飞鼠PowerGate 交易平台编程接口

Version：1.0

2015/10/22

Chapter 1　　简介

飞鼠PowerGate 交易平台 (以下简称 PG)期货交易系统，包含交易服务器，行情服务器和风险管理服务器。

本文档中介绍的 API 用于与 PG 交易服务器进行通信。有了 API，投资者可以向上海期货交易所、郑州商品期货交易所、大连商品期货交易所 和 中国金融期货交易所，以及上海黄金交易所发送交易指令，接收相应的请求响应和市场数据数据。

1.1 API包介绍

飞鼠公司提供c++标准的AP。交易客户端通过API实现和 PG 交易服务器之间的通信。有了 API，交易客户端应用程序可以委托订单，撤单，接受行情，查询委托、成交、资金等。
库包包含以下文件:

File Name	File Description
SgitFtdcTraderApi.h	C++ 头文件　交易接口定义
SgitFtdcMdApi.h	C++ 头文件　行情接口定义
SgitFtdcUserApiStruct.h	C++ 头文件　数据结构体定义
SgitFtdcUserApiDataType.h	C++ h头文件　数据类型定义
Sgittraderapi.dll	交易接口连接库
Sgittraderapi.lib	
Sgitmduserapi.dll	行情接口连接库
Sgitmduserapi.lib	

注: 当编译与 MS Visual c + +，多线程必须打开。

Chapter 2 体系模式

2.1 通信模式

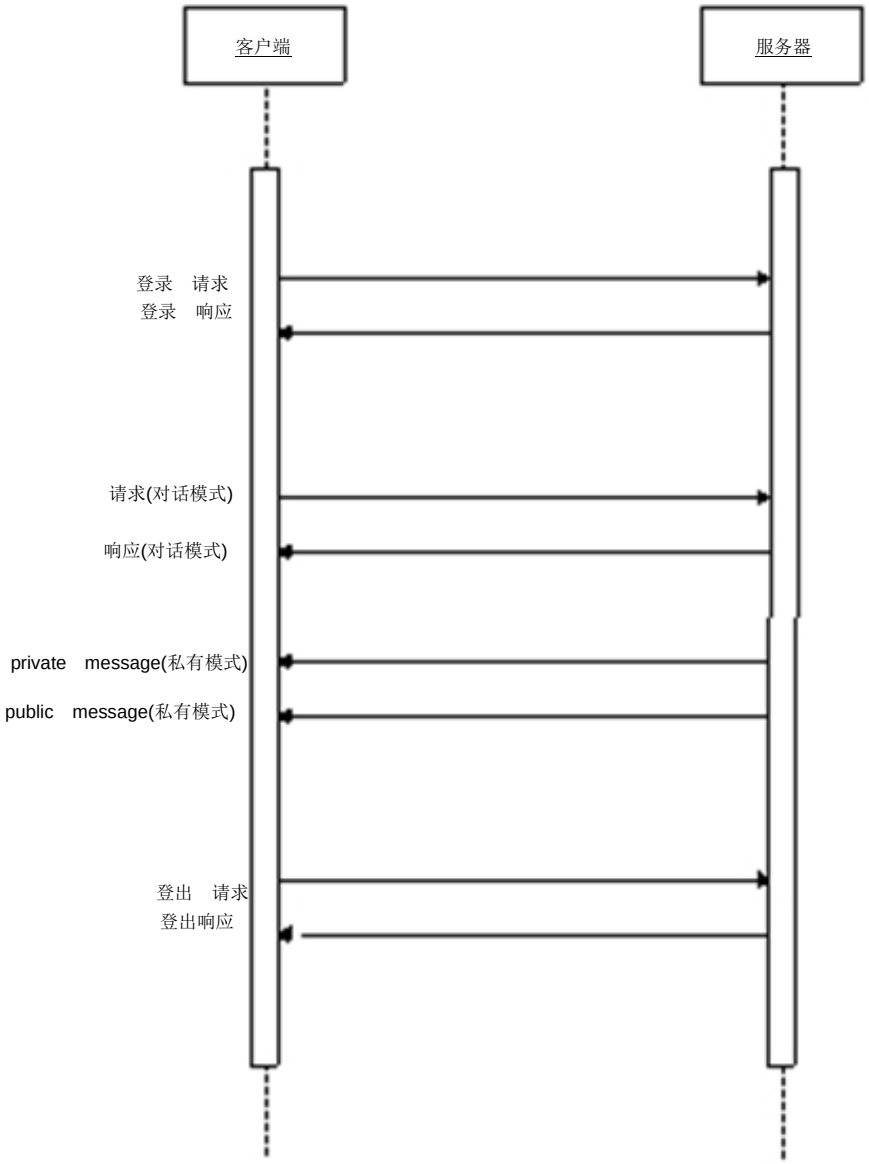
API 客户端与PG服务器以"期货交易数据交换协议 (FTD)"通讯。通讯方式分为以下三种模式：

对话模式： 客户端发送请求到 PG 服务器，后者返回相应的响应。

私有模式:：PG 服务器向特定的客户端发送私有消息。 这些消息都是某个API专有的通知消息如订单状态报告或贸易报告。

广播模式:：PG 将常见的信息发布到所有客户端注册到广播地址.

下面的关系图说明了通信过程中的这三种模式。



通信模式不对网络连接限制。所以，使用单一的网络连接时，客户端可以处理多个通信模式。例如：客户端可以通过单个网络连接，用对话通讯模式模式提交订单和接受相应的响应，使用广播模式 接收交易所状态通知，用私有模式接受委托回报、成交回报等私有的消息。三中通讯模式信息在同一个网络连接中，交叉同时传输。

2.2 数据流

PG 建立在 FTD协议基础上，通过上述的三种通讯模式传输不同的数据流数据流。

对话通信模式传输对话数据流和查询数据的流。对话数据流和查询数据流是双向数据流。客户端应用程序首先提交请求，然后 PG 服务器返回的响应。PG 服务器不保持对话数据流和查询数据流的状态。 出现问题时，例如，网络连接中断，数据流将会在重新连接后重置，连接断开时正在发送的数据将会丢失。

私有通信模式传输私有数据流。私有数据流是以单向流动的方式传送的。即：PG 服务器发送私有消息给相应的客户端应用程序。 私有消息通常包括风委托通知、成交通知。私人的数据流是可靠的当客户端应用程序失去与PG服务器的连接，在同一交易日的任何时间，客户端应用程序可以重新连接到 PG 服务器并重新获得自己相应的私有数据流，无任何其交易数据丢失的风险。

广播通讯模式传输流公共数据。它是就像私人的数据流单向可靠的数据流，它们之间唯一的区别是铸造的广泛通信数据会广播到所有连接的客户端应用程序。它主要用于传输市场数据、 合约状态通知或其他重要的公共信息。

2.3 工作线程

PG 客户端包含三种工作线程：客户应用程序主线程， API交易工作线程和API行情工作线程。API线程通过网络连接到PG服务器，完成客户端应用程序和PG服务器的交互。

API 是线程安全的，支持客户端多线程操作。多线程操作API时，应尽量快的处理完回调函数中的数据，避免阻塞工作线程。为了尽快的处理回调函数，可以在客户端中引入缓冲机制。

2.4 运行时数据存储文件

PG API 动态链接库在运行时，会创建一些本地的文件，用来存储运行时数据，这些文件以“.con”为扩展名，API开发人员可以通过调用函数 静态函数CreateFtdcTraderApi (const char * pszFlowPath) 和 CreateFtdcMdApi (const char * pszFlowPath)来指定文件存储目录。

API 还需要 动态库(dll文件)所处目录的父目录中存在一个名为"store"目录。

Chapter 3　　接口通讯模式

PG　API提供了四个接口类：

二个交易接口类：CSgitFtdcTraderApi和CSgitFtdcTraderSpi。

二个行情接口类：CSgitFtdcMdApi和CSgitFtdcMdSpi。

这四个接口类对FTD协议进行了封装,方便客户端应用程序的开发.

