



FT/API

Programmers Guide

Version 3.6.2

Table of Contents

1	INTRODUCTION	9
1.1	Multi-Thread Programming	9
1.2	Deprecated elements	10
1.3	Discontinued API Functions	10
2	RETRIEVING LIBRARY INFORMATION	11
2.1	FTGetLibraryVersion.....	11
2.2	FTGetErrorString	11
3	SETTING LIBRARY TRACE.....	12
3.1	FTTraceLevelEnum	12
3.2	FTTraceSourceEnum.....	12
3.3	FTLibraryTraceProc	13
3.4	FTGetTraceInfo	13
3.5	FTSetTraceMode.....	13
3.6	FTTraceAppl	15
4	INITIALIZATION AND TERMINATION.....	17
4.1	Access keys.....	18
4.2	FTInitExt	19
4.3	FTExtendClass	19
4.4	FTExtendClassByLibrary	20
4.5	FTStart	21
4.6	FTShutdown	21
4.7	FTRun.....	21
4.8	FTConversationSocketSelect	22
4.9	FTSocketSelect.....	22
4.10	FTEnumConversationSocket	23
4.11	FTSetSelectTimeout(long Sec, long uSec);	24
5	CONVERSATIONS	25
5.1	Common Conversation Types	25
5.1.1	FTUserTypeEnum.....	25
5.1.2	FTRevision	26
5.1.3	FTOpenResStruct.....	26
5.1.4	FTOpenConversResProc	27
5.1.5	FTBrokenTypeEnum	28
5.1.6	FTNotifyBrokenProc.....	28
5.1.7	FTConversationProc	29
5.2	FTContext.....	29
5.2.1	FTCreateContext.....	29
5.2.2	FTSetContextAttribute	30
5.2.3	FTDestroyContext	31
5.3	Conversation management.....	31
5.3.1	FTOpenConversationExt.....	32
5.3.2	FTCloseConversation	34
5.3.3	FTGetConversationSocket	34
5.3.4	FTEnumConversation	35
6	SUBSCRIPTIONS	36
6.1	Common Subscription Types	36
6.1.1	FTComunicationModeEnum	36

6.1.2	FTSubscribeFlowEnum	37
6.1.3	FTTimeStamp	38
6.1.4	FTQueryTypeEnum	38
6.1.5	FTKeyValue	39
6.1.6	FTMakeKeyValue	39
6.1.7	FTSubscribeStatusEnum	40
6.1.8	FTSubscribeResStruct	40
6.1.9	FTEventTypeEnum	41
6.1.10	FTClassActionEnum	42
6.1.11	FTNotifySubscribeResStruct	42
6.1.12	FTSubscribeResProcExt	43
6.1.13	FTNotifySubscribeResProcExt	44
6.2	Masks	44
6.2.1	FTCreateMask	45
6.2.2	FTAddFieldToMask	45
6.2.3	FTDestroyMask	46
6.3	Filters	47
6.3.1	FTFilterCreateEnum	47
6.3.2	FTFilterCreateProc	48
6.3.3	FTFilterCreate	48
6.3.4	FTFilterSetEnum	49
6.3.5	FTFilterSetProc	50
6.3.6	FTFilterSet	50
6.3.7	FTFilterDestroyEnum	51
6.3.8	FTFilterDestroyProc	51
6.3.9	FTFilterDestroy	52
6.4	DiffMasked Subscription	52
6.4.1	FTFiledChanged	53
6.4.2	FTUpdateMarketObject	53
6.5	Subscription Management	53
6.5.1	FTStartSubscribeClassExt	53
6.5.2	FTStopSubscribeClass	56
6.5.3	FTRefreshEntity	56
6.6	Subscriptions Usage	57
6.6.1	Incremental Subscriptions	57
6.6.2	Partial Subscriptions	58
7	TRANSACTIONS	62
7.1	Transaction sending	62
7.1.1	FTTransactionID	62
7.1.2	FTMakeTransactionID	63
7.1.3	FTMakeTransactionIDExt	63
7.1.4	FTSendTransactionStatusEnum	63
7.1.5	FTSendTransactionResStruct	64
7.1.6	FTSendTransactionResProcExt	65
7.1.7	FTSendTransactionExt	66
7.2	Transaction Monitoring	67
7.2.1	FTTransStatusEnum	67
7.2.2	FTTransactionStateStruct	68
7.2.3	FTTransactionMonitorProc	69
7.2.4	FTStartTransactionMonitoring	69
7.2.5	FTQueryTransStatusEnum	70
7.2.6	FTQueryTransStatusStruct	70
7.2.7	FTQueryTransStatusProc	71
7.2.8	FTQueryTransactionStatus	72

8	QUERIES	74
8.1	Query Create	74
8.1.1	FTQueryCreateEnum	74
8.1.2	FTQueryCreateResStruct	75
8.1.3	FTQueryCreateProc	76
8.1.4	FTQueryNotifyStruct	76
8.1.5	FTQueryNotifyProc	77
8.1.6	FTQueryCreate	78
8.2	Query Rows	79
8.2.1	FTQueryRowsEnum	79
8.2.2	FTQueryRowsProc	79
8.2.3	FTQueryRows	80
8.3	Query Destroy	81
8.3.1	FTQueryDestroyEnum	81
8.3.2	FTQueryDestroyProc	81
8.3.3	FTQueryDestroy	82
APPENDIX A:	HEADER FILE FTAPI.H	84
APPENDIX B:	FASTTRACK EXAMPLE: GET ORDER	94
APPENDIX C:	FASTTRACK EXAMPLE: SEND ORDER	101
APPENDIX D:	NAM MARKETS EXAMPLE: GET FILL	113
APPENDIX E:	NAM MARKETS EXAMPLE: SEND PROPOSAL	124
APPENDIX F:	FT/API STATES	137
APPENDIX G:	EXAMPLE OF A CONNECTION TO ASIA	138
	TO CONTACT US	145

Revision History

FT/API version	Changes
3.0.5 March 2003	<ul style="list-style-type: none"> New function <i>FTAddFieldToMask</i> superseded <i>FTMaskAddField</i>. The new function is presented with examples of valid and invalid masked fields. New values <i>FTZeroMaskedFlowLast</i> and <i>FTZeroMaskedFlowAll</i> of <i>FTSubscribeFlowEnum</i>. New field <i>IsMasked</i> of <i>FTNotifySubscribeResStruct</i>. Added some clarification about the choice of an appropriate transaction query function (<i>FTStartTransactionMonitoring</i> vs <i>FTQueryTransactionStatus</i>) depending on the type of the market (NAM market with OMI protocol vs FastTrack service/market with FastTrack protocol).
3.0.6 April 2003	<ul style="list-style-type: none"> Added some clarification about the possible values of some <i>FTStartSubscribeClassExt</i> parameters when the subscription is made on NAM market connected with OMI protocol.
3.0.9 August 2003	<ul style="list-style-type: none"> Documented additional return values for <i>FTQueryCreate</i>. Extended the <i>FTQueryNotifyStruct</i>. Added two more example programs in Appendices E and F to cover NAM markets in addition to FastTrack markets. Added a section on multi-threading programming.
3.1.0 October 2003	<ul style="list-style-type: none"> Tracing features enhanced: <ul style="list-style-type: none"> Added <i>FTTraceAppl</i> to write applicative trace events in the LOGS directory; <i>FTTraceSourceEnum</i> extended with <i>FTSource_Application</i>; Trace options extended with <i>FTTRACE_OPT_Application</i>. Added <i>FTExtendClassByLibrary</i> in addition/alternative to <i>FTExtendClass</i>. Added <i>CTX_CONNECTION_TIMEOUT</i> parameter to <i>FTContext</i>. Added <i>NAMError</i> field to <i>FTTransactionStateStruct</i>. Minor spelling corrections.
3.1.1 November 2003	<ul style="list-style-type: none"> Updated <i>FTExtendClassByLibrary</i>
3.1.5 June 2004	<ul style="list-style-type: none"> APPENDIX F: FT/API STATES updated: removed the arrow from <i>FTShutdown</i> to <i>FTStart</i>. Clarified that an <i>FTOpenConversationExt</i> may successfully return <i>n==0</i> (i.e. <i>FTOK</i>) or <i>n>0</i>. Added some notes about the use of <i>Service</i> parameter of <i>FTOpenConversationExt</i>.
3.1.7 October 2004	<ul style="list-style-type: none"> Conversation attribute: <i>CTX_CONNECTION_PROXYSTRING</i> added.
3.2.1 January 2006	<ul style="list-style-type: none"> Defines: <i>FT_OFFSET</i>, <i>CTX_INIT_MAX_BLOCK_LEN</i>, added. <i>FTSubscribeFlowEnum</i> extended with <i>FTDiffMaskedFlowAll</i>. Type <i>FTDiffMask</i> added. <i>FTNotifySubscribeResStruct</i> extended with <i>DiffMask</i>. Functions: <i>FTInitExt</i>, <i>FTIsFieldChanged</i>, <i>FTUpdateMarketObject</i> added.
3.2.2 April 2006 (10 th)	<ul style="list-style-type: none"> Conversation attribute <i>CTX_CONNECTION_NEWPASSWORD</i> added.
3.2.3 April 2006 (13 th)	Error results added: <i>FTAccountNotActive</i> , <i>FTTooManyTradersConnected</i> , <i>FTPasswordExpired</i> , <i>FTNotPrivilegeChangePassword</i> , <i>FTNewPasswordRepeated</i> , <i>FTInsufficientNewPasswordLength</i> , <i>FTInvalidNewPasswordCharacters</i> , <i>FTNewPasswordTooMuchEasy</i> .
3.2.6 July 2007	General Revision
3.2.7 December 2007	Updated email's addresses
3.2.8 April 2008	<i>FTQueryNotifyStruct</i> : added field "Delete"
3.2.10 December 2010	General Revision
3.2.11 November 2011	No new features. Aligned to the version 3.2.11 of the library

3.3.0 March 2013	Fulfilment for the connection to the ASIA platform
3.3.1 April 2013	No new features
3.3.2 April 2013	Added the FTEnumConversationSocket function
3.4.0 September 2013	It is no longer required to applicatively lock functions FTSendTransactionExt / FTMakeTransactionID when sending transactions on FT_C_ORDER/FT_C_QUOTE in a multithread FTApi program.
3.5.1 January 2014	Added x64 (64 bit) support for MS-Windows (MT9_64). Visual Studio 9 is the minimum requirement to build applications with this architecture. The following functions are now *discontinued*: FTSetLicence, FTLoadLicence, FTInit, FTOpenConversation, FTStartSubscribeClass and FTSendTransaction
3.6.1 June 2015	The SUNXV9_64 architecture support was included
3.6.2 July 2015	No new features

1 INTRODUCTION

This manual describes version 3.6.2 of **FT/API**, the Application Program Interface developed by LIST within FastTrack for access to electronic markets and FastTrack services. The same API is likewise usable to access NAM markets connected with the OMI protocol.

The data structure and the functions of FT/API interface are described, along with the main concepts regarding access to the FastTrack server: connection, data subscription, and submission of transactions.

The implementation of FT/API functionalities is based on an **asynchronous communication model**. Functionality (such as connection opening or a subscription for a class of data) is requested to the library via a C function. This request specifies, among other parameters, some **notification functions**, defined by the user, which will be called by the library when data arrive or when other events occur.

The appendices contain a couple of FT/API examples programs to access both FastTrack markets and NAM markets.

Note ASIA:

Applications using FTApi are now able to connect to a FastTrack platform that supports ASIA technology. With version 3.6.2 of the FTApi, users can exploit both ASIA and non-ASIA connections (R2 and R3) within the same application, without worrying about the management of the dual access connection, public and private, typical of ASIA. The dual ASIA connection is presented in terms of a normal conversation, so that users can use the conversation-Id obtained during the connection and achieve their needs with conventional FTApi methods. To route to the appropriate access, FTApi uses a text file that contains the rules by which a given class-id is sent via one connection rather than on the other one (eg: PUBLMETAMARKET-PRIVMETAMARKET.routing.properties). An ASIA connection can be made via FTApi exclusively by the service name and never directly with an IP-address and a port. The service name string has two tokens "FIRSTSERV|SECONDSERV" separated by a pipe. In the working directory of the user's application, there must be a file named *FIRSTSERV-SECONDSERV.routing.properties* that defines class-id routing rules.

For example, the connection to the MetaMarket service can be done setting the "PUBLMETAMARKET|PRIVMETAMARKET" string to the service name param of the FTOpenConversationExt().

1.1 Multi-Thread Programming

The current version **FT/API** library is not thread-safe: different functions of FT/API cannot be concurrently called, from different threads, in the same process.

In a multi-thread process:

- all FT/API function must be called in the same thread,
- or different calls, in different threads, must all be synchronized in order to guarantee a Complete Mutual Exclusion.

Note: It is no longer required to applicatively lock functions `FTSendTransactionExt` / `FTMakeTransactionID` when sending transactions on `FT_C_ORDER`/`FT_C_QUOTE` in a multithread `FTApi` program.

1.2 Deprecated elements

Interfaces described in this manual refer to January 2014 version of **FT/API**.

In this version there are some **deprecated** elements (values, fields, types, prototypes, functions, etc...) that have been outdated by newer constructs. These deprecated elements are clearly marked as

deprecated.

Deprecated elements may become obsolete in future versions of this library. This library specification indicates alternatives in order to avoid the use of deprecated elements.

1.3 Discontinued API Functions

Interfaces described in this manual refer to January 2014 version of **FT/API**.

In this version there are some **discontinued** elements (values, fields, types, prototypes, functions, etc...) that have been outdated by newer constructs.

The use of any of these functions will result in an error:

`FTSupersededOperation (-39L)`.

The current list of discontinued functions (with the corresponding replacements) is given in the following table:

Discontinued functions	Fonctions to be used
FTSetLicence	Use the concept of <code>CTX_APPL_AUTHKEY</code> in <code>FTContext</code> .
FTLoadLicence	Use the concept of <code>CTX_APPL_AUTHFILE</code> in <code>FTContext</code> .
FTOpenConversation	FTOpenConversationExt
FTStartSubscribeClass	FTStartSubscribeClassExt
FTSendTransaction	FTSendTransactionExt
FTInit	FTInitExt

2 RETRIEVING LIBRARY INFORMATION

FT/API supplies the following general functions:

- **FTGetLibraryVersion**
- **FTGetErrorString**

2.1 FTGetLibraryVersion

The function:

```
char *FTGetLibraryVersion(void);
```

returns a pointer to the string containing the version of the library in use.

2.2 FTGetErrorString

The function:

```
char *FTGetErrorString(long ErrCode);
```

returns a pointer to the string containing the description of the error of code *ErrCode*. The parameter *ErrCode* can be the long value returned by a function of the library, or the one contained in a notification structure.

3 SETTING LIBRARY TRACE

By default the library notifies any exception event (an internal warning or error) occurring during a session by calling the user procedure passed as argument in the *FTInitExt* function (see chapter 4: *Initialization and Termination*).

The *FTSetTraceMode* function can be used at the beginning of the application in order to customize the library's tracing mode, by specifying the **level of detail** and the volume of the generated messages, the **categories** of events of interest, and if or not to automatically generate **dated log files**.

This chapter shows how to use these functions:

- **FTSetTraceMode**
- **FTGetTraceInfo**
- **FTTraceAppl**

3.1 FTTraceLevelEnum

The Enumerated type:

```
typedef enum {  
    FTLevel_Undef,  
    FTLevel_Full,  
    FTLevel_Normal,  
    FTLevel_Warning,  
    FTLevel_Error  
} FTTraceLevelEnum;
```

represents the possible trace message levels.

3.2 FTTraceSourceEnum

The Enumerated types:

```
typedef enum {  
    FTTraceLevel_Undef,  
    FTSource_FTP,  
    FTSource_API,  
    FTSource_FlowControl,  
    FTSource_DataTransfert,
```

```
    FTSource_Application
} FTTraceSourceEnum;
```

represents the trace message categories.

3.3 FTLibraryTraceProc

The user trace call-back function type is:

```
typedef void (*FTLibraryTraceProc)(
    int          EventLevel,
    char*        String);
```

Remarks

This prototype function *FTLibraryTraceProc* will be automatically invoked only if *FTSetTraceMode* or *FTInitExt* returned *FTOK*.

3.4 FTGetTraceInfo

The function:

```
long FTGetTraceInfo (
    long          EventInfo,
    FTTraceSourceEnum * Source,
    FTTraceLevelEnum * Level);
```

can be used in the user trace call-back, in order to retrieve the source category and the notification level of the incoming message. The parameter *EventInfo* has to be set to the received value. An example of user trace call-back is given in the next section.

3.5 FTSetTraceMode

The function:

```
long FTSetTraceMode (
    FTLibraryTraceProc TraceProc,
    FTTraceLevelEnum   Level,
    long               Options
    int                WriteFile,
```

```

int          OldLog,
int          Flush);

```

can be used optionally in order to customize the library's tracing mode.

Parameters

TraceProc	The user call-back. This value overlaps, if present, the one specified in <i>FTInit</i> . Using a <i>NULL</i> value no trace call-back will be called.
Level	This is the minimum FT/API message level that you want to receive. Using the value <i>FTLevel_Full</i> you'll receive all the messages, using <i>FTLevel_Error</i> you'll receive only the error events.
Options	Set this value to the logical OR of the wanted trace options: <ul style="list-style-type: none"> - <i>FTTRACE_OPT_None</i>: No Option. - <i>FTTRACE_OPT_APIInOut</i>: Every call and/or result will be traced. - <i>FTTRACE_OPT_FlowControl</i>: The application Flow Control will be traced. - <i>FTTRACE_OPT_DataTransfert</i>: The application communication will be traced. - <i>FTTRACE_OPT_DataIn</i>: The incoming records (subscription data) will be dumped. - <i>FTTRACE_OPT_DataOut</i>: the sent record (transaction data) will be dumped. - <i>FTTRACE_OPT_Application</i>: messages passed to <i>FTTraceAppl</i> will be dumped.
WriteFile	Set to <i>FTTrue</i> in order to generate automatic dated log files in the LOGS subdirectory of the current working directory. The LOGS directory will be created.
OldLogs	This number indicates the number of old daily log kept in the LOGS subdirectory. Unused if <i>WriteFile</i> is <i>FTFalse</i> .
Flush	This number indicates the file flushing interval, expressed in number of log lines. Unused if <i>WriteFile</i> is <i>FTFalse</i> .

Returned Value

FTOK	Trace mode successfully set.
-------------	------------------------------

Remarks

You can use this function in your application in order to customize (set or change) the tracing mode. All the subsequent library calls and events will be traced on the basis of this setting.

The trace options *FTTRACE_OPT_FlowControl* and *FTTRACE_OPT_DataTransfert* (that generate events of categories: *FTSource_FlowControl* and *FTSource_DataTransfert*) are mainly intended to be used by the LIST helpdesk staff, because the events traced regard aspects of strict internal implementation.

Using the *WriteFile* mode, a subdirectory *LOGS* will be created in the application's working directory, and several log files will be produced. In any case the generated trace messages will be notified to the user call-back as well.

It's possible to avoid excessive user call-back load, by setting to *NULL* the user call-back *TraceProc*, or by filtering the incoming messages on the basis of the level, for example processing only the messages of high level (*FTLevel_Error*). See the example in the following of this section.

Example

The Following program fragment sets a trace mode including FT/API function calls and results, incoming entities dump and outgoing entities dump. The creation of a log file is required, the log file permanence is set to two days, and the generated file is flushed every 3 lines.

```
static void TraceProc (long eventInfo, char *string)
{
    FTTraceSourceEnum source = 0;
    FTTraceLevelEnum level = 0;
    int res = FTGetTraceInfo (eventInfo, &source, &level);

    if (level < FTLevel_Warning)
        return;

    DebugMsg("***FTTRACE** %d %d - %s", source, level, string);
    /* manage exception */
}

/* ... */

int main(int argc, char *argv[])
{
    long res = -1;
    int opt = FTTRACE_OPT_None;

    opt = opt | FTTRACE_OPT_APIInOut | FTTRACE_OPT_DataIn
           | FTTRACE_OPT_DataOut;

    res = FTSetTraceMode(TraceProc, FTLevel_Normal, opt, FTTrue, 2, 3);

    if (res != FTOK)
        printf("FTTrace not active");

    /* ... rest of the program */
}
```

The required detail level is *FTLevel_Normal*, no FT/API message of level *FTLevel_Full* will be generated. The defined user call-back *TraceProc* skips every Normal or Full level trace message.

3.6 FTTraceAppl

The function:

```
void FTTraceAppl (    FTTraceLevelEnum    Level,
                     char *                Fmt,
                     ...);
```

can be used to write applicative trace events (i.e. messages) in the LOGS directory and/or to pass them to the *FTLibraryTraceProc*.

Trace events generated by this call will have a *FTTraceSourceEnum* equal to *FTSource_Application*.

Parameters

Level	Trace message-level associated to the trace message.
Fmt	Trace message format: like to the first argument of <i>printf</i> function.
...	List of zero, one or more parameters to be formatted according to <i>fmt</i> in order to create the trace message.

4 INITIALIZATION AND TERMINATION

The FastTrack library has to be initialized. In particular:

1. The set of **licence keys** provided by LIST has to be passed to the library and
2. The data structure of the markets that are going to be connected must be specified to the library, so as to enable it to format messages exchanged with them.

The functions described in this chapter are:

- **FTInitExt**
- **FTEndClass**
- **FTEndClassByLibrary**
- **FTStart**
- **FTRun**
- **FTShutdown**

4.1 Access keys

An **access key** is an alphanumeric string, in XML notation. It allows the client that uses the library to connect to a server.

An example of a simple access key is the following multiline string:

```
<List_License>
  <License>
    <FTApi>
      <FTID> 1007 </FTID>
      <BusinessServiceID> 100 </BusinessServiceID>
      <ClientServiceID> 87 </ClientServiceID>
      <ExpiryYear> 2003 </ExpiryYear>
      <ExpiryMonth> 1 </ExpiryMonth>
      <ExpiryDay> 1 </ExpiryDay>
      <Transaction> ON </Transaction>
    </FTApi>
    <FTApi>
      <FTID> 1008 </FTID>
      <BusinessServiceID> 101 </BusinessServiceID>
      <ClientServiceID> 88 </ClientServiceID>
      <ExpiryYear> 2003 </ExpiryYear>
      <ExpiryMonth> 1 </ExpiryMonth>
      <ExpiryDay> 1 </ExpiryDay>
      <Transaction> OFF </Transaction>
    </FTApi>
    <FTApi_NAM>
      <ABICode> 21095 </ABICode>
      <Market> EBM </Market>
      <MarketInfo> NLD </MarketInfo>
      <Cash> OFF </Cash>
      <Strip> ON </Strip>
      <Readonly> OFF </Readonly>
      <ExpiryYear> 2003 </ExpiryYear>
      <ExpiryMonth> 1 </ExpiryMonth>
    </FTApi_NAM>
  </License>
  <Sign>
    6192fef7c2de3503f2c3dadf68275404
    9c9d8a15f042e4070264b92287de2b99
    d61e2434ecac79c6bdd18e96c776b72d
    89de704fa089c2e0d34c79c8b3fb36ec
    48c5f046ca4b8df3ffe4b3c8ec2d8dd5
    76a0f6c41c5d6e920450a1af2dd8003f
    8c563ff12844302f9b7dfaa53097dde1
    689fa4bb2521316b23069cb31974b4d4
    32c9703aa4e298dcd557abecbc310feb
    91896df4962743c99539f34ff786491d
    c641b4ece3b07d645e88587649299038
    4a73fc4783723fea9cf9e4329ace5f45
    60e28b78c8163124bdf93b34672d84c2
    80593974da9d4b4e70d298276e3dd0ea
    4326bfcf82cfbdb0d6807906872966d1
    a846c6588550e64eb3c4cb892191d5e3
  </Sign>
</List_License>
```

The example shows an access-key for 3 licenses.

The access keys can be set directly in the library by using the the concepts of CTX_APPL_AUTHKEY or CTX_APPL_AUTHFILE of FTContext.

Remarks

The library can operate in some test environments **without the use of any licence key**. Licence keys can in any case be used in test environments in order to enable the profiling connected to some market classes (eg to filter the Strips bonds alone).

4.2 FTInitExt

Is the extended version of *FTInit*:

```
long FTInitExt( FTLibraryTraceProc TraceProc,  
               FTContext Context);
```

The additional *Context* parameter allows to set special attributes to the whole session.

4.3 FTEndClass

The function:

```
long FTEndClass(void *Sk);
```

extends the number of the market classes that can be manipulated by the FTAPI library with the skeleton¹ pointed by *Sk*.

In particular, *FTEndClass* must be called after *FTInitExt* once for each market or service skeleton which is referenced in the rest of the program.

To facilitate the FastTrack libraries are equipped with several market/service² include files³ and static libraries⁴ each containing:

- **Include file:** C declarations of the market/service structures;
- **Include file and static library:** a function that returns the market/service skeleton.

¹ A skeleton is a structure containing the description of the data structures of the market/service classes. The developer can avoid having to go into detail in the implementation of the skeleton structure, by assuming that it is the equivalent run-time of the information contained in a h file.

² It is thus important that the market library version used is the same as or compatible with the one of the Server accessed.

³ e.g. in Windows "ftmetamarket.h" for the MetaMarket:

this file contains the declaration of the function

```
void *InitSkeletonmetamarket(void);
```

that must be evaluated and passed as actual argument to *FTEndClass*.

⁴ e.g. in Windows "ftmetamarket.lib" for the MetaMarket.

This library contains the body of *InitSkeletonmetamarket*.

Static libraries must be statically linked at compile time.

Returned Value

FTOK	Extension completed.
FTInitError	Library not initialized.

4.4 FTEndClassByLibrary

The function:

```
long FTEndClassByLibrary(char *      Path,
                        char *      LibraryName,
                        char **      LibraryVersion);
```

extends the number of the market classes (that can be manipulated by the FTAPI library) with the skeleton available in the shared⁵ library *LibraryName*. In particular, *FTEndClassByLibrary* must be called after *FTInitExt* once for each market or service skeleton which is referenced in the rest of the program. To facilitate the FastTrack libraries are equipped with several market/service include files⁶ and shared libraries each containing:

- **Include file:** C declarations of the market/service structures;
- **Shared library:** a function that returns the market/service skeleton.

The shared library identified by *LibraryName* must have a name like "ftl<something>"⁷ and it must be present (at run-time) in one of the standard places where the Operating System search for the shared library (e.g. in the same place where the *FTAPI.dll* library is placed⁸). There is no need to statically link this library at compile time.

The *Path* parameter (if different from 0 and "") allows to extend, with an additional folder, the (previously described) list of the standard places where the Operating System search for the shared library.

The *LibraryVersion* parameter (if different from 0) is the address of a string where the library version of the loaded *LibraryName* will be stored.

Returned Value

FTOK	Extension completed.
FTInitError	Library not initialized.

⁵ e.g. in Windows a .DLL library.

⁶ e.g. in Windows "ftlmetamarket.dll" for the MetaMarket:

this file contains only data structures definitions but any function declarations.

⁷ e.g. in Windows "ftlmetamarket.dll" for the MetaMarket:

this library contains a function body that will be automatically invoked by *FTEndClassByLibrary* in order to retrieve the skeleton associated to the MetaMarket.

⁸ e.g. in Windows the current directory, the "C:\WINNT\system32" directory, etc...

FTGenericError	<i>LibraryName</i> not found or malformed.
-----------------------	--

4.5 FTStart

The function:

```
long FTStart(void);
```

concludes the extension process of the library. It has to be called after the *FTExtendClass* and/or *FTExtendClassByLibrary* calls and before any other FT/API function.

Returned Value

FTOK	Initialization completed.
FTInitError	Library not initialized.
FTInternalError	Internal error.

4.6 FTShutdown

The function:

```
long FTShutdown(void);
```

releases the memory used by the library and concludes the session. It must be invoked to properly terminate the client.

Returned Value

FTOK	Closing completed.
FTCloseError	Library not initialized.

4.7 FTRun

The function:

```
long FTRun(void);
```

operates the communication in two directions: Client ⇔ Server.

It:

- sends operation requests to the Server (such as a request for connection opening);
- invokes the notification functions defined by the user when information arrives from the Server.

FTRun has to be systematically called in the main cycle of the client program. Not calling *FTRun* for a period of time longer than the *alive* timeout defined by the protocol communication leads to the server connection being closed.

Returned Value

FTOK	Transfer completed.
FTInitError	Library not initialized.
FTInternalError	Transfer error.

4.8 FTConversationSocketSelect

The function:

```
long FTConversationSocketSelect(long ConversationID);
```

allows the program to read from the socket of the conversation - denoted by the parameter *conversationId* - previously opened with the server.

This function makes obsolete the application function *MarketWait()* used in the previous versions of the *FTApi*.

This function should be called in conjunction with the function *FTRun* as shown below.

