作,而 AE 需要登录指定户口 SPAPI_AccountLogin，
当登录成功后,才能对用户下单与查询用户的信息(见下表)。

4.AE 登录成功后，当子账户下单时都会收到子账户的下单信息.如果想拿到 AE 登录前的所有订单信息，可以用 LoadOrderReport 加载登录前的订单信息.(Trader 相同)

5.AE 下使用子账户,SPAPI 提供子账户登录与登出。(User_Id 是指登录账户，Acc_No 则是指子账户,当普通用户登录时 User_ID 与 Acc_No 是相同的，用 AE 登录例如 AE_01 这时 User_Id 是 AE_01,如果你想操作 1000 你就 AE 下(Access)登录 1000 这时 Acc_No 就等于 1000)

6.AE 下订单操作。AE_01 登录后再指定登录 1000，这时 User_Id=AE_01,Acc_No=1000 就可以对 1000 子账户进行订单操作(添加删除修改订单)

7.AE 下取持仓 户口信息。也是指定登录子账户后可以查询该子账户信息.
8.成交信息. 只能取 AE 下的订单成交记录.(AE_01 用 1000 下了一个 HSIK4 成交了,1000 自己下了一个 HSIM4 成交了.AE 只能拿到自己下的 HSIK4 记录，拿不到 1000 自己下单的成交记录).

9.SPAPI_LoadOrderReport 是请求登入前的在工作中与部分成交工作中的订单，此请求只能请求一次。(请求成交相同)
10.AE Mode 如果想看 Client Account 的订单或成交。需要先 Acc Access 一个客户才能去请求这个客户的订单
11.AE Mode 只能查看自己名下的 Clietnt 的订单与成交。

注：Client Mode 登录后可以对全部功能进行操作(AccountLogin 与 AccountLogin 除外)，AE Mode 下表打“✔”的功能需要 Access Login 后才能使用。(见下表)

SPAPI_AccountLogin 与 SPAPI_AccountLogout 只有是 AE Mode 登录后才能用，Client Mode 这两个方法是不可用的。

	功能	AE Mode	方法
订单(Order)	下单	✔	SPAPI_AddOrder
	删除订单	✔	SPAPI_DeleteOrder
	修改订单	✔	SPAPI_ChangeOrder
	查询订单	✘	SPAPI_GetOrderCount SPAPI_GetOrder SPAPI_GetOrderByOrderNo
	设有效订单	✔	SPAPI_ActivateOrder
	设无效订单	✔	SPAPI_InactivateOrder
持仓(Pos)	查询	✔	SPAPI_GetPosCount SPAPI_GetPos SPAPI_GetPosByProduct
成交(Trader)	查询	✘	SPAPI_LoadTradeReport(加载登录前成交) SPAPI_GetTradeCount SPAPI_GetTrade SPAPI_GetTradeByTradeNo
行情(Price)	订阅与取消	✘	SPAPI_SubscribePrice
	查询	✘	SPAPI_GetPriceCount SPAPI_GetPrice SPAPI_GetPriceByCode
产品系列(Instrument)	加载	✘	SPAPI_LoadInstrumentList 查询前需用它加载
	查询	✘	SPAPI_GetInstrumentCount SPAPI_GetInstrument SPAPI_GetInstrumentByCode
产品信息(Product)	加载	✘	SPAPI_LoadProductInfoListByCode 查询前需用它加载

	查询	✘	SPAPI_GetProductCount SPAPI_GetProduct SPAPI_GetProductByCode
Ticker	订阅与取消	✘	SPAPI_SubscribeTicker
用户信息 (Acc Info)	查询	✔	SPAPI_GetAccInfo
户口资金 (Acc Bal)	查询	✔	SPAPI_GetAccBalCount SPAPI_GetAccBal SPAPI_GetAccBalByCurrency