客户端应用程序可以通过CSgitFtdcTraderApi或CSgitFtdcMdApi对象发送操作请求，通过派生自CSgitFtdcTraderSpi或CSgitFtdcMdSpi的类对象的重载回调函数来处理后台服务的响应。

3.1 对话通讯模式

创建话模式接口的函数模式如下:

请求

int CSgitFtdcTraderApi::ReqXXX(CSgitFtdcXXXField *pReqXXX, int nRequestID)

int CSgitFtdcMDApi::ReqXXX(CSgitFtdcXXXField *pReqXXX, int nRequestID)

响应：

void CSgitFtdcTraderSpi::OnRspXXX(CSgitFtdcXXXField *pRspXXX, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

void CSgitFtdcMDSpi::OnRspXXX(CSgitFtdcXXXField *pRspXXX, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

请求函数的第一个参数是请求内容，不能为空。第二个参数是请求 ID，由客户端应用程序赋值，强烈建议在同一个客户端中nRequestID是唯一增量，当客户端收到来自 PG 服务器的响应消息，客户端可以通过请求ID把请求和响应匹配起来。

当客户端收到 PG 服务器的响应消息时，会回调CSgitFtdcXXXSpi的相应响应函数，如果响应有多条消息，回调函数将被多次调用。

响应函数的第一个参数是应答数据,通常其中包含原始的请求数据。如果发生错误或 PG 无法处理该请求，则该参数将设置为 NULL。第二个参数是一个消息参数，PG 用于显示此请求是否成功的响应。回调函数多次调用时，第二个参数可能会在非首次调用回调函数中为NULL。

第三个参数是其对应的请求函数的请求 ID。最后一个参数是响应的结束标志。如果它的值为"true"指示当前响应是最后一个。

3.2私有通讯模式

私有模式接口实例如下：

```
void CSgitFtdcTraderSpi::OnRtnXXX(CSgitFtdcXXXField *pXXX)

void OnRtnOrder(CSgitFtdcOrderField*pOrder,

CSgitFtdcRspInfoField *pRspInfo)
```

注: 行情数据,没有私有模式。

当 PG 服务器返回私有模式数据时，将调用 CSgitFtdcTradeSpi 派生类对象的重载的回调函数。 所有回调函数的第一个参数是从 PG 服务器返回的内容，如OnRtnOrder的 CSgitFtdcOrderField*pOrder，函数第二个参数是出现错误时,错误详细信息。

3.3 通讯模式和对应的函数

Type	Business	Request interface	Response interface	Stream
Login	Login	CSgitFtdcTraderApi::ReqUserLogin	CSgitFtdcTraderSpi::OnRspUserLogin	dialog
	Logout	CSgitFtdcTraderApi::ReqUserLogout	CSgitFtdcTraderSpi::OnRspUserLogout	dialog
	Password	CSgitFtdcTraderApi::ReqUserPasswordUpdate	CSgitFtdcTraderSpi::OnRspUserPasswordUpdate	dialog
Order	Create	CSgitFtdcTraderApi::ReqOrderInsert	CSgitFtdcTraderSpi::OnRspOrderInsert	dialog

	Modify	CSgitFtdcTraderApi::ReqOrderAction	CSgitFtdcTraderSpi::OnRspOrderAction	dialog
Report	Trade	N/A	CSgitFtdcTraderSpi:OnRtnTrade	private
	Order	N/A	CSgitFtdcTraderSpi:OnRtnOrder	private
Query	Order	CSgitFtdcTraderApi::ReqQryOrder	CSgitFtdcTraderSpi::OnRspQryOrder	query
	Account	CSgitFtdcTraderApi::ReqQryTradingAccount	CSgitFtdcTraderSpi::OnRspQryTradingAccount	query
	Investor	CSgitFtdcTraderApi::ReqQryInvestor	CSgitFtdcTraderSpi::OnRspQryInvestor	query
	position	CSgitFtdcTraderApi::ReqQryInvestorPositonDetail	CSgitFtdcTraderSpi::OnRspQryInvestorPositionDetail	query
	Instrument	CSgitFtdcTraderApi::ReqQryInstrument	CSgitFtdcTraderSpi::OnRspQryInstrument	query
Broadcast	Market Data	N/A	CSgitFtdcMdSpi::OnRtnDepthMarketData	public
Register		CSgitFtdcTraderSpi::SubscribePrivateTopic	N/A	
		CSgitFtdcTraderSpi::SubscribePublicTopic	N/A	
		CSgitFtdcMdSpi::SubscribeMarketTopic	N/A	

Chapter 4 API 接口使用

4.1 接口使用步骤

API 客户端应用程序应该编写程序以下步骤:

1. 创建"CSgitFtdcTraderApi"接口对象.
2. 用" CSgitFtdcTraderApi ::CSgitFtdcTraderSpi"的派生类创建响应对象,并通过函数"RegisterSpi"注册给"CSgitFtdcTraderApi"的接口对象
3. 订阅私有流数据,函数" CSgitFtdcTraderApi ::SubscribePrivateTopic"
4. 订阅公共流数据,函数 " CSgitFtdcTraderApi ::SubscribePublicTopic"
5. 注册交易服务器地址,函数 " CSgitFtdcTraderApi ::RegisterFront"
6. 连接到服务器,函数:" CSgitFtdcTraderApi ::Init"
7. 连接成功,回调函数" CSgitFtdcTraderSpi ::OnFrontConnected" ,开发者可以在这个函数中做登录操作" CSgitFtdcTraderApi ::ReqUserLogin"
8. 登录成功/失败,回调函数" CSgitFtdcTraderSpi ::OnRspUserLogin"
9. 登录成功后,调用 " CSgitFtdcTraderApi ::Ready()" ,然后交易服务器开始传送流数据.
10. 客户端和服务器,功能交互.

对于一些轻型交易程序,不需要以前的流数据,可以忽略第3和4部

4.2 特别声明

有几个编程规则需要注意:

1,所有请求函数的参数不能为NULL.

2,请求函数返回int类型,如果返回值为0,则请求成功;返回值不为0,请求错误,具体参见错误码.

3,创建API对象时，若没有指定运行文件的存储目录，则动态库(dll)的父目录同级,必须有store目录

4,OrderRef必须是前补0的12位字符串.

5,ReqOrderAction 的参数CSgitFtdcInputOrderActionFiel以下字段必填:UserID, ExchangeID, InvestorID, OrderSysID, InstrumentID, ActionFlag.
UserID 登录的交易员ID
ExchangeID 参见SgitFtdcUserApiDataType.h 中的交易所宏定义
InvestorID 交易编码.
OrderSysID 交易所返回的系统报单号,跟交易所返回保持完全一致,不能修改.

6,委托成功,会回调函数OnRtnOrder .参数中的InvestorID和ClentID恢复匹配.

