QUANTHOUSE

FeedOS
The Market Data Operating System

API User Guide & Reference Guide

FeedOS v3.5 2009-04-02

FeedOS API User Guide

Feed Publication

USER AND REFERENCE GUIDE

FeedOS API – Feed Publication

Index

Document History

Date	Author	Action
2006-03-15	D.Fenouil	Initial version (FeedOS Server Framework)
2008-09-16	D.Fenouil	Requests made visible in C++ Client API
2008-09-20	D.Fenouil	Instrument Codes in L1/L2 become Polymorphic; Added support for MarketNews
2009-02-26	D.Fenouil	Command Line Tool Added support for FeedStatus
2009-03-23	D.Fenouil	Added Sample Architecture
2009-04-02	D.Fenouil	TextBridge protocol Minor updates Nota Bene about ranges for custom tags

Overview

This document gives an overview of Feed Publication requests in FeedOS API.

Alternate methods (Command Line Interface and TextBridge protocol) are available to relieve users from integrating C++ or Java API.

Pre-requisite reading

Users need to first read the regular User Guide (either for C++ API or Java API).

The target FeedOS server must be configured to allow publication, and user permissions should be set accordingly for the specified login(s).

Throughout this document, examples will be provided using C++ API. Equivalent code in Java should be very similar.

See `samples/sample_publisher/*` for example code.

Features available

Users can manage instruments and generate market data events. Both custom and existing instruments can be manipulated. Status messages about feeds and markets can be generated, too.

Here is an overview of available requests:

- Referential: add/remove/update instruments
- Level 1: Last price, trades, best bid/ask, pre-defined and custom tags
- Level 1: Custom Values
- Level 2: Order Book
- Status messages: Market News, Feed Status

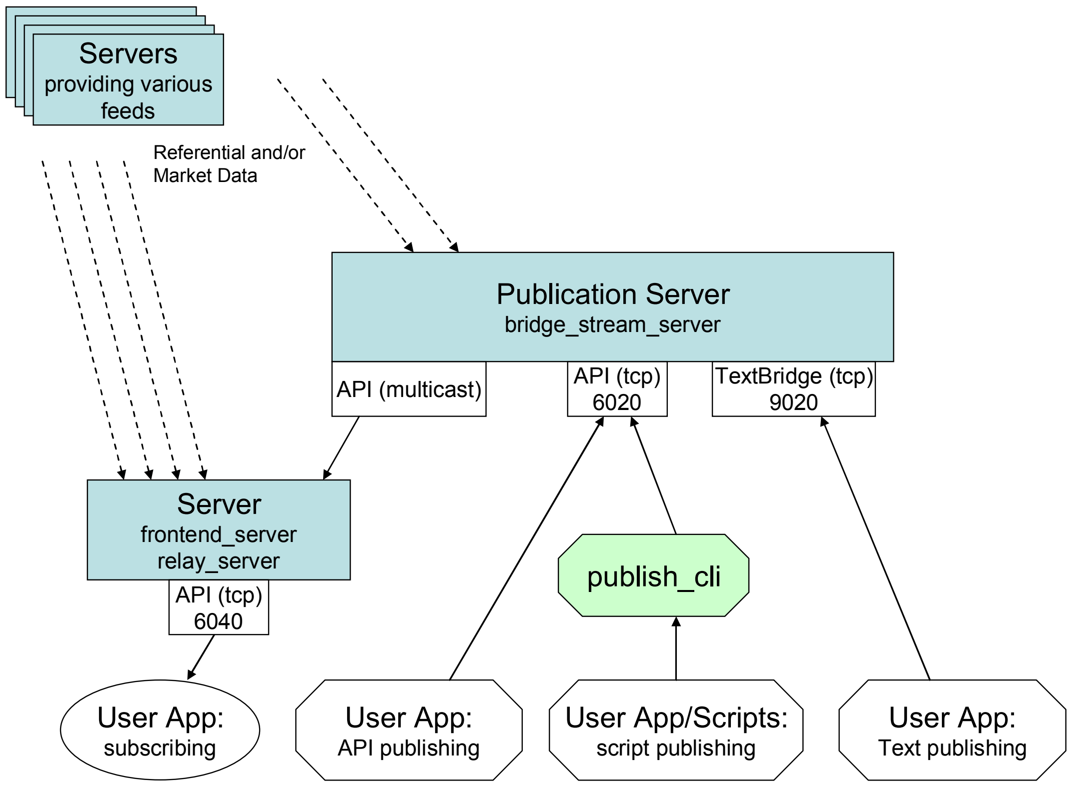
Nota Bene: when publishing custom values, please use tag numbers in reserved ranges: `59000…59999` for quotation data, `60000…60999` for referential.

Sample Architecture

Here is a sample feed publication architecture. It involves the following participants:

- FeedOS servers providing real feeds (referential and/or market data)
  this can be any set of FrontEnds or FeedHandlers
- FeedOS server dedicated to publication
  this is usually a bridge_stream_server
- FeedOS server acting as a FrontEnd
  this is usually a frontend_server or a relay_server
- subscribing application using any API
- publishing application using C++ or Java API
- publishing application using TextBridge protocol
- publishing scripts (or application) using CLI tools

Of course more simple layouts are possible (and more complex ones, too).



API, Initialization

Normal initialisation of API should be performed. Then specific steps are needed to setup feed publication.

Starting a FeedPublisher object

User must instantiate a FeedPublisher object that will be used to send requests. Such object should be started before use.

Example:

```
{

  FeedOS::FeedPublisher publish_obj;

  publish_obj.start_object();

       (…)
```

Terminating a FeedPublisher object

When application exits, it should cleanly terminate the instance of FeedPublisher. Prior destruction the object must be stopped to ensure that all pending requests are completed.

Example:

```
       (…)

  publish_obj.stop_object();

}
```

API, Custom Instruments

If published feed is for custom instruments, user has to create them in the server prior generating market data events.

Creating instruments

Each custom instrument should be created in the server. This implies that an internal numeric code is allocated. To create an instrument and receive the corresponding instrument code, call:

```
ReturnCode FeedPublisher::ref_AllocateNewInstrument
      (
            RequestHandler & req_handler,
            // outputs
            FOSInstrumentCode & Code,
            // inputs
            ListOfReferentialAttribute const & Attributes,
            AllocateNewInstrumentPolicy OverwritePolicy
      );
```

The request is issued synchronously (i.e. blocking function call). The instrument will be created from the given list of referential attributes (see `api/tags_referential.hpp` for the list). An optional "policy" tells what to do when the target instrument already exists. Policy can take the following values:

- **`DoNotOverwrite`** (this is the default)
  If instrument already exists, then the request should fail.

- **`Overwrite`**
  If instrument already exists, then its attributes are overwritten (if needed) by the given list; preserving all other attributes already present.

