CQG

FIX/FAST Direct
Core Functionality

October 08, 2010 | Version 2010-03

Table of Contents

1. Introduction

This document contains information on core features and functionality for Market Data Platform FIX/FAST.

Refer to the following sections for detailed information:

1.1 FIX/FAST Direct vs. CME FIX/FAST

FIX/FAST Direct is designed to be as close to CME FIX/FAST 2.0 as possible. However, to provide maximum productivity, some divergences do exist due to the technical characteristics of the system.

1.1.1 Divergences from the CME FIX/FAST

1. CQG provides one incremental and one snapshot channels with no hot swap possibility: no identical A and B incremental and snapshot UDP multicast groups. A TCP Replay service is used to recover lost/corrupt data when needed.
2. CQG provides a TCP Instrument Definition channel: no public access to Instrument Definition Feed and no security definition flat file.
3. CQG provides data connection configurations through security definitions, not through FTP service as CME does.
4. Unlike CME, CQG does not provide the total traded volume with each trade.
5. As CQG does not limit the DOM depth, the DOM book can grow indefinitely.
6. CQG provides implied and combined (i.e., implied + explicit) book updates.
7. For options, CQG sends combined BBA and only 1-level DOM book updates.

Note:

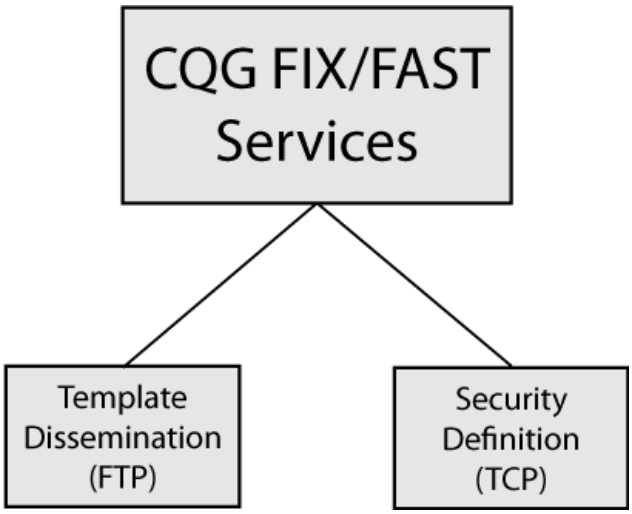
Refer to the **FIX/FAST Direct - Message Specification** document for a more technical and complete presentation of the existing divergences.

2. Architecture

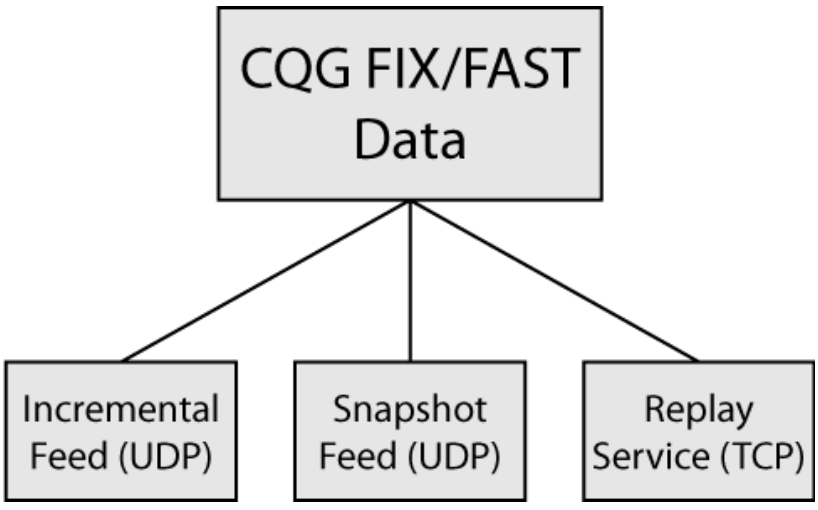
This section contains an architecture overview for FIX/FAST Direct System.

2.1 System Architecture Overview

This section contains a high level look at the production environment.



FIX/FAST clients get the templates from the FTP site, logon to the Security Definition Server, and get security definitions, which also include connection configuration.



Then FIX/FAST clients listen to the Incremental market data feeds. Snapshot feeds and the Replay server are used to restore omitted data.

Figure 1. System Architecture

2.1.1 Market Data Group

Market Data Platform FIX/FAST architecture is explained in this section in terms of a market data group. A market data group is a pair of UDP channels used to produce market data messages for a set of instruments and/or a set of instrument groups.