7,撤单,回调OnRspOrderAction.如果撤单成功参数 CSgitFtdcInputOrderActionField.VolumeChange 表示成功撤单的数量

8,支持FAK指令.

9、CSgitFtdcInvestorPositionDetailField中OpenPrice持仓均价 ，ExchMargin昨持仓量。

10、对于上期所合约，客户可以指定平仓委托的开平仓标志为：平仓/平今/平昨；
如果客户指定开平仓标志为 平仓， 委托量 大于 可平仓量，则报单会被拒绝，报错信息“不足持仓可平”；
如果客户指定开平仓标志为 平昨 或者 平今，不再进行拆单；
使用新版本的sgitapi，如果是上期所合约，委托回报 与 成交回报，将返回平今/平昨标志，不再是平仓。

11、行情api接口CSgitFtdcMdApi支持行情订阅，需要在登录成功之后，调用Ready之前订阅。否则，会接收到所有合约行情。
如果在Ready之前订阅了行情，则在后续交易过程中可再次订阅行情。
如果在Ready之前没有订阅行情，而在后续交易过程中才订阅行情，则先是收到全部合约行情，订阅之后只能收到订阅的行情。

12,请求查询投资者响应(OnRspQryInvestor)结构体CSgitFtdcInvestorField中字段BrokerID为交易编码;

13,黄金交易所,平仓单中投保标志必须为投机+平仓,开仓单投保不限制.

4.3 CSgitFtdcTraderApi

CSgitFtdcTraderApi 是交易 API 的接口类，客户端应用程序通过调用此类的实例的成员函数与 PG 服务器进行交互。 此类的成员函数包括 API 操做、 委托、撤单、 交易所和交易所合约查询和客户端信息、 投资者帐户等其他信息查询...

4.3.1 CreateFtdcTraderApi

创建CSgitFtdcTradeApi*类型的静态函数.

注:不要用new创建api对象

声明：

static CSgitFtdcTradeApi *CreateFtdcTradeApi(const char *pszFlowPath = "");

参数：

pszFlowPath：接口数据文件存储位置,默认为当前目录

返回值：

成功,返回CSgitFtdcTradeApi类型指针;失败,返回 NULL .

4.3.2 Release

Release 释放CSgitFtdcTradeApi对象

注:不能通过 "delete"来释放CSgitFtdcTradeApi对象；

声明：

void Release();

4.3.3 Init

Init初始化api,并启动工作线程,连接服务器.

声明：

void Init();

4.3.4 Join

Join 主线程等待api工作线程.

声明：

void Join();

4.3.5 GetTradingDay

查询当前交易日期.

声明：

const char *GetTradingDay();

返回值：

返回类型为 YYYYMMDD的字符串.

4.3.6 RegisterSpi

Register 注册 CSgitFtdcTraderSpi派生的回调对象

声明：

void RegisterSpi(CSgitFtdcTraderSpi *pSpi) ;

参数：

pSpi： CSgitFtdcTraderSpi派生类的对象指针.

4.3.7 RegisterFront

注册PG服务器地址

声明：

void RegisterFront(char *pszFrontAddress);

参数：

pszFrontAddress： PG服务器地址.

格式"tcp://127.0.0.1:17001",

4.3.8 SubscribePrivateTopic

订阅私有流,在 "Init"之前调用

声明：

void SubscribePrivateTopic(TE_RESUME_TYPE nResumeType);

参数：

nResumeType：

TERT_RESTART:restart模式.

TERT_RESUME: resume模式.

TERT_QUICK: quick模式

4.3.9 SubscribePublicTopic

订阅公共流,必须在"Init"之前调用;

声明：

void SubscribePublicTopic(TE_RESUME_TYPE nResumeType);

参数：

nResumeType：

TERT_RESTART:restart模式.

TERT_RESUME: resume模式.

TERT_QUICK: quick模式

4.3.10 Ready

PG服务器受到Reqdy指令,开始想API推送订阅的私有流和公共流数据.

.

声明：

void Ready();

4.3.11 ReqUserLogin

登录请求函数

声明：

int ReqUserLogin(CSgitFtdcReqUserLoginField *pReqUserLoginField, int nRequestID)

参数：

pReqUserLoginField：

struct CSgitFtdcReqUserLoginField

{

///trading day

TSgitFtdcDateType TradingDay;

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///user id

TSgitFtdcUserIDType UserID; //必填

///password

TSgitFtdcPasswordType Password; //必填

///user product information

TSgitFtdcProductInfoType UserProductInfo;

///interface product information

TSgitFtdcProductInfoType InterfaceProductInfo;

///protocol information

TSgitFtdcProtocolInfoType ProtocolInfo;

};

返回值：

0，成功.

-1,网络连接问题.

-2,超过每秒最大请求量(100).

4.3.12 ReqUserLogout

登出.

声明：

int ReqUserLogout(CSgitFtdcUserLogoutField *pUserLogout, int nRequestID);

参数：

pReqUserLogout：

struct CSgitFtdcUserLogoutField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///user id TSgitFtdcUserIDType UserID; //必填

};

返回值：

0，成功.

-1,网路连接问题.

-2, 超过每秒最大请求数量.

4.3.13 ReqUserPasswordUpdate

修改登录密码.

声明：

int ReqUserPasswordUpdate(CSgitFtdcUserPasswordUpdateField *pUserPasswordUpdate, int nRequestID);

参数：

pUserPasswordUpdate：

struct CSgitFtdcUserPasswordUpdateField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///user id

TSgitFtdcUserIDType UserID;

///old password

TSgitFtdcPasswordType OldPassword;

///new password

TSgitFtdcPasswordType NewPassword;

};

4.3.14 ReqOrderInsert

委托请求

声明：

int ReqOrderInsert(CSgitFtdcInputOrderField *pInputOrder, int nRequestID)

参数：

pInputOrder：

struct CSgitFtdcInputOrderField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

///order reference

TSgitFtdcOrderRefType OrderRef;

///user id

TSgitFtdcUserIDType UserID;

///order price type

TSgitFtdcOrderPriceTypeType OrderPriceType;

///direction

TSgitFtdcDirectionTypeDirection;

```
///combination order's offset flag
TSgitFtdcCombOffsetFlagType CombOffsetFlag;
///combination or hedge flag
TSgitFtdcCombHedgeFlagType CombHedgeFlag;
///price
TSgitFtdcPriceType	LimitPrice;
///volume
TSgitFtdcVolumeType VolumeTotalOriginal;
///valid date
TSgitFtdcTimeConditionType	TimeCondition;
///GTD DATE
TSgitFtdcDateType GTDDate;
///volume condition
TSgitFtdcVolumeConditionType VolumeCondition;
///min volume
TSgitFtdcVolumeType MinVolume;
///trigger condition
TSgitFtdcContingentConditionType	ContingentCondition;
///stop price
TSgitFtdcPriceType	StopPrice;
///force close reason
TSgitFtdcForceCloseReasonType	ForceCloseReason;
/// auto suspend flag
TSgitFtdcBoolType IsAutoSuspend;
///business unit
TSgitFtdcBusinessUnitType BusinessUnit;
///request ID
TSgitFtdcRequestIDType	RequestID;
};
```

InvestorID:填交易编码

OrderRef: 必须是12位前补0字符串.其值唯一递增,可以不连续.上次最大OrderRef在登录请求响应函数 OnRspUserLogin, 中的CSgitFtdcRspUserLoginField.MaxOrderRef.

