

**Leve2 FAST API 使用文档**

**研发中心**

**2014年2月**

目录

[1、接口说明 3](#_Toc380572562)

[1.1 Decoder\_Factory.h 3](#_Toc380572563)

[1.2 Fast\_Decoder.h 3](#_Toc380572564)

[1.2.1 类 IFast\_Decoder 3](#_Toc380572565)

[1.2.2 类 IFast\_Message 3](#_Toc380572566)

[1.2.3 类 IFast\_Record 4](#_Toc380572567)

[2、DEMO 示例 5](#_Toc380572568)

[2.1 win调用库 5](#_Toc380572569)

[2.2 linux调用库 6](#_Toc380572570)

[2.3 解释FAST数据 6](#_Toc380572571)

[2.4 取出记录的字段 7](#_Toc380572572)

[2.5 SH\_Level2\_Tag.h 与 SZ\_Level2\_Tag.h 8](#_Toc380572573)

[3、DEMO测试结果 9](#_Toc380572574)

# 1、接口说明

## 1.1 Decoder\_Factory.h

1）virtual IFast\_Decoder\* FUNCTION\_CALL\_MODE GetFastDecoder (FAST\_TYPE ft) = 0 ;

获取FAST解析器 IFast\_Decoder\*，参数为市场类型，emFAST\_SHLEVEL2 上海市场 , emFAST\_SZLEVEL2 深圳市场 返回一个FAST解释器。

## 1.2 Fast\_Decoder.h

### 1.2.1 类 IFast\_Decoder

从1.1 1）的函数中得到一个 IFast\_Decoder 类对像指针。

1）virtual int FUNCTION\_CALL\_MODE LoadTemplate (const char\* lpFile) = 0 ;

先载入FAST模板，上海市场则载入上海模板、深圳市场刚载入深圳市场模板。

2）virtual int FUNCTION\_CALL\_MODE GetFastMsgLen (const char\* lpData, int nLen) = 0 ;

通过一个原始数据得到获取一条完整FAST消息的长度。

3）virtual IFast\_Message\* FUNCTION\_CALL\_MODE Decode (const char\* lpData, int nLen) = 0 ;

解释一条完整的FAST消息，返回FAST消息指针 IFast\_Message\*

4）virtual const FastField\_Info\* FUNCTION\_CALL\_MODE GetFieldInfo (int nTagID) = 0 ;

通过nTagID 字段ID 获取某FAST字段的信息，必须是在解释一条完整的FAST消息后。

5）virtual const char\* FUNCTION\_CALL\_MODE GetLastError () = 0 ;

获取最后的错误信息

6）virtual void FUNCTION\_CALL\_MODE Release () = 0 ;

使用完FAST解释器后，释放解释器。

### 1.2.2 类 IFast\_Message

从1.2.1 3）的函数一条完整的FAST消息，得到一个 IFast\_Message类对像指针。

1）virtual const char\* FUNCTION\_CALL\_MODE GetMsgType () = 0 ;

获取一个消息的类型，得到一个解释后的FAST 先通过这函数来得到FAST消息的类型然后通过类型进行不同的解释处理。

2）virtual const char\* FUNCTION\_CALL\_MODE GetSendTime () = 0 ;

获取FAST消息的发送时间。

3）virtual int FUNCTION\_CALL\_MODE GetSeqNum () = 0 ;

获取FAST消息的发送序号。

4）virtual int FUNCTION\_CALL\_MODE GetRecordCount () = 0 ;

获取FAST记录的个数，由于每一个FAST消息里可能包含多个记录，所以先用这函数来获取记录个数。

5）virtual IFast\_Record\* FUNCTION\_CALL\_MODE GetRecord (int nSlot) = 0 ;

获取FAST消息的记录，参数为记录的下标。通过4）得到的记录个RecordCount 参数 nSlot 从0 到 RecordCount-1 得到每一条记录。

6）virtual const char\* FUNCTION\_CALL\_MODE GetLastError () = 0 ;

获取最后的错误信息

7）virtual void FUNCTION\_CALL\_MODE Release () = 0 ;