A basic *FTApi* event loop should be:

```
do{
    if(FTRun() != FTOK){
        printf("unrecoverable FTApi Error");
        FTShutdown();
    }
    FTConversationSocketSelect(MyConvId);
}while(1);
```

4.9 FTSocketSelect

The function:

```
long FTSocketSelect();
```

allows the program to read from the sockets of all conversations previously opened with the server. This function makes obsolete the application function *MarketWait()* used in the previous versions of the *FTApi*.

The function should be called in conjunction with the function *FTRun* as shown below:

A basic *FTApi* event loop should be:

```
do{
    if(FTRun() != FTOK){
        printf("unrecoverable FTApi Error");
```

```
        FTShutdown();
    }
    FTSocketSelect();
}while(1);
```

4.10 FTEnumConversationSocket

The function:

```
long FTEnumConversationSocket (FTConversationSocketProc ConvSocketProc);
```

behaves as the FTEnumConversation function and in addition, provides the identifier of the socket associated with the conversation. This function can be used with ASIA conversations: in this case the callback ConvSocketProc is invoked twice (with the same ConversationID), making available to the user two sockets associated with the same conversation.

Params

FTConversationSocketProc ConvSocketProc: The user-defined callback used in the conversations enumeration

- New Api Callbacks:

Synopsis

```
typedef void (*FTConversationSocketProc) (long ConversationID,
int IsWritePending, long Socket);
```

Description

FTConversationSocketProc is the callback used with FTEnumConversationSocket function.

Users can define its function with the same signature of this callback and pass it as argument when they invoke the FTEnumConversationSocket function.

Params

long ConversationID:	The id of the Conversation.
int isWritePending:	Indicates whether there is a request for writing pending on the conversation socket.
long Socket:	The id of the socket associated with this conversation.

Notes

With ASIA Conversation this callback is called twice with the same ConversationID.

4.11 FTSetSelectTimeout(long Sec, long uSec);

The function:

```
void FTSetSelectTimeout();
```

allows the FTApi user to set timeouts - seconds and microseconds - used when POSIX select() function is internally called on opened sockets. This function should be only used when there are some evidences of performance issues related to low level communication between client application and Fasttrack platform. The use of this function must be carefully evaluated.

For example, this call causes select() on all conversations sockets to wait at most 1 sec and 200 microseconds.

```
FTSetSelectTimeout(1, 200);
```


5 CONVERSATIONS

A logical bidirectional connection channel with the Server is called a *Conversation*. Within the same FT/API client program several conversations can be open with the same Server⁹ or with several Servers. The opening of a conversation can be requested to the already started and not closed library.

The functions for managing the connections are:

- **FTOpenConversationExt**
- **FTCloseConversation**
- **FTGetConversationSocket**
- **FTEnumConversation**

The functions for the context management (context is used by **FTOpenConversationExt**) are:

- **FTCreateContext**
- **FTSetContextAttribute**
- **FTDestroyContext**

5.1 Common Conversation Types

5.1.1 FTUserTypeEnum

The enumerated following type indicates the possible types of conversation that can be opened:

```
typedef enum {  
    FTUserUnused, /* deprecated */  
    FTUserTrader,  
    FTUserAutoTrader,  
    FTUserMonitor,  
    FTUserView,  
    FTUserController,  
    FTUserMasterSlave  
} FTUserTypeEnum;
```

⁹ Having several conversations open with the same Server from the same client doesn't usually have any more advantages in functional or efficiency terms than having just one.

In particular, to request market entities to the Server it suffices to open *FTUserView* conversations; to send data variations (transaction), *FTUserTrader* or *FTUserAutoTrader* conversations have to be opened.

FTUserController and *FTUserMasterSlave* values may be used only with *FTOpenConversationExt* function.

|| **FTUserUnused** is a deprecated value that is no more used. ||

5.1.2 FTRevision

The following type is used to code the application version:

```
typedef unsigned long FTRevision[3];
```

The generic H.M.L version (eg 2.0.3) has the following representation in the three-dimensional vector **v**:

$v[0] = H$ (eg. 2);

$v[1] = M$ (eg. 0);

$v[2] = L$ (eg. 3);

5.1.3 FTOpenResStruct

The following structure is used to describe the result of opening a conversation:

```
typedef struct {
    char *          UserName;
    unsigned long   BusinessServiceID;
    unsigned long   ClientServiceID;
    unsigned long   ClientID;
    unsigned long   SystemDate;
    unsigned long   SystemTime;
    FTRevision      MarketRevision;
    unsigned long   ReqID;
    unsigned long   FTID;
    unsigned long   Environment;
} FTOpenResStruct;
```

Fields

UserName	Same <i>UserName</i> used in <i>FTOpenConversationExt</i> .
BusinessServiceID	Business Service ID.
ClientServiceID	Client Service ID.
ClientID	Same <i>ClientID</i> used in <i>FTOpenConversationExt</i> .
SystemDate	System date of the Server.

SystemTime	System time of the Server.
MarketRevision	Version of the Server.
ReqID	Same <i>ReqID</i> given in <i>FTOpenConversationExt</i> .
FTID	FastTrack Server ID.
Environment	FastTrack Server environment (e.g. Production, Testing, etc...): the precise meaning of this value depends on the particular FastTrack server.

The 3 fields *BusinessServiceID*, *ClientServiceID* and *ClientID* are automatically used by *FTMakeTransactionID* to uniquely identify transactions created by this conversation.

5.1.4 FTOpenConversResProc

The following type defines the prototype of the notification function connected to the event *opening of conversation completed*:

```
typedef void (*FTOpenConversResProc) (
    long                ConversationID,
    long                ResultCode,
    FTOpenResStruct *   ResStruct);
```

Parameters

ConversationID	The conversation whose opening has terminated.
ResultCode	Opening result. The possible values are shown below.
ResStruct	Pointer to a structure <i>FTOpenResStruct</i> .

Value of ResultCode

FTOK	Conversation opened successfully.
FTInvalidConfigurationKey	Access rights Invalid.
FTInvalidServerStatus	The Server is in an invalid status (eg. still in a start-up state)
FTInvalidUserType	Invalid <i>UserType</i> .
FTInvalidUserName	Invalid <i>UserName</i> .
FTInvalidPassword	Invalid <i>Password</i> .
FTInvalidRevision	Invalid application version.
FTInvalidCID	Invalid <i>ClientID</i> (eg already in use).
FTAlreadyLog	User specified already connected.
FTExceedSession	Too much open sessions.
FTInvalidProfile	Invalid profile.
FTAccountNotActive	The provided Account is not active.
FTTooManyTradersConnected	There are too many traders connected.
FTPasswordExpired	Provided password is expired.
FTNotPrivilegeChangePassword	The user can't change the password.
FTNewPasswordRepeated	The provided password and the current are equals.
FTInsufficientNewPasswordLength	The provided password is too short.

FTInvalidNewPasswordCharacters	The provided password contains invalid characters.
FTNewPasswordTooMuchEasy	The provided password is too easy.
FTGenericError	A generic error.

Remarks

This prototype function *FTOpenConversResProc* will be automatically invoked only if *FTOpenConversationExt* returned $n \geq 0$.

5.1.5 FTBrokenTypeEnum

The following enumerated type represents the possible causes of loss of conversation:

```
typedef enum {
    FTConnectionLost,
    FTConnectionClosed
} FTBrokenTypeEnum;
```

The value **FTConnectionLost** notifies an unrecoverable closing of a conversation initiated by the server or caused by the non availability of communication resources. The value **FTConnectionClosed** notifies the closing of a conversation caused by the client application itself or engaged by the function *FTCloseConversation* or *FTShutdown*.

5.1.6 FTNotifyBrokenProc

The following type defines the prototype of the notification function connected to the closing event of a conversation:

```
typedef void (*FTNotifyBrokenProc)(
    long ConversationID,
    FTBrokenTypeEnum BrokenType);
```

Parameters

ConversationID	Conversation terminated.
BrokenType	Indicates the type of termination.

Remarks

This prototype function *FTNotifyBrokenProc* will be automatically invoked only if *FTOpenConversationExt* returned $n \geq 0$.

5.1.7 FTConversationProc

The following type defines the prototype of the notification function used by the enumeration of the active conversations of the library (see next section):

```
typedef void (*FTConversationProc)(  
    long    ConversationID,  
    int     isWritePending);
```

Parameters

ConversationID	Conversation.
isWritePending	Indicates whether there is a request for writing pending on the conversation.

Remarks

This prototype function *FTConversationProc* will be automatically invoked only if *FTEnumConversation* returned *FTOK*.

The parameter *isWritePending* can be used to establish whether or not the socket associated with the conversation needs to be considered in the *select* write descriptor.

5.2 FTContext

A *FTContext*, used in *FTOpenConversationExt*, is a set of pairs:

- attribute, identified by a well-defined integer;
- value, represented by a C string *char **.

The list of attributes is shown in the *FTSetContextAttribute* section.

A *FTContext* is:

- created by *FTCreateContext*;
- populated by *FTSetContextAttribute*;
- used in *FTOpenConversationExt*;
- destroyed by *FTDestroyContext*.

5.2.1 FTCreateContext

The function:

```
FTContext FTCreateContext(void)
```

Creates a default empty context (a context where all attributes have no values).

Returned Value

!= NULL	Default empty context.
== NULL	Generic error.

5.2.2 FTSetContextAttribute

The function:

```
long FTSetContextAttribute(
    FTContext    Context,
    long         Attribute,
    char *       Value);
```

set a new *value* as *attribute* of *context*, discarding the previous associated old *value*.

Parameters

Context	<i>FTContext</i> to be updated.
Attribute	Attribute to be set.
Value	New value for <i>attribute</i> .

Returned Value

FTOK	Context updated.
FTInvalidArgument	Invalid <i>Attribute</i> or bad <i>Context</i>
otherwise	Generic error.

Attributes For Session

These attributes can be used in the *FTInitExt* Context.

Attribute	Value Example	Comments
CTX_INIT_MAX_BLOCK_LEN	"1000"	Size of the internal communication buffer. Don't change the default parameter if not needed.

Attributes For Conversation

These attributes can be used in the *FTOpenConversationExt* Context.

Attribute	Value Example	Comments
CTX_COMPRESSED	"ZIP"	Type of compression used in the conversation. Default "" for no compression.
CTX_APPL_AUTHKEY	"<List_License> <License> <fasttrack> <FTID>1007 </FTID> ..."	Authentication key.

CTX_APPL_AUTHFILE	"C:\\license.xml"	File containing one or more authentication keys.
CTX_X509_CERTIFICATE	"...."	X509 certificate.
CTX_ALTERNATIVE_HOST1	"192.56.27.41"	First alternative for server IP address and service port number.
CTX_ALTERNATIVE_PORT1	"1301"	
CTX_ALTERNATIVE_HOST2	"192.56.27.42"	Second alternative for server IP address and service port number.
CTX_ALTERNATIVE_PORT2	"1301"	
CTX_ALTERNATIVE_HOST...	"192.56.27.43"	N th alternative for server IP address and service port number.
CTX_ALTERNATIVE_PORT...	"1301"	
CTX_CONNECTION_TIMEOUT	"1"	Timeout (in seconds ¹⁰) during which the <i>FTOpenConversationExt</i> will remain blocked ¹¹ waiting for a connection to the server.
CTX_CONNECTION_PROXYST RING	"ACCESS_POINT"	Service identifier used by the connected system to route the connection.
CTX_CONNECTION_NEWPAS SWORD	"elisa1984"	New operator's password that will replace the current one.

5.2.3 FTDestroyContext

The function:

```
void FTDestroyContext(FTContext Context)
```

destroy a context, after it is used in *FTOpenConversationExt*.

Parameters

Context	<i>FTContext</i> to be destroyed.
----------------	-----------------------------------

5.3 Conversation management

¹⁰ Minimum value for **CTX_CONNECTION_TIMEOUT** is 1 second.

¹¹ During this period the entire application will be blocked and no more responsive/sensitive to events arriving from the network.

5.3.1 FTOpenConversationExt

The function:

```
long FTOpenConversationExt(
    char *                IPAddress,
    unsigned short        IPPort,
    FTUserTypeEnum         UserType,
    unsigned long          ClientID,
    char *                UserName,
    char *                Password,
    FTRevision            ApplRevision,
    unsigned long          ApplSignature,
    FTOpenConversResProc   OpenResProc,
    FTNotifyBrokenProc     BrokenProc,
    unsigned long          ReqID,
    char *                Service,
    FTContext              contex);
```

requests the library to open a conversation with a service/market on a Server.

Parameters

IPAddress	TCP/IP address of the Server machine.
IPPort	Service port of the Service.
UserType	User type associated with a new conversation.
ClientID	Univocal ID with which the Client will be registered.
UserName	User name.
Password	Password.
ApplRevision	Client application version.
ApplSignature	Signature ID which may be required by the entity that manages the market.
OpenResProc	Function called to notify the result of a conversation.
BrokenProc	Function called to notify the result of a conversation closing.
ReqID	Value used to match this <i>FTOpenConversationExt</i> with the corresponding <i>OpenResProc</i> invocation.
Service	Market/service to open.
Context	Context of the conversation

Returned Value

n (> 0)	OK: <i>n</i> is the conversation ID being opened.
FTOK (== 0)	OK: conversation ID isn't currently available ¹² .
FTInitError	Library not initialized.
FTInvalidCID	Invalid <i>ClientID</i> .
FTInternalError	Generic error.
FTInvalidIPPort	Invalid port.
FTNoServer	<i>Service</i> not answering.
FTUnresolvedService	<i>Service</i> not resolved.

Remarks

OpenResProc and *BrokenProc* will only be called if *FTOpenConversationExt* has not returned an error code (i.e. only when a value ≥ 0 was returned).

Two ways to access a FastTrack service

To connect with a specific FastTrack service (e.g. MetaMarket) there are two different manners:

```
long r1 = FTOpenConversationExt(
    MM_IPAddress, MM_IPPort, UserType, ClientID,
    UserName, Password, ApplRevision, ApplSignature,
    OpenResProc, BrokenProc, ReqID,
    0, context);

long r2 = FTOpenConversationExt(
    YAS_IPAddress, YAS_IPPort, UserType, ClientID,
    UserName, Password, ApplRevision, ApplSignature,
    OpenResProc, BrokenProc, ReqID,
    "METAMARKET", context);
```

With the first manner you ask to connect to a specific service (e.g. MetaMarket) giving the specific address and port on which the service is running (e.g. `MM_IPAddress`, `MM_IPPort`). In this case you have to not specify anything (e.g. `0`) as Service second-last parameter.

A non-negative answer `r1` (typically `r1 > 0`) means that when the connection is established the appropriate *OpenResProc* will be called.

Using this manner every established connection will be always made with the same service instance (exactly with the service running at `MM_IPAddress`, `MM_IPPort`).

With the second manner you ask to YAS service (identified by `YAS_IPAddress`, `YAS_IPPort`) to establish a connection with the less-loaded service specified by the second-last parameter (e.g. `"METAMARKET"`).

A non-negative answer `r2` (typically `r2 == 0`) means that when the connection is established the appropriate *OpenResProc* will be called. Using this manner every

¹² Sometime a specific conversation ID cannot be returned (e.g. when a non NULL and non empty *Service* was requested). In any case the conversation ID of a successfully created conversation may be found in the *ConversationID* parameter of *FTOpenConversResProc*.

established connection to a same logical service (e.g. "METAMARKET") may be physically established with different service instances (e.g. with instances running on different hosts and/or ports).

Obviously it is illegal to ask a non-YAS service for a specific service as in the following example:

```
long r3 = FTOpenConversationExt(
    MM_IPAddress, MM_IPPort, UserType, ClientID,
    UserName, Password, ApplRevision, ApplSignature,
    OpenResProc, BrokenProc, ReqID,
    "METAMARKET", context);
```

Note ASIA:

To connect to a specific ASIA service (e.g. MetaMarket):

```
long r2 = FTOpenConversationExt(
    YAS_IPAddress, YAS_IPPort, UserType, ClientID,
    UserName, Password, ApplRevision, ApplSignature,
    OpenResProc, BrokenProc, ReqID,
    "PUBLMETAMARKET-PRIVMETAMARKET", context);
```

An ASIA connection can be made via FTapi exclusively by service name and never directly with an IP-address and a port; the service name string is composed by two tokens "FIRSTSERV|SECONDSERV" separated by a pipe. There must be present a file named *FIRSTSERV-SECONDSERV.routing.properties* that defines class-id routing rules, in the working dir of the application,
For example, the connection to the MetaMarket service can be done passing "PUBLMETAMARKET|PRIVMETAMARKET" string.

5.3.2 FTCloseConversation

The function:

```
long FTCloseConversation (long ConversationID);
```

closes the conversation identified by *ConversationID*.

Returned Value

FTOK	Conversation closed successfully.
FTCloseError	Conversation already closed.
FTInitError	Library not initialized.
FTInvalidConvIdError	Unknown ID.
FTInternalError	Generic error.

5.3.3 FTGetConversationSocket

The function:

```
long FTGetConversationSocket (long ConversationID);
```

returns the socket associated with the conversation *ConversationID*.
The availability of sockets used by the library means that concurrent client applications can be written, using passive waiting (see also: function *FTEnumConversation*).

Returned Value

n > 0	The socket associated with the conversation.
FTInitError	Library not initialized.
FTInvalidConvIdError	Conversation unknown.
FTInternalError	Generic error.

5.3.4 FTEnumConversation

The following function:

```
long FTEnumConversation (FTConversationProc ConvProc);
```

enumerates all the conversations currently opened by the library, by invoking on each the notification function *ConvProc*.

Returned Value

n ≥ 0	Enumeration of n active conversations successfully completed.
FTInitError	Library not initialized.
FTInternalError	Generic error.

6 SUBSCRIPTIONS

The **FastTrack** library provides functionalities for opening, checking and closing *subscriptions*.

Opening a subscription on a data class of a Server entails both the initial acquisition of the entities of the classes of the Server and the subsequent notification of any additions or cancellations executed by the Server on that class¹³.

Special subscription modalities enable the acquisition of just one subset of the Entities of a Class of the Server (for extra information about special subscriptions modalities, you can see the last section "*Subscription Usage*" of this chapter)

The functions for managing subscriptions are:

- **FTMakeKeyValue**
- **FTStartSubscribeClassExt**
- **FTStopSubscribeClass**
- **FTRefreshEntity**

The functions for mask & filter management (masks and filters are used by **FTStartSubscribeClassExt**) are:

- **FTCreateMask**
- **FTAddFieldToMask**
- **FTDestroyMask**
- **FTFilterCreate**
- **FTFilterSet**
- **FTFilterDestroy**

6.1 Common Subscription Types

6.1.1 FTComunicationModeEnum

The enumerated type:

```
typedef enum {  
    FTAckUnrequired,  
    FTAckRequired  
} FTComunicationModeEnum;
```

¹³ The FastTrack Server keeps information in a relational Data Base. We will call "Classes" the tables of the Data Base, and "Entities" the records of the classes. On each of the server classes one or more keys are defined which induce a corresponding number of possible orders on the Class Entities.

lists the possible communication models to use when receiving Entities subscribed to the Server.

The value **FTAckUnrequired** specifies an asynchronous communication: the Server sends the Entities required by the client at the maximum frequency allowed by the communication structures.

The value **FTAckRequired** specifies a synchronous communication: the Server sends the Entities one by one, and waits for an "ack" from the client after each send.

The asynchronous communication modality maximises the transmission speed and optimizes the exploitation of communication resources.

6.1.2 FTSubscribeFlowEnum

The enumerated type:

```
typedef enum {  
    FTFlowLast,  
    FTFlowAll,  
    FTZeroMaskedFlowLast,  
    FTZeroMaskedFlowAll  
    FTDiffMaskedFlowAll  
} FTSubscribeFlowEnum;
```

lists the possible send policies of the Server.

The values **FTFlowLast** and **FTZeroMaskedFlowLast** adapt the quantity of the data sent by the server to the reception speed of the client. In practice, to each send operation, the server only sends **the most recent** image of the entity, respect the previous send.

The values **FTFlowAll** and **FTZeroMaskedFlowAll** require the server to send **all the variations** of the entities of the subscribed class.

Basically **FTFlowLast** and **FTZeroMaskedFlowLast** adapt the transmission resolution of the Server to the reception band of the Client; **FTFlowAll** and **FTZeroMaskedFlowAll**, on the other hand, requires each intermediate variation of the Server DataBase to be submitted. Thus delays in the acquisition by the Client may occur due to the increase in load of the buffer inside the communication channel.

The **FTZeroMaskedFlowLast** and **FTZeroMaskedFlowAll** are equivalent to the non-ZeroMasked version, apart the fact the transmission is optimized in a manner that only all non-zero values are sent from server to client. This optimization is transparent to the client application, e.g. the behavior of *IsMasked* field of *FTNotifySubscribeResStruct* does not change for masked or non-ZeroMasked flow values.

The **FTDiffMaskedFlowAll** mode is an advanced subscription opening mode. This mode will produce a field-by-field notification of the subscription. Suitable functions allow to add the incoming field-update to the local record image (see *FTIsFieldChanged* and *FTUpdateMarketObject* functions).

6.1.3 FTTimeStamp

The structure:

```
typedef unsigned long FTTimeStamp[2];
```

will be used to represent a *time stamp*, or rather temporal indicators associated with each market entity¹⁴.

Moreover, the function:

```
int FTCompareTS (FTTimeStamp TS1, FTTimeStamp TS2);
```

allows the comparison of two time stamps.

Returned Value

-1	If TS1 is before TS2.
0	If TS1 represents the same time as TS2.
+1	If TS1 is after TS2.

6.1.4 FTQueryTypeEnum

The enumerated type:

```
typedef enum {
    FTQueryAll,
    FTQuerySet,
    FTQueryEQ, /* deprecated */
    FTQueryLT, /* deprecated */
    FTQueryGT, /* deprecated */
    FTQueryBW /* deprecated */
    FTQueryPast,
    FTQueryOnTime,
} FTQueryTypeEnum;
```

lists the possible selection criteria that can be set in a subscription.

Values

FTQueryAll	To subscribe all the entities in a class.
-------------------	---

¹⁴ The Market Server updates the time stamp of an entity after each update/insertion. The first long field is set to the value `time(0)`, while the second one is used incrementally to make the time stamps univocal when it was generated within the same time unit.

FTQuerySet	To subscribe the subset of the entities that correspond to the partial key value contained in <i>KeyValue1</i> .
FTQueryEQ	To subscribe the only entity that coincides in key with the complete key value contained in <i>KeyValue1</i> .
FTQueryLT	To subscribe the subset of the entities that precede in key the complete key value contained in <i>KeyValue1</i> .
FTQueryGT	To subscribe the subset of the entities that follow in key the complete key value contained in <i>KeyValue1</i> .
FTQueryBW	To subscribe the subset of the entities that follow in key the complete key value contained in <i>KeyValue1</i> and precede in key the complete key value contained in <i>KeyValue2</i> .
FTQueryPast	Same as FTQueryAll but subscription stops automatically when the last past variations are sent. Whith this value only past variations are received, live variations are never sent/received. This value may be used with <i>FTStartSubscribeClassExt</i> function only.
FTQueryOnTime	Same as FTQueryAll but subscription does not include past variations. Whith this value only live variations are received, past variations are never sent/received. This value may be used with <i>FTStartSubscribeClassExt</i> function only.

Values *FTQueryEQ*, *FTQueryLT*, *FTQueryGT* and *FTQueryBW* are **deprecated**. Similar functionalities may be achieved using the filtering mechanism of *FTStartSubscribeClassExt* or using the query mechanism of *FTQueryCreate*.

6.1.5 FTKeyValue

The structure:

```
typedef struct{
    char    buffer[FTKEY_MAXLEN];
    long    len;
} FTKeyValue;
```

is used to represent key values.

6.1.6 FTMakeKeyValue

The function:

```
long FTMakeKeyValue (
    long        ClassId,
    void *      Entity,
    long        keyId,
    long        NumSegments,
    FTKeyValue * KeyValue);
```

calculates in *KeyValue* the key value relating to the key *keyId* of order *NumSegments*¹⁵, on the base of the entity (partially or completely filled) *Entity* of the class *ClassId*.

Parameters

ClassId	Class of the <i>Entity</i> .
Entity	Pointer to a record-entity in which at least the key fields of interest have been assigned.
keyId	A key for the class <i>ClassId</i> .
NumSegments	The number of key segments that have to be acquired from <i>Entity</i> , or rather the order of the partial key calculated. If 0 all the key segments are considered.
KeyValue	A pointer to an <i>FTKeyValue</i> structure in which the value of the calculated key will be loaded.

Returned value

If successful *FTMakeKeyValue* returns *FTOK*.

Remarks

To set the parameter *NumFields* equal to 0, entails calculating a complete key value; it is thus equivalent to a value of *NumSegments* equal to the overall number of segments associated with *keyId*.

6.1.7 FTSubscribeStatusEnum

The enumerated type:

```
typedef enum {
    FTSubscribeStart,
    FTSubscribeStop
} FTSubscribeStatusEnum;
```

represents subscription opening and subscription closing events.

6.1.8 FTSubscribeResStruct

The structure:

¹⁵ Please see the last section "Subscription Usage" of this chapter in order to understand the concepts of:

- key value of order *K*,
- complete key value,
- key segments.


```
typedef struct {
    unsigned long        ClassId;
    FTSubscribeStatusEnum SubscribeStatus;
    int                  ResetClass;
    unsigned long        ClassVersion;
    FTSubscribeFlowEnum  SubscribeFlowEnum;
    FTQueryTypeEnum       QueryType;
    long                  SubscriptionID;
} FTSubscribeResStruct;
```

will be used to return the result of a subscription opening.

Parameters

ClassId	Subscribed Class.
SubscribeStatus	Indicates whether the subscription is open or closed.
ResetClass(*)	Indicates whether a Down Load is needed on all the classes.
ClassVersion(*)	Class Version in the market Server.
SubscribeFlowEnum(*)	Requested transmission modality.
FTQueryTypeEnum(*)	Requested selection modality.
SubscriptionID	Identifier of the open or closed Subscription.

Fields marked with (*) are only available when *SubscribeStatus* is equal to *FTSubscribeStart*.

6.1.9 FTEventTypeEnum

The enumerated type:

```
typedef enum {
    FTClassInstance,
    FTConnectionBroken,
    FTClassIdle
} FTEventTypeEnum;
```

will be used in the notification function of a subscription to indicate the type of event notified.

Values

FTClassInstance	Notifies rewritings, additions or cancellations of an entity.
FTConnectionBroken	Notifies the closing of the communication structures used by the subscription.

FTClassIdle	Notifies the aligning with the data class of the server, ie. end of past data and start of live data. This value may be seen only if <i>QueryType</i> is not equal to <i>FTQueryPast</i> nor <i>FTQueryOnTime</i> in <i>FTStartSubscribeClassExt</i> .
--------------------	---

FTClassIdle stands for the reception of all the entities currently present in the server class subscribed. It is received exactly once for each open subscription. Even after *FTClassIdle* it is still possible to receive on a subscription other notifications of *FTClassInstance* events caused by modifications on the DataBase of the Server.

6.1.10 FTClassActionEnum

The enumerated type:

```
typedef enum {
    FTEntityADD,
    FTEntityDEL,
    FTEntityRWT,
    FTEntityKIL
} FTClassActionEnum;
```

is used to indicate the possible operations carried out by the Server on an entity of a data-class.

Values

FTEntityADD	The entity has been added to the data class.
FTEntityDEL	The entity has been logically removed from the data class.
FTEntityRWT	The entity has been rewritten in the data class.
FTEntityKIL	The entity has been physically removed from the data class.

An *FTEntityADD* is functionally equivalent to an *FTEntityRWT*, if an instance with the same key value is already present in the DataBase.

6.1.11 FTNotifySubscribeResStruct

The structure:

```
typedef struct {
    FTClassActionEnum  EntityAction;
    unsigned long      ClassId;
    long               KeyId;
    FTTimeStamp        TimeStamp;
    void *             MarketObject;
```

```

        unsigned long    LenMarketObject;
        long             SubscriptionID;
        int              IsMasked;
        FTDiffMask       DiffMask;
    } FTNotifySubscribeResStruct;

```

will be used to represent data arriving from a subscription.

Parameters

EntityAction	Operation carried out by the Server on the entity <i>MarketObject</i> on its DataBase (addition, removal, etc).
ClassId	Class of the entity in <i>MarketObject</i> .
KeyId	Key on the basis of which the server has carried out <i>EntityAction</i> .
TimeStamp	Time stamp associated with the entity contained in <i>MarketObject</i> .
MarketObject	Pointer to the entity containing the market data. Usually a cast is needed on this variable on the basis of <i>ClassId</i> .
LenMarketObject	Quantity of memory used in <i>MarketObject</i> .
SubscriptionID	Subscription on which the data has been received.
IsMasked	A non-zero value indicates that some fields of <i>MarketObject</i> have zero values because they are fields of a masked subscription. A non zero-value is present only for data arriving on masked subscriptions that are not answer to a <i>FTRefreshEntity</i> action. In all other cases this field evaluate to zero.
DiffMask	Valorized for <i>FTDiffMaskedFlowAll</i> subscription type only. This data is used in order to update the local client database with the field(s) received.