4.3.15 ReqOrderAction

撤单

声明：

int ReqOrderAction(CSgitFtdcOrderActionField *pOrderAction, int nRequestID)

参数：

pOrderAction ：
struct CSgitFtdcOrderActionField

{

///broker id

TSgitFtdcBrokerIDType

BrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///order action reference

TSgitFtdcOrderActionRefType OrderActionRef;

///order reference

TSgitFtdcOrderRefType OrderRef;

///request ID

TSgitFtdcRequestIDType RequestID;

///front ID

TSgitFtdcFrontIDType FrontID;

///session ID

TSgitFtdcSessionIDType SessionID;

///exchange ID

TSgitFtdcExchangeIDType ExchangeID;

///order system ID

TSgitFtdcOrderSysIDType OrderSysID;

///action flag

```
TSgitFtdcActionFlagType    ActionFlag;
///price
TSgitFtdcPriceType         LimitPrice;
///volume change
TSgitFtdcVolumeType VolumeChange;
///action date
TSgitFtdcDateType          ActionDate;
///action time
TSgitFtdcTimeType          ActionTime;
///trader ID
TSgitFtdcTraderIDType TraderID;
///install ID
TSgitFtdcInstallIDType    InstallID;
///order local ID
TSgitFtdcOrderLocalIDType          OrderLocalID;
///action local ID
TSgitFtdcOrderLocalIDType          ActionLocalID;
///participant ID
TSgitFtdcParticipantIDType         ParticipantID;
///trading code
TSgitFtdcClientIDTypeClientID;
///business unit
TSgitFtdcBusinessUnitType BusinessUnit;
///order action status
TSgitFtdcOrderActionStatusType OrderActionStatus;
///user id
 TSgitFtdcUserIDType UserID;
///status message
TSgitFtdcErrorMsgType StatusMsg;
};
```

4.3.16 ReqQryOrder

委托单查询.

声明：

int ReqQryOrder(CSgitFtdcQryOrderField *pQryOrder, int nRequestID)**;**

参数：

pQryOrder：

struct CSgitFtdcQryOrderField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

///exchange ID

TSgitFtdcExchangeIDType ExchangeID;

///order system ID

TSgitFtdcOrderSysIDType OrderSysID;

};

4.3.17 ReqQryInvestor

投资者查询。

声明：

int ReqQry Investor (CSgitFtdcQryInvestorField *pQryInvestor, int nRequestID)

参数：

pQryInvestor：

struct CSgitFtdcQryInvestorField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

};

4.3.18 ReqQryTradingAccount

查询资金. 不查询也会定时推送.

声明：

int ReqQryTradingAccount(CSgitFtdcQryTradingAccountField *pQryTradingAccount, int nRequestID);

参数：

pQryTradingAccount：

struct CSgitFtdcQryTradingAccountField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

};

4.3.19 ReqQryInstrument

查询交易所合约

声明：

int ReqQryInstrument(CSgitFtdcQryInstrumentField *pQryInstrument, int nRequestID);

参数：

pQryInstrument：

struct CSgitFtdcQryInstrumentField

{

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

///exchange ID

TSgitFtdcExchangeIDType ExchangeID;

///exchange instrument ID

TSgitFtdcExchangeInstIDType ExchangeInstID;

///product ID

TSgitFtdcInstrumentIDType ProductID;

};

4.3.20 ReqQryInvestorPositionDetail

查询持仓明细

声明：

int ReqQryInvestorPositionDetail(CSgitFtdcQryInvestorPositionDetailField *pQryInvestorPositionDetail, int nRequestID)**;**

参数：

pQryInvestorPositionDetail：
requeststruct CSgitFtdcQryInvestorPositionDetailField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

};

4.4 CSgitFtdcTraderSpi

API回调类.通过继承CSgitFtdcTraderSpi类,重载回调函数.

4.4.1 OnFrontConnected

API客户端网络连接到 PG 服务器时,回调改函数.
可以在该函数中调用"ReqUserLogin" 进行登录操作

声明：

void OnFrontConnected();

4.4.2 OnFrontDisconnected

API客户端到PG服务器的网络连接断开,回调此函数.

声明：

void OnFrontDisconnected (char *pErrMsg);

参数：

pErrMsg: 错误信息

4.4.3 OnRspUserLogin

登录请求响应函数.

声明：

void OnRspUserLogin(CSgitFtdcRspUserLoginField *pRspUserLogin, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pRspUserLogin：

struct CSgitFtdcRspUserLoginField

{

///trading day

TSgitFtdcDateType TradingDay;

///time of login

TSgitFtdcTimeType LoginTime;

///broker id

TSgitFtdcBrokerIDType BrokerID;

///user id

TSgitFtdcUserIDType UserID;

///trade system name

TSgitFtdcSystemNameType SystemName;

};

pRspInfo：

struct CSgitFtdcRspInfoField

{

///error id

TSgitFtdcErrorIDType ErrorID;

///error information

TSgitFtdcErrorMsgType ErrorMsg;

};

4.4.4 OnRspUserLogout

登出请求响应函数

声明：

Void OnRspUserLogout(CSgitFtdcUserLogoutField *pUserLogout, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pRspUserLogout：

struct CSgitFtdcUserLogoutField

{

///broker id

TSgitFtdcBrokerIDType BrokerID;

///user id

TSgitFtdcUserIDType UserID;

};

4.4.5 OnRspUserPasswordUpdate

修改登录密码响应函数.

definition：

void OnRspUserPasswordUpdate(CSgitFtdcUserPasswordUpdateField *pUserPasswordUpdate, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

parameters：

pUserPasswordUpdate：Address of the structure for the password modification response

struct CSgitFtdcUserPasswordUpdateField

{

///broker id

TSgitFtdcBrokerIDType BrokerID;

///user id

TSgitFtdcUserIDType UserID;

///old password

TSgitFtdcPasswordType OldPassword;

///new password

TSgitFtdcPasswordType NewPassword;

};

4.4.6 OnRspOrderInsert

This function is reserved, turn to OnRtnOrder to handle new order reports.

4.4.7 OnRspOrderAction

Response callback function of the order action request **"**ReqOrderAction**"** .

definition：

void OnRspOrderAction(CSgitFtdcOrderActionField *pOrderAction, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)**;**

parameters：

pOrderAction：Address of the structure for the response of order action

```
struct CSgitFtdcOrderActionField
{
///broker id
TSgitFtdcBrokerIDTypeBrokerID;
///investor ID
TSgitFtdcInvestorIDType InvestorID;
///order action reference
TSgitFtdcOrderActionRefType OrderActionRef;
///order reference
TSgitFtdcOrderRefType OrderRef;
///request ID
TSgitFtdcRequestIDType    RequestID;
///front ID
TSgitFtdcFrontIDType FrontID;
///session ID
TSgitFtdcSessionIDType              SessionID;
///exchange ID
TSgitFtdcExchangeIDType ExchangeID;
///order system ID
TSgitFtdcOrderSysIDType OrderSysID;
///action flag
TSgitFtdcActionFlagType    ActionFlag;
///price
TSgitFtdcPriceType          LimitPrice;
///volume change
TSgitFtdcVolumeType VolumeChange;
///action date
TSgitFtdcDateType          ActionDate;
///action time
TSgitFtdcTimeType          ActionTime;
///trader ID
TSgitFtdcTraderIDTypeTraderID;
```

///install ID

TSgitFtdcInstallIDType InstallID;

///order local ID

TSgitFtdcOrderLocalIDType OrderLocalID;