- **`ResetAndOverwrite`**
  If instrument already exists, then its existing attributes are cleared before the given list is loaded.

If return code is not RC_OK, then the request failed. If request succeeded, the internal numeric code that has been allocated is returned.

Nota Bene: When creating instruments, two attributes are mandatory, because they are used to uniquely identify an instrument:

- FOSMarketId: tells what is the parent market

- LocalCodeStr:: unique identifier within the parent market

Updating instruments

Referential attributes of any instrument (custom or not) can be updated. Call:

```
ReturnCode FeedPublisher::ref_UpdateInstrumentAttributes
      (     RequestHandler & req_handler,
            // inputs
            PolymorphicInstrumentCode const & Code,
            ListOfReferentialAttribute const & Attributes
      );
```

Deleting instruments

Custom instruments that have been created (via "Allocate" request) can be deleted (i.e. "unallocated"). Call:

```
ReturnCode FeedPublisher::ref_UnallocateInstruments
      (
            RequestHandler & req_handler,
            // inputs
            ListOfPolymorphicInstrumentCode const & Codes
      );
```

Depending on server configuration, instruments are either "hidden" or really deleted.

Nota Bene: FeedOS server remembers about previously-allocated instruments. In case of deletion followed by re-creation, a given instrument will get the same internal code as the first time it was created.

API, Publishing Market Data

Once target instruments have been located (or created), user can start publishing market data events. Instruments are referenced through thie internal numeric code.

Level 1 events

Level 1 stream carries trades, best bid/ask, open/high/low/close prices, trading status and other price-related values.

An all-purpose request is available to generate a `TradeEventExt` message. This message can carry any combination of trade, open/high/low/close signal, best bid/ask, trade conditions and other "contextual flags" or any other value (see `api/tags_quotation.hpp` for the list).

```
  void FeedPublisher::quot_GenerateTradeEventExt
      (     RequestHandler & req_handler,
            // inputs
            PolymorphicInstrumentCode const & Code,
            Timestamp const & ServerUTCTimestamp,
            Timestamp const & MarketUTCTimestamp,
            QuotationTradeEventExt const & Data
      );
```

A few simplified functions are available. They are dedicated to sending trades, best bid/ask and miscellaneous values. As an example, here is the function dedicated to updating one tag with a value whose type is `float64` :

```
void FeedPublisher::quot_simple_generate_other_value
      (     RequestHandler & req_handler,
            // inputs
            PolymorphicInstrumentCode const & Code,
            Timestamp const & MarketUTCTimestamp,
            TagNumber   quotation_tag_number,
            float64     tag_value
      );
```

See class `framework/FeedPublisher` for details.

Nota Bene: to publish custom values (not standard tags like LastPrice, DailySettlementPrice,etc) please select tag numbers in the range dedicated to user-specific values. Example:

```
      TagNumber my_tag = TAG_QUOT_USER_0 + N;
```

Where ***N*** ranges from 0 to `NB_TAG_QUOT_USER`.

Level 2 events

Level 2 stream carries top-of-the-book prices. For each side of the book (Bid and Ask) a list `OrderBookEntry` is available. At the moment each entry contains only a price and a quantity.

Two requests are available to generate **incremental** order book updates:

```
// update whole or part of order book
void FeedPublisher::quot_GenerateOrderBookRefresh
      (     RequestHandler & req_handler,
            // inputs
            PolymorphicInstrumentCode const & Code,
            Timestamp const & ServerUTCTimestamp,
            int8 BidChangeIndicator,
            int8 AskChangeIndicator,
            ListOfOrderBookEntry const & partial_BidLimits,
            ListOfOrderBookEntry const & partial_AskLimits
      );

// small incremental update of order book
void FeedPublisher::quot_GenerateOrderBookDeltaRefresh
      (     RequestHandler & req_handler,
            // inputs
            PolymorphicInstrumentCode const & Code,
            Timestamp const & ServerUTCTimestamp,
            OrderBookDeltaRefresh const & Delta
      );
```

See `samples/sample_cli/perform_subscribe_book2.cpp` to know how Refresh and DeltaRefresh can be used to incrementally update order book.

A simplified version exists that allows refreshing the whole order book (Bid and Ask, all limits) in a single request:

```
// update order book as a whole
void FeedPublisher::quot_simple_generate_full_book
      (     RequestHandler & req_handler,
            // inputs
            PolymorphicInstrumentCode const & Code,
            ListOfOrderBookEntry const & full_BidLimits,
            ListOfOrderBookEntry const & full_AskLimits
      );
```

API, Other Events

Market News

Market news are usually available from exchanges, and carry various messages: administrative notices, corporate events, etc.
Users can publish them in FeedOS by call the following synchronous function:

```
// generate a market news
ReturnCode FeedPublisher::status_GenerateMarketNews
      (     RequestHandler & req_handler,
            // inputs
            MarketNews const & News
      );
```

Structure `FeeOS::Types::MarketNews` provides the following methods:

```
void setOrigMarketId     (FOSMarketId origmarketid)
void setOrigUTCTime      (Timestamp const & origutctime)
void setUrgency          (FIXMarketNewsUrgency urgency)
void setHeadline         (String const & headline)
void setURLLink          (String const & urllink)
void setContent          (String const & content)
void setRelatedInstruments
                  (ListOfPolymorphicInstrumentCode const &)
```

The parameter Urgency can take the following values:

- `FIXMarketNewsUrgency_Normal`
- `FIXMarketNewsUrgency_Flash`
- `FIXMarketNewsUrgency_Background`

Feed Status

Such events are used to inform client applications of disruptions or degradation of feeds.

```
            // generate a FeedStatus message
      ReturnCode status_GenerateFeedStatusUpdate
            (
                  RequestHandler & req_handler,
                  // inputs
                  String const & Sender,
                  Timestamp const & SenderUTCTimestamp,
                  FeedOS::Types::FeedStatus const & Status
            );
```

Structure `FeeOS::Types::FeedStatus` provides the following methods:

```
void setFeed(FeedDescription const & feed);
void setOverallUsability(FeedUsability const & overall_u);
void setServices(ListOfFeedServiceStatus const & services);
```

Structure `FeeOS::Types::FeedDescription` provides the following methods:

```
void setFeedName(String const & feedname);
void setInternalSourceIDs
                  (ListOfFeedInternalSourceID const & l);
```

Structure `FeeOS::Types::FeedUsability` provides the following methods:

```
void setState(FeedState state);
void setLatencyPenalty(bool latencypenalty);
void setOutOfDateValues(bool outofdatevalues);
void setBadDataQuality(bool baddataquality);
```

The parameter `FeedState` can take the following values:

- `FeedState_Active`
- `FeedState_ProbablyNormal`
- `FeedState_ProbablyDisrupted`
- `FeedState_Disrupted_TechnicalLevel`
- `FeedState_Disrupted_ExchangeLevel`

Command Line Tool

A CLI tool can be used to generate Publication requests. This can be used either interactively or from scripts. PUBLISH_CLI is an extension of FEEDOS_CLI that brings more commands. See FEEDOS_CLI documentation first.