使用完后，释放FAST消息。

### 1.2.3 类 IFast\_Record

从1.2.5 5）的函数一条完整的FAST记录，得到一个 IFast\_Record

类对像指针。

1）virtual uint8 FUNCTION\_CALL\_MODE GetInt8 (int nTagID) = 0 ;

获取记录中的8-bit的字段值，参数 nTagID 为字段的ID号。

2）virtual uint16 FUNCTION\_CALL\_MODE GetInt16 (int nTagID) = 0 ;

获取记录中的16-bit的字段值，参数 nTagID 为字段的ID号。

3）virtual uint32 FUNCTION\_CALL\_MODE GetInt32 (int nTagID) = 0 ;

获取记录中的32-bit的字段值，参数 nTagID 为字段的ID号。

4）virtual uint64 FUNCTION\_CALL\_MODE GetInt64 (int nTagID) = 0 ;

获取记录中的64-bit的字段值，参数 nTagID 为字段的ID号。

5）virtual const char\* FUNCTION\_CALL\_MODE GetString (int nTagID) = 0 ;

获取记录中的 String 的字段值，参数 nTagID 为字段的ID号。

6）virtual bool FUNCTION\_CALL\_MODE IsExist(int nTagID) = 0 ;

判断某个字段存不存在。

7）virtual int FUNCTION\_CALL\_MODE GetSubCount (int nTagID) = 0 ;

如果记录里嵌套着一个 sequence 记录，则通过sequence 的ID　来得到sequence的记录数。

8）virtual IFast\_Record\* FUNCTION\_CALL\_MODE GetSubRecord (int nTagID, int nSlot) = 0 ;

通过　７）　得到sequence的记录数，通过sequence ID号（nTagID）和记录的下标(nSlot) 下标为 0 – GetSubCount()-1，得到每一条sequence 记录指针 IFast\_Record\* ，通过字段类型的字段ID得到值。

# 2、DEMO 示例

## 2.1 win调用库

IDecoder\_Factory\* lpFactory = GetDecoderFactoryInstance();

//从解释工厂得到解释器

IFast\_Decoder\* m\_Decoder = lpFactory->GetFastDecoder(IDecoder\_Factory::emFAST\_SHLEVEL2) ; //上海市场

//加载模板 上海市场模板

m\_Decoder->LoadTemplate(“template.xml”);

（参见 Level2\_Decoder\_Demo.cpp Init() 函数）

用VS工程编译要将FAST\_API.lib链接进来（参考示例VS工程）

## 2.2 linux调用库

IDecoder\_Factory\* lpFactory = GetDecoderFactoryInstance();

//linux 直接得到解释工厂

//从解释工厂得到解释器

IFast\_Decoder\* m\_Decoder = lpFactory->GetFastDecoder(IDecoder\_Factory::emFAST\_SHLEVEL2) ; //上海市场

//加载模板 上海市场模板

m\_Decoder->LoadTemplate(“template.xml”);

（参见 Level2\_Decoder\_Demo.cpp Init() 函数）

（编译运行参考示例的makefile 和 rundemo脚本）

## 2.3 解释FAST数据

先获取一个完整的Fast包，参考代码实现。

//解释FAST数据得到一个Fast\_Message 消息

IFast\_Message \*FastMsg=m\_Decoder->Decode(buff,length); //buff 为FAST数据 length为度

//得到消息的类型

const char \* strType=FastMsg->GetMsgType();

//判断消息类型进行不同处理

if( strType==0 )

{

FastMsg->Release();

return;

}

else if (!strcmp (strType, "UA3201")) //处理 UA3201的消息 逐笔成交

{

//得到FAST 消息里Record的数量

int count=FastMsg->GetRecordCount();

std::cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<std::endl ;

std::cout<<"MsgType:"<<FastMsg->GetMsgType ()<<" SendTime:"<<FastMsg->GetSendTime ()

<<" SeqNum:"<<FastMsg->GetSeqNum ()<<"RecordCount:"<<count<<std::endl ;

//输出消息时间、消息类型、消息序号、消息记录数

for (int i=0;i<count;i++)

{

//得到每一条Record

IFast\_Record \*Recoder=FastMsg->GetRecord(i);

if (!Recoder)

{

FastMsg->Release();

return;

}

//处理Record

Deal\_SH\_NGTSTransaction(Recoder);