///action local ID

TSgitFtdcOrderLocalIDType ActionLocalID;

///participant ID

TSgitFtdcParticipantIDType ParticipantID;

///trading code

TSgitFtdcClientIDTypeClientID;

///business unit

TSgitFtdcBusinessUnitType BusinessUnit;

///order action status

TSgitFtdcOrderActionStatusType OrderActionStatus;

///user id

TSgitFtdcUserIDType UserID;

///status message

TSgitFtdcErrorMsgType StatusMsg;

};

4.4.8 OnRspQryOrder

委托查询响应函数

声明：

void OnRspQryOrder(CSgitFtdcOrderField *pOrder, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pOrder：

struct CSgitFtdcOrderField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

```
///investor ID
TSgitFtdcInvestorIDType InvestorID;
///instrument ID
TSgitFtdcInstrumentIDType InstrumentID;
///order reference
TSgitFtdcOrderRefType OrderRef;
///user id
TSgitFtdcUserIDType UserID;
///order price type
TSgitFtdcOrderPriceTypeType OrderPriceType;
///direction
TSgitFtdcDirectionTypeDirection;
///combination order's offset flag
TSgitFtdcCombOffsetFlagType CombOffsetFlag;
///combination or hedge flag
TSgitFtdcCombHedgeFlagType CombHedgeFlag;
///price
TSgitFtdcPriceType          LimitPrice;
///volume
TSgitFtdcVolumeType VolumeTotalOriginal;
///valid date type
TSgitFtdcTimeConditionType          TimeCondition;
///GTD DATE
TSgitFtdcDateType GTDDate;
///volume condition TSgitFtdcVolumeConditionType VolumeCondition;
///min volume
TSgitFtdcVolumeType MinVolume;
///trigger condition
TSgitFtdcContingentConditionType    ContingentCondition;
///stop price
TSgitFtdcPriceType          StopPrice;
///force close reason
```

TSgitFtdcForceCloseReasonType ForceCloseReason;

/// auto suspend flag

TSgitFtdcBoolType IsAutoSuspend;

///business unit

TSgitFtdcBusinessUnitType BusinessUnit;

///request ID

TSgitFtdcRequestIDType RequestID;

///order local ID

TSgitFtdcOrderLocalIDType OrderLocalID;

///exchange ID

TSgitFtdcExchangeIDType ExchangeID;

///participant ID

TSgitFtdcParticipantIDType ParticipantID;

///trading code

TSgitFtdcClientIDTypeClientID;

///exchange instrument ID

TSgitFtdcExchangeInstIDType ExchangeInstID;

///trader ID

TSgitFtdcTraderIDTypeTraderID;

///install ID

TSgitFtdcInstallIDType InstallID;

///order submit status

TSgitFtdcOrderSubmitStatusType OrderSubmitStatus;

///order notify sequence

TSgitFtdcSequenceNoType NotifySequence;

///trading day

TSgitFtdcDateType TradingDay;

///settlement ID

TSgitFtdcSettlementIDType SettlementID;

///order system ID

TSgitFtdcOrderSysIDType OrderSysID;

///order source

TSgitFtdcOrderSourceType OrderSource;

///order status

TSgitFtdcOrderStatusType OrderStatus;

///order type

TSgitFtdcOrderTypeType OrderType;

///volume traded

TSgitFtdcVolumeType VolumeTraded;

/// total volume

TSgitFtdcVolumeType VolumeTotal;

///insert date

TSgitFtdcDateType InsertDate;

///insert time

TSgitFtdcTimeType InsertTime;

///active time

TSgitFtdcTimeType ActiveTime;

///suspend time

TSgitFtdcTimeType SuspendTime;

///update time

TSgitFtdcTimeType UpdateTime;

///cancel time

TSgitFtdcTimeType CancelTime;

///active trader ID

TSgitFtdcTraderIDTypeActiveTraderID;

///clear participant ID

TSgitFtdcParticipantIDType ClearingPartID;

///sequence No.

TSgitFtdcSequenceNoType SequenceNo;

///front ID

TSgitFtdcFrontIDType FrontID;

///session ID

TSgitFtdcSessionIDType SessionID;

///user product information

TSgitFtdcProductInfoType UserProductInfo;

///status message

TSgitFtdcErrorMsgType StatusMsg;

};

4.4.9 OnRspQryInvestor

投资者查响应函数.

声明：

void OnRspQry Investor (CSgitFtdcInvestorField *pInvestor, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)**;**

参数：

pInvestor：

struct CSgitFtdcInvestorField

{

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor group ID

TSgitFtdcInvestorIDType InvestorGroupID;

///investor name

TSgitFtdcPartyNameType InvestorName;

///Identified Card Type

TSgitFtdcIdCardTypeType IdentifiedCardType;

///Identified Card No.

TSgitFtdcIdentifiedCardNoType IdentifiedCardNo;

///is active

TSgitFtdcBoolType IsActive;

};

4.4.10 OnRspQryTradingAccount

客户资金查询响应函数.

声明：

void OnRspQryTradingAccount(CSgitFtdcTradingAccountField *pTradingAccount, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pTradingAccount ：

struct CSgitFtdcTradingAccountField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///account id

TSgitFtdcAccountIDType AccountID;

///previous mortgage

TSgitFtdcMoneyType PreMortgage;

///previous credit

TSgitFtdcMoneyType PreCredit;

///previous deposit

TSgitFtdcMoneyType PreDeposit;

///previous balance

TSgitFtdcMoneyType PreBalance;

///premargin

TSgitFtdcMoneyType PreMargin;

///interest base

TSgitFtdcMoneyType InterestBase;

///interest

TSgitFtdcMoneyType Interest;

///deposit

TSgitFtdcMoneyType Deposit;

```
///withdraw
TSgitFtdcMoneyType Withdraw;
///frozen margin
TSgitFtdcMoneyType FrozenMargin;
///frozen cash
TSgitFtdcMoneyType FrozenCash;
///frozen commission
TSgitFtdcMoneyType FrozenCommission;
///current margin
 TSgitFtdcMoneyType CurrMargin;
///cash in TSgitFtdcMoneyType CashIn;
///commission
TSgitFtdcMoneyType Commission;
///close profit
TSgitFtdcMoneyType CloseProfit;
///position profit
TSgitFtdcMoneyType PositionProfit;
///balance
TSgitFtdcMoneyType Balance;
///available
TSgitFtdcMoneyType Available;
///withdraw quota
TSgitFtdcMoneyType WithdrawQuota;
///reserve
TSgitFtdcMoneyType Reserve;
///trading day
TSgitFtdcDateType       TradingDay;
///settlement ID
TSgitFtdcSettlementIDType SettlementID;
///credit
TSgitFtdcMoneyType Credit;
///Mortgage
```

TSgitFtdcMoneyType Mortgage;

///excahnge margin

TSgitFtdcMoneyType ExchangeMargin;

};

4.4.11 OnRspQryInstrument

交易所合约查询响应函数

声明：

void OnRspQryInstrument(CSgitFtdcInstrumentField *pInstrument, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pInstrument：

struct CSgitFtdcInstrumentField

{

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

///exchange ID

TSgitFtdcExchangeIDType ExchangeID;

///instrument name

TSgitFtdcInstrumentNameType InstrumentName;

///exchange instrument ID

TSgitFtdcExchangeInstIDType ExchangeInstID;

///product ID

TSgitFtdcInstrumentIDType ProductID;

///product class

TSgitFtdcProductClassType ProductClass;