2.1.1.1 Incremental Feed (UDP)

The Incremental Feed is used to disseminate FIX/FAST Direct incremental market data using the bandwidth-efficient FAST encoded FIX messages. All FIX message types are sent through Incremental Feed applicable Market Data Groups (book update, statistics, quotes, trades).

2.1.1.2 Snapshot Feed (UDP)

The Snapshot Feed is used to disseminate market data snapshots for all books with any activity (i.e., bid, ask, or trade) since the beginning of the week. Book updates (limit and implied) and statistics are sent through the applicable market data channel. A single FIX/FAST message contains the market state for a given instrument. Snapshots are replayed at a constant frequency. Expired instruments are included on the Snapshot Feed until the end of the week (i.e., Sunday 0:00 Line Time or Chicago time).

Note:

CQG strongly recommends that the Snapshot Feed be used for recovery purposes only. Once client systems have retrieved data, they should stop listening to the Snapshot Feeds.

2.1.1.3 Security Definition Server (TCP)

A **Security Definition** message provides information about one financial instrument. It includes:

- Connection configurations for the Incremental, Snapshot, and Replay data feeds.

Note:

Connection configurations for instruments from one source (i.e., Feed ID) are the same.

- Security ID, which is needed to decipher data feeds.

The **Security Definition Server (SDS)** identifies the FIX/FAST clients and sends them security definitions for instruments they are enabled for. FIX/FAST clients are required to provide their username/password in order to log on to SDS.

SDS sends security definitions to qualified FIX/FAST clients that request them.

SDS does not inform about expired instruments.

New security definitions are sent to connected FIX/FAST clients as soon as they emerge.

2.1.1.4 Replay (TCP – Historical)

The TCP Replay component allows requesting a replay of a set of messages already published on the UDP Incremental or Snapshot feeds. The request specifies the messages to be replayed. The request uses the FIX Market Data Request message (35=V).

This type of request is sent through a new TCP connection established by the customer. The responses are sent by CQG through this same connection and the connection is then closed by the Replay Server once the resend is complete. All responses are FIX/FAST encoded (including the reject response). Replay is limited in the number of messages that can be requested.

Note:

This is the recommended recovery method when small outages occur, like when the FIX/FAST client detects that a few messages are missing.
Snapshots are better used to recover instrument states on huge outages.

2.1.2 Template Dissemination

FIX/FAST is a template based protocol. As a result, messages can only be interpreted using a template. Each message contains a unique Template ID that references the template to use to interpret the message.

The template dissemination service provides a method for client systems to receive all of the CQG active templates, or the templates associated with either a Template ID or a Feed ID.

FIX/FAST Direct supplies its templates through this FTP site:

For additional information on this, refer to Template Distribution.

2.1.2.1 FTP Site Information – Template

FIX/FAST Direct uses the following FTP site to disseminate its templates:

ftp://develop.cqg.com

This FTP site contains the template files and configuration files for all environments. The FTP site is a secure site that requires a user name and password for access. Template and market data configuration details for the production environment are only available to customers after the certification process is complete.

Information applies as follows in the table:

- **Environment** - specific environment (i.e., Certification, Production).

Note:

In order to get access to the production environment, first your FIX/FAST client must be certified.
For more information, and to pass the certification process, visit:
https://develop.cqg.com/fixfast/

- **Service** – Specifies the Template or the Configuration Service.
- **FTP Site** – Specifies the address of the FTP site.
- **User Name** – Specifies the username.
- **Password** – Specifies the password.
- **Directory Location** – Identifies the directory.
- **Client System Update Schedule** – Client systems should download updates according to the specified schedule.

Environment	Service	FTP Site Folder Location	Client System Update Schedule
Testing	Template	ftp://develop.cqg.com/Testing/	Sunday 0:00 (Line Time)
Certification	Template	ftp://develop.cqg.com/Certification/	Sunday 0:00 (Line Time)
Production	Template	ftp://develop.cqg.com/Production/	Sunday 0:00 (Line Time)

2.2 System Startup

This section contains a high level overview of the startup process for the production environment.

Upon startup, the FIX/FAST client should perform the following steps:

1. Get the templates from the FTP Template Dissemination Service. Refer to Template Dissemination for more information.
2. Logon to SDS and request the security definitions.
3. Get channel configurations from the corresponding security definition messages.