Creating Instruments

Syntax

```
PUBLISH.AllocateNewInstrument <Arguments>
```

Where <Arguments> is:

```
       ( <attributes:ReferentialAttribute> )
                    { <num:TagNumber>
                      <value:Any>
                    }
       <overwritepolicy:AllocateNewInstrumentPolicy>
```

Example Input Script

```
set POLICY ResetAndOverWrite
set TAGS_ID { FOSMarketId XTKS } { LocalCodeStr my-indicator }
set TAG_DESCR { Description "sample custom instrument" }
set TAG_CFICODE { CFICode MXXXXX }
set ALL_TAGS ( $TAGS_ID $TAG_DESCR $TAG_CFICODE )
PUBLISH.AllocateNewInstrument $ALL_TAGS $POLICY
```

Checking Result

```
publish_cli (joe@localhost:6020) > geti XTKS@my-indicator
instr # 147/700000 = 308981344
    Description                  string{sample custom instrument}
    SecurityType                 string{NONE}
    FOSMarketId                  XTKS
    CFICode                      string{MXXXXX}
    InternalCreationDate         Timestamp{2009-02-27 18:27:02:079}
    InternalModificationDate     Timestamp{2009-02-27 18:27:02:079}
    InternalSourceId             uint16{10000}
    LocalCodeStr                 string{my-indicator}
```

Updating Instruments

Syntax

```
PUBLISH. UpdateInstrumentAttributes <Arguments>
```

Where <Arguments> is:

```
        <code:PolymorphicInstrumentCode>
        ( <attributes:ReferentialAttribute> )
                       { <num:TagNumber>
                         <value:Any>
                       }
```

Example Input Script

```
PUBLISH.UpdateInstr XTKS@my-indicator ( {Symbol FOO} )
```

Checking Result

```
publish_cli (joe@localhost:6020) > geti XTKS@my-indicator
instr # 147/700000 = 308981344
    Symbol                      string{FOO}
    Description                 string{sample custom instrument}
    SecurityType                string{NONE}
    FOSMarketId                 XTKS
    CFICode                     string{MXXXXX}
    InternalCreationDate        Timestamp{2009-02-27 18:27:02:079}
    InternalModificationDate    Timestamp{2009-02-27 18:27:02:079}
    InternalSourceId            uint16{10000}
    LocalCodeStr                string{my-indicator}
```

Deleting Instruments

Syntax

```
PUBLISH.UnallocateInstruments <Arguments>
```

Where <Arguments> is:

```
        ( <codes:PolymorphicInstrumentCode> )
```

Example Input Script

```
PUBLISH.UnallocateInstruments ( XTKS@my-indicator XCME@another )
```

Checking Result

```
publish_cli (joe@localhost:6020) > geti XTKS@my-indicator
instr # 147/700000 = 308981344
    Symbol                      string{FOO}
    Description                 string{sample custom instrument}
    SecurityType                string{NONE}
    FOSMarketId                 XTKS
    CFICode                     string{MXXXXX}
    InternalCreationDate        Timestamp{2009-02-27 18:27:02:079}
    InternalModificationDate    Timestamp{2009-02-27 18:27:02:079}
    InternalHideFromLookup      bool{True}
    InternalSourceId            uint16{10000}
    LocalCodeStr                string{my-indicator}
```

Level 1 events

Syntax

```
PUBLISH.GenerateTradeEventExt <Arguments>
```

Where <Arguments> is:

```
       <code:PolymorphicInstrumentCode>
       <serverutctimestamp:Timestamp>
       <marketutctimestamp:Timestamp>
       {
         ### THIS INDICATES WHAT FIELDS ARE RELEVANT
         <contentmask:QuotationContentMask>
         ### BID
         { <price:float64>
           <qty:float64>
           <nborders:int32>
         }
         ### ASK
         { <price:float64>
           <qty:float64>
           <nborders:int32>
         }
         ### PRICE/TRADE
         <lasttradeqty:float64>
         <price:float64>
         ### LIST OF "CONTEXT FLAGS"
         ( <context:QuotationContextFlag> )
         ### LIST OF MISC. VALUE
         ( <values:QuotationVariable> )
       }
```

Example Input Script

```
set INSTR XTKS@my-indicator
set SERVER_TIME now
set MARKET_TIME "2009-01-25 08:00:00:125"
set TIMESTAMPS $SERVER_TIME $MARKET_TIME
set CONTENT Bid|LastPrice|LastTradeQty|OtherValues
set BID { 2 1000 -1 }
set ASK { 0 0 0 }
set TRADE 3 200
set CONTEXT ()
set MISC_VALUES ( { DailyTotalVolumeTraded 9999 } )
set EVENT_DATA { $CONTENT $BID $ASK $TRADE $CONTEXT $MISC_VALUES }

PUBLISH.GenerateTradeEventExt $INSTR $TIMESTAMPS $EVENT_DATA
```

Checking Result

NB: execute before issuing publication request

```
publish_cli (joe@localhost:6020) > sub1 XTKS@my-indicator
EV 147/700000   MarketUTCTime: 2009-01-25 08:00:00:125
                ServerUTCTime: 2009-02-27 20:14:37:100
content: Bid LastPrice LastTradeQty OtherValues
        BestBid      = 2      1000
        LastTradeQty = 3
        LastPrice    = 200
VALUES:
    DailyTotalVolumeTraded      float64{9999}
```

Level 2 events

Here is an example that generates a full refresh of top-of-the-book data, width depth=3.

Incremental refreshes are possible by using combinations of GenerateOrderBookDeltaRefresh and GenerateOrderBookRefresh.