///delivery year

TSgitFtdcYearType DeliveryYear;

///delivery month

TSgitFtdcMonthType DeliveryMonth;

///max volume for market order

TSgitFtdcVolumeType MaxMarketOrderVolume;

///min volume for market order

TSgitFtdcVolumeType MinMarketOrderVolume;

///max volume for limit order

TSgitFtdcVolumeType MaxLimitOrderVolume;

///min volume for limit order

TSgitFtdcVolumeType MinLimitOrderVolume;

///volume multiple of instrument

TSgitFtdcVolumeMultipleType VolumeMultiple;

///price tick

TSgitFtdcPriceType PriceTick;

///create date

TSgitFtdcDateType CreateDate;

///open date

TSgitFtdcDateType OpenDate;

///expire date

TSgitFtdcDateType ExpireDate;

///start delivery date

TSgitFtdcDateType StartDelivDate;

///end delivery date

TSgitFtdcDateType EndDelivDate;

///instrument life phase

TSgitFtdcInstLifePhaseTypeInstLifePhase;

///is trading

TSgitFtdcBoolType IsTrading;

///position type

TSgitFtdcPositionTypeType PositionType;

///position date type

TSgitFtdcPositionDateTypeType PositionDateType;

///long margin ratio

TSgitFtdcRatioType LongMarginRatio;

///short margin ratio

TSgitFtdcRatioType ShortMarginRatio;

};

4.4.12 OnRspQryInvestorPositionDetail

持仓明细查询响应函数

声明：

void OnRspQryInvestorPositionDetail(CSgitFtdcInvestorPositionDetailField *pInvestorPositionDetail, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pInvestorPositionDetail ：
ReqQryInvestorPositionDetailstruct CSgitFtdcInvestorPositionDetailField

{

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///hedge flag

TSgitFtdcHedgeFlagType HedgeFlag;

///direction

TSgitFtdcDirectionTypeDirection;

///open date

TSgitFtdcDateType OpenDate;

///trade ID TSgitFtdcTradeIDType TradeID;

///volume

TSgitFtdcVolumeType Volume;

///open price

TSgitFtdcPriceType OpenPrice;

///trading day

TSgitFtdcDateType TradingDay;

///settlement ID

TSgitFtdcSettlementIDType SettlementID;

///trade type

TSgitFtdcTradeTypeType TradeType;

///combination instrument ID

TSgitFtdcInstrumentIDType CombInstrumentID;

};

4.4.13 OnRtnTrade

成交回报响应函数

声明：

void OnRtnTrade(CSgitFtdcTradeField *pTrade)

参数：

pTrade：

struct CSgitFtdcTradeField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

///order reference

TSgitFtdcOrderRefType OrderRef;

///user id

TSgitFtdcUserIDType UserID;

///exchange ID

TSgitFtdcExchangeIDType ExchangeID;

///trade ID

TSgitFtdcTradeIDType TradeID;

```
///direction
TSgitFtdcDirectionTypeDirection;
///order system ID
TSgitFtdcOrderSysIDType OrderSysID;
///participant ID
TSgitFtdcParticipantIDType          ParticipantID;
///trading code
TSgitFtdcClientIDTypeClientID;
///trading role
TSgitFtdcTradingRoleType TradingRole;
///exchange instrument ID
TSgitFtdcExchangeInstIDType ExchangeInstID;
///offset flag
TSgitFtdcOffsetFlagType   OffsetFlag;
///hedge flag
TSgitFtdcHedgeFlagType   HedgeFlag;
///price
TSgitFtdcPriceType          Price;
///volume
TSgitFtdcVolumeType Volume;
///trade date
TSgitFtdcDateType          TradeDate;
///trade time
TSgitFtdcTimeType          TradeTime;
///trade type
TSgitFtdcTradeTypeType   TradeType;
///price source
TSgitFtdcPriceSourceType PriceSource;
///trader ID
TSgitFtdcTraderIDTypeTraderID;
///order local ID
TSgitFtdcOrderLocalIDType          OrderLocalID;
```

///clear participant ID

TSgitFtdcParticipantIDType ClearingPartID;

///business unit

TSgitFtdcBusinessUnitType BusinessUnit;

///sequence No.

TSgitFtdcSequenceNoType SequenceNo;

///trading day

TSgitFtdcDateType TradingDay;

///settlement ID

TSgitFtdcSettlementIDType SettlementID;

};

4.4.14 OnRtnOrder

委托响应函数

声明：

void OnRtnOrder(CSgitFtdcOrderField *pOrder, CSgitFtdcRspInfoField *pRspInfo);

参数：

pOrder：Address of the structure for the order information

struct CSgitFtdcOrderField

{

///broker id

TSgitFtdcBrokerIDTypeBrokerID;

///investor ID

TSgitFtdcInvestorIDType InvestorID;

///instrument ID

TSgitFtdcInstrumentIDType InstrumentID;

///order reference

TSgitFtdcOrderRefType OrderRef;

///user id TSgitFtdcUserIDType

UserID;

```
///order price type
TSgitFtdcOrderPriceTypeType OrderPriceType;
///direction
TSgitFtdcDirectionTypeDirection;
///combination order's offset flag
TSgitFtdcCombOffsetFlagType CombOffsetFlag;
///combination or hedge flag
TSgitFtdcCombHedgeFlagType CombHedgeFlag;
///price
TSgitFtdcPriceType          LimitPrice;
///volume
TSgitFtdcVolumeType VolumeTotalOriginal;
///    valid date
TSgitFtdcTimeConditionType          TimeCondition;
///GTD DATE
TSgitFtdcDateType GTDDate;
///volume condition
TSgitFtdcVolumeConditionType VolumeCondition;
///min volume
TSgitFtdcVolumeType MinVolume;
///trigger condition
TSgitFtdcContingentConditionType    ContingentCondition;
///stop price
TSgitFtdcPriceType          StopPrice;
///force close reason
TSgitFtdcForceCloseReasonType               ForceCloseReason;
/// auto suspend flag
TSgitFtdcBoolType IsAutoSuspend;
///business unit
TSgitFtdcBusinessUnitType BusinessUnit;
///request ID
TSgitFtdcRequestIDType    RequestID;
```

```
///order local ID
TSgitFtdcOrderLocalIDType	OrderLocalID;
///exchange ID
TSgitFtdcExchangeIDType ExchangeID;
///participant ID
TSgitFtdcParticipantIDType	ParticipantID;
///trading code
TSgitFtdcClientIDTypeClientID;
///exchange instrument ID
TSgitFtdcExchangeInstIDType ExchangeInstID;
///trader ID
TSgitFtdcTraderIDTypeTraderID;
///install ID
TSgitFtdcInstallIDType	InstallID;
///order submit status
TSgitFtdcOrderSubmitStatusType	OrderSubmitStatus;
///notify sequence
TSgitFtdcSequenceNoType NotifySequence;
///trading day
TSgitFtdcDateType	TradingDay;
///settlement ID
TSgitFtdcSettlementIDType SettlementID;
///order system ID
TSgitFtdcOrderSysIDType OrderSysID;
///order source
TSgitFtdcOrderSourceType OrderSource;
///order status
TSgitFtdcOrderStatusType OrderStatus;
///order type
TSgitFtdcOrderTypeType	OrderType;
///volume traded
TSgitFtdcVolumeType VolumeTraded;
```

```
///volume total
TSgitFtdcVolumeType VolumeTotal;
///insert date
TSgitFtdcDateType       InsertDate;
///insert time
TSgitFtdcTimeType       InsertTime;
///active time
TSgitFtdcTimeType       ActiveTime;
///suspend time
TSgitFtdcTimeType       SuspendTime;
///update time
TSgitFtdcTimeType       UpdateTime;
///cancel time
TSgitFtdcTimeType       CancelTime;
///active trader ID
TSgitFtdcTraderIDTypeActiveTraderID;
///clear participant ID
TSgitFtdcParticipantIDType ClearingPartID;
///sequence No.
TSgitFtdcSequenceNoType SequenceNo;
///front ID
TSgitFtdcFrontIDType FrontID;
///session ID
TSgitFtdcSessionIDType          SessionID;
///user product information
TSgitFtdcProductInfoType UserProductInfo;
///status message
TSgitFtdcErrorMsgType StatusMsg;
};
```