Conventionally, an *EntityAction=FTEntityKIL* with *KeyId* ≤ 0 indicates that the Server has removed all the entities of the class *ClassId*.

6.1.12 FTSubscribeResProcExt

The type :

```

typedef void (*FTSubscribeResProcExt)
    (long                ConversationID,
     unsigned long       ReqID,
     long               ResultCode,
     FTSubscribeResStruct * ResStruct);

```

defines the prototype of the notification function of the opening or closing of a subscription.

Parameters

ConversationID	<i>ConversationID</i> in <i>FTStartSubscribeClassExt</i> .
-----------------------	--

ReqID	Same <i>ReqID</i> given in <i>FTStartSubscribeClassExt</i> .
ResultCode	Code indicating operation result: <i>FTOK</i> or <i>FTNoSubscribedClassError</i> if subscription opening fails.
ResStruct	Contains information regarding the subscription only if the resultCode gives <i>FTOK</i> .

Remarks

This prototype function *FTSubscribeResProcExt* will be automatically invoked only if *FTStartSubscribeClassExt* returned $n > 0$.

6.1.13 FTNotifySubscribeResProcExt

The type :

```
typedef void (*FTNotifySubscribeResProcExt)(
    long                ConversationID,
    unsigned long       ReqID,
    FTEventTypeEnum     EventType,
    FTNotifySubscribeResStruct * ResStruct);
```

defines the prototype of the function to define in order to receive the events and the subscribed data.

Parameters

ConversationID	Conversation on which the subscription has been opened.
ReqID	Same <i>ReqID</i> given in <i>FTStartSubscribeClassExt</i>
EventType	Type of the new event on the subscription.
ResStruct	Pointer to the structure containing the data received.

Remarks

This prototype function *FTNotifySubscribeResProcExt* will be automatically invoked only if *FTStartSubscribeClassExt* returned $n > 0$.

The parameter *ResStruct* is not significant in the case of *EventType=FTConnectionBroken*. It is significant only in the field *ClassId* in the case of *EventType= FTClassIdle*.

6.2 Masks

A *FTMask*, used in *FTStartSubscribeClassExt* or in *FTSendTransactionExt*, denotes a set of fields in a same class. During the subscription-life (between a *FTStartSubscribeClassExt* and the corresponding *FTStopSubscribeClass*) only the fields described by the mask will be available to the client in the *FTNotifySubscribeResProcExt*.

A *FTMask* is:

- created by *FTCreateMask*;
- populated by *FTAddFieldToMask*;
- used in *FTStartSubscribeClassExt* or in *FTSendTransactionExt*;
- destroyed by *FTDestroyMask*.

6.2.1 FTCreateMask

The function:

FTMask FTCreateMask(unsigned long ClassID)

Creates a default empty mask (a mask with no fields) for the class *ClassID*.

Returned Value

!= NULL	Empty mask (a mask with no fields) created
== NULL	Generic error.

6.2.2 FTAddFieldToMask

The function:

```
long FTAddFieldToMask(FTMask Mask,
                     char *   MaskedField)
```

add *MaskedField* to the set of fields described by *Mask*.

Parameters

Mask	<i>FTMask</i> to be updated.
MaskedField	Field to be added to the mask. See below for a detailed description on fields.

Returned Value

FTOK	Mask updated.
FTInvalidArgument	Error in <i>MaskedField</i>

A masked field of a class C may be:

- Rule 1: a string (array of char) field of C.
- Rule 2: a primitive (i.e. char or numeric type or enum) field of C.
- Rule 3: a component of an array, of primitive types (but here chars are not allowed!), field of C.
In this case the component is identified by an unsigned integer between square brackets [].

- Rule 4: a masked field (recursive definition!) of a class field of C.
In this case the masked field is identified by the dot notation "field.field".
- Rule 5: a masked field (recursive definition!) of a component of an array, of classes, field of C.
In this case the component is identified by an unsigned integer between square brackets [].

Spaces (blank, tab, carriage return, etc...) are not allowed inside a masked field representation.

As an example please consider the following two classes:

```
typedef struct {
    int    n;
    int    v[10];
    char   s[10];
} TypeA;
typedef struct {
    TypeA  a[10];
    TypeA  x;
    double d[10];
    char   c[10];
    char   b;
    char   ss[10];
} TypeB
```

Here a list of **valid** TypeB masked fields:

Rule 1: "ss"

Rule 2: "b"

Rule 3: "d[2]"

Rule 4: "x.n", "x.v[3]", "x.s"

Rule 5: "a[8].n", "a[0].v[9]", "a[5].s"

Here a list of **invalid** TypeB masked fields:

Rule 1: "s s", "t"

Rule 2: "x"

Rule 3: "c[2]", "d[+2]", "d[20]", "d[2]", "d[2]", "ss[3]"

Rule 4: "a.n", "x.y"

Rule 5: "a[8]. n", "a[8]", "a[0].v", "a[5].s[3]"

6.2.3 FTDestroyMask

The function:

```
void FTDestroyMask(FTMask Mask)
```

destroy a mask, after it is used in *FTStartSubscribeClassExt* or in *FTSendTransactionExt*.

Parameters

Mask	<i>FTMask</i> to be destroyed.
-------------	--------------------------------

6.3 Filters

A filter, used in *FTStartSubscribeClassExt*, is used in a subscription to restrict (into the server) the set of values available (into the client) during the *FTNotifySubscribeResProcExt* invocations.

A filter is:

- created by *FTFilterCreate*;
- extended by *FTFilterSet*;
- used in *FTStartSubscribeClassExt*;
- destroyed by *FTFilterDestroy*.

A filter is defined by 3 things:

- A filter type (defined during the filter creation);
- A (eventually empty) filter definition (defined as above);
- A (eventually empty) filter value (defined in the filter extension).

The precise meaning of these 3 things depends from the particular filter and, in general, must be agreed between the client and the server.

6.3.1 FTFilterCreateEnum

The enumerated type:

```
typedef enum {  
    FTFilterCreateOK,  
    FTFilterCreateSyntaxError,  
    FTFilterCreateInvalidFilterLen,  
    FTFilterCreateInvalidClassID,  
    FTFilterCreateInvalidFilterType,  
    FTFilterCreateNotImplemented  
} FTFilterCreateEnum;
```

is used to indicate the possible results carried out by the Server on filter creation.

Values

FTFilterCreateOK	Filter created on the server.
FTFilterCreateSyntaxError	Syntax error in the <i>FilterDefinition</i> .
FTFilterCreateInvalidFilterLen	Wrong <i>FilterDefinitionLen</i>
FTFilterCreateInvalidClassID	Wrong <i>ClassID</i>
FTFilterCreateInvalidFilterType	Wrong <i>FilterType</i>
FTFilterCreateNotImplemented	Filter non implemented on the server.

6.3.2 FTFilterCreateProc

The following type defines the prototype of the notification function connected to the filter creation:

```
typedef void (*FTFilterCreateProc) (
    long                ConversationID,
    unsigned long       ReqID,
    long                FilterID,
    FTFilterCreateEnum  Result);
```

Parameters

ConversationID	The conversation on which the filter was created.
ReqID	Same <i>ReqID</i> given in <i>FTFilterCreate</i> .
FilterID	Id of filter created. It may be used in <i>FTFilterSet</i> , <i>FTFilterDestroy</i> , <i>FTStartSubscribeClassExt</i> .
Result	Result of filter creation into the server.

Remarks

This prototype function *FTFilterCreateProc* will be automatically invoked only if *FTFilterCreate* returned *FTOK*.

6.3.3 FTFilterCreate

The function:

```
long FTFilterCreate(
    long                ConversationID,
    unsigned long       ReqID,
    unsigned long       ClassID,
    unsigned long       FilterType,
    unsigned long       FilterDefinitionLen,
    char *              FilterDefinition,
                        /* max len: FT_FILTERDEF_MAXLEN */
    FTFilterCreateProc  FilterCreateProc);
```

creates the filter.

When the *FilterCreateProc* is called the filter is create in the server also and then it may be referenced by the *FilterID* value.

Parameters

ConversationID	Conversation on which the filter will be created.
-----------------------	---

ReqID	Value used to match this <i>FTFilterCreate</i> with the corresponding <i>FilterCreateProc</i> invocation.
ClassId	Class on which the filter will be used. Must be the same <i>ClassId</i> used in <i>FTStartSubscribeClassExt</i> .
FilterType	Type of the filter. The precise meaning of this field must be agreed between the client and the server.
FilterDefinitionLen	Length (in bytes) of <i>FilterDefinition</i> . Must be $\leq FT_FILTERDEF_MAXLEN$.
FilterDefinition	Definition of the filter. The precise meaning of this field must be agreed between the client and the server.
FilterCreateProc	Notification function of filter creation results.

Returned Value

FTOK	Filter created on the client.
FTInvalidRevision	A stale server don't implement this function.
otherwise	Generic error.

Remarks

If successful the function returns *FTOK*. In this case *FilterCreateProc* will be automatically invoked.

6.3.4 FTFilterSetEnum

The enumerated type:

```
typedef enum {
    FTFilterSetOK,
    FTFilterSetSyntaxError,
    FTFilterSetInvalidFilterID,
    FTFilterSetInvalidFilterLen,
    FTFilterSetAlreadySet
} FTFilterSetEnum;
```

is used to indicate the possible results carried out by the Server on filter extension.

Values

FTFilterSetOK	Filter extended on the server.
FTFilterSetSyntaxError	Syntax error in the <i>FilterVal</i> .
FTFilterSetInvalidFilterID	Wrong <i>FilterID</i> .
FTFilterSetInvalidFilterLen	Wrong <i>FilterLen</i> .
FTFilterSetAlreadySet	Filter already extended.

6.3.5 FTFilterSetProc

The following type defines the prototype of the notification function connected to the filter extension:

```
typedef void (*FTFilterSetProc) (
    long          ConversationID,
    unsigned long ReqID,
    long          FilterID,
    FTFilterSetEnum Result);
```

Parameters

ConversationID	The conversation on which the filter was extended.
ReqID	Same <i>ReqID</i> given in <i>FTFilterSet</i> .
FilterID	Id of filter extended. It may be used in <i>FTFilterSet</i> , <i>FTFilterDestroy</i> , <i>FTStartSubscribeClassExt</i> .
Result	Result of filter extension into the server.

Remarks

This prototype function *FTFilterSetProc* will be automatically invoked only if *FTFilterSet* returned *FTOK*.

6.3.6 FTFilterSet

The function:

```
long FTFilterSet(
    long          ConversationID,
    unsigned long FilterID,
    unsigned long ReqID,
    unsigned long FilterLen,
    char *        FilterVal,
    /* maxlen: FT_FILTERVAL_MAXLEN */
    FTFilterSetProc FilterSetProc);
```

extends the filter denoted by *FilterID* with a value *FilterVal*.

Parameters

ConversationID	Conversation on which the filter will be updated.
FilterID	Filter to be extended.
ReqID	Value used to match this <i>FTFilterSet</i> with the corresponding <i>FilterSetProc</i> invocation.
FilterLen	Length (in bytes) of <i>FilterVal</i> . Must be \leq <i>FT_FILTERVAL_MAXLEN</i> .

FilterVal	Value of the filter. The precise meaning of this field must be agreed between the client and the server.
FilterSetProc	Notification function of filter extension results.

Returned Value

FTOK	Filter extended on the client.
FTInvalidRevision	A stale server don't implement this function.
otherwise	Generic error.

Remarks

If successful the function returns *FTOK*. In this case *FilterSetProc* will be automatically invoked.

6.3.7 FTFilterDestroyEnum

The enumerated type:

```
typedef enum {
    FTFilterDestroyOK,
    FTFilterDestroyInvalidFilterID
} FTFilterDestroyEnum;
```

is used to indicate the possible results carried out by the Server on filter destruction.

Values

FTFilterDestroyOK	Filter destroyed on the server.
FTFilterDestroyInvalidFilterID	Wrong <i>FilterID</i> .

6.3.8 FTFilterDestroyProc

The following type defines the prototype of the notification function connected to the filter destruction:

```
typedef void (*FTFilterDestroyProc) (
    long          ConversationID,
    unsigned long ReqID,
    long          FilterID,
    FTFilterDestroyEnum Result);
```

Parameters

ConversationID	The conversation on which the filter was extended.
ReqID	Same <i>ReqID</i> given in <i>FTFilterDestroy</i> .
FilterID	Id of filter destroyed.
Result	Result of filter destruction into the server.

Remarks

This prototype function *FTFilterDestroyProc* will be automatically invoked only if *FTFilterDestroy* returned *FTOK*.

6.3.9 FTFilterDestroy

The function:

```
long FTFilterDestroy(
    long ConversationID,
    unsigned long FilterID,
    unsigned long ReqID,
    FTFilterDestroyProc FilterDestroyProc);
```

destroy a filter, after it is used in *FTStartSubscribeClassExt*.

Parameters

ConversationID	Conversation on which the filter will be updated.
FilterID	Filter to be extended.
ReqID	Value used to match this <i>FTFilterDestroy</i> with the corresponding <i>FilterDestroyProc</i> invocation.
FilterDestroyProc	Notification function of filter destroy results.

Returned Value

FTOK	Filter destroyed on the client.
FTInvalidRevision	A stale server don't implement this function.
otherwise	Generic error.

Remarks

If successful the function returns *FTOK*. In this case *FilterDestroyProc* will be automatically invoked.

6.4 DiffMasked Subscription

DiffMasked subscriptions are advanced subscriptions used to optimize the data traffic. With this type of subscription data are sent field-by-field rather than record-by-record.

This subscription type is required via the setting of the enumerated value *FTDiffMaskedFlowAll* in the parameter *SubscribeFlow* of *FTStartSubscribeClassExt*.

Subscription opening will fail in case this subscription type is not supported by the connected system.

Special functionalities can be used in order to update the client database image when a record field arrive from a DiffMasked subscription:

- *FTIsFieldChanged*;
- *FTUpdateMarketObject*;

6.4.1 FTFiledChanged

The function:

```
long FTIsFieldChanged (unsigned long Offset,  
                      FTDiffMask DiffMask);
```

tests if the received updates affect the field indicated in *Offset*.
The macro definition *FT_OFFSET* can be used to retrieve field offset.

Returned Value

0	The provided <i>DiffMask</i> doesn't include the field of offset <i>Offset</i> .
1	The provided <i>DiffMask</i> includes the field of offset <i>Offset</i> .

6.4.2 FTUpdateMarketObject

The function:

```
long FTUpdateMarketObject (  
    char *CurrentMarketObject,  
    char *DiffMarketObject,  
    FTDiffMask DiffMask);
```

updates the current image of the MarketObject record *CurrentMarketObject*, by applying the field updates received for a DiffMask subscription: *DiffMarketObject* and *DiffMask*.

Returned Value

FTOK	Update completed without errors
FTGenericError	Update failed

6.5 Subscription Management

6.5.1 FTStartSubscribeClassExt

The function:

```
long FTStartSubscribeClassExt (  
    long                      ConversationID,  
    FTCommunicationModeEnum  CommunicationMode,  
    FTSubscribeFlowEnum      SubscribeFlow,
```

unsigned long	ClassId,
unsigned long	ClassVersion,
FTTimeStamp	ClassTimeStamp,
FTQueryTypeEnum	QueryType,
unsigned long	KeyID,
FTKeyValue *	KeyValue1,
FTKeyValue *	KeyValue2,
	/* deprecated */
FTSubscribeResProcExt	SubscrResProc,
FTNotifySubscribeResProcExt	NotifyResProc,
unsigned long	ReqID,
unsigned long	FilterID,
FTMask	SubscrMask);

entails opening a subscription.

Parameters

ConversationID	Conversation on which a subscription will be opened.
CommunicationMode	Communication model to adopt for data transmission.
SubscribeFlow	Data transmission policy.
ClassId	Data class to subscribe.
ClassVersion	Last managed version.
ClassTimeStamp	Last time stamp acquired by the class ClassId.
QueryType	Selection criteria of the entities of the class. On NAM market, connected with OMI protocol, this value must not be <i>FTQueryPast</i> nor <i>FTQueryOnTime</i> .
KeyID	Key used to define special selection criteria. Used when <i>QueryType</i> = <i>FTQuerySet</i>
KeyValue1	First key value. Used when <i>FTQuerySet</i> ≤ <i>QueryType</i> ≤ <i>FTQueryBW</i> It is suggested to use only when <i>QueryType</i> = <i>FTQuerySet</i> , because the other <i>QueryType</i> values are deprecated .
KeyValue2	Second key value. Used when <i>QueryType</i> = <i>FTQueryBW</i> . The use of this parameter is deprecated because <i>FTQueryBW</i> is deprecated .
SubscrResProc	Notification function of subscription opening and closing.
NotifyResProc	Notification function of the subscription events: data in arrival, idle event, connection closing.
ReqID	Value used to match this <i>FTStartSubscribeClassExt</i> with the corresponding <i>NotifyResProc</i> and <i>SubscrResProc</i> invocations.

FilterID	Identifier of the filter that restricts the set of class instances availables. The value <i>NULL</i> indicates that no filtering is needed. On NAM market, connected with OMI protocol, this value must be <i>NULL</i> .
SubscrMask	Mask that denotes the interesting set of fields in the class <i>ClassId</i> . The value <i>NULL</i> indicates that all fields are subscribed. On NAM market, connected with OMI protocol, this value must be <i>NULL</i> .

Partial subscriptions for the acquisition of only entities with a partial key given, can be requested by setting *QueryType* to values between *FTQuerySet* and *FTQueryBW*, and by suitably setting parameters *KeyID*, *KeyValue1* and *KeyValue2* (the latter only for *FTQueryBW* criteria).

If non partial subscriptions are required, set *QueryType* to *FTQueryAI* or *FTQueryPast* or *FTQueryOnTime* and *KeyID*, *KeyValue1* and *KeyValue2* to *NULL*.

Incremental subscriptions for the acquisition of only those entities that follow a certain instant, can be requested by suitably setting *ClassVersion* and *ClassTimeStamp*.

Such parameters can be set to *NULL* to receive all the entities irrespectively of their time stamp.

Set *QueryType* to *FTQueryPast* or *FTQueryOnTime* to acquire all past variations or only current variations.

Note that the setting of the parameter **CommunicationMode** (with the value **FTAckUnrequired** or **FTAckRequired**), doesn't affect in any manner the use of the functions in the rest of the code. That's only an internal setting for the communication protocol.

Returned Value

n (>0)	ID of subscription being opened.
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid conversation.
FTInvalidClassIdError	Invalid data class.
FTTransportError	Communication error.
FTInvalidConfigurationKey	The class is not configured in the licence.
FTInvalidMask	<i>SubscrMask</i> refers a class different from <i>ClassId</i>
FTInvalidRevision	A stale server don't implement this function.
FTInternalError	Generic error.

Remarks

If successful the function returns $n > 0$. In this case *SubscrResProc* and *NotifyResProc* will be automatically invoked

6.5.2 FTStopSubscribeClass

The function:

```
long FTStopSubscribeClass (
    long          ConversationID,
    unsigned long  ClassId,
    long          SubscriptionID);
```

requires the closing of a subscription.

Parameters

ConversationID	Conversation on which the subscription has been opened.
ClassId	Data class on which the subscription has been opened.
SubscriptionID	Subscription ID.

Returned Value

FTOK	Closing completed.
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid <i>ConversationID</i> .
FTInvalidClassIdError	Invalid <i>ClassId</i> .
FTNoSubscribedClassError	Invalid <i>SubscriptionID</i> or <i>ClassId</i> not subscribed within <i>SubscriptionID</i>
FTInternalError	Generic error.

Remarks

If successful the function returns *FTOK*. In this case *SubscrResProc* (registered in *FTStartSubscribeClassExt*) will be automatically invoked.

6.5.3 FTRefreshEntity

The function:

```
FTRefreshEntity(
    long          ConversationID
    unsigned long  ClassID,
    long          SubscriptionID,
    void *        Entity,
    unsigned long  KeyID);
```

query the server to re-send (on the opened subscription *SubscriptionID*) the single complete (not masked) *Entity* described by the key *KeyID*.

Parameters

ConversationID	Conversation on which the subscription has been opened.
-----------------------	---

ClassId	Data class on which the subscription has been opened.
SubscriptionID	Subscription ID.
Entity	Pointer to a record-entity in which at least the key fields have been assigned
KeyID	A key for the class <i>ClassId</i> .

Returned Value

FTOK	<i>QueryRecordResProc</i> will be called with the single result found on the server.
FTInitError	Library not initialized.
FTInvalidRevision	A stale server don't implement this function.
FTInvalidConvIdError	Invalid <i>ConversationID</i> .
FTInvalidClassIdError	Invalid <i>ClassId</i> .
FTInvalidSubscriptionID	Invalid <i>SubscriptionID</i> .
FTInternalError	Internal error.

Remarks

If successful the function returns *FTOK*. In this case the function *FTNotifySubscribeResProcExt*, registered with *FTStartSubscribeClassExt*, may be invoked with the single result found on the server: in this case the *IsMasked* field of *FTNotifySubscribeResStruct* will have a non-zero value to indicate that all fields of the entity was received. The *FTNotifySubscribeResProcExt* will not be invoked if there is not an entity with the desired key.

6.6 Subscriptions Usage

These notes outline some basic concepts for managing **incremental** and **partial** subscriptions via this **FT/API** library.

6.6.1 Incremental Subscriptions

This section outlines how to manage incremental subscriptions, in which the server is only required to update the contents of a data class, rather than sending all its records.

A client application can consequently keep a local data base aligned with the market server's, or more generally avoid processing data twice, by minimizing, at the same time, the volume of data to transfer and the time needed.

In particular, the *FTStartSubscribeClassExt* subscription functionality allows you to specify the pair of values *ClassVersion* and *ClassTimeStamp*, thus respectively, the last version index of the class and the time-stamp of the last entity (record) received of that class during the previous subscription.

Thus, if the version of the class maintained in the server coincides with the one supplied, only those entities with a time stamp that is later than the one supplied will be sent.

On the other hand if the version of the class is earlier than the server's current one, the field *ResetClass* of the structure *FTSubscribeResStruct* will be set. This indicates a

general invalidation of any entities from that class that have been archived until that moment. In the latter case it will not be possible to proceed to an incremental acquisition, but all the entities in the class will have to be received.

In order to be able to make this data available the client application has to maintain for each class the following information:

- Current time stamp,
- Current version.

The current time stamp can be maintained by updating local data of the type *FTTimeStamp* whenever *FTEntityADD*, *FTEntityRWT* or *FTEntityDEL* operations are made, on the basis of the field *TimeStamp* of the structure *FTNotifySubscribeResStruct*.

The current version can be maintained by acquiring the value of the version at the opening of the subscription by the field *ClassVersion* of the structure *FTSubscribeResStruct*, and updating it on every version variation of the server class, this basically means at each notification of an *FTEntityKIL* operation, by setting it to the *TimeStamp[0]* received.

Remarks

FTEntityDEL, and *FTEntityKIL* operations with *KeyId>0*, should be intended as notifications of cancellation of an individual entity at a server level. The entity in question is the one, from those acquired beforehand, which coincides with the entity received in *MarketObject* on the fields associated with the key *KeyId*. The entity pointed by *MarketObject* is generally undefined on any fields apart from those associated with *KeyId*.

Operations of *FTEntityKIL* with *KeyId<=0* should be intended as the physical cancellation of every entity in the specified class that has been acquired beforehand. In this case also *MarketObject=NULL* and *LenMarketObject=0*.

If the *ClassVersion* supplied in *FTStartSubscribeClassExt* is different from the version of the class of the server, the library notifies a request for reset on the class, both by setting the field *ResetClass* of the structure *FTSubscribeResStruct*, and by notifying an operation of *FTEntityKIL* with *KeyId=0* on this class.

6.6.2 Partial Subscriptions

This section outlines how to manage partial subscriptions, in which the server is requested to send only those entities in a class that satisfy certain constraints.

Using this type of subscription helps to decrease the flow of data on the network, whenever the client and server have been activated on separate machines, and the ratio is high between the total size of the class and the total size of the subclass.

Partial subscriptions can be formulated by setting the parameter *QueryType* of *FTStartSubscribeClassExt* to a value different from *FTQueryAll* and exploiting the parameter *KeyVal1* to define the subclass of interest.

Relational Keys

A class may have zero, one or more keys.

A key for a class corresponds to an ordered list of some key segments of the class and it is identified by an ID (e.g. KeyID).

A key segment of a class may be:

- an elementary field or subfield of the class, or
- a string (array of chars) field or subfield of the class, or
- an elementary array-component of an array which is a field or subfield of the class.

An elementary field (or array-component) is a field (or array-component) of type:

- char, unsigned char,
- short, unsigned short,
- int, unsigned int,
- long, unsigned long,
- float,
- double.

Example:

```
typedef {  
    long          SequenceNumber;  
    char          SequenceType;  
} SEQUENCERec;  
  
typedef { /* PROPOSAL_ID */  
    SEQUENCERec  Sequence;  
    char         Verb;  
    char         Operator[5];  
    long         BondCode;  
    int          PriceQuantity[2];  
} PROPOSALRec;
```

These are Key-segments of PROPOSALRec = {
 Sequence.SequenceNumber, Sequence.SequenceType,
 Verb, Operator, BondCode, PriceQuantity[0],
 PriceQuantity[1] }

These are **not** Key-segments of PROPOSALRec = {
 Sequence, Operator[0], Operator[1], Operator[2],
 Operator[3], Operator[4], PriceQuantity }

eskey = {
 Verb, PriceQuantity[0], PriceQuantity[1],
 Sequence.SequenceNumber, BondCode };

eskeyID = 2000.

A **key value** k of order r for a key *KeyID* of key segments F_1, \dots, F_n ($n \geq r$) is a vector of memory of size $\text{sizeof}(F_1) + \dots + \text{sizeof}(F_r)$, loaded with r values V_1, \dots, V_r , respectively for the key segments F_1, \dots, F_r .

If $r=n$, k is called a **complete key value**.

Example: (continued)

A key value of order 3 for the key *eskeyID* is given by 9 bytes (1+4+4):
`['B'][103][10]`

where:

- `[xxx]` indicates the machine representation of `xxx`;
- we assume `sizeof(int) = sizeof(long) = 4`.

A complete key value is 21 bytes long:
`['B'][103][10][9876][333]`.

Basically the library makes available the type:

```
typedef struct{
    char    buffer[FTKEY_MAXLEN];
    long    len;
} FTKeyValue;
```

to represent key values;

Example: (continued)

The key value of order 3 in the preceding example is represented by the variable *kv* of type *FTKeyValue* where:

- *kv.buffer*: contains in the first 9 bytes the representation of 'B', 103 and 10;
- *kv.len*: is 9.

The library also defines the function:

```
long FTMakeKeyValue (
    long        ClassId,
    void *      Entity,
    long        keyId,
    long        NumSegments,
    FTKeyValue * KeyValue);
```

for the automatic calculation of key value of the class *Entity*, with *NumSegments* key segments, thus facilitating their construction and length determination.

By setting *NumSegments* = 0, the complete key value is constructed from the entity.

Example: (continued)

The key value of order 3 in the previous example can be constructed as follows:

```
{  
  
    PROPOSALRec  pr;  
    FTKeyValue   kv;  
  
    memset(&pr, 0, sizeof(pr));  
    memset(&kv, 0, sizeof(kv));  
  
    pr.Verb = 'B';  
    pr.PriceQuantity[0] = 103;  
    pr.PriceQuantity[1] = 10;  
  
    FTMakeKeyValue(PROPOSAL_ID, &pr, eskeyID, 3, &kv);  
}
```

Using the value *NULL* for *NumSegments*, and exploiting the other fields in the key *eskeyID* of *pr*, a complete key value would have been obtained.

Selection Criteria

Hereafter the following notations will be used:

- *KeyID(E, r)*: indicates the only key value of order *r* derived from the entity *E*;
- *order(kv)*: indicates the order of a key value *kv*.

Given a key identified by *KeyID*, the selection criteria that can be specified in subscription are:

- *FTQueryAll*: Entails all the class, it doesn't use any of the *KeyValue1* parameter;
- *FTQuerySet*: Entails all the entities *E* such that: *KeyID(E, order(KeyValue1)) == KeyValue1* parameter;
- *FTQueryPast* or *FTQueryOnTime*: their have the same selection criteria of *FTQueryAll*, but they differs from *FTQueryAll* as specified in the "*FTQueryTypeEnum*" section.

7 TRANSACTIONS

A transaction is a client's request to the server to add, remove or modify an entity in its own Data Base. For example, a transaction can be sent in order to present a new proposal on the electronic market.