Syntax

```
PUBLISH.GenerateOrderBookRefresh <Arguments>
```

Where <Arguments> is:

```
       <code:PolymorphicInstrumentCode>
       <serverutctimestamp:Timestamp>
       <bidchangeindicator:int8>
       <askchangeindicator:int8>
       ( <bidlimits:OrderBookEntry> )
             { <price:float64>
               <qty:float64>
             }
       ( <asklimits:OrderBookEntry> )
             { <price:float64>
               <qty:float64>
             }
```

Example Input Script

```
set INSTR XTKS@my-indicator
set SERVER_TIME now
set BID_INDICATOR -1
set ASK_INDICATOR -1
set BIDS ( { 3 100 } { 2 100 } { 1 100 } )
set ASKS ( { 4 100 } { 5 100 } { 6 100 } )
set TOP_OF_BOOK $BID_INDICATOR $ASK_INDICATOR $BIDS $ASKS
PUBLISH.GenerateOrderBookRefresh $INSTR $SERVER_TIME $TOP_OF_BOOK
```

Checking Result

NB: execute before issuing publication request

```
publish_cli (joe@localhost:6020) > sub2 XTKS@my-indicator
SubscribeInstrumentL2_Started
ticket = 2
InstrumentStatusL2
147/700000
OR 147/700000 bid(0*3) ask(0*3)
   ServerUTCTime=2009-02-25 12:46:19:601
         0      BID    3.0000 x    100  ASK    4.0000 x    100
         1      BID    2.0000 x    100  ASK    5.0000 x    100
         2      BID    1.0000 x    100  ASK    6.0000 x    100
         3      BID ****************    ASK ****************
```

MarketNews events

Syntax

```
PUBLISH.GenerateMarketNews <Arguments>
```

Where <Arguments> is:

```
             { <origmarketid:FOSMarketId>
               <origutctime:Timestamp>
               <urgency:FIXMarketNewsUrgency>
               <headline:String>
               <urllink:String>
               <content:String>
               ( <relatedinstruments:PolymorphicInstrumentCode> )
             }
```

Example Input Script

```
set MIC misc
set TS now
set URG Flash
set HEAD "Great news"
set URL "http://nowhere.com"
set CONTENT "performed +1000% !"
set INSTR_LIST ( XTKS@my-indicator )
set NEWS_DATA { $MIC $TS $URG $HEAD $URL $CONTENT $INSTR_LIST }
PUBLISH.GenerateMarketNews $NEWS_DATA
```

Checking Result

NB: execute before issuing publication request

```
publish_cli (joe@localhost:6020) > SubscribeAllStatus
SubscribeAllStatus_Started
MarketNews
        OrigMarketId    misc
        OrigUTCTime     2009-02-25 18:47:07:429
        Urgency Flash
        Headline        Great news
        URLLink http://nowhere.com
        Content performed +1000% !
        RelatedInstruments      XTKS@my-indicator
```

FeedStatus events

<table>
<tr><td>**Syntax**</td><td>

```
PUBLISH.GenerateFeedStatusUpdate <Arguments>
```

Where <Arguments> is:

```
              <sender:String>
              <senderutctimestamp:Timestamp>
              {
                   { <feedname:String>
                     ( <internalsourceids:FeedInternalSourceID> )
                   }
                   { <state:FeedState>
                     <latencypenalty:bool>
                     <outofdatevalues:bool>
                     <baddataquality:bool>
                   }
                   ( <services:FeedServiceStatus> )
                     { <servicename:String>
                       { <state:FeedState>
                         <latencypenalty:bool>
                         <outofdatevalues:bool>
                         <baddataquality:bool>
                       }
                     }
              }
```

</td></tr>
<tr><td>**Example Input Script**</td><td>

```
set SENDER_N_TIME publisher-app now
set FEED_NAME MY-FEED
set FEED_INTERNAL_IDS ( 10000 )
set FEED_STATE Disrupted_TechnicalLevel
set DEGRADATION_FLAGS false false false
set SERVICES_DETAILS ()
set FEED { $FEED_NAME $FEED_INTERNAL_IDS }
set STATUS { $FEED_STATE $DEGRADATION_FLAGS } $SERVICES_DETAILS
PUBLISH.GenerateFeedStatusUpdate $SENDER_N_TIME { $FEED $STATUS }
```

</td></tr>
<tr><td>**Checking Result**

NB: execute before issuing publication request</td><td>

```
publish_cli (joe@localhost:6020) > feed_status
*** FeedStatusUpdate from publisher-app at 2009-02-25 16:44:04:847
feed name: MY-FEED
internal source IDs: 10000
    feed state          : Disrupted_TechnicalLevel
    latency penalty     : false
    out of date values  : false
    bad data quality    : false
```

</td></tr>
</table>

TextBridge protocol

A simple text-oriented protocol allows performing basic operations through a basic TCP connection.

The FeedOS "bridge_stream_server" allows sending publication requests using a simple text-oriented protocol. A TCP connection should be established by the publishing application towards the appropriate port on the "bridge_stream_server" (usually 9020).
Nota Bene: This port is NOT the one used to connect using API-based application (usually 6020 or 6040).

Overview of Protocol

TextBridge Protocol is made of ASCII commands. Character set is essentially ASCII. Some values of type string allow any kind of 8bit characters (ISO 8859-1, typically).

The protocol is a one-way stream of commands sent by the publishing app towards the server. "One-way" means that no acknowledgement or feeback is sent back to the publishing application. In case of serious error in the protocol the server shall simply close the TCP connection. You should look for error message in the server's log file.

There are two kinds of commands:

1. **Action** commands.
   Based on previous declarations, an event is generated (trade, order book update, creation of custom instrument, etc).

2. **Declaration** commands.
   They carry information: prices, timestamps, etc.
   There are two kinds of Declarations:

   a. **Persistent Declaration**.
      Related information is valid throughout all subsequent Action commands.

   b. **Volatile Declaration**.
      Related information is valid only for the upcoming Action command.

For a given Action, only a subset of declarations is relevant. Non-relevant declarations that may have been issued are simply ignored.

A command is a set of tokens, separated by the space character (ASCII 0x20). Commands are terminated by the null character (ASCII 0x00).
The first token is the command code (a few characters long). Following tokens, if any, are the parameters of the command. Although most commands take only 1 parameter, some of them can take a variable list or some parameters can be optional (i.e. they have default values).

Parameters can be of the following types:

- Character string (no blank allowed, unless it is the latest parameter)
- Integer
- Float
- Boolean, as a 1-char string (`f`=false and `t`=true)
- Price (float + a few special values as character strings)
- Date, as a character string. Format is `YYYYMMDD`
- Time, as a character string. Format is `HHMMSSmmm` or `HHMMSS`
- Timestamp, as a character string.
  Format is `YYYY-MM-DD HH:MM:SS:mmm`

A definition of all supported commands is available as a header file in C language. Each command is described (using C macro definition) with the following information:

- Command code (short character string)
- Command name (verbose description)
- List of arguments (in C language)

This header file should be used as a reference guide. See appendix.
A tiny C++ API is also available. It wraps the C definition and provides a C++ class that relieves users from crafting commands at character level.