4.5 CSgitFtdcMdApi

CSgitFtdcMdApi 作为行情 API 的接口类，客户端应用程序通过调用此类的实例的成员函数与 PG 行情服务器进行交互。
CSgitFtdcMdApi 的结构类似于 CSgitFtdcTraderApi，只是 CSgitFtdcMdApi 交互的是行情数据而不是顺序的交易数据。
只有不同的功能在以下各小节中列出。

4.5.1 CreateFtdcMdApi

创建行情API对象,参照CSgitFtdcMdApi::CreateFtdcTradeApi

4.5.2 SetMultiCastAddr

注册多播地址.

声明：

void SetMultiCastAddr(char *szMlCastAddr);

参数：

szMlCastAddr:
字符串类型如下
//hostIP$multiaddress:port

//ANY$multiaddress:port

4.5.1 SubscribeMarketTopic

订阅行情流数据,必须在 "Init"之前调用

声明：

void SubscribeMarketTopic(TE_RESUME_TYPE nResumeType);

参数：

nResumeType：

TERT_RESTART: restart模式 .

TERT_RESUME: resume模式.

TERT_QUICK: quick模式.

4.6 CSgitFtdcMdSpi

行情回调响应类.

4.6.1 OnRtnDepthMarketData

行情更新响应函数.

声明：

void OnRtnDepthMarketData(CSgitFtdcDepthMarketDataField *pDepthMarketData);

参数：

pDepthMarketData:

struct CSgitFtdcDepthMarketDataField

{

TSgitFtdcDateType TradingDay;

TSgitFtdcInstrumentIDType InstrumentID;

TSgitFtdcExchangeIDType ExchangeID;

TSgitFtdcExchangeInstIDType ExchangeInstID;

TSgitFtdcPriceType LastPrice;

TSgitFtdcPriceType PreSettlementPrice;

TSgitFtdcPriceType PreClosePrice;

TSgitFtdcLargeVolumeType PreOpenInterest;

TSgitFtdcPriceType OpenPrice;

```
TSgitFtdcPriceType              HighestPrice;
TSgitFtdcPriceType              LowestPrice;
TSgitFtdcVolumeType     Volume;
TSgitFtdcMoneyType      Turnover;
TSgitFtdcLargeVolumeType                OpenInterest;
TSgitFtdcPriceType              ClosePrice;
TSgitFtdcPriceType              SettlementPrice;
TSgitFtdcPriceType              UpperLimitPrice;
TSgitFtdcPriceType              LowerLimitPrice;
TSgitFtdcRatioType              PreDelta;
TSgitFtdcRatioType              CurrDelta;
TSgitFtdcTimeType               UpdateTime;
TSgitFtdcMillisecType   UpdateMillisec;
TSgitFtdcPriceType              BidPrice1;
TSgitFtdcVolumeType     BidVolume1;
TSgitFtdcPriceType              AskPrice1;
TSgitFtdcVolumeType     AskVolume1;
TSgitFtdcPriceType              BidPrice2;
TSgitFtdcVolumeType     BidVolume2;
TSgitFtdcPriceType              AskPrice2;
TSgitFtdcVolumeType     AskVolume2;
TSgitFtdcPriceType              BidPrice3;
TSgitFtdcVolumeType     BidVolume3;
TSgitFtdcPriceType              AskPrice3;
TSgitFtdcVolumeType     AskVolume3;
TSgitFtdcPriceType              BidPrice4;
TSgitFtdcVolumeType     BidVolume4;
TSgitFtdcPriceType              AskPrice4;
TSgitFtdcVolumeType     AskVolume4;
TSgitFtdcPriceType              BidPrice5;
TSgitFtdcVolumeType     BidVolume5;
TSgitFtdcPriceType              AskPrice5;
```

```
    TSgitFtdcVolumeType     AskVolume5;
    TSgitFtdcPriceType      AveragePrice;
};
```