The FastTrack library has the following functions for sending transactions:

- **FTMakeTransactionID**
- **FTSendTransactionExt**
- **FTStartTransactionMonitoring**
- **FTQueryTransactionStatus**

7.1 Transaction sending

7.1.1 FTTransactionID

The structure:

```
typedef struct {  
    unsigned long    BusinessServiceID;  
    unsigned long    ClientServiceID;  
    unsigned long    ClientID;  
    unsigned long    TimeStamp[2];  
} FTTransactionID;
```

will be used as a univocal ID of a transaction. An instance of such a structure must be created before sending each transaction and may be used to reference the transaction, for example during a recovery or when routing monitoring transaction information.

Parameters

BusinessServiceID	Business Service ID to which the transaction will be sent.
ClientServiceID	Client Service ID (of BusinessServiceID) to which the transaction will be sent.
ClientID	Client ID from which the transaction will be sent.
TimeStamp	Long couple for the representation of the creation instant.

7.1.2 FTMakeTransactionID

The function:

```
long FTMakeTransactionID (  
    long          ConversationID,  
    FTTransactionID* TransactionID);
```

is responsible for generating a new TransactionID on the basis of the parameters associated with the specified conversation (see fields *BusinessServiceId*, *ClientServiceId* and *ClientID* of *FTOpenResStruct*) and of the time of the client activation machine. Note: It is no longer required to applicatively lock the function FTMakeTransactionID when sending transactions on FT_C_ORDER/FT_C_QUOTE in a multithread FTApi program.

Parameters

ConversationID	Conversation on which the transaction will be sent.
TransactionID	Pointer to the already allocated FTTransactionID structure.

Returned Value

FTOK	The TransactionID is correctly constructed.
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid conversation.
FTInvalidCID	The CID associated with the conversation is not correct.
FTInternalError	Internal overflow.

7.1.3 FTMakeTransactionIDExt

The function:

```
long FTMakeTransactionIDExt(long ConversationID, unsigned long ClassID,  
FTTransactionID *TransactionID);
```

behaves exactly like the FTMakeTransactionID() function, except that it accepts as a parameter the ID of the class that will be sent in the transaction.

The use of this function is required when the FTApi are connected to an ASIA platform.

7.1.4 FTSendTransactionStatusEnum

The enumerated type:

```
typedef enum {  
    FTTransOK,  
    FTTransNO,  
    FTTransInvalSessionID,
```

```

        FTTransInvalidTID,
        FTTransPending,
        FTTransConversationClosed
    } FTSendTransactionStatusEnum;

```

will be used to indicate the acceptance or refusal of the transaction.

Values

FTTransOK	Transaction accepted ¹⁶ .
FTTransNO	Transaction aborted.
FTTransInvalidSessionID	Transaction aborted: internal error.
FTTransInvalidTID	Transaction aborted: an invalid TransactionID was given in <i>FTSendTransactionExt</i> or <i>FTSendTransaction</i>
FTTransPending	Transaction flying.
FTTransConversationClosed	Transaction flying: response of the transaction lost due to conversation closing.

7.1.5 FTSendTransactionResStruct

The structure:

```

typedef struct {
    FTSendTransactionStatusEnum    TransactionStatus;
    long                          MarketReasonCode;
    unsigned long                  ClassId;
    unsigned long                  KeyID;
    FTClassActionEnum              EntityAction;
    FTTransactionID                 TransactionID;
    FTTimeStamp                     TimeStamp
    long                          ResultClassID;
    void *                         ResultObject;
} FTSendTransactionResStruct;

```

¹⁶ With NAM market **FTTransOK** means the transaction is still flying and you have to make an explicit call to *FTStartTransactionMonitoring* to see if the transaction is still flying or it's aborted or committed.

With generic (no NAM) market **FTTransOK** means the transaction is committed: in this case you do not need to make an explicit call to *FTQueryTransactionStatus*.

will be used to return the result of the submission of a transaction.

Parameters

TransactionStatus	Indicates whether the transaction has been accepted or not.
MarketReasonCode	If <i>TransactionStatus</i> is <i>FTTransNO</i> then this field indicates the error that caused the transaction to fail. The exact meaning of this value is market-dependent, except for the following values: 10000 → internal error 10001 → not logged 10002 → inadequate privileges 10003 → invalid request action 10004 → invalid Transaction ID
ClassId	Class on which the transaction was sent.
KeyID	Key of the class on which the transaction was sent.
EntityAction	Requested action.
TransactionID	Transaction ID to which the structure refers.
TimeStamp	TimeStamp of the result. This field may be used only with <i>FTSendTransactionExt</i> .
ResultClassID	Class of the result referenced by <i>ResultObject</i> . This field may be used only with <i>FTSendTransactionExt</i> function when <i>ResClassRequired = FTTrue</i> . This field may be <i>NULL</i> if the server choose to not send the result.
ResultObject	Address of the result (of class <i>ResultClassID</i>). This field may be used only with <i>FTSendTransactionExt</i> function when <i>ResClassRequired = FTTrue</i> . This field may be <i>NULL</i> if the server choose to not send the result.

7.1.6 FTSendTransactionResProcExt

The type:

```
typedef void (*FTSendTransactionResProcExt) (
    long                    ConversationID,
    unsigned long           ReqID,
    FTSendTransactionResStruct * ResStruct);
```

defines the prototype of the function to define in order to be notified about the occurrence of the submission of a transaction.

Parameters

ConversationID	Conversation on which the transaction was sent and the response received.
ReqID	Same <i>ReqID</i> given in <i>FTSendTransactionExt</i>
ResStruct	Pointer to a structure indicating the outcome of the transaction sent.

Remarks

This prototype function *FTSendTransactionResProcExt* will be automatically invoked only if *FTSendTransactionExt* returned *FTOK*.

7.1.7 FTSendTransactionExt

The function:

```
long FTSendTransactionExt (
    long ConversationID,
    FTTransactionID TransactionID,
    FTClassActionEnum Action,
    unsigned long ClassID,
    unsigned long KeyID,
    void * MarketObject,
    FTSendTransactionResProcExt ResProc,
    long ResClassRequired,
    FTMask MarketObjectMask,
    unsigned long ReqID);
```

sends a transaction to the Server.

Parameters

ConversationID	An already open conversation.
TransactionID	Transaction ID, obtained with <i>FTMakeTransactionID</i> and not used as yet.
Action	Operation requested.
ClassID	Class on which an <i>Action</i> has to be carried out.
KeyID	Key used in the operation.
MarketObject	Pointer to a memory area containing an entity of <i>ClassID</i> : the transaction argument. If <i>MarketObjectMask</i> is not <i>NULL</i> then the memory area contains only the fields described by the <i>MarketObjectMask</i> mask.
ResProc	An <i>FTSendTransactionResProcExt</i> type function, which will be called subsequently to notify the acquisition by (and at the level of) the market server of the transaction.
ResClassRequired	<i>FTTrue</i> or <i>FTFalse</i> . If it is <i>FTTrue</i> then <i>ResultClassID</i> and <i>ResultClassObject</i> fields of <i>FTSendTransactionResStruct</i> may contain the result sent by the server. If it is <i>FTFalse</i> then <i>ResultClassID</i> and <i>ResultClassObject</i> fields of <i>FTSendTransactionResStruct</i> will be always equal to <i>NULL</i> .
MarketObjectMask	Mask that denotes which fields of class <i>ClassId</i> are available in <i>MarketObject</i> . The value <i>NULL</i> indicates that all fields are available.
ReqID	Value used to match this <i>FTSendTransactionExt</i> with the corresponding <i>ResProc</i> invocation.

Returned Value

FTOK	The transaction was sent correctly.
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid conversation.
FTInvalidClassIdError	Invalid class ID.
FTMemoryFullError	Insufficient memory.
FTInvalidRevision	A stale server don't implement this function.
FTInternalError	Internal error.

Remarks

If successful the function returns *FTOK*. In this case *ResProc* will be automatically invoked: the parameter *ResStruct* of *ResProc* will subsequently indicate (during *ResProc* invocation) whether the transaction has been correctly processed at later levels.

The transaction can be monitored after the acceptance of the market server, by using the function *FTQueryTransactionStatus* (as shown in the following sections).

It is no longer required to applicatively lock the function *FTSendTransactionExt* when sending transactions on *FT_C_ORDER/FT_C_QUOTE* in a multithread *FTApi* program.

7.2 Transaction Monitoring

To monitor the status of a transaction there are two available methods:

- *FTStartTransactionMonitoring* (on NAM markets connected with the OMI protocol),
- *FTQueryTransactionStatus* (on FastTrack markets connected with the FastTrack protocol).

7.2.1 FTTransStatusEnum

The enumerated type:

```
typedef enum {  
    FTFlying,  
    FTCommitted,  
    FTLABorted,  
    FTHAborted,  
    FTNotManaged  
} FTTransStatusEnum;
```

will be used to represent the status of a transaction sent to the electronic circuit.

Values

FTFlying	The transaction is in progress.
FTCommitted	The transaction has been successfully completed.
FTLAborted	The transaction was aborted.
FTHAborted	The transaction was aborted. In this case the field <i>NAMEError</i> of <i>FTTransactionStateStruct</i> may be inspected in order to see the reason.
FTNotManaged	The transaction cannot be monitored (this is possible only for particular market classes).

7.2.2 FTTransactionStateStruct

The structure:

```
typedef struct {
    unsigned long    ClassID;
    unsigned long    KeyID;
    FTClassActionEnum CA;
    FTTransactionID  TID;
    FTTransStatusEnum Status;
    unsigned long    MarketError;
    unsigned long    NAMEError;
    unsigned long    LocalError;
} FTTransactionStateStruct;
```

will be used to represent monitoring events coming from transactions sent to the Server.

Parameters

ClassID	Class of the transaction.
KeyID	Key of the transaction.
CA	Action of the transaction.
TID	Transaction ID.
Status	Status of sent transaction.
MarketError	Market error.
NAMEError	NAM error. This field is meaning only if <i>Status</i> is equal to <i>FTHAborted</i> .
LocalError	Local error.

Events notified via the monitoring function can be filtered by managing the Transaction ID.

7.2.3 FTTransactionMonitorProc

The type :

```
typedef void (*FTTransactionMonitorProc) (  
    long ConversationID  
    FTTransactionStateStruct * TransactionState);
```

defines the prototype of the function to define in order to receive the monitoring events of transactions relating to all levels of the market network.

Parameters

ConversationID	Conversation of the monitored transactions.
TransactionState	Pointer to the structure containing the current status of the transaction.

Remarks

This prototype function *FTTransactionMonitorProc* will be automatically invoked only if *FTStartTransactionMonitoring* returned *FTOK*.

7.2.4 FTStartTransactionMonitoring

The function:

```
long FTStartTransactionMonitoring (  
    long ConversationID,  
    FTTransactionMonitorProc MonitorProc);
```

sets a monitoring function on a conversation with a NAM market connected with the OMI protocol.

Parameters

ConversationID	Conversation on which transactions to monitoring will be sent.
MonitorProc	Monitoring function.

Returned Value

FTOK	Monitoring function active.
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid conversation.
FTInternalError	Internal error.

FTInvalidRevision	A stale server don't implement this function. ¹⁷
--------------------------	---

Remarks

If successful the function returns *FTOK*. In this case *MonitorProc* will be invoked in correspondence with each status variation of each transaction sent by Client with the ClientID, on the specified conversation.

Status variations in transactions that took place before the invocation of *FTStartTransactionMonitoring* will also be notified.

7.2.5 FTQueryTransStatusEnum

The enumerated type:

```
typedef enum {
    FTTransStatusOK,
    FTTransStatusNO,
    FTTransStatusInvalSessionID,
    FTTransStatusInvalidTID,
    FTTransStatusPending
} FTQueryTransStatusEnum;
```

will be used to represent the status of a query status transaction.

Values

FTTransStatusOK	The transaction has been successfully completed.
FTTransStatusNO	The transaction was aborted. The reason is described in the field <i>ReasonCode</i> of <i>FTQueryTransStatusStruct</i> .
FTTransStatusInvalSessionID	Internal error.
FTTransStatusInvalidTID	Invalid <i>TransID</i> in <i>FTQueryTransactionStatus</i> .
FTTransStatusPending	The transaction is in progress.

7.2.6 FTQueryTransStatusStruct

The structure:

```
typedef struct {
```

¹⁷ **FTInvalidRevision** is returned from a FastTrack (no NAM) market. In this case the *FTQueryTransactionStatus* must be used instead of *FTStartTransactionMonitoring*.

```

    FTQueryTransStatusEnum  Result;
    long                    ReasonCode;
    unsigned long           ClassID;
    unsigned long           KeyID;
    FTClassActionEnum       ClassAction;
    FTTimeStamp             TimeStamp;
    unsigned long           ResClassID;
    void *                  ResEntity;
} FTQueryTransStatusStruct;

```

will be used to return (via the *FTStartTransactionMonitoring* call) the result of the query of a transaction sent to a NAM market connected with the OMI protocol.

Parameters

Result	Indicates the result of <i>FTQueryTransactionStatus</i>
ReasonCode	If <i>Result</i> is <i>FTTransStatusNO</i> then this field indicates the error that caused the transaction to fail. For additiona info see the description of <i>MarketReasonCode</i> field in <i>FTSendTransactionResStruct</i> .
ClassId	Class on which the transaction was sent.
KeyID	Key of the class on which the transaction was sent.
ClassAction	Requested action.
TimeStamp	TimeStamp of the result.
ResClassID	Class of the result referenced by <i>ResEntity</i> . This field may be used only with <i>FTQueryTransactionStatus</i> function when <i>ResClassRequired</i> = <i>FTTrue</i> . This field may be <i>NULL</i> if the server choose to not send the result.
ResEntity	Address of the result (of class <i>ResEntity</i>). This field may be used only with <i>FTQueryTransactionStatus</i> function when <i>ResClassRequired</i> = <i>FTTrue</i> . This field may be <i>NULL</i> if the server choose to not send the result.

7.2.7 FTQueryTransStatusProc

The type :

```

typedef void (*FTQueryTransStatusProc) (
    long                    ConversationID,
    unsigned long           ReqID,
    FTTransactionID *       TransID,
    FTQueryTransStatusStruct * ResStruct);

```

defines the prototype of the notification function of the querying of a transaction.

Parameters

ConversationID	Same <i>ConversationID</i> given in <i>FTQueryTransactionStatus</i> .
ReqID	Same <i>ReqID</i> given in <i>FTQueryTransactionStatus</i> .
TransID	Same <i>TransID</i> given in <i>FTQueryTransactionStatus</i> .
ResStruct	Transaction status

Remarks

This prototype function *FTQueryTransStatusProc* will be automatically invoked only if *FTQueryTransactionStatus* returned *FTOK*.

7.2.8 FTQueryTransactionStatus

The function:

```
long FTQueryTransactionStatus(
    long                ConversationID,
    unsigned long       ReqID,
    FTTransactionID *   TransID,
    FTQueryTransStatusProc ResProc,
    long                ResClassRequired);
```

query the status of a transaction sent to a FastTrack market connected with the FastTrack protocol.

Parameters

ConversationID	Conversation on which transactions was sent.
ReqId	Value used to match this <i>FTQueryTransactionStatus</i> with the corresponding <i>ResProc</i> invocation.
TransID	Transaction ID.
ResProc	Query function with available results.
ResClassRequired	<i>FTTrue</i> or <i>FTFalse</i> . If it is <i>FTTrue</i> then <i>ResClassID</i> and <i>ResEntity</i> fields of <i>FTQueryTransStatusStruct</i> may contain the result sent by the server. If it is <i>FTFalse</i> then <i>ResClassID</i> and <i>ResEntity</i> fields of <i>FTQueryTransStatusStruct</i> will be always equal to <i>NULL</i> .

Returned Value

FTOK	<i>ResProc</i> will be called with available results.
FTInitError	Library not initialized.

FTInvalidConvIdError	Invalid conversation.
FTInvalidRevision	A stale server don't implement this function. ¹⁸
FTInternalError	Internal error.

Remarks

If successful the function returns *FTOK*. In this case *ResProc* will be invoked with available results.

¹⁸ **FTInvalidRevision** is returned from a NAM market connected with the OMI protocol. In this case the *FTStartTransactionMonitoring* must be used instead of *FTQueryTransactionStatus*.

8 QUERIES

A query is a client's request to the server to obtain a set of entities from its own Data Base.

To obtain this you have to:

- Create a query for a set of rows;
- Retrieve the entire results set (one row at time) or retrieve a specific subset (again one row at time);
- Destroy the query.

The library has the following functions to handling:

- **FTQueryCreate**
- **FTQueryRows**
- **FTQueryDestroy**

8.1 Query Create

8.1.1 FTQueryCreateEnum

The enumerated type:

```
typedef enum {  
    FTQueryOK,  
    FTQueryBadParameters,  
    FTQueryWrongQueryID,  
    FTQueryGenericError  
} FTQueryCreateEnum;
```

will be used to indicate the result of Query creation.

Values

FTQueryOK	Query accepted by FastTrack.
FTQueryBadParameters	Bad parameters used in <i>FTQueryCreate</i> .
FTQueryWrongQueryID	Bad <i>QueryID</i> in <i>FTQueryCreate</i> .
FTQueryGenericError	Others generic errors.

8.1.2 FTQueryCreateResStruct

The structure:

```
typedef struct {
    unsigned long    QueryKey;
    FTQueryCreateEnum Result;
    int              CanComputeRowNumber;
    unsigned long    RowNumber;
    unsigned long    TimeToLive;
    int              ResultSetFollows;
} FTQueryCreateResStruct;
```

will be used to return the result of a Query creation.

Parameters

QueryKey	Query identifier computed by the server. It may be used in <i>FTQueryRows</i> and <i>FTQueryDestroy</i>
Result	Result of Query creation
CanComputeRowNumber	<i>FTTrue</i> or <i>FTFalse</i> . If it is <i>FTTrue</i> then the <i>RowNumber</i> field is significant.
RowNumber	Number of rows in the result-set as compute by the server. It is significant only if <i>CanComputeRowNumber</i> = <i>FTTrue</i> .
TimeToLive	Interval time (in seconds) during which the server cache the result-set. During this interval the client may issue <i>FTQueryRows</i> calls to the server. It may be <i>NULL</i> if the server is unable to compute it.
ResultSetFollows	<i>FTTrue</i> or <i>FTFalse</i> . If it is <i>FTTrue</i> then the <i>FTQueryNotifyProc</i> will be automatically called N+1 times: N (≥ 0) times (with <i>EOQ</i> = <i>FTFalse</i>) for each of the N rows in the result-set; + 1 additional time (with <i>EOQ</i> = <i>FTTrue</i>) to indicate the end of the result-set. If it is <i>FTFalse</i> then the <i>FTQueryNotifyProc</i> will not be automatically called (as result of <i>FTQueryCreate</i>) and the client must issue a specific <i>FTQueryRows</i> to obtains a subset of the result-set.

Remarks

Apart from the *Result* field, all other fields are significant only if *Result* = *FTQueryOK*.
RowNumber is significant only if *CanComputeRowNumber* is *FTTrue*.

8.1.3 FTQueryCreateProc

The type:

```
typedef void (*FTQueryCreateProc) (
    long                ConversationID,
    unsigned long        ReqID,
    FTQueryCreateResStruct * ResStruct);
```

defines the prototype of the notification function of a query creation result.

Parameters

ConversationID	Same <i>ConversationID</i> given in <i>FTQueryCreate</i> .
ReqID	Same <i>ReqID</i> given in <i>FTQueryCreate</i> .
ResStruct	Results of query creation into the server.

Remarks

This prototype function *FTQueryCreateProc* will be automatically invoked only if *FTQueryCreate* returned *FTOK*. Only after this invocation additional N+1 automatic invocations of *FTQueryNotifyProc* may happen as described in the notes of *ResultSetFollows* field of *FTQueryCreateResStruct*.

8.1.4 FTQueryNotifyStruct

The structure:

```
typedef struct {
    unsigned long    QueryKey;
    unsigned long    RowNumber;
    long             ClassID;
    void *           Entity;
    unsigned long    ClassVersion;
    FTTimeStamp      TimeStamp;
    unsigned long    Delete;
    void *           Info;
    int              InfoSize;
    int              EOQ;
} FTQueryNotifyStruct;
```

is used to return one row of a result-set, as a result of a *FTQueryCreate* or *FTQueryRows*.

Parameters

QueryKey	Query identifier
RowNumber	Index (1-based) of this row in the result-set. This field is significant only if <i>EOQ = FTFalse</i> .
ClassID	Class of the <i>Entity</i> returned. This field is significant only if <i>EOQ = FTFalse</i> .
Entity	Pointer to the row returned (of class <i>ClassID</i> and index <i>RowNumber</i>) This field is significant only if <i>EOQ = FTFalse</i>
ClassVersion	Class version of last <i>Entity</i> modification. This field, together with <i>TimeStamp</i> , uniquely identifies the <i>Entity</i> time. This field is significant only if <i>EOQ = FTFalse</i> .
TimeStamp	Timestamp of last <i>Entity</i> modification. This field, together with <i>ClassVersion</i> , uniquely identifies the <i>Entity</i> time. This field is significant only if <i>EOQ = FTFalse</i> .
Delete	Indicates if the record is logically cancelled. This field is significant only if <i>EOQ = FTFalse</i> .
Info	The meaning of this field depends on the particular query made. Generally (on the most of queries) this field is unused. This field is significant only if <i>EOQ = FTFalse</i> .
InfoSize	The meaning of this field depends on the particular query made. Generally (on the most of queries) this field is unused. This field is significant only if <i>EOQ = FTFalse</i> .
EOQ	<i>FTTrue</i> to indicate the end of the result-set: no more invocation to <i>FTQueryNotifyProc</i> will be done. <i>FTFalse</i> to indicate the presence of a row (<i>Entity</i>) as described by the previous 7 fields.

8.1.5 FTQueryNotifyProc

The type:

```
typedef void (*FTQueryNotifyProc) (
    long                ConversationID,
    FTQueryNotifyStruct * QueryNotifyStruct);
```

defines the prototype of the notification function for each row of the result-set of a *FTQueryCreate* or *FTQueryRows*.

Parameters

ConversationID	Same <i>ConversationID</i> given in <i>FTQueryCreate</i> or <i>FTQueryRows</i> .
QueryNotifyStruct	Results of query creation into the server.

Remarks

In case of *FTQueryCreate* this prototype function *FTQueryNotifyProc* may be automatically invoked N+1 times as described in the notes of *ResultSetFollows* field of *FTQueryCreateResStruct*.

In case of *FTQueryRows* this prototype function *FTQueryNotifyProc* may be automatically invoked N+1 times as described in the remarks of *FTQueryRowsProc*.

8.1.6 FTQueryCreate

The function:

```
long FTQueryCreate(
    long                ConversationID,
    unsigned long       QueryID,
    unsigned long       ReqID,
    unsigned long       ClassID,
    void *              Entity,
    FTQueryCreateProc   QueryCreateProc,
    FTQueryNotifyProc   QueryNotifyProc);
```

Create a query into the server and eventually retrieve the entire corresponding result-set.

Parameters

ConversationID	Conversation on which create the query.
QueryID	Identifies the query into the server. This value must be agreed between the client and the server.
ReqID	Value used to match this <i>FTQueryCreate</i> with the corresponding <i>QueryCreateProc</i> invocation.
ClassID	Class of the <i>Entity</i>
Entity	Pointer to a <i>ClassID</i> instance.
QueryCreateProc	Notification function of server result correctness.
QueryNotifyProc	Notification function to access every single row of the result-set.

Returned Value

FTOK	<i>QueryRowsProc</i> will be called with server results eventually followed by N+1 invocations of <i>QueryNotifyProc</i>
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid conversation.
FTInvalidClassIdError	Invalid data class.
FTInvalidRevision	A stale server don't implement this function.
FTMemoryFullError	Insufficient memory.
FTInternalError	Internal error.

Remarks

If successful the function returns *FTOK*. In this case *QueryCreateProc* will be invoked with the result into the server and then *FTQueryNotifyProc* may be automatically invoked N+1 as described in the notes of *ResultSetFollows* field of *FTQueryCreateResStruct*

8.2 Query Rows

8.2.1 FTQueryRowsEnum

The enumerated type:

```
typedef enum {  
    FTQueryRowsOK,  
    FTQueryRowsWrongFirstRow,  
    FTQueryRowsWrongRowNumber,  
    FTQueryRowsWrongQueryID,  
    FTQueryRowsGenericError  
} FTQueryRowsEnum;
```

will be used to indicate the result of a *FTQueryRows*.

Values

FTQueryRowsOK	Query accepted by FastTrack.
FTQueryRowsWrongFirstRow	Bad <i>FirstRow</i> in <i>FTQueryRows</i> .
FTQueryRowsWrongRowNumber	Bad <i>RowNumber</i> in <i>FTQueryRows</i> .
FTQueryRowsWrongQueryID	Bad <i>QueryID</i> in <i>FTQueryRows</i> .
FTQueryRowsGenericError	Others generic errors.

8.2.2 FTQueryRowsProc

The type:

```
typedef void (*FTQueryRowsProc) (  
    long                ConversationID,  
    unsigned long       ReqID,  
    unsigned long       QueryKey,  
    FTQueryRowsEnum     Result);
```

defines the prototype of the notification function of a *FTQueryRows* result into the server.

Parameters

ConversationID	Same <i>ConversationID</i> given in <i>FTQueryRows</i> .
QueryID	Same <i>QueryID</i> given in <i>FTQueryRows</i> .
QueryKey	Same <i>QueryKey</i> given in <i>FTQueryRows</i> .
Result	Result of <i>FTQueryRows</i> into the server.

Remarks

This prototype function *FTQueryRowsProc* will be automatically invoked only if *FTQueryRows* returned *FTOK*.

If *Result* is *FTQueryRowsOK* then the *FTQueryNotifyProc* (registered with *FTQueryCreate*) will be automatically called N+1 times:

- N ($\leq \text{RowNumber}$) times (with *EOQ* = *FTFalse*) for each of the N rows in the subset of result-set;
- + 1 additional time (with *EOQ* = *FTTrue*) to indicate the end of the subset of result-set.

If *Result* is not *FTQueryRowsOK* then the *FTQueryNotifyProc* will never be called.

8.2.3 FTQueryRows

The function:

```
long FTQueryRows(
    long            ConversationID,
    unsigned long   QueryKey,
    unsigned long   ReqID,
    unsigned long   FirstRow,
    unsigned long   RowNumber,
    FTQueryRowsProc QueryRowsProc);
```

retrieve a subset of the result-set of a query from the server.

Parameters

ConversationID	Conversation on which retrieve the results.
QueryKey	Query to be used. Use the value returned in <i>FTQueryCreateResStruct</i> .
ReqID	Value used to match this <i>FTQueryRows</i> with the corresponding <i>QueryRowsProc</i> invocation.
FirstRow	Index (1-based) of the first row of the subset of result-set to be retrieved.
RowNumber	Number of rows of the subset of result-set to be retrieved
QueryRowsProc	Notification function of server result correctness.

Returned Value

FTOK	<i>QueryRowsProc</i> will be called with server results eventually followed by N+1 invocations of <i>QueryNotifyProc</i>
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid conversation.
FTInvalidRevision	A stale server don't implement this function.
FTInternalError	Internal error.

Remarks

This function may be invoked if *ResultSetFollows* is *FTFalse* in *FTQueryCreateResStruct*.

If successful the function returns *FTOK*. In this case *QueryRowsProc* will be invoked with the result into the server and then additional N+1 automatic invocations of *FTQueryNotifyProc* (registered with *FTQueryCreate*) may happen as described in the remarks of *FTQueryRowsProc*.

8.3 Query Destroy

8.3.1 FTQueryDestroyEnum

The enumerated type:

```
typedef enum {
    FTDestroyOK,
    FTDestroyWrongQueryID,
    FTDestroyError
} FTQueryDestroyEnum;
```

will be used to indicate the result of Query destruction.

Values

FTDestroyOK	Query accepted by FastTrack.
FTDestroyWrongQueryID	Bad <i>QueryID</i> in <i>FTQueryDestroy</i> .
FTDestroyError	Others generic errors.

8.3.2 FTQueryDestroyProc

The type:

```
typedef void (*FTQueryDestroyProc) (
    long ConversationID,
```

```

unsigned long      ReqID,
unsigned long      QueryKey,
FTQueryDestroyEnum Result);

```

defines the prototype of the notification function of a query destruction.

Parameters

ConversationID	Same <i>ConversationID</i> given in <i>FTQueryDestroys</i> .
ReqID	Same <i>ReqID</i> given in <i>FTQueryDestroys</i>
QueryID	Same <i>QueryID</i> given in <i>FTQueryDestroys</i> .
Result	Results of <i>FTQueryDestroys</i> into the server.

Remarks

This prototype function *FTQueryDestroyProc* will be automatically invoked only if *FTQueryDestroys* returned *FTOK*.

8.3.3 FTQueryDestroy

The function:

```

long FTQueryDestroy(
long      ConversationID,
unsigned long      QueryKey,
unsigned long      ReqID,
FTQueryDestroyProc QueryDestroyProc);

```

Destroy a query into the server.