Example

As an example, we'll take the command that tells to insert a Bid entry in order book. Here is the definition of the command, in C syntax:

```
FEEDOS_BRIDGE_INTERFACE_CMD(
  "2IB",                                                         // command code
  QUOT_L2_SEND_rt_order_book_insert_BidAtLevel,                  // command name
  (unsigned int n, double price, double qty, int nb_orders)      // parameters
)
```

The C++ method corresponding to the command above has the following signature:

```
void
FeedOS::TextBridge::Writer::
bridge_cmd_QUOT_L2_SEND_rt_order_book_insert_BidAtLevel(
      unsigned int level,
      double price,
      double qty,
      int nb_orders);
```

Thus if we need to build a command to tell "insert a Bid entry in order book at Level=0 with Price=10.25 ; Quantity=50 ; NbOrders=10", we can use the following C++ code:

```
std::stringstream buffer;
FeedOS::TextBridge::Writer w (buffer);
      /* should select_instrument() first */
w.bridge_cmd_QUOT_L2_SEND_rt_order_book_insert_BidAtLevel(
      0     ,     // level
      10.25 ,     // price
      50    ,     // quantity
      10    );    // nb_orders
std::string result = buffer.str();
char const * result_ptr    = result.data();
size_t       result_length = result.size();
      /* should send result in TCP socket */
buffer.str(std::string()); // reset the buffer for next set of commands
```

Here is the resulting character buffer. This should be pushed in the TCP socket connected to the server:

ASCII string	`2IB 0 10.25 50 10`
Dump in hexadecimal	`32 49 42 20 30 20 31 30 2e 32 35 20 35 30 20 31 30 00`

Alternatively, we could have sent that "by hand" from a unix-like command line:

```
echo "2IB 0 10.25 50 10" | tr '\n' '\0' | nc -w1 SERVER PORT
```

Overview of commands

Prior issuing an **Action** command, a few **Declarations** are usually required. Volatile Declarations pertain only to the upcoming Action whereas Persistent Declarations remain valid forever.

Action commands can be grouped in 3 categories:

1. Referential Data: creation / modification of custom instruments
2. Quotation Data: publishing Level 1 events (trades, BBO, status)
3. Quotation Data: publishing Level 2 events (Market By Level order book)

See reference document "TextBridgeInterface_cmd.hpp", where commands are grouped according to their characteristics:

- Action vs Declaration
- Volatile vs Persistent
- Referential vs Quotation

The most common **Declaration commands** are:

What For		**Command Code**	**Command Name**	**Parameters**
Selecting a target instrument		`I`	`select_instrument`	`char const * instrument`
Declaring Date & Time (**PERSISTENT**)	"official" Market Date	`@MD`	`QUOT_set_market_date`	`char const * date`
	"official" Market Time	`@MD`	`QUOT_set_market_time`	`char const * time`
	Date (part of internal timestamp)	`@SD`	`QUOT_set_server_date`	`char const * date`
	Time (internal timestamp)	`@ST`	`QUOT_set_server_time`	`char const * time`

By default, "official market timestamp" is null (not set) whereas "internal timestamp" is set to the current system clock by the publication server.
Instruments can be referenced either by MIC@LocalCodeStr or internal numeric.

Initialization and administrative commands

Initialization

When connecting to a FeedOS publication server via TextBridge protocol, user should declare which protocol version he's using:

TextBridge protocol	`PROTOCOL_VERSION 1.0`
C++	`w.bridge_cmd_select_protocol_version ( FeedOS::TextBridge::CurrentProtocolVersion );`

Sending an informative message

The given message will appear in the log file of the server.
Look for "`TextBridge user msg:`"

TextBridge protocol	`MSG this is an informative message`
C++	`w.bridge_cmd_msg  ("this is an informative message");`

Sending an error message

The given message will appear in the log file of the server.
Look for "`TextBridge user error msg:`"

TextBridge protocol	`ERROR this is an ERROR message`
C++	`w.bridge_cmd_error ("this is an ERROR message");`

Forcing the server to wait a bit

This can be used to perform basic timing. Useful when piping to the server a large set of pre-computed commands.

TextBridge protocol	`s 250`
C++	`w.bridge_cmd_msleep (250);`

Generating basic requests

Here is a list of common requests.

Referential: defining a custom instrument

This is needed if you plan to generate market data for custom instruments (indicators, instruments carrying aggregated prices/order books, etc). User-defined values can been set besides predefined tags.

TextBridge protocol

```
REF_SELECT_MIC XTKS
REF_Description sample custom instrument
REF_CFICode MXXXXX
A/str 60000 my str
A/f64 60001 3.14159
*REF_CREATE my-indicator
```

C++

```
w.bridge_cmd_REF_select_MarketId ("XTKS");
w.bridge_cmd_REF_declare_Description ("sample custom instrument");
w.bridge_cmd_REF_declare_CFICode ("MXXXXX");
w.bridge_cmd_REF_declare_attribute_Syntax_String (60000,"my str");
w.bridge_cmd_REF_declare_attribute_Syntax_float64 (60001, 3.14159);
w.bridge_cmd_REF_SEND_create_or_update ("my-indicator");
```

Checking Result

```
feedos_cli (joe@localhost:6020) > geti XTKS@my-indicator
instr # 147/700000 = 308981344
    Description                 string{sample custom instrument}
    SecurityType                string{NONE}
    FOSMarketId                 XTKS
    CFICode                     string{MXXXXX}
    InternalCreationDate        Timestamp{2009-02-27 18:27:02:079}
    InternalModificationDate    Timestamp{2009-02-27 18:27:02:079}
    InternalSourceId            uint16{10000}
    LocalCodeStr                string{my-indicator}
    REF_USER_0                  string{my str}
    REF_USER_1                  float64{3.14159}
```

Nota Bene: you can set custom referential tags (in the range 60000…60999) using commands starting with “A”. See reference document.

Level 1: Generating a LastPrice

This can be used to declare a price (index spot, custom indicator, indicative price, etc). In the following example, some user-defined value sent in tag # 59000 besides the Last Price.

TextBridge protocol

```
I XTKS@my-indicator
1P 1000
V/f64 59000 0.0001
*1
```

C++

```
w.bridge_cmd_select_instrument ("XTKS@my-indicator");
w.bridge_cmd_QUOT_L1_rt_price (1000);
w.bridge_cmd_QUOT_L1_value_Syntax_float64 (59000, 0.0001);
w.bridge_cmd_QUOT_L1_SEND ();
```

Checking Result

```
feedos_cli (joe@localhost:6020) > sub1 XTKS@my-indicator
EV 147/700000   MarketUTCTime: null
                ServerUTCTime: 2009-04-01 10:04:17:999
content: LastPrice OtherValues
        LastPrice     = 1000
VALUES:
    QUOT_USER_0                  float64{0.0001}
```

Nota Bene: you can set custom quotation tags (in the range 59000…59999) using commands starting with "V". See reference document.