2.2.1 Pre-Opening Startup

For a startup prior to the weekly market open, all market data (book updates, statistics, quotes, trades). Follow the process below to ensure that all necessary market data is received:

1. Listen to the Incremental feed for incremental market data and start normal processing.

2.2.2 Late Joiner Startup

For a late joiner startup, follow the process below to ensure that all necessary market data is received:

1. Listen to the Incremental feed for incremental market data. Begin the natural refresh process and begin queuing messages. A natural refresh is when the DOM snapshot is empty and further updates reconstruct the book.

> **Note:**
>
> The incremental market data may complete a natural refresh that would construct the current, correct state of a book.

2. Listen to the Market Recovery feed for the latest snapshots.

Use the snapshots to construct the book state.

Since the size of the book is not known in this stage, the FIX/FAST client cannot determine when the whole book is fully refreshed. Therefore, there is a chance that DOM might not be recovered completely even if all levels were retained during the incremental refreshes.

When the latest snapshots are received, the value for tag 369-LastMsgSeqNumProcessed in the snapshot matches tag 34-MsgSeqNum in the incremental feed message. Also, the value for tag 83-RptSeq in the snapshot matches tag 83-RptSeq in the incremental feed for the same instrument (i.e., the same Security ID).

If the book for an instrument was not completely constructed using natural refresh, then apply the snapshot.

3. Stop listening to the Snapshot feed.
4. Start normal processing.

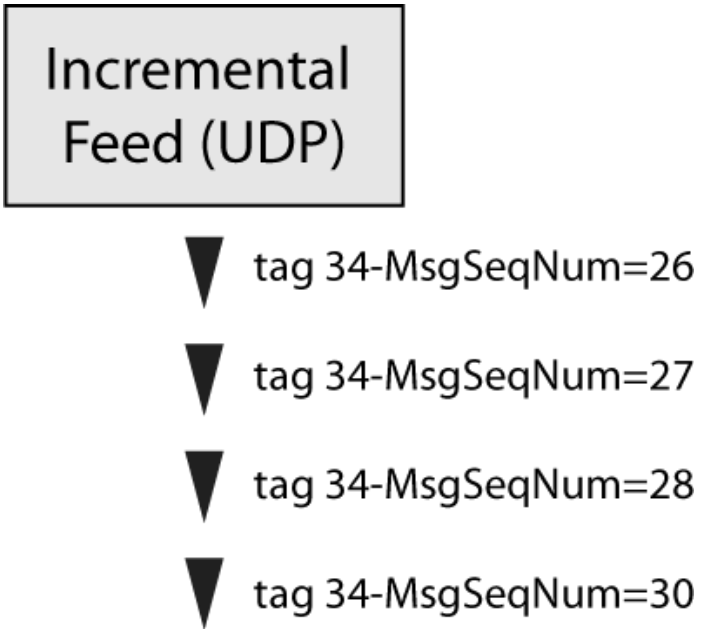
2.3 Processing the Incremental Feed

FIX/FAST Direct diverges from CME FIX/FAST in regard of Incremental data arbitration.

In order to provide a faster service, CQG utilizes one Incremental UDP feed and one Replay TCP Service. The client systems shall check the FIX messages to find a missing message, which may occur due to UDP connection nature. If such occurs, the client system shall request the required messages from the Replay server using standard FIX message.

The client system should process the feed as follows:

1. Listen to the Incremental feed
2. Process messages by incremental sequence number (tag 34-MsgSeqNum).
3. If a sequence number (tag 34-MsgSeqNum) gap is detected – this would indicate a packet was lost on the Incremental Feed. Client systems would need to initiate the Recovery process (refer to Recovery for additional information).



Processing of messages as follows:

Messages are received at client system from Incremental Feed as following:

1. Receive message on Incremental Feed – tag 34-MsgSeqNum=26
   - Process the message tag 34-MsgSeqNum=26
2. Receive message on Incremental Feed – tag 34-MsgSeqNum=27
   - Process the message tag 34-MsgSeqNum=27
3. Receive message on Incremental Feed – tag 34-MsgSeqNum=28
   - Process the message tag 34-MsgSeqNum=28

4. Receive message on Incremental Feed – tag 34-MsgSeqNum=30
   - Process the message tag 34-MsgSeqNum=30
   - **Gap detected!**
   - Begin the recovery process as applicable.

3. FAST Implementation

This section describes how to implement FIX Adapted for Streaming (FAST) protocol.