Parameters

ConversationID	Conversation on which send the query destruction.
QueryKey	Query to be destroyed. Use the value returned in <i>FTQueryCreateResStruct</i> .
ReqID	Value used to match this <i>FTQueryDestroy</i> with the corresponding <i>QueryDestroyProc</i> invocation
QueryDestroyProc	Notification function of query destruction into the server.

Returned Value

FTOK	<i>QueryDestroyProc</i> will be called with the single result.
FTInitError	Library not initialized.
FTInvalidConvIdError	Invalid conversation.
FTInvalidRevision	A stale server don't implement this function.
FTInternalError	Internal error.

Remarks

If successful the function returns *FTOK*. In this case *QueryDestroyProc* will be invoked with the result of the query destruction into the server.

APPENDIX A: HEADER FILE FTAPI.H

Here there is the verbatim copy of <ftapi.h> include file.

```
#ifndef FTAPI_H
#define FTAPI_H

/*****
FTAPI
FastTrack Access API
c)2003-13 List SpA
*****/

/*****
Comments Tags:

deprecated : not to be used. In case of function Fun there is an equivalent
            FunExt available.
superseded : not to be used, using it will result in an error.
Vxyz      : added, starting from the x.y.z release version.
*****/

#if defined(WIN32) && !defined(NODLL)
    #ifdef FTDLL_EXPORTS
        #define FT_API __declspec(dllexport)
    #else
        #define FT_API __declspec(dllimport)
    #endif
#else
    #define FT_API
#endif

#define FT_OFFSET(Type, Field) ((unsigned long)&((Type*)0)->Field)

#define FTTrue          1
#define FTFalse         0

#define FTKEY_MAXLEN    (256)

/* Return codes */

#define FTOK              ( 0L)
#define FTDoubleInitError (-1L)
#define FTInternalError  (-2L)
#define FTInitError      (-3L)
#define FTCloseError     (-4L)
#define FTInvalidIPAddress (-5L)
#define FTInvalidIPPort  (-6L)
#define FTNoServer       (-7L)
#define FTTransportError (-8L)
#define FTInvalidUserType (-9L)
#define FTInvalidUserName (-10L)
#define FTInvalidPassword (-11L)
#define FTInvalidRevision (-12L)
#define FTInvalidCID      (-13L)
#define FTAlreadyLog      (-14L)
#define FTInvalidConvIdError (-15L)
#define FTGenericError    (-16L)
#define FTInvalidSubscribeTypeError (-17L)
#define FTInvalidClassIDError (-18L)
#define FTDoubleSubscription (-19L)
#define FTNoSubscribedClassError (-20L)
```

```

#define FTInvalidConfigurationKey      (-21L)
#define FTMemoryFullError              (-22L)
#define FTInvalidServerStatus          (-23L)
#define FTInvalidArgument              (-24L)
#define FTUnresolvedService            (-25L)
#define FTInvalidMask                  (-26L)
#define FTExceedSession                (-27L)
#define FTInvalidProfile                (-28L)
#define FTInvalidFilter                 (-29L)
#define FTAccountNotActive              (-30L)      /* V323 */
#define FTTooManyTradersConnected      (-31L)      /* V323 */
#define FTPasswordExpired               (-32L)      /* V323 */
#define FTNotPrivilegeChangePassword   (-33L)      /* V323 */
#define FTNewPasswordRepeated           (-34L)      /* V323 */
#define FTInsufficientNewPasswordLength (-35L)      /* V323 */
#define FTInvalidNewPasswordCharacters (-36L)      /* V323 */
#define FTNewPasswordTooMuchEasy        (-37L)      /* V323 */
#define FTUnsupportedOperation          (-38L)      /* V330 */
#define FTSupersededOperation           (-39L)      /* V350 */

/* Trace Options */

#define FTTRACE_OPT_None                (0)
#define FTTRACE_OPT_APIInOut            (1<<0)
#define FTTRACE_OPT_FlowControl         (1<<1)
#define FTTRACE_OPT_DataTransfert      (1<<2)
#define FTTRACE_OPT_DataIn             (1<<3)
#define FTTRACE_OPT_DataOut            (1<<4)
#define FTTRACE_OPT_Application        (1<<5)      /* V310 */

/* Filter sizes */

#define FT_FILTERDEF_MAXLEN             (1024)
#define FT_FILTERVAL_MAXLEN             (6144)

/* FTContext, conversation attributes */

#define CTX_COMPRESSED                   (1)
#define CTX_APPL_AUTHKEY                 (2)
#define CTX_APPL_AUTHFILE                (3)
#define CTX_X509_CERTIFICATE             (4)
#define CTX_CONNECTION_TIMEOUT           (5)      /* V310 */
#define CTX_CONNECTION_PROXYSTRING       (6)      /* V316 */
#define CTX_CONNECTION_NEWPASSWORD       (7)      /* V322 */

#define CTX_ALTERNATIVE_HOST1            (111)
#define CTX_ALTERNATIVE_PORT1           (112)
#define CTX_ALTERNATIVE_HOST2           (121)
#define CTX_ALTERNATIVE_PORT2           (122)
#define CTX_ALTERNATIVE_HOST3           (131)
#define CTX_ALTERNATIVE_PORT3           (132)

/* FTContext, initialization attributes */

#define CTX_INIT_MAX_BLOCK_LEN           (1000)

/* -----
ENUMS
----- */

typedef enum {
    FTLevel_Undef,
    FTLevel_Full,
    FTLevel_Normal,
    FTLevel_Warning,
    FTLevel_Error
} FTTraceLevelEnum;

typedef enum {
    FTSource_Unused,

```

```

    FTSource_FTP,
    FTSource_API,
    FTSource_FlowControl,
    FTSource_DataTransfert,
    FTSource_Application /* V310 */
} FTTraceSourceEnum;

typedef enum {
    FTUserUnused,
    FTUserTrader,
    FTUserAutoTrader,
    FTUserMonitor,
    FTUserView,
    FTUserController,
    FTUserMasterSlave
} FTUserTypeEnum;

typedef enum {
    FTConnectionLost,
    FTConnectionClosed
} FTBrokenTypeEnum;

typedef enum {
    FTSubscribeStart,
    FTSubscribeStop
} FTSubscribeStatusEnum;

typedef enum {
    FTAckUnrequired,
    FTAckRequired
} FTCommunicationModeEnum;

typedef enum {
    FTFlowLast,
    FTFlowAll,
    FTZeroMaskedFlowLast,
    FTZeroMaskedFlowAll,
    FTDiffMaskedFlowAll /* V321 */
} FTSubscribeFlowEnum;

typedef enum {
    FTQueryAll,
    FTQuerySet,
    FTQueryEQ, /* deprecated */
    FTQueryLT, /* deprecated */
    FTQueryGT, /* deprecated */
    FTQueryBW, /* deprecated */
    FTQueryPast,
    FTQueryOnTime
} FTQueryTypeEnum;

typedef enum {
    FTEntityADD,
    FTEntityDEL,
    FTEntityRWT,
    FTEntityKIL
} FTClassActionEnum;

typedef enum {
    FTClassInstance,
    FTConnectionBroken,
    FTClassIdle
} FTEventTypeEnum;

typedef enum {
    FTFlying,
    FTCommitted,
    FTAborted,
    FTHAborted,
    FTNotManaged
} FTTransStatusEnum;

typedef enum {
    FTTransOK,
    FTTransNO,

```

```

    FTTransInvalSessionID,
    FTTransInvalidTID,
    FTTransPending,
    FTTransConversationClosed
} FTSendTransactionStatusEnum;

/* enum types */
typedef enum {
    FTFilterCreateOK,
    FTFilterCreateSyntaxError,
    FTFilterCreateInvalidFilterLen,
    FTFilterCreateInvalidClassID,
    FTFilterCreateInvalidFilterType,
    FTFilterCreateNotImplemented
} FTFilterCreateEnum;

typedef enum {
    FTFilterDestroyOK,
    FTFilterDestroyInvalidFilterID
} FTFilterDestroyEnum;

typedef enum {
    FTFilterSetOK,
    FTFilterSetSyntaxError,
    FTFilterSetInvalidFilterID,
    FTFilterSetInvalidFilterLen,
    FTFilterSetAlreadySet
} FTFilterSetEnum;

typedef enum {
    FTQueryOK,
    FTQueryBadParameters,
    FTQueryWrongQueryID,
    FTQueryGenericError
} FTQueryCreateEnum;

typedef enum {
    FTQueryRowsOK,
    FTQueryRowsWrongFirstRow,
    FTQueryRowsWrongRowNumber,
    FTQueryRowsWrongQueryID,
    FTQueryRowsGenericError
} FTQueryRowsEnum;

typedef enum {
    FTDestroyOK,
    FTDestroyWrongQueryID,
    FTDestroyError
} FTQueryDestroyEnum;

typedef enum {
    FTQueryRecOK,
    FTQueryRecNO,
    FTQueryRecGenericError
} FTQueryRecordEnum;

typedef enum {
    FTTransStatusOK,
    FTTransStatusNO,
    FTTransStatusInvalSessionID,
    FTTransStatusInvalidTID,
    FTTransStatusPending
} FTQueryTransStatusEnum;

/* -----
Data Types
----- */

typedef void * FTContext;
typedef void * FTMask;
typedef void * FTDiffMask;

/* V321 */

/* -----

```

```

Structures
----- */
typedef unsigned long FTTimeStamp[2];
typedef unsigned long FTRevision[3];

typedef struct{
    char buffer[FTKEY_MAXLEN];
    long len;
} FTKeyValue;

typedef struct {
    char *UserName;
    unsigned long BusinessServiceID;
    unsigned long ClientServiceID;
    unsigned long ClientID;
    unsigned long SystemDate;
    unsigned long SystemTime;
    FTRevision MarketRevision;
    unsigned long ReqID;
    unsigned long FTID;
    unsigned long Environment;
} FTOpenResStruct;

typedef struct {
    unsigned long ClassID;
    FTSubscribeStatusEnum SubscribeStatus;
    int ResetClass;
    unsigned long ClassVersion;
    FTSubscribeFlowEnum SubscribeFlowEnum;
    FTQueryTypeEnum QueryType;
    long SubscriptionID;
} FTSubscribeResStruct;

typedef struct {
    FTClassActionEnum EntityAction;
    unsigned long ClassID;
    long KeyID;
    FTTimeStamp TimeStamp;
    void *MarketObject;
    unsigned long LenMarketObject;
    long SubscriptionID;
    int IsMasked;
    FTDiffMask DiffMask; /* V321 */
} FTNotifySubscribeResStruct;

typedef struct {
    unsigned long BusinessServiceID;
    unsigned long ClientServiceID;
    unsigned long ClientID;
    unsigned long TimeStamp[2];
} FTTransactionID;

typedef struct {
    FTSendTransactionStatusEnum TransactionStatus;
    long MarketReasonCode;
    unsigned long ClassID;
    unsigned long KeyID;
    FTClassActionEnum EntityAction;
    FTTransactionID TransactionID;
    FTTimeStamp TimeStamp;
    long ResultClassID;
    void *ResultObject;
} FTSendTransactionResStruct;

typedef struct {
    unsigned long ClassID;
    unsigned long KeyID;
    FTClassActionEnum CA;
    FTTransactionID TID;
    FTTransStatusEnum Status;
    unsigned long MarketError;
    unsigned long NAMEError;
    unsigned long LocalError;
} FTTransactionStateStruct;

```



```

/* struct types */

typedef struct {
    unsigned long      QueryKey;
    FTQueryCreateEnum  Result;
    int                CanComputeRowNumber;
    unsigned long      RowNumber;
    unsigned long      TimeToLive;
    int                ResultSetFollows;
} FTQueryCreateResStruct;

typedef struct {
    unsigned long      QueryKey;
    unsigned long      RowNumber;
    long               ClassID;
    void *             Entity;
    unsigned long      ClassVersion; /* V308 */
    FTTimeStamp        TimeStamp; /* V308 */
    unsigned long      Delete; /* V328 */
    void *             *Info; /* V308 */
    int                InfoSize; /* V308 */
    int                EOQ;
} FTQueryNotifyStruct;

typedef struct {
    FTQueryTransStatusEnum Result;
    long                   ReasonCode;
    unsigned long          ClassID;
    unsigned long          KeyID;
    FTClassActionEnum      ClassAction;
    FTTimeStamp            TimeStamp;
    unsigned long          ResClassID;
    void *                 ResEntity;
} FTQueryTransStatusStruct;

/* -----
Function types
----- */

typedef void (*FTLibraryTraceProc) (long EventInfo, char *Message);
typedef void (*FTNotifySubscribeResProc) (long ConversationID, FTEventTypeEnum EventType,
    FTNotifySubscribeResStruct *ResStruct); /* superseded since V350 */
typedef void (*FTSubscribeResProc) (long ConversationID, long ResultCode,
    FTSubscribeResStruct *ResStruct); /* superseded since V350 */
typedef void (*FTOpenConversResProc) (long ConversationID, long ResultCode, FTOpenResStruct
    *ResStruct);
typedef void (*FTNotifyBrokenProc) (long ConversationID, FTBrokenTypeEnum BrokenType);
typedef void (*FTSendTransactionResProc) (long ConversationID, FTSendTransactionResStruct
    *ResStruct); /* superseded since V350 */
typedef void (*FTTransactionMonitorProc) (long ConversationID, FTTransactionStateStruct
    *TransactionState);
typedef void (*FTConversationProc) (long ConversationID, int IsWritePending);
typedef void (*FTConversationSocketProc) (long ConversationID, int IsWritePending, long
    Socket);

/* function types */
typedef void (*FTNotifySubscribeResProcExt) (long ConversationID, unsigned long ReqID,
    FTEventTypeEnum EventType, FTNotifySubscribeResStruct *ResStruct);
typedef void (*FTSubscribeResProcExt) (long ConversationID, unsigned long ReqID, long
    ResultCode, FTSubscribeResStruct *ResStruct);
typedef void (*FTFilterCreateProc) (long ConversationID, unsigned long ReqID, long
    FilterID, FTFilterCreateEnum Res);
typedef void (*FTFilterSetProc) (long ConversationID, unsigned long ReqID, long
    FilterID, FTFilterSetEnum Res);
typedef void (*FTFilterDestroyProc) (long ConversationID, unsigned long ReqID, long
    FilterID, FTFilterDestroyEnum Res);
typedef void (*FTQueryCreateProc) (long ConversationID, unsigned long ReqID,
    FTQueryCreateResStruct *ResStruct);
typedef void (*FTQueryNotifyProc) (long ConversationID, FTQueryNotifyStruct
    *QueryNotifyStruct);
typedef void (*FTQueryRowsProc) (long ConversationID, unsigned long ReqID, unsigned
    long QueryKey, FTQueryRowsEnum Res);
typedef void (*FTQueryDestroyProc) (long ConversationID, unsigned long ReqID, unsigned

```

```

    long QueryKey, FTQueryDestroyEnum Res);
typedef void (*FTSendTransactionResProcExt) (long ConversationID, unsigned long ReqID,
    FTSendTransactionResStruct *ResStruct);
typedef void (*FTQueryTransStatusProc) (long ConversationID, unsigned long ReqID,
    FTTransactionID *TransID, FTQueryTransStatusStruct *ResStruct);

/* -----
Functions
----- */

FT_API long FTSetTraceMode (FTLibraryTraceProc          TraceProc,
    FTTraceLevelEnum      Level,
    long                   Options,
    int                    WriteFile,
    int                    OldLogs,
    int                    Flush);

FT_API void FTTraceAppl (FTTraceLevelEnum      level,
    char *          fmt,
    ...);          /* V310. Up to 10Kb
    messages */

FT_API long FTSetLicence ( char *          Key);
/* superseded since V350 */
FT_API long FTLoadLicence ( char *          KeyFile);
/* superseded since V350 */
FT_API long FTInit ( FTLibraryTraceProc      TraceProc);
/* superseded since V350 */
FT_API long FTEndClass ( void *          Sk);
FT_API long FTEndClassByLibrary ( char *          Path,
    char *          LibraryName,
    char **         LibraryVersion); /* V310 */

FT_API long FTStart ( void);
FT_API long FTShutdown ( void);
FT_API long FTRun ( void);
FT_API char *FTGetLibraryVersion ( void);
FT_API char *FTGetErrorString ( long          ErrCode);
FT_API int FTCompareTS ( FTTimeStamp          TS1,
    FTTimeStamp          TS2);

FT_API long FTGetTraceInfo ( long          EventInfo,
    FTTraceSourceEnum *   Source,
    FTTraceLevelEnum *   Level);

FT_API long FTOpenConversation ( char *          IPAddress,
    unsigned short        IPPort,
    FTUserTypeEnum         UserType,
    unsigned long         ClientID,
    char *                UserName,
    char *                Password,
    FTRevision            ApplRevision,
    unsigned long         ApplSignature,
    FTOpenConversResProc  OpenResProc,
    FTOpenResStruct *     ResStruct,
    FTNotifyBrokenProc     BrokenProc); /*
    superseded since V350 */

FT_API long FTCloseConversation ( long          ConversationID);

FT_API long FTGetConversationSocket ( long          ConversationID);
FT_API long FTEnumConversation ( FTConversationProc      ConvProc);

FT_API long FTMakeKeyValue ( long          ClassID,
    void *          Entity,
    long            KeyId,
    long            NumFields,
    FTKeyValue *    KeyValue);
FT_API long FTStartSubscribeClass ( long          ConversationID,
    FTCommunicationModeEnum      CommunicationMode,
    FTSubscribeFlowEnum          SubscribeFlow,
    unsigned long                ClassID,
    unsigned long                ClassVersion,
    FTTimeStamp                  ClassTimeStamp,
    long                          NotifyConversationID,

```

```

        FTQueryTypeEnum          QueryType,
        unsigned long            KeyID,
        FTKeyValue *             KeyValue1,
        FTKeyValue *             KeyValue2,
        FTSubscribeResProc       SubscrResProc,
        FTSubscribeResStruct *    ResStruct,
        FTNotifySubscribeResProc NotifyResProc); /*
    superseded since V350 */

FT_API long FTStopSubscribeClass (long ConversationID,
        unsigned long ClassID,
        long SubscriptionID);

FT_API long FTStartTransactionMonitoring( long
    ConversationID,
        FTTransactionMonitorProc MonitorProc);
FT_API long FTMakeTransactionID (long ConversationID,
        FTTransactionID * TransactionID);
FT_API long FTSendTransaction (long ConversationID,
        FTTransactionID TransactionID,
        FTClassActionEnum Action,
        unsigned long ClassID,
        unsigned long KeyID,
        void * MarketObject,
        FTSendTransactionResProc ResProc,
        FTSendTransactionResStruct * ResStruct); /*
    superseded since V350 */

FT_API FTContext FTCreateContext (void);
FT_API long FTSetContextAttribute (FTContext Context,
        long Attribute,
        char * Value);
FT_API void FTDestroyContext (FTContext Context);

FT_API long FTOpenConversationExt (char * IPAddress,
        unsigned short IPPort,
        FTUserTypeEnum UserType,
        unsigned long ClientID,
        char * UserName,
        char * Password,
        FTRevision ApplRevision,
        unsigned long ApplSignature,
        FTOpenConversResProc OpenResProc,
        FTNotifyBrokenProc BrokenProc,
        unsigned long ReqID,
        char * Service,
        FTContext Context);

FT_API FTMask FTCreateMask (unsigned long ClassID);
FT_API void FTDestroyMask (FTMask Mask);
FT_API long FTAddFieldToMask (FTMask Mask,
        char * Field);

FT_API long FTInitExt (FTLibraryTraceProc TraceProc,
        FTContext Context);

FT_API long FTStartSubscribeClassExt (long ConversationID,
        FTCommunicationModeEnum CommunicationMode,
        FTSubscribeFlowEnum SubscribeFlow,
        unsigned long ClassID,
        unsigned long ClassVersion,
        FTTimeStamp ClassTimeStamp,
        FTQueryTypeEnum QueryType,
        unsigned long KeyID,
        FTKeyValue * KeyValue1,
        FTKeyValue * KeyValue2,
        FTSubscribeResProcExt SubscribeResProc,
        FTNotifySubscribeResProcExt NotifyResProc,
        unsigned long ReqID,
        unsigned long FilterID,
        FTMask SubscrMask);

FT_API long FTRefreshEntity (long ConversationID,
        unsigned long ClassID,
        long SubscriptionID,
        void * Entity,

```

```

                                unsigned long      KeyID);

FT_API long FTFilterCreate      ( long      ConversationID,
                                unsigned long ReqID,
                                unsigned long ClassID,
                                unsigned long FilterType,
                                unsigned long FilterDefinitionLen,
                                char *      FilterDefinition, /* max len:
                                FT_FILTERDEF_MAXLEN */
                                FTFilterCreateProc FilterCreateProc);
FT_API long FTFilterSet        ( long      ConversationID,
                                unsigned long FilterID,
                                unsigned long ReqID,
                                unsigned long FilterLen,
                                char *      FilterVal, /*
                                max len: FT_FILTERVAL_MAXLEN */
                                FTFilterSetProc FilterSetProc);
FT_API long FTFilterDestroy    ( long      ConversationID,
                                unsigned long FilterID,
                                unsigned long ReqID,
                                FTFilterDestroyProc FilterDestroyProc);

FT_API long FTQueryCreate      ( long      ConversationID,
                                unsigned long QueryID,
                                unsigned long ReqID,
                                unsigned long ClassID,
                                void *      Entity,
                                FTQueryCreateProc QueryCreateProc,
                                FTQueryNotifyProc QueryNotifyProc);
FT_API long FTQueryDestroy    ( long      ConversationID,
                                unsigned long QueryKey,
                                unsigned long ReqID,
                                FTQueryDestroyProc QueryDestroyProc);
FT_API long FTQueryRows       ( long      ConversationID,
                                unsigned long QueryKey,
                                unsigned long ReqID,
                                unsigned long FirstRow,
                                unsigned long RowNumber,
                                FTQueryRowsProc QueryRowsProc);

FT_API long FTSendTransactionExt ( long      ConversationID,
                                FTTransactionID * TransID,
                                FTClassActionEnum Action,
                                unsigned long ClassID,
                                unsigned long KeyID,
                                void *      MarketObject,
                                FTSendTransactionResProcExt ResProc,
                                long      ResClassRequired,
                                FTMask     MarketObjectMask,
                                unsigned long ReqID);
FT_API long FTQueryTransactionStatus ( long      ConversationID,
                                unsigned long ReqID,
                                FTTransactionID * TransID,
                                FTQueryTransStatusProc ResProc,
                                long      ResClassRequired);

FT_API long FTIsFieldChanged    ( unsigned long      Offset,
                                /* v321 */
                                FTDiffMask DiffMask);
FT_API long FTUpdateMarketObject ( char      *CurrentMarketObject, /*
                                v321 */
                                char      *DiffMarketObject,
                                FTDiffMask DiffMask);

FT_API long FTConversationSocketSelect ( long      ConversationID);
/* v330 */
FT_API long FTSocketSelect      ( void);
/* v330 */
FT_API void FTSetSelectTimeout  ( long Sec,
                                long uSec);
/* v330 */

FT_API long FTMakeTransactionIDExt ( long      ConversationID,
                                unsigned long ClassID,
                                FTTransactionID * TransactionID);
/*

```

```
    v330 */  
FT_API long FTEnumConversationSocket ( FTConversationSocketProc      ConvSocketProc );  
    /* v332 */  
#endif
```

APPENDIX B: FASTTRACK EXAMPLE: GET ORDER

This example shows how to use FT/API library to connect a FastTrack system.

The example subscribes some data classes such as Order.

In order to work with this example you need to configure the `ft.cfg` file including: host, port, hostAlter1, portAlter1, username, password (that will be used for the login) separated by newlines, for example:

```
ft_machine
12000
ft_machine_alternative
12000
jack
*
```

To build an executable binary you must compile this file and link it along with the libraries: `ftapi` and `ftmetamarket`.

As an alternative you can substitute the include

```
#include "ftmetamarket.h"
```

and the call

```
res = FTEndClass(InitSkeletonmetamarket());
```

with the following include

```
#include "ftlmetamarket.h"
```

and the following call

```
res = FTEndClassByLibrary(0, "ftlmetamarket", 0);
```

In this case you need not to link this file with the `ftmetamarket` static library, but you need to have, at run-time, the shared `ftlmetamarket.dll` library in an appropriate place.

As an exercise for the reader you can delete the definitions of `DebugMsg` and `DebugInFile` functions and substitute all `DebugMsg` calls with a corresponding `FTTraceAppl` call.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <stdarg.h>
#include <memory.h>
#include <time.h>

#include "ftapi.h"
#include "ftmetamarket.h"

#ifdef ALPHA
#include <netdb.h>
#include <fcntl.h>
#endif

#ifdef WIN32
#include "winsock.h"
#else
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#endif

#ifdef AX42
#define _POSIX_SOURCE
#include <sys/types.h>
#include <sys/select.h>
#endif

static int Abort = 0;

static FTRevision MyVersion = {1,0,0};

typedef enum {
    Sconnected,
    WaitOnConversation,
    Broken,
    Connected
} ConnectionStatus;

typedef struct {
    ConnectionStatus Status;
    long ConversationID;
    int Socket;
} MarketInfo;

static MarketInfo MetamarketInfo;

static void MarketInfoReset(MarketInfo *mi)
{
    memset((char*)mi, 0, sizeof(MarketInfo));
    mi->Status=Sconnected;
    mi->ConversationID=-1;
    mi->Socket=-1;
}

static void DebugInFile(char *msg) {
    FILE *pf;
    static int first = 1;

    pf = fopen("ftexamp.log",first? "w" : "a");
    fprintf(pf, "%s\n", msg);
    fclose(pf);
    first = 0;
}

static void DebugMsg(char *fmt,...)
{
    static char buffer[1024];
    va_list ap;
```

```

    va_start(ap,fmt);
    vsprintf(buffer, fmt, ap);
    va_end(ap);

    DebugInFile(buffer);
    printf("%s\n", buffer);
}

static void UserAbort(int dummy)
{
    DebugMsg(" *** USER ABORT ***");
    Abort = 1;
}

static int GetConfig(char *filename,
                    char *IPAddress, unsigned short *PortNum,
                    char *IPAlternative, char *PortAlternative,
                    char *UserName, char *UserPasswd)
{
    FILE *f;
    int port;

    if ((f = fopen(filename,"r"))) {
        fscanf(f, "%s%d%s%s%s", IPAddress, &port,
              IPAlternative, PortAlternative, UserName, UserPasswd);
        fclose(f);
        *PortNum = (unsigned short)port;
        return 0;
    } else
        return -1;
}

/*****
    SUBSCRIBE HANDLER
*****/

static void SubscribeResProc (long convID, unsigned long ReqID,
                             long ResultCode, FTSubscribeResStruct *resStruct)
{
    DebugMsg("=====> FTOpenSubscribeResProc: classID [%lu], SubscriptionID [%lu]",
            resStruct->ClassID, resStruct->SubscriptionID);
    DebugMsg(" - Result: %s", FTGetErrorString(ResultCode));

    if (ResultCode == FTOK){
        DebugMsg(" - Status: %s",
            (resStruct->SubscribeStatus==FTSubscribeStart)? "Start" : "Stop");
        if (resStruct->SubscribeStatus==FTSubscribeStart)
            DebugMsg(" - New Table version: %lu; To-Reset signal: %d",
                resStruct->ClassVersion, resStruct->ResetClass);
    }
    if( convID==MetamarketInfo.ConversationID )
        DebugMsg(" - ConvID: %d (Metamarket)", convID);
}

/*****
    MESSAGE HANDLER
*****/

static void IncomingOrder(FT_C_ORDERPtr Order)
{
    DebugMsg(" -> Order: %s", Order->OrderID);
}

static void IncomingMarket(FT_C_MARKETPtr Market)
{
    DebugMsg(" -> Market: %s", Market->MarketID);
}

static void NotifyClass(long
                        convID,
```



```

        unsigned long          ReqID,
        FTEventTypeEnum          EventType,
        FTNotifySubscribeResStruct * ResStruct)
{
    DebugMsg("=====> NotifyClass: Conversation [%d], ReqID [%d], Class [%d], SubscriptionID
[%d]",
        convID, ReqID, ResStruct->ClassID, ResStruct->SubscriptionID);

    switch (EventType) {
        case FTConnectionBroken:
            DebugMsg(" - Event: ConnectionBroken");
            break;
        case FTClassIdle:
            DebugMsg(" - Event: Idle signal");
            break;
        case FTClassInstance:
            DebugMsg(" - Event: ClassInstance: TimeStamp (%lu, %lu)",
                ResStruct->TimeStamp[0], ResStruct->TimeStamp[1]);
            switch (ResStruct->EntityAction) {
                case FTEntityKIL:
                    if (ResStruct->KeyID <= 0 && !ResStruct->MarketObject)
                        DebugMsg(" - Action: Class Reset Signal - New version: %d",
                            (ResStruct->TimeStamp)[0]);
                    else
                        DebugMsg(" - Action: EntityKIL - KeyID [%d]",
                            ResStruct->KeyID);
                    break;
                case FTEntityDEL:
                    DebugMsg(" - Action: EntityDEL - KeyID [%d]",
                        ResStruct->KeyID);
                    break;
                case FTEntityADD:
                case FTEntityRWT:
                    DebugMsg(" - Action: EntityADD/RWT - KeyID [%d]",
                        ResStruct->KeyID);
                    switch ((int)ResStruct->ClassID) {
                        case FT_C_ORDER_ID:
                            IncomingOrder((FT_C_ORDERPtr)ResStruct->MarketObject);
                            break;
                        case FT_C_MARKET_ID:
                            IncomingMarket((FT_C_MARKETPtr)ResStruct->MarketObject);
                            break;
                        default:
                            DebugMsg(" - ?: Entity Not managed ");
                            break;
                    }
                    break;
            }
            break;
        default:
            DebugMsg(" - Event: ?");
            break;
    }
}