Level 1: Generating a Trade

TextBridge protocol

```
I XTKS@my-indicator
@MD 20090401
@MT 100500
1T 25.5 20000
*1
```

C++

```
w.bridge_cmd_select_instrument ("XTKS@my-indicator");
w.bridge_cmd_QUOT_set_market_date (2009, 4, 1);
w.bridge_cmd_QUOT_set_market_time (  10, 5, 0, 0);
w.bridge_cmd_QUOT_L1_rt_trade (25.5 , 20000);
w.bridge_cmd_QUOT_L1_SEND ();
```

Checking Result

NB: execute before issuing publication request

```
feedos_cli (joe@localhost:6020) > sub1 XTKS@my-indicator
EV 147/700000   MarketUTCTime: 2009-04-01 10:05:00:000
                ServerUTCTime: 2009-04-01 10:07:56:245
content: LastPrice LastTradeQty
        LastTradeQty  = 20000
        LastPrice     = 25.5
```

Level 1: Generating BBO

It has to be noted that special values can be used when declaring Bid / Ask:

- Price = "`AT_BEST`" which in C++ is:
  `FeedOS::TextBridge::ORDERBOOK_MAGIC_PRICE_AT_BEST`
- Price = "`AT_OPEN`" which in C++ is:
  `FeedOS::TextBridge::ORDERBOOK_MAGIC_PRICE_AT_OPEN`
- NbOrders can be omitted when it's unknown. In C++, use:
  `FeedOS::TextBridge::ORDERBOOK_NB_ORDERS_UNKNOWN`

TextBridge protocol	`I XTKS@my-indicator`
	`1B 24 100`
	`1A 25 250`
	`*1`

C++	`w.bridge_cmd_select_instrument ("XTKS@my-indicator");`
	`w.bridge_cmd_QUOT_L1_rt_best_bid`
	`      (24, 100, FeedOS::TextBridge::ORDERBOOK_NB_ORDERS_UNKNOWN);`
	`w.bridge_cmd_QUOT_L1_rt_best_ask`
	`      (25, 250, FeedOS::TextBridge::ORDERBOOK_NB_ORDERS_UNKNOWN);`
	`w.bridge_cmd_QUOT_L1_SEND ();`

Checking Result	`feedos_cli (joe@localhost:6020) > sub1 XTKS@my-indicator`
	`EV 147/700000    MarketUTCTime: null`
	`                 ServerUTCTime: 2009-04-01 10:08:31:112`
	`content: Bid Ask`
	`        BestBid       = 24       100`
	`        BestAsk       = 25       250`

Nota Bene: use commands "`1b`" and "`1a`" to clear bid/ask limits, respectively.

Level 2: Generating full, partial or incremental updates

A few commands exist to generate order books, either full depth or top-of-the-book (best 5 or 10 limits, typically).
Nota Bene: the best price is at level 0.

Full Refresh

Here is a full refresh of an order book. Of course it's possible to update only one side.

TextBridge protocol

```
I XTKS@my-indicator
2Ob 0 t
2b 3 100
2b 2 50
2b 1 10
2Oa 0 t
2a 4 200
2a 5 5430
2a 6 111
*2
```

Checking Result

```
feedos_cli (joe@localhost:6020) > sub2 XTKS@my-indicator
SubscribeInstrumentL2_Started
ticket = 2
InstrumentStatusL2
147/700000
OR 147/700000 bid(0*3) ask(0*3)
   ServerUTCTime=2009-04-01 12:46:19:601
         0      BID    3.0000 x    100  ASK    4.0000 x    200
         1      BID    2.0000 x     50  ASK    5.0000 x   5430
         2      BID    1.0000 x     10  ASK    6.0000 x    111
         3      BID ****************    ASK ****************
```

Partial Refresh

Although sending full order books is alright, it's usually more efficient to refresh only those entries that changed.

You can, in a single event, update whole or part of entries by specifying the right "start level" (depth at which update commences). The boolean indicator tells if the update spans up to the last (worst price) entry. If indicator is true, then previous entries that are deeper than the update, if any, are cleared. Else those entries are kept.

Here is an example of a partial Bid and Ask update:

TextBridge protocol

```
I XTKS@my-indicator
20b 1 f
2b 2 40
20a 0 t
2a 4.5 250
*2
```

Checking Result

```
feedos_cli (joe@localhost:6020) > sub2 XTKS@my-indicator
SubscribeInstrumentL2_Started
ticket = 2
InstrumentStatusL2
ob 147/700000          bid(0*3) ask(0*3)
   ServerUTCTime=2009-04-01 20:12:10:700
         0      BID    3.0000 x    100  ASK    4.0000 x    200
         1      BID    2.0000 x    50   ASK    5.0000 x    5430
         2      BID    1.0000 x    10   ASK    6.0000 x    111
         3      BID ****************    ASK ****************
OR 147/700000          bid(1+1) ask(0*1)
   ServerUTCTime=2009-04-02 20:12:39:855
         0      BID                     ASK    4.5000 x    250
         1      BID    2.0000 x     40  ASK ****************
         2      BID                     ASK ****************

feedos_cli (joe@localhost:6020) > snap2 XTKS@my-indicator
InstrumentStatusL2
ob 147/700000          bid(0*3) ask(0*3)
   ServerUTCTime=2009-04-02 20:13:43:112
         0      BID    3.0000 x    100  ASK    4.5000 x    250
         1      BID    2.0000 x     40  ASK ****************
         2      BID    1.0000 x     10  ASK ****************
         3      BID ****************    ASK ****************
```

Incremental Updates

It's possible to send incremental updates using the following Action commands:

Command Code	Command Name	Meaning
2C	QUOT_L2_SEND_rt_order_book_clear_FromLevel	Clear both sides of order book starting at given level (included)
2CB	QUOT_L2_SEND_rt_order_book_clear_BidFromLevel	Clear Bid side starting at given level (included)
2IB	QUOT_L2_SEND_rt_order_book_insert_BidAtLevel	Insert price and qty at given level. Any existing limits should be shifted "down". Any limit going past the "max visible depth" should be dropped
2RB	QUOT_L2_SEND_rt_order_book_remove_BidAtLevel	Remove limit at given level. Existing ones that are deeper, if any, should be shifted "up"
2Rb	QUOT_L2_SEND_rt_order_book_remove_BidAtLevel_and_append	Same as 2RB, plus given price and qty should be appended at bottom (it's the new worst limit visible)
2QB	QUOT_L2_SEND_rt_order_book_change_BidQtyAtLevel	Change quantity at given level
2MVD	QUOT_L2_SEND_rt_order_book_max_visible_depth	Declaration of the "max visible depth" for this instrument. This means that only top-of-the-book is sent. See command "insert_xxxAtLevel".

Nota Bene: only Bid variants are listed… check reference document for Ask counterparts.