}

}

（参见 Level2\_Decoder\_Demo.cpp Deal\_SH\_Level2\_Data(char \*buff,int length) 函数）

## 2.4 取出记录的字段

上海在demo中已实现 逐笔成交、行情快照、指数行情。

深圳在demo中已实现 指数行情 、逐笔委托、逐笔成交 。

（参见 Level2\_Decoder\_Demo.cpp 中其实现的函数）

下面以 上海市场 行情快照 为例说明，具体参见demo的实现。

const char\* temp;

int temp\_int32=0;

long long temp\_int64=0;

//nTagID 参照SH\_Level2\_Tag.h 对应的注释模板和宏

//SH\_DataTimestamp 为 时间时间戳 ID 号的宏定义

//根据模板 对不同类型 进行不同类型的取值 如 int32 用 GetInt32

//int 64 用 GetInt64 string 用GetString

temp\_int32=Record->GetInt32(SH\_DataTimestamp); //获取DataTimestamp

temp\_int32=Record->GetInt32(SH\_DataStatus); //获取DataStatus

temp=Record->GetString(SH\_SecurityID); //获取SecurityID

.

.

.

.

.

temp\_int64=Record->GetInt64(SH\_TotalVolumeTrade); //获取TotalVolumeTrade

temp\_int64=Record->GetInt64(SH\_TotalValueTrade); //获取TotalValueTrade

.

.

.

///存在一个 sequence

// SH\_NoBidLevel 为 sequence 的 ID

int subCount1=Record->GetSubCount(SH\_NoBidLevel); //得到sequence的记录数

for (int i=0;i<subCount1;i++)

{

//通过sequence ID 和 记录 下标 得到每一条 sequence记录

IFast\_Record \*subRecord=Record->GetSubRecord(SH\_NoBidLevel,i);

if (!subRecord)

{

return;

}

//获取 sequence 里 PriceLeveOperator 字段

temp\_int32=subRecord->GetInt32(SH\_PriceLeveOperator);

.

.

// sequence里还存在一个sequence

//通过这个sequence ID号得到 子sequence 的 记录数

int tsubCount=subRecord->GetSubCount(SH\_Orders);

for (int j=0;j<tsubCount;j++)

{

//通过sequence ID 和 记录 下标 得到每一条 sequence记录

IFast\_Record \*tsubRecord=subRecord->GetSubRecord(SH\_Orders,j);

if (!tsubRecord)

{

return;

}

//获取 sequence 里 OrderQueueOperator 字段

temp\_int32=subRecord->GetInt32(SH\_OrderQueueOperator);

.

.

}

}

//其他的sequence 记录

int subCount2=Record->GetSubCount(SH\_NoOfferLevel);

for (int i=0;i<subCount2;i++)

{

IFast\_Record \*subRecord=Record->GetSubRecord(SH\_NoBidLevel,i);

.

.

.

.

}

## 2.5 SH\_Level2\_Tag.h 与 SZ\_Level2\_Tag.h

这两个文件中分别定义了 上海市场、深圳市场 每个字段ID号对应的宏定义，以及每一个模板每个字段ID号对应的宏定义。

例如：

//逐笔成交　<template name="NGTSTransaction" id="3201">

/\*\*\*

<string name="MessageType" id="35" SH\_MessageType

<int32 name="DataStatus" id="10121" SH\_DataStatus

<int32 name="TradeIndex" id="10011" SH\_TradeIndex

<int32 name="TradeChannel" id="10115" SH\_TradeChannel

<string name="SecurityID" id="48" SH\_SecurityID

<int32 name="TradeTime" id="10013" SH\_TradeTime

<int32 name="TradePrice" id="10014" SH\_TradePrice

<int64 name="TradeQty" id="10015" SH\_TradeQty

<int64 name="TradeMoney" id="10016" SH\_TradeMoney

\*\*\*/

<int32 name="DataStatus" id="10121" SH\_DataStatus

10121 为 字段 DataStatus ID号 而宏定义

#define SH\_DataStatus 10121

# 3、DEMO测试结果

上海市场

1. 测试过程 连接上海的FAST数据转发源得到原始的FAST数据。
2. 通过FAST解释得到对应的消息

逐笔的结果