/*****
OPEN CONV. HANDLER
*****/

static void OpenProc(long convID, long ResultCode, FTOpenResStruct *ResStruct)
{
    int res;

    DebugMsg("=====> OpenProc: Login [%d - %s]",
        ResultCode, FTGetErrorString(ResultCode));
    if (ResultCode == FTOK) {
        MetamarketInfo.Status = Connected;

        DebugMsg(" - Market Revision: V%d.%d.%d",
            ResStruct->MarketRevision[0],
            ResStruct->MarketRevision[1],
            ResStruct->MarketRevision[2]);
    }
}

```

```

        res = FTStartSubscribeClassExt(convID, FTAckUnrequired, FTFlowAll,
                                      FT_C_ORDER_ID, 0, 0, FTQueryAll, 0, 0, 0,
                                      SubscribeResProc, NotifyClass, 1, 0, 0);
        DebugMsg("FTStartSubscribeClassExt[FT_C_ORDER_ID]: [%d], [%s]",
                res, FTGetErrorString(res));

        res = FTStartSubscribeClassExt(convID, FTAckUnrequired, FTFlowAll,
                                      FT_C_MARKET_ID, 0, 0, FTQueryAll, 0, 0, 0,
                                      SubscribeResProc, NotifyClass, 2, 0, 0);
        DebugMsg("FTStartSubscribeClassExt[FT_C_MARKET_ID]: [%d], [%s]",
                res, FTGetErrorString(res));
    }
}

/*****
    CLOSING HANDLE
*****/

static void BrokenProc (long conversationID, FTBrokenTypeEnum brokenType)
{
    DebugMsg("=====> BrokenProc: %s", (brokenType==FTConnectionClosed)?
            "FTConnectionClosed" : "FTConnectionLost" );
    if(conversationID==MetamarketInfo.ConversationID){
        DebugMsg("    - Broken Detected on FT conversation: %d", conversationID);
        MarketInfoReset(&MetamarketInfo);
        MetamarketInfo.Status = Broken;
        /* TryToLink(); */
    }
}

```

```

/*****
CONNECTION OPENING
*****/

static void TryToLink()
{
    long            convID=-1;
    unsigned short  PortNum;
    char            IPAddress[128], IPAlternative[128], PortAlternative[128],
                    UserName[128], UserPasswd[128];
    FTContext       ctx;

    if (!GetConfig("ft.cfg", IPAddress, &PortNum,
                  IPAlternative, PortAlternative, UserName, UserPasswd)){
        DebugMsg("Connection Params Used: [%s] [%d] [%s] [%s] [%s] [%s]",
                  IPAddress, PortNum, IPAlternative, PortAlternative,
                  UserName, UserPasswd);
        ctx = FTCreateContext();
        if(! ctx ||
            FTSetContextAttribute(ctx, CTX_ALTERNATIVE_HOST1, PortAlternative) != FTOK ||
            FTSetContextAttribute(ctx, CTX_ALTERNATIVE_PORT1, PortAlternative) != FTOK)
            DebugMsg("Cannot create or update context");
        convID = FTOpenConversationExt(IPAddress, PortNum, FTUserTrader, 520,
                                       UserName, UserPasswd, MyVersion, 0,
                                       OpenProc, BrokenProc, 0, 0, ctx);

        FTDestroyContext(ctx);
        DebugMsg("FTOpenConversationExt: %d, [%s]", convID, FTGetErrorString(convID));
        if (convID>=0){
            MetamarketInfo.Status = WaitOnConversation;
            MetamarketInfo.ConversationID = convID;
            MetamarketInfo.Socket=(int)FTGetConversationSocket(convID);
            DebugMsg("Connection Socket: [%d]",MetamarketInfo.Socket);
        }
    }
    else
        DebugMsg("Cannot read cfg file");
}

/*****
SOCKETS MANAGEMENT
*****/

static fd_set readfds, writefds;
static int NumConv = 0;

static void ConversationProc(long ConvID, int isWritePending)
{
    const long socket = FTGetConversationSocket(ConvID);

    FD_SET(socket, &readfds);
    if (isWritePending)
        FD_SET(socket, &writefds);

    NumConv++;
}

static void MarketWait()
{
    int res;
    struct timeval max_sleep;

    FD_ZERO(&readfds);
    FD_ZERO(&writefds);

    NumConv = 0;
    FTEnumConversation(ConversationProc);
#ifdef NT
    if (!NumConv){
        Sleep(10);
        DebugMsg("No Connection");
        return;
    }
#endif
}

```

```

max_sleep.tv_sec    = 3;
max_sleep.tv_usec   = 0;

res = select(FD_SETSIZE, (void *)&readfds, (void *)&writefds, 0, &max_sleep);
if (res<0)
    perror("select");
if (res==0)
    DebugMsg("*** select: Time-Out ***");
}

/*****
MAIN
*****/

int main(int argc, char *argv[])
{
    long res = -1;
    int opt = FTTRACE_OPT_None;

    /* possible setting of tracing feature */
    opt |= FTTRACE_OPT_APIInOut | FTTRACE_OPT_DataIn;
    res = FTSetTraceMode(0, FTLevel_Normal, opt, 2, 1, 1);

    signal(SIGINT, UserAbort);

    MarketInfoReset(&MetamarketInfo);
    DebugMsg("FT/API Example Start: [%s]", FTGetLibraryVersion());

    res = FTInit(0);
    DebugMsg("FTInit: %s", FTGetErrorString(res));
    if (res != FTOK) {
        DebugMsg("FTInit: %s", FTGetErrorString(res));
        return -1;
    }

    res = FTExtendClass(InitSkeletonmetamarket());
    if (res != FTOK) {
        DebugMsg("FTExtendClass: %s", FTGetErrorString(res));
        return -1;
    }

    res = FTStart();
    if (res != FTOK) {
        DebugMsg("FTStart: %s", FTGetErrorString(res));
        return -1;
    }

    TryToLink();

    do {
        res = FTRun();
        MarketWait();
        if (MetamarketInfo.Status==Broken)
            TryToLink();
    } while ( !Abort );

    if (MetamarketInfo.Status==Connected){
        res = FTCloseConversation(MetamarketInfo.ConversationID);
        DebugMsg("FTCloseConversation: %s", FTGetErrorString(res));
    }

    DebugMsg("FT/API Example Stop: [%s]", FTGetLibraryVersion());

    res = FTShutdown();
    DebugMsg("FTShutdown: %s", FTGetErrorString(res));
    return 0;
}

```

APPENDIX C: FASTTRACK EXAMPLE: SEND ORDER

This example shows how to use FT/API library to connect a FastTrack system.

The example subscribes some data classes such as Order and send to the market all the orders contained in the file FTORDER.TXT, if present.

The command line accepts a parameter:

-wait ms,

where ms is in milliseconds the minimal time interval between two order sending.

By default the application sends orders waiting 1 sec.

In order to work with this example you need to configure the ft.cfg file including: host, port, hostAlter1, portAlter1, username, password (that will be used for the login) separated by newlines, for example:

```
ft_machine
12000
ft_machine_alternative
12000
jack
*
```

Moreover you can specify your orders in the FTORDER.TXT file following the format used in the fscanf parsing (TAB spaced).

To build an executable binary you must compile this file and link it along with the libraries: ftapi and ftmetamarket.

As an alternative you can substitute the include

```
#include "ftmetamarket.h"
```

and the call

```
res = FTEndClass(InitSkeletonmetamarket());
```

with the following include

```
#include "ftlmetamarket.h"
```

and the following call

```
res = FTEndClassByLibrary(0, "ftlmetamarket", 0);
```

In this case you need not to link this file with the *ftmetamarket* static library, but you need to have, at run-time, the shared *ftlmetamarket.dll* library in an appropriate place.

As an exercise for the reader you can delete the definitions of *DebugMsg* and *DebugInFile* functions and substitute all *DebugMsg* calls with a corresponding *FTTraceAppl* call.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <stdarg.h>
#include <memory.h>
#include <time.h>

#include "ftapi.h"
#include "ftmetamarket.h"

#ifdef ALPHA
#include <netdb.h>
#include <fcntl.h>
#endif

#ifdef WIN32
#include "winsock.h"
#else
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#endif

#ifdef AX42
#define _POSIX_SOURCE
#include <sys/types.h>
#include <sys/select.h>
#endif

static int Abort = 0;
static char Operator[128] = "-";

static FTRevision MyVersion = {1,0,0};

typedef enum {
    Sconnected,
    WaitOnConversation,
    Broken,
    Connected
} ConnectionStatus;

typedef struct {
    ConnectionStatus Status;
    long ConversationID;
    int Socket;
} MarketInfo;

static MarketInfo MetamarketInfo;
static int waitSleep;
struct timeval max_sleep;

static void MarketInfoReset(MarketInfo *mi)
{
    memset((char*)mi, 0, sizeof(MarketInfo));
    mi->Status=Sconnected;
    mi->ConversationID=-1;
    mi->Socket=-1;
}

static void DebugInFile(char *msg) {
    FILE *pf;
    static int first = 1;

    pf = fopen("ftapiex.log",first? "w" : "a");
    fprintf(pf, "%s\n", msg);
    fclose(pf);
    first = 0;
}
```

```
static void DebugMsg(char *fmt,...)
{
    static char buffer[1024];
    va_list ap;

    va_start(ap,fmt);
    vsprintf(buffer, fmt, ap);
    va_end(ap);

    DebugInFile(buffer);
    printf("%s\n", buffer);
}

static void UserAbort(int dummy)
{
    DebugMsg(" *** USER ABORT ***");
    Abort = 1;
}

unsigned long GetTimeInt(void) /* ms */
{
    struct tm      *tmnow;
    unsigned long  msec;
    struct timeval  tp;
#ifdef WIN32
    extern int gettimeofday(struct timeval *, struct timezone *);
#endif

    gettimeofday(&tp,(struct timezone *)0);
    tmnow = localtime((time_t *)&tp.tv_sec);
    msec = (tp.tv_usec / 1000);

    return  (((tmnow->tm_hour * 60) + tmnow->tm_min) * 60 + tmnow->tm_sec) * 1000 + msec;
}

static int GetConfig(char *filename,
                    char *IPAddress, unsigned short *PortNum,
                    char *IPAlternative, char *PortAlternative,
                    char *UserName, char *UserPasswd)
{
    FILE *f;
    int port;

    if ((f = fopen(filename,"r")) {
        fscanf(f, "%s%d%s%s%s", IPAddress, &port,
              IPAlternative, PortAlternative, UserName, UserPasswd);
        fclose(f);
        *PortNum = (unsigned short)port;
        return 0;
    } else
        return -1;
}

static void ConfirmExample()
{
    char c;

    DebugMsg("");
    DebugMsg("*****");
    DebugMsg(" NOTE:");
    DebugMsg(" ----");
    DebugMsg(" This example send an ORDER to the meta market");
    DebugMsg("*****");
    DebugMsg("");
    DebugMsg(" Press 'c' to continue, 's' to stop.");
    printf("ftapiex> ");

    c = getchar();

    if (c!='c' && c!='C'){
        DebugMsg("Example Stopping: no order sending.");
        exit(EXIT_SUCCESS);
    }
}
```

```

/*****
SUBSCRIBE HANDLER
*****/

static void SubscribeResProc (long convID, unsigned long ReqID,
                             long ResultCode, FTSubscribeResStruct *resStruct)
{
    DebugMsg("=====> FTOpenSubscribeResProc: ClassID [%lu], SubscriptionID [%lu]",
             resStruct->ClassID, resStruct->SubscriptionID);
    DebugMsg("    - Result: %s", FTGetErrorString(ResultCode));

    if (ResultCode == FTOK){
        DebugMsg("    - Status: %s",
                 (resStruct->SubscribeStatus==FTSubscribeStart)? "Start" : "Stop");
        if (resStruct->SubscribeStatus==FTSubscribeStart)
            DebugMsg("    - New Table version: %lu; To-Reset signal: %d",
                     resStruct->ClassVersion, resStruct->ResetClass);
    }
    if( convID==MetamarketInfo.ConversationID )
        DebugMsg("    - ConvID: %d (Metamarket)", convID);
}

```



```

/*****
MESSAGE HANDLER
*****/

static void IncomingOrder(FT_C_ORDERPtr Order)
{
    if (!strcmp(Order->OperatorID, Operator))
        DebugMsg("  -> Order: [%s] [%s] [%s] %0.21f %0.21f",
                  Order->OrderID, Order->FTSecID, Order->OperatorID,
                  Order->Price, Order->Qty);
    else{
        /* DebugMsg("  -> Order: [%s] no owner", Order->OrderID); */
    }
}

static void IncomingMarket(FT_C_MARKETPtr Market)
{
    DebugMsg("  -> Market: %s", Market->MarketID);
}

static void NotifyClass(long                convID,
                        unsigned long        ReqID,
                        FTEventTypeEnum      EventType,
                        FTNotifySubscribeResStruct * ResStruct)
{
    /*
    DebugMsg("=====> NotifyClass: Conversation [%d], ReqID [%d], Class [%d], SubscriptionID
    [%d]",
              convID, ReqID, ResStruct->ClassID, ResStruct->SubscriptionID);
    */

    switch (EventType) {
        case FTConnectionBroken:
            DebugMsg("  - Event: ConnectionBroken");
            break;
        case FTClassIdle:
            DebugMsg("  - Event: Idle signal %d", ResStruct->ClassID);
            break;
        case FTClassInstance:
            /* DebugMsg("  - Event: ClassInstance: TimeStamp (%lu, %lu)",
              ResStruct->TimeStamp[0], ResStruct->TimeStamp[1]);
            */
            switch (ResStruct->EntityAction) {
                case FTEntityKIL:
                    if (ResStruct->KeyID <= 0 && !ResStruct->MarketObject)
                        DebugMsg("  - Action: Class Reset Signal - New version: %d",
                                  (ResStruct->TimeStamp)[0]);
                    else
                        DebugMsg("  - Action: EntityKIL - KeyID [%d]",
                                  ResStruct->KeyID);
                    break;
                case FTEntityDEL:
                    DebugMsg("  - Action: EntityDEL - KeyID [%d]", ResStruct->KeyID);
                    break;
                case FTEntityADD:
                case FTEntityRWT:
                    /* DebugMsg("  - Action: EntityADD/RWT - KeyID [%d]",
                      ResStruct->KeyID); */
                    switch ((int)ResStruct->ClassID) {
                        case FT_C_ORDER_ID:
                            IncomingOrder((FT_C_ORDERPtr)ResStruct->MarketObject);
                            break;
                        case FT_C_MARKET_ID:
                            IncomingMarket((FT_C_MARKETPtr)ResStruct->MarketObject);
                            break;

                        default:
                            DebugMsg("  - ?: Entity Not managed ");
                            break;
                    }
                    break;
            }
    }
}

```

```
        break;
    default:
        DebugMsg("    - Event: ?");
        break;
    }
}
```

```

/*****
OPEN CONV. HANDLER
*****/

static void OpenProc(long convID, long ResultCode, FTOpenResStruct *ResStruct)
{
    int res;

    DebugMsg("=====> OpenProc: Login [%d - %s]",
        ResultCode, FTGetErrorString(ResultCode));
    if(ResultCode == FTOK) {
        MetamarketInfo.Status = Connected;

        DebugMsg("    - Market Revision: V%d.%d.%d",
            ResStruct->MarketRevision[0],
            ResStruct->MarketRevision[1],
            ResStruct->MarketRevision[2]);

        res = FTStartSubscribeClassExt(convID, FTAckUnrequired, FTFlowAll,
            FT_C_ORDER_ID, 0, 0, FTQueryAll, 0, 0, 0,
            SubscribeResProc, NotifyClass, 1, 0, 0);
        DebugMsg("FTStartSubscribeClassExt[FT_C_ORDER_ID]: [%d], [%s]",
            res, FTGetErrorString(res));

        res = FTStartSubscribeClassExt(convID, FTAckUnrequired, FTFlowAll,
            FT_C_MARKET_ID, 0, 0, FTQueryAll, 0, 0, 0,
            SubscribeResProc, NotifyClass, 2, 0, 0);
        DebugMsg("FTStartSubscribeClassExt[FT_C_MARKET_ID]: [%d], [%s]",
            res, FTGetErrorString(res));
    }
}

/*****
CLOSING HANDLE
*****/

static void BrokenProc (long conversationID, FTBrokenTypeEnum brokenType)
{
    DebugMsg("=====> BrokenProc: %s",
        (brokenType==FTConnectionClosed)? "FTConnectionClosed" : "FTConnectionLost" );
    if(conversationID==MetamarketInfo.ConversationID){
        DebugMsg("    - Broken Detected on FT conversation: %d", conversationID);
        MarketInfoReset(&MetamarketInfo);
        MetamarketInfo.Status = Broken;
        /* TryToLink(); */
    }
}

/*****
CONNECTION OPENING
*****/

static int TryToLink()
{
    long convID=-1;
    unsigned short PortNum;
    char IPAddress[128], IPAlternative[128], PortAlternative[128],
        UserName[128], UserPasswd[128];
    FTContext ctx;

    if (!GetConfig("ft.cfg", IPAddress, &PortNum,
        IPAlternative, PortAlternative, UserName, UserPasswd)){
        DebugMsg("Connection Params Used: [%s] [%d] [%s] [%s] [%s] [%s]",
            IPAddress, PortNum, IPAlternative, PortAlternative,
            UserName, UserPasswd);
        ctx = FTCreateContext();
        if(! ctx ||
            FTSetContextAttribute(ctx, CTX_ALTERNATIVE_HOST1, PortAlternative) != FTOK ||
            FTSetContextAttribute(ctx, CTX_ALTERNATIVE_PORT1, PortAlternative) != FTOK) {
            DebugMsg("Cannot create or update context");
            if(ctx)
                FTDestroyContext(ctx);
            return -1;
        }
    }
}

```

```

        convID = FTOpenConversationExt(IPAddress, PortNum, FTUserTrader, time(0)%32000,
                                      UserName, UserPasswd, MyVersion, 0,
                                      OpenProc, BrokenProc, 0, 0, ctx);

    FTDestroyContext(ctx);
    strcpy(Operator, UserName);
    DebugMsg("FTOpenConversationExt: %d, [%s]", convID, FTGetErrorString(convID));
    if (convID>=0){
        MetamarketInfo.Status = WaitOnConversation;
        MetamarketInfo.ConversationID = convID;
        MetamarketInfo.Socket=(int)FTGetConversationSocket(convID);
        DebugMsg("Connection Socket: [%d]",MetamarketInfo.Socket);

        return 1;
    }
    else
        return -1;
}
else
    DebugMsg("Cannot read cfg file");
return -1;
}

/*****
    SOCKETS MANAGEMENT
*****/

static fd_set readfds, writefds;
static int NumConv = 0;

static void ConversationProc(long ConvID, int isWritePending)
{
    const long socket = FTGetConversationSocket(ConvID);

    FD_SET(socket, &readfds);
    if (isWritePending)
        FD_SET(socket, &writefds);

    NumConv++;
}

static void MarketWait()
{
    int res;

    FD_ZERO(&readfds);
    FD_ZERO(&writefds);

    NumConv = 0;
    FTEnumConversation(ConversationProc);
#ifdef WIN32
    if (!NumConv){
        Sleep(5000);
        DebugMsg("No Connection");
        return;
    }
#endif
    res = select(FD_SETSIZE, (void *)&readfds, (void *)&writefds, 0, &max_sleep);
    if (res<0)
        perror("select");
    if (res==0)
        DebugMsg("*** select: Time-Out ***");
}

/*****
    ORDER SENDING
*****/

static FILE *orderfile = 0;

static int GetNextOrder(FT_C_ORDERRec *Order)
{
    int res = 0;

    if (!orderfile)

```

```

        orderfile = fopen("ORDER.TXT", "r");
    if (!orderfile)
        return -1;

    /* Verb: a single character: '0'==Buy '1'==Sell */
    res = fscanf(orderfile, "%c %s %lf %lf %s\n",
        (char*)&Order->Verb, Order->FTSecID, &Order->Price,
        &Order->Qty, Order->Client.ClientAccount);

    if (res<1){
        fclose(orderfile);
        return -1;
    }

    if (res!=5){
        DebugMsg("Syntax Error in ORDER.TXT");
        return -1;
    }

    Order->Verb -= '0'; /* Verb: a single character: '0'==Buy '1'==Sell */
    strcpy(Order->OperatorID, Operator);
    return 1;
}

static void DumpTransactioID(FTTransactionID TransactionID)
{
    DebugMsg("      - FTID:      %lu", TransactionID.OpenServerID);
    DebugMsg("      - ServiceID: %lu", TransactionID.ServiceID);
    DebugMsg("      - ClientID:   %lu", TransactionID.ClientID);
    DebugMsg("      - TimeStamp: (%lu,%lu)", TransactionID.TimeStamp[0],
        TransactionID.TimeStamp[1]);
}

static char *StatusToString(FTSendTransactionStatusEnum status)
{
    switch (status){
        case FTTransOK:                return "FTTransOK";
        case FTTransNO:                return "FTTransNO";
        case FTTransInvalSessionID:    return "FTTransInvalSessionID";
        case FTTransInvalidTID:        return "FTTransInvalidTID";
        case FTTransPending:           return "FTTransPending";
        case FTTransConversationClosed: return "FTTransConversationClosed";
        default:                       return "?";
    }
}

static void TransactionResProc(long                conversationID,
                                unsigned long      ReqID,
                                FTSendTransactionResStruct * resStruct)
{
    DebugMsg("----- Transaction Response -----");
    DebugMsg("      Status:      [%s]", StatusToString(resStruct->TransactionStatus));
    DebugMsg("      MarketReason: %d (-1 = undef)", resStruct->MarketReasonCode);
    DebugMsg("      Class:        %lu", resStruct->ClassID);
    DebugMsg("      KeyID:        %lu", resStruct->KeyID);
    DebugMsg("      Action:       %d (ADD, DEL, RWT, ...)", (long) resStruct->EntityAction);
    DebugMsg("      TRANSACTIONID:");
    DumpTransactioID(resStruct->TransactionID);

    if(resStruct->ResultClassID==FT_C_ERROR_INFO_ID && resStruct->ResultObject){
        FT_C_ERROR_INFOPtr info = (FT_C_ERROR_INFOPtr)resStruct->ResultObject;
        DebugMsg("      ReasonCode:   %lu", info->ReasonCode);
        DebugMsg("      ErrorString:  %s", info->ErrorString);
    }

    DebugMsg("-----\n");

    if(resStruct->ResultClassID!=FT_C_ERROR_INFO_ID){
        DebugMsg("WARNING: Transaction Response Class: %d not managed",
            resStruct->ResultClassID);
    }

    if (resStruct->TransactionStatus==FTTransConversationClosed)
        DebugMsg("WARNING: Transaction response lost.");
}

```

```

}

static int SendOrder(FT_C_ORDERRec *Order)
{
    long res = FTInternalError;
    FTTransactionID TransactionID;
    long convID = MetamarketInfo.ConversationID;

    res = FTMakeTransactionID (convID, &TransactionID);
    DebugMsg("FTMakeTransactionID [%s]", FTGetErrorString(res));

    if (res != FTOK)
        return -1;

    DebugMsg("Done TransactionID:");
    DumpTransactioID(TransactionID);

    res = FTSendTransactionExt( convID,
                                &TransactionID,
                                FTEntityADD,
                                FT_C_ORDER_ID,
                                FT_C_ORDERKey,
                                (void*) Order,
                                TransactionResProc,
                                FTFalse,
                                0,
                                3);
    DebugMsg("Send Order on Security [%s].", Order->FTSecID);
    DebugMsg("FTSendTransactionExt [%s]", FTGetErrorString(res));
    DebugMsg("");

    return res;
}

static void TrySend()
{
    static int        lastsend_ms = 0;
    static int        FileDone = 0;
    int              res = 0;
    int              now = 0;
    FT_C_ORDERRec    Order;

    if (FileDone)
        return;

    now = GetTimeInt();

    if (lastsend_ms && now-lastsend_ms<waitSleep)
        return;

    memset(&Order, 0, sizeof(Order));

    res = GetNextOrder(&Order);

    FileDone = res==-1;

    if (FileDone){
        max_sleep.tv_sec    = 3;
        max_sleep.tv_usec   = 0;
    }

    if (res>0){
        res = SendOrder(&Order);
        if (res>=0)
            lastsend_ms = now;
    }
}

/*****
MAIN
*****/

int main(int argc, char *argv[])

```

```
{
    long res = -1;

    /* Setting for tracing features
    int opt = FTTRACE_OPT_None;
    res = FTSetTraceMode(0, FTLevel_Normal, opt, 2, 1, 1);
    */

    if (argc>2 && !strcmp(argv[1], "-wait"))
        waitSleep = atoi(argv[2]);
    else
        waitSleep = 1000;                                /* default */

    max_sleep.tv_sec    = waitSleep/1000;
    max_sleep.tv_usec   = (waitSleep%1000)*1000;          /* microsec*/

    signal(SIGINT, UserAbort);

    MarketInfoReset(&MetamarketInfo);
    DebugMsg("FT/API Example Start: [%s]", FTGetLibraryVersion());

    ConfirmExample();

    res = FTInit(0);
    DebugMsg("FTInit: %s", FTGetErrorString(res));
    if (res != FTOK) {
        DebugMsg("FTInit: %s", FTGetErrorString(res));
        return -1;
    }

    res = FTExtendClass(InitSkeletonmetamarket());
    if (res != FTOK) {
        DebugMsg("FTExtendClass: %s", FTGetErrorString(res));
        return -1;
    }

    res = FTStart();
    if (res != FTOK) {
        DebugMsg("FTStart: %s", FTGetErrorString(res));
        return -1;
    }
}
```

```
res = TryToLink();

if (res>0){
    do {
        res = FTRun();
        MarketWait();

        if (MetamarketInfo.Status==Connected)
            TrySend();

        if (MetamarketInfo.Status==Broken)
            TryToLink();

    } while ( !Abort );
}

if (MetamarketInfo.Status==Connected){
    res = FTCloseConversation(MetamarketInfo.ConversationID);
    DebugMsg("FTCloseConversation: %s", FTGetErrorString(res));
}

DebugMsg("FT/API Example Stop: [%s]", FTGetLibraryVersion());

res = FTShutdown();
DebugMsg("FTShutdown: %s", FTGetErrorString(res));

if (orderfile)
    fclose(orderfile);

return 0;
}
```


APPENDIX D: NAM MARKETS EXAMPLE: GET FILL

This example shows how to use FT/API library to connect a NAM market via a OMI protocol.

This example shows how get data classes (BOND, FILL, etc...) from a NAM Market Server 'Telematico Exchange Place' (JGB and EBM), by using subscriptions.

In order to work with this example you need to configure the ft.cfg file including: host, port, username, password (that will be used for the login) separated by newlines, for example:

```
ft_machine
12000
jack
*
```

To build an executable binary you must compile this file and link it along with the libraries: ftapi and omiexp.

As an alternative you can substitute the include

```
#include "omiexp.h"
```

and the call

```
res = FTExtendClass(InitSkeletonexp());
```

with the following include

```
#include "ftlexp.h"
```

and the following call

```
res = FTExtendClassByLibrary(0, "ftlexp", 0);
```

In this case you need not to link this file with the *omiexp* static library, but you need to have, at run-time, the shared *ftlexp.dll* library in an appropriate place.