Appendix A: list of TextBridge commands

Here is the list of TextBridge commands, in C syntax. See file "TextBridgeInterface_cmd.hpp"

```
/************************************/
/** FeedOS TextBridge command      **/
/** copyright QuantHouse 2007      **/
/************************************/

#ifdef FEEDOS_BRIDGE_INTERFACE_PROTOCOL_VERSION
                FEEDOS_BRIDGE_INTERFACE_PROTOCOL_VERSION("1.0")
#undef FEEDOS_BRIDGE_INTERFACE_PROTOCOL_VERSION
#endif

////////////////////////////////////////////////////////////////////
//
//       ACTION commands
//
////////////////////////////////////////////////////////////////////

// MISC

// can be used to insert "comments" in a script of commands
FEEDOS_BRIDGE_INTERFACE_CMD("#"                      ,NOP                          ,(char const * dummy)  )

// declare what is the protocol version
FEEDOS_BRIDGE_INTERFACE_CMD("PROTOCOL_VERSION"       ,select_protocol_version      ,(char const * version) )

// send an error message
FEEDOS_BRIDGE_INTERFACE_CMD("ERROR"                  ,error                        ,(char const * error) )

// send an informative message
FEEDOS_BRIDGE_INTERFACE_CMD("MSG"                    ,msg                          ,(char const * msg) )

// sleep for N milliseconds
FEEDOS_BRIDGE_INTERFACE_CMD("s"                      ,msleep                       ,(unsigned int nb_millisec) )

// manage instruments
FEEDOS_BRIDGE_INTERFACE_CMD("*REF_CREATE"            ,REF_SEND_create_or_update    ,(char const * instrument) )
FEEDOS_BRIDGE_INTERFACE_CMD("*REF_DELETE"            ,REF_SEND_delete              ,(char const * instrument) )

// generate Level1 event (bid/ask/trade and misc values)
FEEDOS_BRIDGE_INTERFACE_CMD("*1"      ,QUOT_L1_SEND                  , () )
```

```
// generate Level2/MBL event: override a set of bid/ask values (based on previous "Level2 declarations")
FEEDOS_BRIDGE_INTERFACE_CMD("*2"     ,QUOT_L2_SEND_rt_order_book_override  , ()    )


// generate Level2/MBL event: generate a delta refresh
FEEDOS_BRIDGE_INTERFACE_CMD("2C" ,QUOT_L2_SEND_rt_order_book_clear_FromLevel     ,(unsigned int n) )
FEEDOS_BRIDGE_INTERFACE_CMD("2CB",QUOT_L2_SEND_rt_order_book_clear_BidFromLevel   ,(unsigned int n) )
FEEDOS_BRIDGE_INTERFACE_CMD("2CA",QUOT_L2_SEND_rt_order_book_clear_AskFromLevel   ,(unsigned int n) )
FEEDOS_BRIDGE_INTERFACE_CMD("2IB",QUOT_L2_SEND_rt_order_book_insert_BidAtLevel,(unsigned int n, double price, double qty, int nb_orders))
FEEDOS_BRIDGE_INTERFACE_CMD("2IA",QUOT_L2_SEND_rt_order_book_insert_AskAtLevel,(unsigned int n, double price, double qty, int nb_orders))
FEEDOS_BRIDGE_INTERFACE_CMD("2RB",QUOT_L2_SEND_rt_order_book_remove_BidAtLevel    ,(unsigned int n) )
FEEDOS_BRIDGE_INTERFACE_CMD("2RA",QUOT_L2_SEND_rt_order_book_remove_AskAtLevel    ,(unsigned int n) )

FEEDOS_BRIDGE_INTERFACE_CMD("2Rb",QUOT_L2_SEND_rt_order_book_remove_BidAtLevel_and_append ,
        (unsigned int n, double price, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("2Ra",QUOT_L2_SEND_rt_order_book_remove_AskAtLevel_and_append ,
        (unsigned int n, double price, double qty, int nb_orders) )

FEEDOS_BRIDGE_INTERFACE_CMD("2QB",QUOT_L2_SEND_rt_order_book_change_BidQtyAtLevel ,(unsigned int n, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("2QA",QUOT_L2_SEND_rt_order_book_change_AskQtyAtLevel ,(unsigned int n, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("2MVD",QUOT_L2_SEND_rt_order_book_max_visible_depth   ,(unsigned int n) )


////////////////////////////////////////////////////////////////////
//
//      DECLARATION commands
//
////////////////////////////////////////////////////////////////////


/////////////////////////////////////
// common, volatile commands
/////////////////////////////////////
FEEDOS_BRIDGE_INTERFACE_CMD("I",     select_instrument, (char const * instrument) )       // select an existing instrument


/////////////////////////////////////
// REFERENTIAL, persistent commands
/////////////////////////////////////
FEEDOS_BRIDGE_INTERFACE_CMD("REF_SELECT_MIC"        ,REF_select_MarketId          , (char const * mic)          )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_SET_TIMEOFFSET4MIC",REF_set_MarketId_timeoffset  , (char const * mic, int offset_local2UTC_minute))


/////////////////////////////////////
// REFERENTIAL, volatile commands
/////////////////////////////////////
FEEDOS_BRIDGE_INTERFACE_CMD("A/?"    ,REF_declare_attribute_Syntax_UNKNOWN        , (unsigned int tag) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/str"  ,REF_declare_attribute_Syntax_String         , (unsigned int tag, char const * v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/ui8"  ,REF_declare_attribute_Syntax_uint8          , (unsigned int tag, unsigned int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/ui16"  ,REF_declare_attribute_Syntax_uint16          , (unsigned int tag, unsigned int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/ui32"  ,REF_declare_attribute_Syntax_uint32          , (unsigned int tag, unsigned int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/i8"    ,REF_declare_attribute_Syntax_int8            , (unsigned int tag, int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/i16"   ,REF_declare_attribute_Syntax_int16           , (unsigned int tag, int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/i32"   ,REF_declare_attribute_Syntax_int32           , (unsigned int tag, int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/f64"   ,REF_declare_attribute_Syntax_float64         , (unsigned int tag, double v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/t"     ,REF_declare_attribute_Syntax_Timestamp       , (unsigned int tag, char const * v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/e"     ,REF_declare_attribute_Syntax_Enum            , (unsigned int tag, unsigned int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/b"     ,REF_declare_attribute_Syntax_bool            , (unsigned int tag, bool v) )