Chapter 5 示例

The attached file is a simple sample to demonstrate the usage of both
SgitFtdcMduserApi and CSgitFtdcMduserSpi in a single application..

```
// SgitApiDemo.cpp :
// A simple sample to demonstrate the usage of SgitFtdcMduserApi and CSgitFtdcMduserSpi.
#include "stdio.h"
#include <string.h>
#include "SgitFtdcMdApi.h"

#include "SgitFtdcTraderApi.h"

#ifdef _DEBUG
#pragma comment(lib,"sgitquotapid.lib ")
#pragma comment(lib,"sgittradeapid.lib ")
#else
#pragma comment(lib,"sgitquotapi.lib ")
#pragma comment(lib,"sgittradeapi.lib ")
#endif

// sample username and password
char gszUserID[] = { "7001" };
char gszPwd[] = { "888888" };

static bool bOrder = false;
static int nRequestId = 0;

// Trade API handler class
class CTradeHandle : public CSgitFtdcTraderSpi
```

```
{
public:
        CTradeHandle(CSgitFtdcTraderApi *pUserMDApi) : m_pUserMDApi(pUserMDApi) {}
        ~CTradeHandle() {}

        void OnFrontConnected()
        {
                CSgitFtdcReqUserLoginField reqUserLogin;
                strcpy(reqUserLogin.TradingDay, m_pUserMDApi->GetTradingDay());
                strcpy(reqUserLogin.BrokerID, "yyy");
                strcpy(reqUserLogin.UserID, gszUserID);
                strcpy(reqUserLogin.Password, gszPwd);
                // send login request here on connect
                int iRet = m_pUserMDApi->ReqUserLogin(&reqUserLogin,nRequestId++);
                printf("login: %d\n",iRet);
        }

        void OnFrontDisconnected(char *pErrMsg)
        {
                printf("OnFrontDisconnected: %s\n",pErrMsg);
        }

        // login response handler
        void OnRspUserLogin(CSgitFtdcRspUserLoginField *pRspUserLogin, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
                printf("OnRspUserLogin:\n");
                printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);
                printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
```

```
                if (pRspInfo->ErrorID == 0)
                {
                        m_pUserMDApi->Ready();
                }
                if (pRspInfo->ErrorID != 0)
                {
                        // something error occured on logon
                        printf("Failed to login, errormsg=%s errorcode=%d requestid=%d chain=%d",pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
                        return;
                }

                // query for instrument on successful logon
                CSgitFtdcQryInstrumentField reqInstrument;
                memset(&reqInstrument,0,sizeof(CSgitFtdcQryInstrumentField));
                if (m_pUserMDApi->ReqQryInstrument(&reqInstrument,nRequestId++) == 0)
                {
                        printf("send query Instrument success.\n");
                }
                else
                {
                        printf("send query Instrument error\n");
                }
        }

        // query instrument response handler
        void OnRspQryInstrument(CSgitFtdcInstrumentField *pInstrument, CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
                printf("OnRspQryInstrument \n");
                //
                if (pInstrument != NULL)
```

```
                {
                        printf("Instrument info: %s,%.2f\n",pInstrument->InstrumentID,pInstrument->PriceTick);
                }

                // demonstrate how to send new order, on the first response of query instrument
                // to make sure only one order is created
                if (bOrder == false)
                {
                        CSgitFtdcInputOrderField ReqOrderAdd;

                        // fill the new order request
                        memset(&ReqOrderAdd,0,sizeof(ReqOrderAdd));
        strncpy(ReqOrderAdd.UserID,"7001",sizeof(ReqOrderAdd.UserID) - 1);
                        ReqOrderAdd.RequestID = 1;
                        const char *pch = "000000000050";
        strncpy(ReqOrderAdd.OrderRef,pch,sizeof(ReqOrderAdd.OrderRef) - 1);
                        ReqOrderAdd.OrderPriceType = Sgit_FTDC_OPT_LimitPrice;
                        sprintf(ReqOrderAdd.CombHedgeFlag,"%s","1");            // 1 for speculation

                        ReqOrderAdd.TimeCondition = Sgit_FTDC_TC_GFD;           // Good For Day
                        ReqOrderAdd.VolumeCondition = Sgit_FTDC_VC_AV;          // Any Volume
                        ReqOrderAdd.MinVolume = 1;
                // minimum match volume
                        ReqOrderAdd.ContingentCondition = Sgit_FTDC_CC_Immediately;     // trigger immediately
                        ReqOrderAdd.ForceCloseReason = Sgit_FTDC_FCC_NotForceClose;     // non force close

                        sprintf(ReqOrderAdd.InvestorID,"%s","30057712");        // sample investor
```

```
                    sprintf(ReqOrderAdd.InstrumentID,"%s","IF1303");          //
sample instrument

                    ReqOrderAdd.Direction = '0';
            // 0 for buy, 1 for sell

                    sprintf(ReqOrderAdd.CombOffsetFlag,"%s","0");
        // 0 for open, 1 for close, 2 for force close, 3 for close new position, 4 for close old
position, 5 for force reduction

                    ReqOrderAdd.LimitPrice = 2090.2;
            // limit price

                    ReqOrderAdd.VolumeTotalOriginal = 1;
        // volume


                    // send the order as the second

                    int          iRet          =          m_pUserMDApi-
>ReqOrderInsert(&ReqOrderAdd,nRequestId++);

                    if (iRet == 0)

                    {

                            printf("ReqOrderInsert success:%d \n",iRet);

                    }

                    else

                    {

                            printf("ReqOrderInsert error:%d \n",iRet);

                    }

                    //

                    bOrder = true;

                }

            }


            // order report handler

            void        OnRtnOrder(CSgitFtdcOrderField    *pOrder,CSgitFtdcRspInfoField
*pRspInfo)

            {

                if (pRspInfo->ErrorID == 0)

                {

                    printf("OnRtnOrder success\n");
```

```
                }
                else
                {
                        printf("OnRtnOrder error:%d - %s\n",pRspInfo->ErrorID,pRspInfo->ErrorMsg);
                }
        }

        // trade report handler
        void OnRtnTrade(CSgitFtdcTradeField *pTrade)
        {
                printf("OnRtnTrade \n");
        }

        // instrument status notification handler
        void OnRtnInstrumentStatus(CSgitFtdcInstrumentStatusField *pInstrumentStatus)
        {
                printf("OnRtnInstrumentStatus: InstrumentID=%s,ExchangeID=%s\n",
                        pInstrumentStatus->InstrumentID,pInstrumentStatus->ExchangeID);
        }

private:
        CSgitFtdcTraderApi *m_pUserMDApi;
};

// market data API handler class
class CSimpleHandler : public CSgitFtdcMdSpi
{
public:

        // constructor, an instance of CSgitFtdcMdApi is a must
```

```
CSimpleHandler(CSgitFtdcMdApi *pUserMDApi) : m_pUserMDApi(pUserMDApi) {}
~CSimpleHandler() {}

void OnFrontConnected() {
        CSgitFtdcReqUserLoginField reqUserLogin;
        strcpy(reqUserLogin.UserID, gszUserID);
        strcpy(reqUserLogin.Password, gszPwd);
        // logon on connect
        m_pUserMDApi->ReqUserLogin(&reqUserLogin, 0);
}

void OnFrontDisconnected() {
        // just print message, reconnect will be done automatically
        printf("OnFrontDisconnected.\n");
}

// login response handler
void OnRspUserLogin(CSgitFtdcRspUserLoginField *pRspUserLogin,
CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {
        printf("OnRspUserLogin:\n");
        printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo-
>ErrorMsg);
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

        if (pRspInfo->ErrorID == 0) {
                m_pUserMDApi->Ready();
        }
        if (pRspInfo->ErrorID != 0) {
                // something error
                printf("Failed to login, errorcode=%d errormsg=%s requestid=%d
chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
        }
}
```

```
	// market data notification
	void OnRtnDepthMarketData(CSgitFtdcDepthMarketDataField *pMarketData)
	{
		// just show the data
		printf("receive contract=%s Volume=%d lastprice=%f\n",pMarketData->InstrumentID,pMarketData->Volume, pMarketData->LastPrice);
	}

	void OnRspError(CSgitFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {
		printf("OnRspError:\n");
		printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);
		printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
	}

private:
	// point to an instance of CSgitFtdcMduserApi
	CSgitFtdcMdApi *m_pUserMDApi;
};

int main()
{
	// create an trader API instance
	CSgitFtdcTraderApi *pUserTradeApi = CSgitFtdcTraderApi::CreateFtdcTraderApi();
	// the handler for trader API
	CTradeHandle th(pUserTradeApi);
	// register the handler
	pUserTradeApi->RegisterSpi(&th);
	pUserTradeApi->SubscribePrivateTopic(Sgit_TERT_RESTART);
	pUserTradeApi->SubscribePublicTopic(Sgit_TERT_RESTART);

	// register the trade server
	pUserTradeApi->RegisterFront("tcp://192.168.1.22:7776");
```

```
// start the trader API
pUserTradeApi->Init(false);

// create an market data API instance
CSgitFtdcMdApi *pUserMDApi = CSgitFtdcMdApi::CreateFtdcMdApi();
// the handler for market data API
CSimpleHandler sh(pUserMDApi);
// register the handler
pUserMDApi->RegisterSpi(&sh);
printf("I am here 2\n");

// subscribe for market topic from the start
/// TERT_RESTART:        from the start
/// TERT_RESUME:         from the last position
/// TERT_QUICK:                  start with a snapshot of all the market data
pUserMDApi-> SubscribeMarketTopic (Sgit_TERT_RESTART);

// register the market data server
pUserMDApi->RegisterFront("tcp://192.168.1.22:7777");

// call this to enable multicast mode
pUserMDApi->SetMultiCastAddr("MlCast://ANY$224.0.1.12:5555");

// start the market data API
pUserMDApi->Init(false);

// join the trader api thread
pUserTradeApi->Join();

// join the market data api thread
pUserMDApi->Join() ;
```

```
        // release the market data API
        pUserMDApi->Release();

        // release the trader API
        pUserTradeApi->Release();

        return 0;
}
```