As an exercise for the reader you can delete the definitions of *DebugMsg* and *DebugInFile* functions and substitute all *DebugMsg* calls with a corresponding *FTTraceAppl* call.

```

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdarg.h>
#include <memory.h>
#include <time.h>

#ifdef ALPHA
#include <netdb.h>
#include <fcntl.h>
#endif

#ifdef WIN32
#include "winsock.h"
#else
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#endif

#ifdef AX42
#define _POSIX_SOURCE
#include <sys/types.h>
#include <sys/select.h>
#endif

#include "ftapi.h" /* Communication Layer */
#include "omiexp.h" /* Market Layer */

#define BUFF_LEN 128
#define CONNECTION_TIMER_SEC 8 /* Connection retry interval */

/*****
GLOBALS
*****/
static int Abort = 0;

static FTRevision MyVersion = {1,1,1};

typedef enum {
    Sconnected,
    WaitOnConversation,
    Broken,
    Connected
} ConnectionStatus;

typedef struct {
    ConnectionStatus Status;
    long ConversationID;
    int Socket;
} MarketInfo;

static MarketInfo EBMInfo;

static void MarketInfoReset(MarketInfo *mi)
{
    memset((char*)mi, 0, sizeof(MarketInfo));
    mi->Status=Sconnected;
    mi->ConversationID=-1;
    mi->Socket=-1;
}

/*****
UTILITIES
*****/

static void DebugMsg(char *fmt,...)
{
    va_list ap;

```

```
va_start(ap, fmt);  
vprintf(fmt, ap);  
va_end(ap);  
printf("\n");  
}
```

```

static void TraceProc (long eventInfo, char *string)
{
    FTTraceSourceEnum source = 0;
    FTTraceLevelEnum level = 0;

    /*
       Here we trace only exceptional (Warning or greater) events, on screen.
       All events (including Normal) will be traced automatically in FTapi LOGS
       files, in force of the WriteFile flag used in FTSetTraceMode call.
    */

    int res = FTGetTraceInfo (eventInfo, &source, &level);

    if (level<FTLevel_Warning)
        return;

    DebugMsg("**TRACE** %d %d - %s", source, level, string);
}

static void UserAbort(int dummy)
{
    DebugMsg(" *** USER ABORT ***");
    Abort = 1;
}

static int GetConfig(char *filename,
                    char *IPAddress,
                    unsigned short *PortNum,
                    char *UserName,
                    char *UserPasswd)
{
    FILE *f;
    int port;

    f = fopen(filename,"r");
    if (!f)
        return -1;

    fscanf(f, "%s%d%s", IPAddress, &port, UserName, UserPasswd);
    fclose(f);
    *PortNum = (unsigned short)port;

    return 0;
}

/*****
DATA NOTIFICATION
*****/

static void SubscribeResProcExt(long convID,
                               unsigned long ReqID,
                               long ResultCode,
                               FTSubscribeResStruct *resStruct)
{
    DebugMsg("=====> FTOpenSubscribeResProc: classID [%lu], SubscrID [%lu]",
            resStruct->ClassID, resStruct->SubscriptionID);
    DebugMsg(" - Result: %s", FTGetErrorString(ResultCode));

    if (ResultCode == FTOK){
        DebugMsg(" - Status: %s",
            (resStruct->SubscribeStatus==FTSubscribeStart)? "Start" : "Stop");
        DebugMsg(" - New Table version: %lu; To-Reset signal: %d",
            resStruct->ClassVersion, resStruct->ResetClass);
    }
    if( convID==EBMInfo.ConversationID )
        DebugMsg(" - ConvID: %d (EBM)", convID);
}

static void DumpFill (EBM_FILLPtr fill)
{
    FILE *f;

```

```
static int first = 1;

/*
   Here you can manage the incoming new/changed record.
*/

f = fopen("Fill.txt", first? "w" : "a");
first = 0;

if (f){
    fprintf(f, "%s\t", fill->MarketCode);
    fprintf(f, "%lu\t", fill->Date);
    /*fprintf(f, "%lu\t", fill->Time);*/
    fprintf(f, "%lu\t", fill->Type);
    fprintf(f, "%s\t", fill->BondCode);
    fprintf(f, "%s\t", fill->Description);
    fprintf(f, "%f\t", fill->Price);
    fprintf(f, "%f\t", fill->Yield);
    fprintf(f, "%f\t", fill->Qty);

    fprintf(f, "%s\t", fill->OrderMemberInfo.MemberCode);
    fprintf(f, "%s\t", fill->OrderMemberInfo.Operator);

    fprintf(f, "%s\t", fill->FillMemberInfo.MemberCode);
    fprintf(f, "%s\t", fill->FillMemberInfo.Operator);

    fprintf(f, "\n");
    fclose(f);
}

static void DumpBond (EBM_BONDPtr bond)
{
    FILE *f;
    static int first = 1;

    f = fopen("Bond.txt", first? "w" : "a");
    first = 0;

    if (f){
        fprintf(f, "%s\t", bond->BondCode);
        fprintf(f, "%s\t", bond->Description);
        fprintf(f, "%d\n", bond->StripFlag);
        fclose(f);
    }
}

static void DumpBondMarket (EBM_BOND_MARKETPtr bondmarket)
{
    FILE *f;
    static int first = 1;

    f = fopen("BondMarket.txt", first? "w" : "a");
    first = 0;

    if (f){
        fprintf(f, "%s\t", bondmarket->BondCode);
        fprintf(f, "%s\n", bondmarket->MarketCode);
        fclose(f);
    }
}

static void DumpProposal (EBM_PROPOSALPtr proposal)
{
    FILE *f;
    static int first = 1;

    f = fopen("Proposal.txt", first? "w" : "a");
    first = 0;

    if (f){
```

```

        fprintf(f, "%d\t", proposal->SeqNo);
        fprintf(f, "%s\t", proposal->MarketCode);
        fprintf(f, "%s\t", proposal->BondCode);
        fprintf(f, "%s\n", proposal->BondCode);
        fclose(f);
    }
}

static void NotifyClassExt(long convID,
                          unsigned long ReqID,
                          FTEventTypeEnum EventType,
                          FTNotifySubscribeResStruct *ResStruct)
{
    DebugMsg("=====> NotifyClassEBM: Conversation [%d], Class [%d], SubscrID [%d]",
            convID, ResStruct->ClassID, ResStruct->SubscriptionID);

    switch (EventType) {
        case FTConnectionBroken:
            DebugMsg(" - Event: ConnectionBroken");
            break;
        case FTClassIdle:
            DebugMsg(" - Event: Idle signal");
            break;
        case FTClassInstance:
            DebugMsg(" - Event: ClassInstance: TimeStamp (%lu, %lu)",
                    ResStruct->TimeStamp[0], ResStruct->TimeStamp[1]);
            switch (ResStruct->EntityAction) {
                case FTEntityKIL:
                    if (ResStruct->KeyID <= 0 && !ResStruct->MarketObject)
                        DebugMsg(" - Action: Class Reset Signal - New version: %d",
                                (ResStruct->TimeStamp)[0]);
                    else
                        DebugMsg(" - Action: EntityKIL - KeyID [%d]",
                                ResStruct->KeyID);
                    break;
                case FTEntityDEL:
                    DebugMsg(" - Action: EntityDEL - KeyID [%d]",
                            ResStruct->KeyID);
                    break;
                case FTEntityADD:
                case FTEntityRWT:
                    DebugMsg(" - Action: EntityADD/RWT - KeyID [%d]",
                            ResStruct->KeyID);
                    switch (ResStruct->ClassID) {
                        case EBM_BOND_ID:
                            DumpBond((EBM_BONDPtr)ResStruct->MarketObject);
                            break;
                        case EBM_BOND_MARKET_ID:
                            DumpBondMarket((EBM_BOND_MARKETPtr)
                                    ResStruct->MarketObject);
                            break;
                        case EBM_OPERATOR_ID:
                            static count = 0;

                            count ++;

                            /* Example:
                               Stop the Operator Subscription after 3 records
                               */
                            if (count==3){
                                int res = FTStopSubscribeClass(convID,
                                                                EBM_OPERATOR_ID,
                                                                ResStruct->
                                                                SubscriptionID);
                                DebugMsg("FTStopSubscribeClass %s",
                                        FTGetErrorString(res));
                            }
                        }
                    break;
                case EBM_FILL_ID:
                    DumpFill((EBM_FILLPtr)ResStruct->MarketObject);
                    break;
                case EBM_EXEC_ID:
                    DebugMsg((EBM_EXECPtr)ResStruct->MarketObject)
    }
}

```

```
                                ->BondCode);  
    break;  
    case EBM_PROPOSAL_ID: {  
        DumpProposal((EBM_PROPOSALPtr)ResStruct->MarketObject);  
    }  
    break;  
  
    /*  
    Add here other ClassID cases,  
    to manage subscribed classes.  
    */  
  
    default:  
        DebugMsg("    - ?: Entity Not managed ");  
        break;  
    }  
    break;  
}  
break;  
default:  
    DebugMsg("    - Event: ?");  
    break;  
}  
}
```

```

/*****
CONVERSATION
*****/

static int SubscribeClass(long convID, long ClassID, char *ClassName)
{
    long res = FTStartSubscribeClassExt(    convID,
                                           FTAckUnrequired,
                                           FTFlowAll,
                                           ClassID,
                                           0,
                                           0,
                                           FTQueryAll,
                                           0,
                                           0,
                                           0,
                                           SubscribeResProcExt,
                                           NotifyClassExt,
                                           124,
                                           0,
                                           0);

    DebugMsg("Subscription[%s]: [%d], [%s]", ClassName, res, FTGetErrorString(res));

    return res;
}

static void OpenProcEBM(long convID, long ResultCode, FTOpenResStruct *ResStruct)
{
    int res = 0;

    DebugMsg("=====> OpenProc EBM: Login [%d - %s]",
            ResultCode, FTGetErrorString(ResultCode));
    if(ResultCode == FTOK) {
        EBMInfo.Status = Connected;

        DebugMsg("    - Market Revision: V%d.%d.%d",
            ResStruct->MarketRevision[0],
            ResStruct->MarketRevision[1],
            ResStruct->MarketRevision[2]);

        res = SubscribeClass(convID, EBM_BOND_ID, "EBM_BOND");
        res = SubscribeClass(convID, EBM_BOND_MARKET_ID, "EBM_BOND_MARKET");
        res = SubscribeClass(convID, EBM_OPERATOR_ID, "EBM_OPERATOR");

        res = SubscribeClass(convID, EBM_EXEC_ID, "EBM_EXEC");
        res = SubscribeClass(convID, EBM_FILL_ID, "EBM_FILL");
        res = SubscribeClass(convID, EBM_PROPOSAL_ID, "EBM_PROPOSAL");

        /*
        Add here other Class subscription requests
        */
    }
}

static void BrokenProc (long conversationID, FTBrokenTypeEnum brokenType)
{
    DebugMsg("=====> BrokenProc: %s", (brokenType==FTConnectionClosed)
            ? "FTConnectionClosed" : "FTConnectionLost" );
    if(conversationID==EBMInfo.ConversationID){
        DebugMsg("    - Broken Detected on EBM conversation: %d", conversationID);
        MarketInfoReset(&EBMInfo);
        EBMInfo.Status = Broken;
    }
}

static void TryToConnectEBMServer()
{
    long convID=-1;
    unsigned short PortNum;
    char IPAddress[BUFF_LEN], UserName[BUFF_LEN], UserPasswd[BUFF_LEN];
    FTContext ctx = 0;

```



```

int res = 0;

if( EBMInfo.Status==Connected || EBMInfo.Status==WaitOnConversation )
    return;

if (!GetConfig("ft.cfg",IPAddress, &PortNum, UserName, UserPasswd)){
    unsigned int ClientID = 0;

    DebugMsg("EBM Params Used: [%s] [%d] [%s] [%s]",
        IPAddress, PortNum, UserName, UserPasswd);
    ClientID = time(0)%33333+30000;

    ctx = FTCreateContext();
    res = FTSetContextAttribute(ctx, CTX_APPL_AUTHFILE, "license.xml");
    convID = FTOpenConversationExt( IPAddress,
                                    PortNum,
                                    FTUserTrader,
                                    ClientID,
                                    UserName,
                                    UserPasswd,
                                    MyVersion,
                                    0,
                                    OpenProcEBM,
                                    BrokenProc,
                                    12345,
                                    0,
                                    ctx);

    FTDestroyContext(ctx);

    DebugMsg("FTOpenConversation: %d, [%s]", convID, FTGetErrorString(convID));
    if (convID>=0){
        EBMInfo.Status = WaitOnConversation;
        EBMInfo.ConversationID = convID;
        EBMInfo.Socket=(int)FTGetConversationSocket(convID);
        DebugMsg("EBM Socket: [%d]",EBMInfo.Socket);
    }
    else
        DebugMsg("Cannot read file: ft.cfg");
}

/*
    Implements a Timer to keep alive the connection
*/
static void KeepEBMConnection()
{
    static int LastAttempt = 0;

    if(time(0)-LastAttempt<CONNECTION_TIMER_SEC)
        return;

    TryToConnectEBMServer();
    LastAttempt = time(0);
}

/*****
    SELECT
*****/

static fd_set readfds, writefds;

static void ConversationProc(long ConvID, int isWritePending)
{
    const long socket = FTGetConversationSocket(ConvID);

    FD_SET(socket, &readfds);
    if (isWritePending)
        FD_SET(socket, &writefds);
}

/*
    Implements a process wait on the icoming/outgoing sockets.

```

```

*/
static void MarketWait()
{
    int res, num_conv;
    struct timeval max_sleep;

    FD_ZERO(&readfds);
    FD_ZERO(&writefds);

    num_conv = FTEnumConversation(ConversationProc);

#ifdef WIN32
    /*
     * If no conversation is active
     * implements a 3 sec. timeout
     */
    if (num_conv<1){
        Sleep(3*1000);
        DebugMsg("*** No Connections ***");
        return;
    }
#endif

    max_sleep.tv_sec    = 1;
    max_sleep.tv_usec   = 0;

    res = select(FD_SETSIZE, (void *) &readfds, (void *) &writefds, 0, &max_sleep);
    if (res<0)
        perror("select");
    if (res==0)
        DebugMsg("*** select: Time-Out ***");
}

static void SetTrace()
{
    int res = -1;
    int opt = FTTRACE_OPT_None;

    /*
     Note:
     FTTRACE_OPT_DataTransfert is a large-detailed log mode,
     to use in specific cases only.
    */

    opt =    opt
            | FTTRACE_OPT_APIInOut
            | FTTRACE_OPT_FlowControl
            | FTTRACE_OPT_DataIn
            | FTTRACE_OPT_DataOut
            /*|FTTRACE_OPT_DataTransfert*/
            ;

    res = FTSetTraceMode(0, FTLevel_Full, opt, 1, 1, 0);
    /* Redirect Log-events from handler to file */
    DebugMsg("FTSetTraceMode: [%s]", FTGetErrorString(res));
}

/*****
MAIN
*****/

int main(int argc, char *argv[])
{
    long res = -1;

    signal(SIGINT, UserAbort);
    MarketInfoReset(&EBMInfo);
    DebugMsg("FT/API Example Start: [%s]", FTGetLibraryVersion());

    SetTrace();

    /*res = FTLoadLicence(0);*/      /* Deprecated licence call */

```

```
res = FTInit(TraceProc);
DebugMsg("FTInit: %s", FTGetErrorString(res));
if (res != FTOK) {
    DebugMsg("FTInit: %s", FTGetErrorString(res));
    return -1;
}

res = FTEExtendClass(InitSkeletonexp());
if (res != FTOK) {
    DebugMsg("FTEExtendClass EBM: %s", FTGetErrorString(res));
    return -1;
}

res = FTStart();
if (res != FTOK) {
    DebugMsg("FTStart: %s", FTGetErrorString(res));
    return -1;
}

do {
    KeepEBMConnection();
    FTRun();
    MarketWait();
} while ( !Abort && EBMInfo.Status != Broken);

if (EBMInfo.Status==Connected){
    res = FTCloseConversation(EBMInfo.ConversationID);
    DebugMsg("FTCloseConversation EBM: %s", FTGetErrorString(res));
}

DebugMsg("FT/API Example Stop: [%s]", FTGetLibraryVersion());
res = FTShutdown();
DebugMsg("FTShutdown: %s", FTGetErrorString(res));

return 0;
}
```

APPENDIX E: NAM MARKETS EXAMPLE: SEND PROPOSAL

This example shows how to use FT/API library to connect a NAM market via a OMI protocol.

The focus of this example is to show how to send transaction on a new proposal (EBM_PROPOSAL).

In order to work with this example you need to configure the ft.cfg file including: host, port, username, password (that will be used for the login) separated by newlines, for example:

```
ft_machine
12000
jack
*
```

To build an executable binary you must compile this file and link it along with the libraries: ftapi and omiexp.

As an alternative you can substitute the include

#include "omiexp.h"

and the call

```
res = FTExtendClass(InitSkeletonexp());
```

with the following include

#include "ftlexp.h"

and the following call

```
res = FTExtendClassByLibrary(0, "ftlexp", 0);
```

In this case you need not to link this file with the *omiexp* static library, but you need to have, at run-time, the shared *ftlexp.dll* library in an appropriate place.

As an exercise for the reader you can delete the definitions of *DebugMsg* and *DebugInFile* functions and substitute all *DebugMsg* calls with a corresponding *FTTraceAppl* call.

```
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdarg.h>
#include <memory.h>
#include <time.h>

#ifdef ALPHA
#include <netdb.h>
#include <fcntl.h>
#endif

#ifdef WIN32
#include "winsock.h"
#else
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#endif

#if defined AX42
#define _POSIX_SOURCE
#include <sys/types.h>
#include <sys/select.h>
#endif

#include "ftapi.h" /* Communication Layer */
#include "omiexp.h" /* Market Layer */

#define CONNECTION_TIMER_SEC 8 /* Connection timer interval */

/*****
GLOBALS
*****/
static int Abort = 0;

static FTRevision MyVersion = {1,1,1};

typedef enum {
    Sconnected,
    WaitOnConversation,
    Broken,
    Connected
} ConnectionStatus;

typedef struct {
    ConnectionStatus Status;
    long ConversationID;
    int Socket;
} MarketInfo;

static MarketInfo EBMInfo;

static void MarketInfoReset(MarketInfo *mi)
{
    memset((char*)mi, 0, sizeof(MarketInfo));
    mi->Status=Sconnected;
    mi->ConversationID=-1;
    mi->Socket=-1;
}

/*****
UTILITIES
*****/

static void DebugMsg(char *fmt,...)
{
    va_list ap;
```

```
va_start(ap,fmt);  
vprintf(fmt,ap);  
va_end(ap);  
printf("\n");  
}
```

```
static void TraceProc (long eventInfo, char *string)
{
    FTTraceSourceEnum source = 0;
    FTTraceLevelEnum level = 0;

    /*
       Note:
       Here we trace exceptional (Warning or greater) events only, on screen.
       All the events (Normal also) will be traced automatically in FTapi LOGS
       files, in force of the WriteFile flag used in FTSetTraceMode call.
    */

    int res = FTGetTraceInfo (eventInfo, &source, &level);

    if (level<FTLevel_Warning)
        return;

    DebugMsg("***TRACE** %d %d - %s", source, level, string);
}

static void UserAbort(int dummy)
{
    DebugMsg(" *** USER ABORT ***");
    Abort = 1;
}

static int GetConfig(char *filename,
                    char *IPAddress,
                    unsigned short *PortNum,
                    char *UserName,
                    char *UserPasswd)
{
    FILE *f;
    char b1[64], b2[64], b3[64], b4[64];

    if (f = fopen(filename,"r")) {
        fscanf(f, "%s%s%s%s", b1, b2, b3, b4);
        fclose(f);

        if (IPAddress)
            strcpy(IPAddress, b1);
        if (PortNum)
            *PortNum = (unsigned short) (atoi(b2));
        if (UserName)
            strcpy(UserName, b3);
        if (UserPasswd)
            strcpy(UserPasswd, b4);

        return 0;
    } else
        return -1;
}

static void ConfirmExample()
{
    char c;

    DebugMsg("");
    DebugMsg("*****");
    DebugMsg(" NOTE:");
    DebugMsg(" ----");
    DebugMsg(" This example send a PROPOSAL into the market");
    DebugMsg("*****");
    DebugMsg("");
    DebugMsg(" Press 'c' to continue, 's' to stop.");
    printf("ftexample> ");

    c = getchar();

    if (c!='c' && c!='C'){
        DebugMsg("Example Stopping: no proposal sending.");
        exit(EXIT_SUCCESS);
    }
}
```

```

    }
}

static void DumpTransactionID(FTTransactionID TransactionID)
{
    DebugMsg("      | BusinessServiceID: %lu", TransactionID.BusinessServiceID);
    DebugMsg("      | ClientServiceID: %lu", TransactionID.ClientServiceID);
    DebugMsg("      | ClientID: %lu", TransactionID.ClientID);
    DebugMsg("      | TimeStamp: (%lu,%lu)",
        TransactionID.TimeStamp[0], TransactionID.TimeStamp[1]);
}

static char *StatusToString(FTSendTransactionStatusEnum status)
{
    switch (status){
        case FTTransOK: return "FTTransOK";
        case FTTransNO: return "FTTransNO";
        case FTTransInvalSessionID: return "FTTransInvalSessionID";
        case FTTransInvalidTID: return "FTTransInvalidTID";
        case FTTransPending: return "FTTransPending";
        case FTTransConversationClosed: return "FTTransConversationClosed";
    }

    return "??";
}

#define TS_ITEM(TS, item) (TS? TS[item] : 0)

/* deprecated Transaction Callback */
/*
static void TransactionResProc(long ConversationID,
                               FTSendTransactionResStruct *resStruct)
{
    DebugMsg("----- Transaction Response -----");
    DebugMsg("      Status: [%s]", StatusToString(resStruct->TransactionStatus));
    DebugMsg("      MarketReason: %d (-1 = undef)", resStruct->MarketReasonCode);
    DebugMsg("      Class: %lu", resStruct->ClassID);
    DebugMsg("      KeyID: %lu", resStruct->KeyID);
    DebugMsg("      Action: %d (ADD, DEL, RWT, ...)",
        (long) resStruct->EntityAction);
    DebugMsg("      TimeStamp: %d,%d",
        TS_ITEM(resStruct->TimeStamp, 0), TS_ITEM(resStruct->TimeStamp, 1));
    DebugMsg("      ResultClassID %d", resStruct->ResultClassID);
    DebugMsg("      ResultObject %x", resStruct->ResultObject);
    DebugMsg("      TRANSACTIONID:");
    DumpTransactionID(resStruct->TransactionID);
    DebugMsg("-----\n");
    if (resStruct->TransactionStatus==FTTransConversationClosed)
        DebugMsg("WARNING: Transaction response lost.");
}
*/

static void TransactionResProcExt(long ConversationID,
                                  unsigned long ReqID,
                                  FTSendTransactionResStruct *resStruct)
{
    DebugMsg("----- Transaction Response -----");
    DebugMsg("      ReqID: %d", ReqID);
    DebugMsg("      Status: [%s]", StatusToString(resStruct->TransactionStatus));
    DebugMsg("      MarketReason: %d (-1 = undef)", resStruct->MarketReasonCode);
    DebugMsg("      Class: %lu", resStruct->ClassID);
    DebugMsg("      KeyID: %lu", resStruct->KeyID);
    DebugMsg("      Action: %d (ADD, DEL, RWT, ...)",
        (long) resStruct->EntityAction);
    DebugMsg("      TimeStamp: %d,%d",
        TS_ITEM(resStruct->TimeStamp, 0), TS_ITEM(resStruct->TimeStamp, 1));
    DebugMsg("      ResultClassID %d", resStruct->ResultClassID);
    DebugMsg("      ResultObject %x", resStruct->ResultObject);
    DebugMsg("      TRANSACTIONID:");
    DumpTransactionID(resStruct->TransactionID);
    DebugMsg("-----\n");
    if (resStruct->TransactionStatus==FTTransConversationClosed)

```



```

        DebugMsg("WARNING: Transaction response lost.");
    }

    /*****
    TRANSACTION SENDING
    *****/
    /*
    Send an EBM_PROPOSAL
    */

static void TransactionTest()
{
    long res = -1;
    long convID = EBMInfo.ConversationID;
    EBM_PROPOSALRec proposal = {0};
    char MyOperator[64] = {0};
    FTTransactionID TransactionID = {0};

    ConfirmExample();

    res = FTMakeTransactionID (convID, &TransactionID);
    DebugMsg("FTMakeTransactionID [%s]", FTGetErrorString(res));

    if (res != FTOK)
        return;

    DumpTransactionID(TransactionID);

    GetConfig("ft.cfg", NULL, NULL, MyOperator, NULL);

    memset((char*)&proposal, 0, sizeof(proposal));

    /*
    Fill the required proposal fields
    on the base of the EBM specifications
    */

    strcpy(proposal.BondCode          , "AT0000383518");
    strcpy(proposal.BondType          , "BTP");
    strcpy(proposal.MarketCode        , "EBM");
    proposal.Status                   = EBM_PROPOSAL_STATUS_Active;
    strcpy(proposal.Operator          , MyOperator);
    proposal.Ask.Price                = 130.00;
    proposal.Ask.EbmDrip.Qty          = 10;
    proposal.Ask.DomDrip.Qty          = 10;
    proposal.Bid.Price                = 123.12;
    proposal.Bid.EbmDrip.Qty          = 20;
    proposal.Bid.DomDrip.Qty          = 20;

    res = FTSendTransactionExt( convID,
                                &TransactionID,
                                FTEntityADD,
                                EBM_PROPOSAL_ID,
                                EBM_PROPOSALKey,
                                (void*) &proposal,
                                TransactionResProcExt,
                                FTFalse,
                                0,
                                2233);

    /* deprecated send */
    /*
    res = FTSendTransaction( convID,
                            TransactionID,
                            FTEntityADD,
                            EBM_PROPOSAL_ID,
                            EBM_PROPOSALKey,
                            (void*) &proposal,
                            TransactionResProc,
                            0);

    */

    DebugMsg("SEND PROPOSAL on security [%s].", proposal.BondCode);
    DebugMsg("FTSendTransactionExt [%s]", FTGetErrorString(res));

```

```
}

/*****
DATA NOTIFICATION
*****/

static void SubscribeResProcExt (long convID,
                                unsigned long ReqID,
                                long ResultCode,
                                FTSubscribeResStruct *resStruct)
{
    DebugMsg("=====> FTOpenSubscribeResProc: classID [%lu], SubscrID [%lu]",
             resStruct->ClassID, resStruct->SubscriptionID);
    DebugMsg("    - Result: %s", FTGetErrorString(ResultCode));

    if (ResultCode == FTOK){
        DebugMsg("    - Status: %s",
                 (resStruct->SubscribeStatus==FTSubscribeStart)? "Start" : "Stop");
        DebugMsg("    - New Table version: %lu; To-Reset signal: %d",
                 resStruct->ClassVersion, resStruct->ResetClass);
    }
    if ( convID==EBMInfo.ConversationID )
        DebugMsg("    - ConvID: %d (EBM)", convID);
}
```

```

static void NotifyClassExt(long convID,
                          unsigned long ReqID,
                          FTEventTypeEnum EventType,
                          FTNotifySubscribeResStruct *ResStruct)
{
    DebugMsg("=====> NotifyClassEBM: Conversation [%d], Class [%d], SubscrID [%d]",
            convID, ResStruct->ClassID, ResStruct->SubscriptionID);

    switch (EventType) {
        case FTConnectionBroken:
            DebugMsg(" - Event: ConnectionBroken");
            break;
        case FTClassIdle:
            DebugMsg(" - Event: Idle signal");
            if (ResStruct->ClassID==EBM_PROPOSAL_ID){
                TransactionTest();
            }
            break;
        case FTClassInstance:
            DebugMsg(" - Event: ClassInstance: Timestamp (%lu, %lu)",
                    ResStruct->TimeStamp[0], ResStruct->TimeStamp[1]);
            switch (ResStruct->EntityAction) {
                case FTEntityKIL:
                    if (ResStruct->KeyID <= 0 && !ResStruct->MarketObject)
                        DebugMsg(" - Action: Class Reset Signal - New version: %d",
                                (ResStruct->TimeStamp)[0]);
                    else
                        DebugMsg(" - Action: EntityKIL - KeyID [%d]",
                                ResStruct->KeyID);
                    break;
                case FTEntityDEL:
                    DebugMsg(" - Action: EntityDEL - KeyID [%d]",
                            ResStruct->KeyID);
                    break;
                case FTEntityADD:
                case FTEntityRWT:
                    DebugMsg(" - Action: EntityADD/RWT - KeyID [%d]",
                            ResStruct->KeyID);
                    switch (ResStruct->ClassID) {
                        case EBM_FILL_ID:{
                            EBM_FILLPtr FILL = (EBM_FILLPtr)ResStruct->MarketObject;
                            DebugMsg(" - EBM_FILL ==> [%d] [%s]",
                                    FILL->ContractNo, FILL->BondCode);
                        }
                        break;

                        case EBM_PROPOSAL_ID:{
                            EBM_PROPOSALPtr PROPOSAL = (EBM_PROPOSALPtr)
                                    ResStruct->MarketObject;
                            DebugMsg(" - EBM_PROPOSAL ==> [%s] [%s]",
                                    PROPOSAL->MemberCode, PROPOSAL->Description);
                        }
                        break;

                        case EBM_ORDER_ID:{
                            EBM_ORDERPtr ORDER = (EBM_ORDERPtr)
                                    ResStruct->MarketObject;
                            DebugMsg(" - EBM_ORDER ==> [%s] [%s]",
                                    ORDER->BondCode, ORDER->Operator);
                        }
                        break;

                        /*
                         Add here other ClassID cases,
                         to manage subscribed classes.
                        */

                        default:
                            DebugMsg(" - ?: Entity Not managed ");
                            break;
                    }
                    break;
            }
            break;
        default:
    }
}