FEEDOS_BRIDGE_INTERFACE_CMD("A/ch"    ,REF_declare_attribute_Syntax_char            , (unsigned int tag, char v) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_Description",REF_declare_Description               , (char const * description) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_Symbol"     ,REF_declare_Symbol                    , (char const * symbol) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_CFICode"    ,REF_declare_CFICode                   , (char const * cficode) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_SecurityType"      ,REF_declare_SecurityType       , (char const * security_type) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_SecuritySubType"   ,REF_declare_SecuritySubType    , (char const * security_subtype)     )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_ISIN"              ,REF_declare_ISIN               , (char const * isin)                         )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_SEDOL"             ,REF_declare_SEDOL              , (char const * sedol) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_CUSIP"             ,REF_declare_CUSIP              , (char const * cusip) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_MaturityYear"      ,REF_declare_MaturityYear       , (unsigned int y) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_MaturityMonth"     ,REF_declare_MaturityMonth      , (unsigned int m) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_MaturityDay"       ,REF_declare_MaturityDay        , (unsigned int d) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_StrikePrice"       ,REF_declare_StrikePrice        , (double strike_price) )
FEEDOS_BRIDGE_INTERFACE_CMD("REF_PriceCurrency"     ,REF_declare_PriceCurrency      , (char const * currency) )


/////////////////////////////////////
// QUOTATION, persistent commands
/////////////////////////////////////
FEEDOS_BRIDGE_INTERFACE_CMD("@MD"    ,      QUOT_set_market_date          , (char const * date)  )
FEEDOS_BRIDGE_INTERFACE_CMD("@MT"    ,      QUOT_set_market_time          , (char const * time)  )
FEEDOS_BRIDGE_INTERFACE_CMD("@SD"    ,      QUOT_set_server_date          , (char const * date)  )
FEEDOS_BRIDGE_INTERFACE_CMD("@ST"    ,      QUOT_set_server_time          , (char const * time)  )


/////////////////////////////////////
// QUOTATION Level 1, volatile commands
/////////////////////////////////////
FEEDOS_BRIDGE_INTERFACE_CMD("V/?"    ,QUOT_L1_value_Syntax_UNKNOWN        , (unsigned int tag) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/str"  ,QUOT_L1_value_Syntax_String         , (unsigned int tag, char const * v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/ui8"  ,QUOT_L1_value_Syntax_uint8          , (unsigned int tag, unsigned int v)  )
FEEDOS_BRIDGE_INTERFACE_CMD("V/ui16" ,QUOT_L1_value_Syntax_uint16         , (unsigned int tag, unsigned int v)  )
FEEDOS_BRIDGE_INTERFACE_CMD("V/ui32" ,QUOT_L1_value_Syntax_uint32         , (unsigned int tag, unsigned int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/i8"   ,QUOT_L1_value_Syntax_int8           , (unsigned int tag, int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/i16"  ,QUOT_L1_value_Syntax_int16          , (unsigned int tag, int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/i32"  ,QUOT_L1_value_Syntax_int32          , (unsigned int tag, int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/f64"  ,QUOT_L1_value_Syntax_float64        , (unsigned int tag, double v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/t"    ,QUOT_L1_value_Syntax_Timestamp      , (unsigned int tag, char const * v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/e"    ,QUOT_L1_value_Syntax_Enum           , (unsigned int tag, unsigned int v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/b"    ,QUOT_L1_value_Syntax_bool           , (unsigned int tag, bool v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V/ch"   ,QUOT_L1_value_Syntax_char           , (unsigned int tag, char v) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_SessionVWAPPrice"     ,QUOT_L1_value_SessionVWAPPrice       , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_LastPrice"            ,QUOT_L1_value_LastPrice              , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_DailyOpeningPrice"    ,QUOT_L1_value_DailyOpeningPrice      , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_DailyClosingPrice"    ,QUOT_L1_value_DailyClosingPrice      , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_DailyHighPrice"       ,QUOT_L1_value_DailyHighPrice         , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_DailyLowPrice"        ,QUOT_L1_value_DailyLowPrice          , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_DailyTotalVolumeTraded"              ,QUOT_L1_value_DailyTotalVolumeTraded         , (double volume) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_DailySettlementPrice"                ,QUOT_L1_value_DailySettlementPrice           , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_PreviousDailyClosingPrice"           ,QUOT_L1_value_PreviousDailyClosingPrice      , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("V_PreviousDailyTotalVolumeTraded"      ,QUOT_L1_value_PreviousDailyTotalVolumeTraded, (double volume) )
FEEDOS_BRIDGE_INTERFACE_CMD("1X"      ,QUOT_L1_rt_context            , (char const * context) )
FEEDOS_BRIDGE_INTERFACE_CMD("1SDO"    ,QUOT_L1_rt_signal_DailyOpen   , () )
FEEDOS_BRIDGE_INTERFACE_CMD("1SDC"    ,QUOT_L1_rt_signal_DailyClose  , () )
FEEDOS_BRIDGE_INTERFACE_CMD("1SDH"    ,QUOT_L1_rt_signal_DailyHigh   , () )
FEEDOS_BRIDGE_INTERFACE_CMD("1SDL"    ,QUOT_L1_rt_signal_DailyLow    , () )
FEEDOS_BRIDGE_INTERFACE_CMD("1P"      ,QUOT_L1_rt_price              , (double price) )
FEEDOS_BRIDGE_INTERFACE_CMD("1T"      ,QUOT_L1_rt_trade              , (double price, double qty) )
FEEDOS_BRIDGE_INTERFACE_CMD("1S"      ,QUOT_L1_rt_status             , (::FeedOS::TextBridge::SecurityTradingStatus s) )
FEEDOS_BRIDGE_INTERFACE_CMD("1B"      ,QUOT_L1_rt_best_bid           , (double price, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("1A"      ,QUOT_L1_rt_best_ask           , (double price, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("1b"      ,QUOT_L1_rt_best_bid_clear     , () )
FEEDOS_BRIDGE_INTERFACE_CMD("1a"      ,QUOT_L1_rt_best_ask_clear     , () )


//////////////////////////////////////
// QUOTATION Level 2, volatile commands
//////////////////////////////////////
FEEDOS_BRIDGE_INTERFACE_CMD("2Ob",QUOT_L2_rt_order_book_override_BidStartAtLevel,  (unsigned int n, bool is_complete) )
FEEDOS_BRIDGE_INTERFACE_CMD("2Oa",QUOT_L2_rt_order_book_override_AskStartAtLevel,  (unsigned int n, bool is_complete) )
FEEDOS_BRIDGE_INTERFACE_CMD("2b", QUOT_L2_rt_order_book_override_BidNextLevel,     (double price, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("2a", QUOT_L2_rt_order_book_override_AskNextLevel,     (double price, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("2B", QUOT_L2_rt_order_book_override_BidAtLevel, (unsigned int n, double price, double qty, int nb_orders) )
FEEDOS_BRIDGE_INTERFACE_CMD("2A", QUOT_L2_rt_order_book_override_AskAtLevel, (unsigned int n, double price, double qty, int nb_orders) )


#undef FEEDOS_BRIDGE_INTERFACE_CMD
```