```

```
        DebugMsg("    - Event: ?");  
        break;  
    }  
}
```

```

/*****
CONVERSATION
*****/

static int SubscribeClass(long convID, long ClassID, char *ClassName)
{
    long res = FTStartSubscribeClassExt(    convID,
                                           FTAckUnrequired,
                                           FTFlowAll,
                                           ClassID,
                                           0,
                                           0,
                                           FTQueryAll,
                                           0,
                                           0,
                                           0,
                                           SubscribeResProcExt,
                                           NotifyClassExt,
                                           124,
                                           0,
                                           0);

    DebugMsg("Subscription[%s]: [%d], [%s]", ClassName, res, FTGetErrorString(res));

    return res;
}

static void OpenProcEBM(long convID, long ResultCode, FTOpenResStruct *ResStruct)
{
    int res;

    DebugMsg("=====> OpenProc EBM: Login [%d - %s]",
            ResultCode, FTGetErrorString(ResultCode));
    if(ResultCode == FTOK) {
        EBMInfo.Status = Connected;

        DebugMsg("    - Market Revision: V%d.%d.%d",
            ResStruct->MarketRevision[0],
            ResStruct->MarketRevision[1],
            ResStruct->MarketRevision[2]);

        res = SubscribeClass(convID, EBM_FILL_ID,          "EBM_FILL");
        res = SubscribeClass(convID, EBM_PROPOSAL_ID,       "EBM_PROPOSAL");
        res = SubscribeClass(convID, EBM_ORDER_ID,          "EBM_ORDER");
        /*
        Add here other Class subscription requests
        */
    }
}

static void BrokenProc (long conversationID, FTBrokenTypeEnum brokenType)
{
    DebugMsg("=====> BrokenProc: %s", (brokenType==FTConnectionClosed)
        ? "FTConnectionClosed"
        : "FTConnectionLost" );
    if(conversationID==EBMInfo.ConversationID){
        DebugMsg("    - Broken Detected on EBM conversation: %d", conversationID);
        MarketInfoReset(&EBMInfo);
        EBMInfo.Status = Broken;
    }
}

static void TryToConnectEBMServer()
{
    long convID=-1;
    unsigned short PortNum;
    char IPAddress[128], UserName[128], UserPasswd[128];
    FTContext ctx = 0;
    int res = 0;

```

```

if( EBMInfo.Status==Connected || EBMInfo.Status==WaitOnConversation )
    return;

if (!GetConfig("ft.cfg",IPAddress, &PortNum, UserName, UserPasswd)){
    unsigned int ClientID = 0;

    DebugMsg("EBM Params Used: [%s] [%d] [%s] [%s]",
        IPAddress, PortNum, UserName, UserPasswd);
    ClientID = time(0)%33333+30000;

    ctx = FTCreateContext();
    res = FTSetContextAttribute(ctx, CTX_APPL_AUTHFILE, "license.xml");

    convID = FTOpenConversationExt( IPAddress,
                                    PortNum,
                                    FTUserTrader,
                                    ClientID,
                                    UserName,
                                    UserPasswd,
                                    MyVersion,
                                    0,
                                    OpenProcEBM,
                                    BrokenProc,
                                    12345,
                                    0,
                                    ctx);

    FTDestroyContext(ctx);

    DebugMsg("FTOpenConversation: %d, [%s]", convID, FTGetErrorString(convID));
    if (convID>=0){
        EBMInfo.Status = WaitOnConversation;
        EBMInfo.ConversationID = convID;
        EBMInfo.Socket=(int)FTGetConversationSocket(convID);
        DebugMsg("EBM Socket: [%d]",EBMInfo.Socket);
    }
}
else
    DebugMsg("Cannot read file: ft.cfg");
}

/*
    Implements a Timer to keep alive the connection
*/
static void KeepEBMConnection()
{
    static int LastAttempt = 0;

    if(time(0)-LastAttempt<CONNECTION_TIMER_SEC)
        return;

    TryToConnectEBMServer();
    LastAttempt = time(0);
}

/*****
    SELECT
*****/

static fd_set readfds, writefds;

static void ConversationProc(long ConvID, int isWritePending)
{
    const long socket = FTGetConversationSocket(ConvID);

    FD_SET(socket, &readfds);
    if (isWritePending)
        FD_SET(socket, &writefds);
}

/* Implements a process wait on the icoming/outgoing sockets. */
static void MarketWait()
{
    int res, num_conv;
    struct timeval max_sleep;

```

```

    FD_ZERO(&readfds);
    FD_ZERO(&writefds);

    num_conv = FTEnumConversation(ConversationProc);

#ifdef WIN32
    /* If no conversation is active
     * implements a 3 sec. timeout
     */
    if (num_conv<1){
        Sleep(3*1000);
        DebugMsg("*** No Connections ***");
        return;
    }
#endif

    max_sleep.tv_sec    = 1;
    max_sleep.tv_usec   = 0;

    res = select(FD_SETSIZE, (void *)&readfds, (void *)&writefds, 0, &max_sleep);
    if (res<0)
        perror("select");
    if (res==0)
        DebugMsg("*** select: Time-Out ***");
}

static void SetTrace()
{
    int res = -1;
    int opt = FTTRACE_OPT_None;

    /*
     Note:
     FTTRACE_OPT_DataTransfert is a large-detailed log mode,
     to use in specific cases only.
    */

    opt =    opt
            | FTTRACE_OPT_APIInOut
            | FTTRACE_OPT_FlowControl
            | FTTRACE_OPT_DataIn
            | FTTRACE_OPT_DataOut
            /*|FTTRACE_OPT_DataTransfert*/
            ;

    res = FTSetTraceMode(0, FTLevel_Full, opt, 1, 1, 0);
    /* Redirect Log-events from handler to file */
    DebugMsg("FTSetTraceMode: [%s]", FTGetErrorString(res));
}

/*****
MAIN
*****/

int main(int argc, char *argv[])
{
    long res = -1;

    signal(SIGINT, UserAbort);
    MarketInfoReset(&EBMInfo);
    DebugMsg("FT/API Example Start: [%s]", FTGetLibraryVersion());

    SetTrace();

    /*res = FTLoadLicence(0);*/      /* Deprecated licence call */

    res = FTInit(TraceProc);
    DebugMsg("FTInit: %s", FTGetErrorString(res));
    if (res != FTOK) {
        DebugMsg("FTInit: %s", FTGetErrorString(res));
        return -1;
    }

    res = FTExtendClass(InitSkeletonexp());

```

```
if (res != FTOK) {
    DebugMsg("FTExtendClass EBM: %s", FTGetErrorString(res));
    return -1;
}

res = FTStart();
if (res != FTOK) {
    DebugMsg("FTStart: %s", FTGetErrorString(res));
    return -1;
}

do {
    KeepEBMConnection();
    FTRun();
    MarketWait();
} while ( !Abort && EBMInfo.Status != Broken);

if (EBMInfo.Status==Connected){
    res = FTCloseConversation(EBMInfo.ConversationID);
    DebugMsg("FTCloseConversation EBM: %s", FTGetErrorString(res));
}

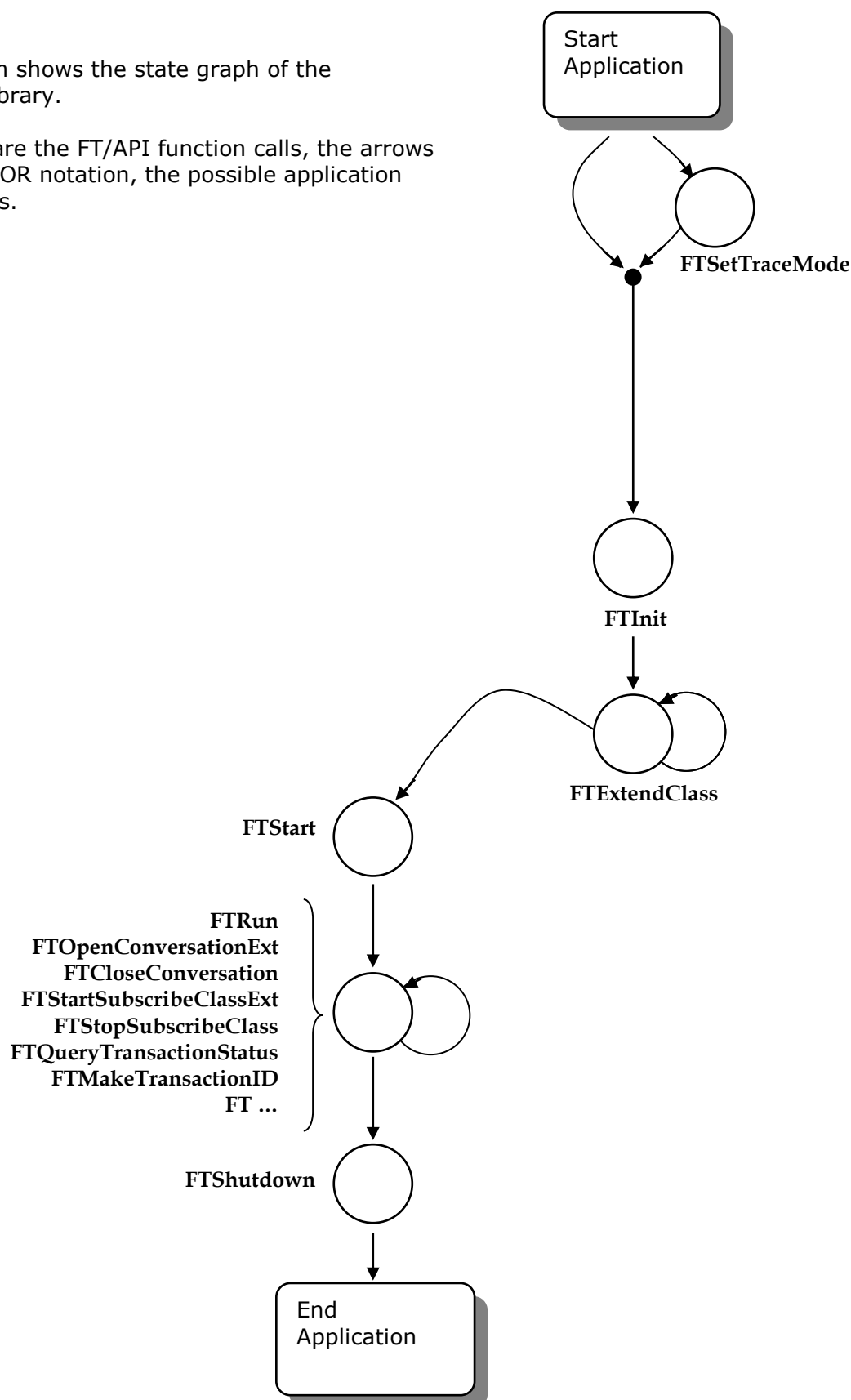
DebugMsg("FT/API Example Stop: [%s]", FTGetLibraryVersion());
res = FTShutdown();
DebugMsg("FTShutdown: %s", FTGetErrorString(res));

return 0;
}
```


APPENDIX F: FT/API STATES

The diagram shows the state graph of the FastTrack library.

The nodes are the FT/API function calls, the arrows indicate, in OR notation, the possible application control flows.



APPENDIX G: EXAMPLE OF A CONNECTION TO ASIA

This example shows how to use FT/API library to connect an ASIA platform.

```
#include "ftapi.h"
#include "ftlmetamarket.h"

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#ifdef WIN32
#include <windows.h>
#include <WinSock.h>
#else
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#endif

static FTRevision appVer = {1,0,0};
static unsigned long applSign = 732;

#define CSTR_BUF_LEN 256

/*****
 *  STATIC UTILS AND FUNCTIONS
 *****/

void mysleep(long msec){
#ifdef WIN32
    Sleep(msec);
#else
    sleep(msec/1000);
#endif
}

void traceCb(long Eventinfo, char* String){
    FTTraceLevelEnum tlv1 = FTLevel_Undef;
    FTTraceSourceEnum src = FTSource_Used;
    FTGetTraceInfo(Eventinfo, &src, &tlv1);
    if(src == FTSource_Application && tlv1 >= FTLevel_Normal){
        printf("%s\n", String);
    }
}

typedef struct{
    long open_conv_StatusCode;
    long ConversationID;
}conv_desc_rec, *conv_desc_ptr;

//METAMARKET CONNECTION DESCRIPTOR
conv_desc_rec mm_conv_desc;

//an int used to generate a pseudo-random cliendID
static unsigned int cid_ = 30001;

static unsigned int nextCID(){
    if(!cid_){
        cid_ = rand() % 65535;
    }else{
```

```

        cid_++;
    }
    return cid_;
}

static void TransactionID_TO_Str(FTTransactionID *txid, char *stxid){
    sprintf(stxid, "%lu%lu%lu%lu%lu",
        txid->BusinessServiceID,
        txid->ClientServiceID,
        txid->ClientID,
        txid->TimeStamp[0],
        txid->TimeStamp[1]);
}

static void dump_tx_res_info(long conversationID, unsigned long ReqID,
    FTSendTransactionResStruct *resStruct, char *desc){
    char stxid[CSTR_BUF_LEN];
    TransactionID_TO_Str(&resStruct->TransactionID, stxid);

    FTTraceAppl(FTLevel_Normal, "\n--Transaction Response--");
    FTTraceAppl(FTLevel_Normal, "@@%s@@", desc);
    FTTraceAppl(FTLevel_Normal, "  TXRES:      %d - %s", resStruct->TransactionStatus,
        resStruct->TransactionStatus == FTTransOK ? "TXOK" : "TXKO");
    FTTraceAppl(FTLevel_Normal, "  TXID:      %s", stxid);
    FTTraceAppl(FTLevel_Normal, "  ReqID:     %d", ReqID);
    FTTraceAppl(FTLevel_Normal, "  MarketReason: %d", resStruct->MarketReasonCode);
    FTTraceAppl(FTLevel_Normal, "  Class:     %lu", resStruct->ClassID);
    FTTraceAppl(FTLevel_Normal, "  KeyID:     %lu", resStruct->KeyID);
    FTTraceAppl(FTLevel_Normal, "  Action:    %d", (long)resStruct->EntityAction);

    if(resStruct->ResultClassID==FT_C_ERROR_INFO_ID && resStruct->ResultObject){
        FT_C_ERROR_INFOPtr info = (FT_C_ERROR_INFOPtr)resStruct->ResultObject;
        FTTraceAppl(FTLevel_Normal, "    MMReasonCode: %lu", info->MMReasonCode);
        FTTraceAppl(FTLevel_Normal, "    MMErrString:  %s", info->MMErrString);
        FTTraceAppl(FTLevel_Normal, "    MrkReasonCode: %lu", info->MarketReasonCode);
        FTTraceAppl(FTLevel_Normal, "    MrkErrString:  %s", info->MarketErrorString);
    }
    FTTraceAppl(FTLevel_Normal, "-----\n");
}

/*****
* Some req-Id(s)
*****/

#define REQ_ID_MM 20001

#define REQID_SUBSCR_DEPTH      4
#define REQID_FILTER_DEPTH     11

/*****
* FTApiConf // FTApiConfLoader
*****/

typedef struct{
    char host_addr_[CSTR_BUF_LEN];
    unsigned int host_port_;
    char username_[CSTR_BUF_LEN];
    char password_[CSTR_BUF_LEN];
    char license_file_[CSTR_BUF_LEN];
    char serv_name_[CSTR_BUF_LEN];
    int load_mm_dynamic_;
    FTRevision *app_ver_;
    unsigned long app_signature_;
}FTApiConf, *FTApiConfPtr;

//load a configuration
int loadConfig(int argc, char *argv[], FTApiConfPtr conf){

    printf("-----\n");
    printf("host: %s \n", argv[1]);
    printf("port: %s \n", argv[2]);
    printf("username: %s \n", argv[3]);

```

```

printf("password: %s \n", argv[4]);
printf("license: %s \n", argv[5]);
printf("serv name: %s \n", argv[6]);
printf("-----\n");

strcpy(conf->host_addr_, argv[1]);
conf->host_port_ = atoi(argv[2]);
strcpy(conf->username_, argv[3]);
strcpy(conf->password_, argv[4]);
strcpy(conf->license_file_, argv[5]);
strcpy(conf->serv_name_, argv[6]);
conf->app_ver_ = &appVer;
conf->app_signature_ = applSign;
conf->load_mm_dinamic_ = 1;
return 0;
}

/////////////////////////////////REQUEST INFO/////////////////////////////////

static void ReqInfoTransactionResProc(long conversationID, unsigned long ReqID,
FTSendTransactionResStruct *resStruct){
    char stxid[CSTR_BUF_LEN];
    dump_tx_res_info(conversationID, ReqID, resStruct, "REQ-INFO");

    TransactionID_TO_Str(&resStruct->TransactionID, stxid);
}

/////////////////////////////////APPLICATION LOGIC ON SUBSCRIBED CLASSES
/////////////////////////////////

long handleFT_C_DEPTH(long ConversationID, unsigned long ReqID, FTEventTypeEnum EventType,
FTNotifySubscribeResStruct *ResStruct){
    FT_C_DEPTHPtr obj = (FT_C_DEPTHPtr)ResStruct->MarketObject;
    switch (ResStruct->EntityAction) {
        case FTEntityKIL:
            if (ResStruct->KeyID <= 0 && !ResStruct->MarketObject){
                // This means a class reset at the server level.
                // Every entity received previously have to be invalidated!
                FTTraceAppl(FTLevel_Normal, "FTEntityKIL Action - Class Reset Signal - New
version: %d", (ResStruct->TimeStamp)[0]);
            }
            else{
                FTTraceAppl(FTLevel_Normal, "FTEntityKIL Action - KeyID [%d]", ResStruct-
>KeyID);
            }
            break;
        case FTEntityDEL:
            FTTraceAppl(FTLevel_Normal, "EntityDEL Action - KeyID [%d]", ResStruct->KeyID);
            break;
        case FTEntityADD:
        case FTEntityRWT:
            FTTraceAppl(FTLevel_Normal, "EntityADD/RWT Action - KeyID [%d]", ResStruct-
>KeyID);
            FTTraceAppl(FTLevel_Normal, "FT_C_DEPTH [BID QTY:%f BID PR:%f] [ASK QTY:%f ASK
PR:%f]", obj->Bid[0].Qty, obj->Bid[0].Price, obj->Ask[0].Qty, obj->Ask[0].Price);
            break;
    }
    return 0;
}

/////////////////////////////////SUBSCRIPTION
CALLBACKS/////////////////////////////////

static void NotifyClass(long ConversationID, unsigned long ReqID, FTEventTypeEnum EventType,
FTNotifySubscribeResStruct *ResStruct){
    switch(EventType){
        case FTConnectionBroken:
            FTTraceAppl(FTLevel_Error, "onMMSsubscribeNotify: connection broken (%d,%d)" ,
ConversationID, ReqID);
            break;
    }
}

```

```

case FTClassIdle:
    FTTraceAppl(FTLevel_Normal, "idle received on class %d", ResStruct->ClassID);
    break;
case FTClassInstance: {
    switch(ResStruct->ClassID){
        case FT_C_DEPTH_ID:
            handleFT_C_DEPTH(ConversationID, ReqID, EventType, ResStruct);
            break;
        default:
            break;
    }
}
default:
    break;
}
}

static void onSubscribeResProc(long convID, unsigned long ReqID, long ResultCode,
FTSubscribeResStruct *resStruct){
    FTTraceAppl(FTLevel_Normal, "----- FTOpenSubscribeResProc: classID [%lu],
SubscriptionID [%lu]", resStruct->ClassID, resStruct->SubscriptionID);
    FTTraceAppl(FTLevel_Normal, "----- Result: %s", FTGetErrorString(ResultCode));

    if (ResultCode == FTOK){
        FTTraceAppl(FTLevel_Normal, "----- Status: %s", (resStruct-
>SubscribeStatus==FTSubscribeStart)? "Start" : "Stop");
        FTTraceAppl(FTLevel_Normal, "----- New Table version: %lu; To-Reset signal: %d",
resStruct->ClassVersion, resStruct->ResetClass);

        if (resStruct->ResetClass){
            // In this case your request can't be served in incremental manner!
            // Any previous record received have to be considered invalidated and
            // the server will resend all the class content.
            FTTraceAppl(FTLevel_Normal, "----- Class Reset - Class %d: New version [%lu]",
resStruct->ClassID, resStruct->ClassVersion);
        }
    }
}

////////////////////////////////////MM OPEN-CONV
CALLBACKS////////////////////////////////////
void onMMOpenConvCallback(long ConversationID, long ResultCode, FTOpenResStruct *ResStruct){
    long res = 0;
    char filter[256] = {'\0'};

    if((mm_conv_desc.open_conv_ResultCode = ResultCode) == FTOK){
        FTTraceAppl(FTLevel_Normal, "metamarket connection is ON!");
        FTTraceAppl(FTLevel_Normal, "Market Revision: V%d.%d.%d", ResStruct->MarketRevision[0],
ResStruct->MarketRevision[1], ResStruct->MarketRevision[2]);

        mm_conv_desc.ConversationID = ConversationID;

        FTTraceAppl(FTLevel_Normal, "subscribing FT_C_DEPTH class..");

        //subscribe something

        res = FTStartSubscribeClassExt(ConversationID,
            FTAckUnrequired,
            FTFlowLast,
            FT_C_DEPTH_ID,
            0, 0,
            FTQueryAll,
            FT_C_DEPTHKey,
            0, 0,
            onSubscribeResProc,
            NotifyClass,
            REQID_SUBSCR_DEPTH,
            0, 0);

        //send req info
        {
            FT_C_REQUEST_INFOWRec req;
            long res = 0;

```

```

FTTransactionID TransactionID;
memset(&req,0,sizeof(req));
strcpy(req.Securities[0].SecurityID, "FTSECID");
req.NSecurities = 1;
req.ReqCode = FT_C_REQUEST_CODE_SubExt;

res = FTMakeTransactionIDExt (ConversationID, FT_C_REQUEST_INFO_ID, &TransactionID);
FTTraceAppl(FTLevel_Normal, "FTMakeTransactionIDExt [%s]", FTGetErrorString(res));

res = FTSendTransactionExt(ConversationID,
                           &TransactionID,
                           FTEntityADD,
                           FT_C_REQUEST_INFO_ID,
                           1,
                           &req,
                           ReqInfoTransactionResProc,
                           FTFalse,
                           0,
                           999);

FTTraceAppl(FTLevel_Normal, "FTSendTransactionExt [%s]", FTGetErrorString(res));
}

}else{
    FTTraceAppl(FTLevel_Normal, "metamarket connection FAILED! (%d)", ResultCode);
}
}

void onMMBrokenConvCallback(long ConversationID, FTBrokenTypeEnum BrokenType){
    FTTraceAppl(FTLevel_Error, "onMMBrokenConvCallback: connection broken (%d, %d)" ,
    ConversationID, BrokenType);
}

//////////***** INITIALIZE FT FRAMEWORK *****///////////
long initializeFT(FTApiConfPtr conf_, FTContext ft_ctx_){
    long result = FTOK;

    //ftinit
    if((result = FTInitExt(traceClbk, ft_ctx_)) != FTOK){
        return result;
    }

    //metamarket classes loading
    if(conf_>load_mm_dynamic){
        //load classes by runtime
        if((result = FTEndClassByLibrary(NULL, "ftlmetamarket", NULL)) != FTOK){
            FTTraceAppl(FTLevel_Error, "metamarket classes loading failed with error: %d",
            result);
            return result;
        }
    }else{
        printf("unsupported\n");
        return -1;
    }

    //ftstart
    if((result = FTStart()) != FTOK){
        return result;
    }

    return result;
}

//you can even open connection to metamarket on another thread.. only remember to synchronize
all FT calls!!!
long connectToServer(FTApiConfPtr conf_, FTContext ft_ctx_){
    long result = FTOK;
    int n = 1;
    unsigned int mm_cid_ = nextCID();

    //opening conversation.
    //a little logic is made here to attempt to reconnect if an invalidCID is provided.
    while((result = FTOpenConversationExt(conf_>host_addr_, conf_>host_port_, FTUserTrader,

```

```

mm_cid_, conf_>username_, conf_>password_, *conf_>app_ver_, 0, onMMOpenConvCallback,
onMMBrokenConvCallback, REQ_ID_MM, conf_>serv_name_, ft_ctx_) != FTOK){
    switch(result){
        case FTExceedSession:
            FTTraceAppl(FTLevel_Normal, "FTOpenConversationExt: retrying in 1 sec..");
            mysleep(2000);
            continue;
        case FTInvalidCID:
            FTTraceAppl(FTLevel_Normal, "FTOpenConversationExt: attempt to connecting
            #d failed due to invalid CID", n++);
            mm_cid_ = nextCID();
            break;
        default:
            FTTraceAppl(FTLevel_Error, "FTOpenConversationExt: attempt to connecting
            failed due to an error: %d", result);
            return result;
    }
}
return result;
}

int price_exec_subscribed = 0;

////////////////////MAIN////////////////////////////////////
// launch with params: ip port user passw licencefile service

//replace your connection params
static char *argv[] = {"", "10.91.195.211", "20000", "FASTTRACK", "*", "license.xml",
    "PUBLMETAMARKET|PRIVMETAMARKET"};

int main(int argc, char *argv[]){
    long result = FTOK;
    FTContext ft_ctx_ = NULL;

    FTApiConf conf_;
    memset(&conf_, 0, sizeof(FTApiConf));
    if(loadConfig(argc, argv, &conf_)){
        printf("PARAMETERS ERROR! exiting..\n");
        return 1;
    }

    //create an ft-context and set its properties
    ft_ctx_ = FTCreateContext();
    FTSetContextAttribute(ft_ctx_, CTX_COMPRESSED, "ZIP");
    FTSetContextAttribute(ft_ctx_, CTX_APPL_AUTHFILE, conf_.license_file);

    //initialize ft-framework
    if(initializeFT(&conf_, ft_ctx_) != FTOK){
        //error
        FTShutdown();
        return 1;
    }

    //metamarket connection.
    //remember that connection will not be done until you call at least one FTRun() call.
    if(connectToServer(&conf_, ft_ctx_) != FTOK){
        //error
        FTShutdown();
        return 1;
    }

    //////////////////////**** MAIN LOOP ****////////////////////////////////////

    //main loop is *necessary* to call FTRun, and it is must be synchronized with all other
    threads you eventually start.
    //calling FTRun() ensures that all requests (subscriptions, transactions ..) will be
    handled by FT framework durig program lifecycle.
    //you can also eventually move it to another thread.
    do{
        if(FTRun() != FTOK){
            printf("unrecoverable FTApi Error");
            FTShutdown();
        }
        FTSocketSelect();
    }while(1);
}

```

```
//unreachable code  
return 0;  
}
```


To CONTACT Us

Any comments or requests for clarifications are welcome.

Email

Marketing: marketing@list-group.com
General Support: helpdesk@list-group.com
FT/API Programming Support: ftapi@list-group.com

Website

www.list-group.com

Offices

List SpA

Via Pietrasantina, 123 56122 **Pisa** - Italy
Tel. +39 050 80 01 51 – Fax +39 050 80 01 701

Foro Buonaparte, 76 20121 **Milano** - Italy
Tel. +39 02 80 28 91 – Fax +39 02 80 51 040

Via Cavour, 24 10123 **Torino** - Italy
Tel. +39 011 81 01 211 – Fax +39 011 83 58 83

Via Camporegio, 5 53100 **Siena** - Italy
Tel. +39 0577 05 741 – Fax +39 0577 05 74 99

Via Carducci, 20 34125 **Trieste** - Italy
Tel. +39 040 98 5100 – Fax +39 040 98 51 099

Piazza Duomo, 57 27058 **Voghera** (PV) - Italy
Tel + 39 0383 64 35 11 - Fax + 39 0383 64 35 10

List UK Ltd

4th floor, 45 Ludgate Hill **London** EC4M 7JU - UK
Tel. +44 (0)203 393 43 70 - Fax +44 (0)203 393 43 72

List USA Inc

5 Penn Plaza, Suite 3600 – **New York**, NY, 10119 USA
Tel. +1 212 83 51 622 - Fax +1 212 84 96 901

List Polska S.A.

Plac Trzech Krzyzy, 3 - 00-535 **Warszawa** - Poland
Tel +48 22 584 70 11 - Fax +48 22 584 70 14

List Technology Iberica SA

C/Zurbano, 5 – 1º 28010 **Madrid** - Spain
Tel +34 917 88 82 00 - Fax +34 917 88 82 32

List Sdn Bhd

35-2 Jalan Putra Mahkota 7/7B - Putra Point Business Centre - Putra Heights
47650 Subang Jaya - Selangor Darul Ehsan – Malaysia
Tel. +603 5191 1522 - Fax +603 5192 2312

List India Private Limited

S-11, 2nd Floor, Haware's Centurion - Plot No. 88-91, Sector 19-A, **Nerul**
400706 Navi Mumbai India
Tel. +91 22 27703445 - Fax +91 22 27703445