3.1 Introduction

The FIX Adapted for Streaming (FAST) Protocol has been developed as part of the FIX Market Data Optimization Working Group. FAST is designed to optimize electronic exchange of financial data, particularly for high volume, low latency data dissemination. This document describes implementation of FAST in receiving and processing FIX/FAST Direct-encoded electronic market data feed.

For more information see the FIX FAST (version 1.x.1) specification at:

http://www.fixprotocol.org/documents/3066/FAST%20Specification%201%20x%201.pdf

FAST is a data compression algorithm that significantly reduces bandwidth requirements and latency between sender and receiver. FAST works especially well at improving performance during periods of peak message rates. FAST extends the base FIX specification and assumes the use of FIX message formats and data structures. FAST is a standalone specification that uses templates to inform the receiver which operations to use in decoding. Templates allow FAST to achieve high levels of data compression with low processing overhead and latency compared to other compression utilities such as Zlib.

Note:

This document describes concepts applicable to CQG-specific FAST implementation; which is supplementary to the FAST specification referenced above.

3.1.1 Stop Bit Encoding

Stop bit encoding is a process incorporated in FAST that eliminates redundancy at the data field level by using a stop bit instead of the traditional separator byte. In FAST, a stop bit is used instead of FIX's traditional <SOH> separator byte. Thus 7 bits of each byte are used to transmit data and the eighth bit is used to indicate the end of a field.

3.1.1.1 FAST 7-Bit Binary Representation

FAST renders numbers into binary across the 7 data bits in each byte. Thus a number equal to or less than 2^7-1, (127) occupies one byte, a number between 2^7 and 2^(7*2)-1(16,383), occupies two bytes, etc.

3.1.2 Implicit Tagging

In traditional FIX messages each field takes the form "Tag=Value<SOH>", where the tag is a number representing which field is being transmitted and the value is the actual data content. The ASCII <SOH> character is used as a byte delimiter to terminate the field. For example:

```
35=x|268=3 (message header)
279=0|269=2|270=9462.50|271=5|48=800123 (trade)
279=0|269=0|270=9462.00|271=175|1023=1|48=800123|346=15 (new bid 1)
```

```
279=0|269=0|270=9461.50|271=133|1023=2|48=800123|346=12 (new bid 2)
```

FAST eliminates redundancy with a template that describes the message structure. This technique is known as implicit tagging as the FIX tags become implicit in the data. A FAST template replaces the tag=value syntax with "implicit tagging" as follows:

- Tag numbers are not present in the message but specified in the template
- Fields in a message occur in the same sequence as tags in the template
- The template specifies an ordered set of fields with operators

3.1.3 Data Types

A field within a FAST template will have one of the following Data Types indicating the required decoding action:

- **String –** used to represent ASCII or Unicode values using the FAST 7-bit binary encoding.
- **Signed Integer –** used to represent a signed integer using the FAST 7-bit binary encoding. A twos complement integer representation is used, with the most significant data bit being the sign bit. Note that contiguous leading bits of the same value must be dropped (so, for instance, only 1 byte is required to encode -1, 0xGG). In some cases, a 0x00 byte will be the most significant byte sent to preserve sign, so 64 is represented 0x00 0xC0.
- **Unsigned Integer –** used to represent unsigned integers using the FAST 7- bit binary encoding.
- **Decimal –** used to represent a floating point number as exponent and mantissa. The exponent is a signed integer used to express precision and the mantissa is a signed integer used to express the value. The numerical value is obtained by multiplying the mantissa with the base-10 power of the exponent expressed as:

  $$number = mantissa * 10^{exp}$$

  The exponent and mantissa can be encoded as a single, composite field or as individual conjoined fields.

3.1.4 Field Encoding Operators

FAST functions as a state machine and must know which field values to keep in memory. FAST compares the current value of that field and determines if the new value should be constant, default, copy, delta (integer or string), increment, or tail.

3.1.4.1 Dictionary Context

CQG uses a dictionary context on a per-packet basis. A dictionary is a cache in which previous values are maintained. All dictionary entries are reset to the initial values specified after each UDP packet. Currently, CQG sends one message per UDP packet.

3.1.4.2 Field Operators

Note:

The following are general descriptions and may have exceptions depending upon the scenario.

A field within a FAST template will generally have one of the Field Operators described below indicating the required decoding action. Please note that in some cases it is possible for a field to have no Field Operator.

- **Constant** – indicates that the field will always contain a predetermined value as specified by the value attribute.
  If the value is optional this field will contain a Presence Map (Pmap) bit; if mandatory this field will not contain a Pmap bit.
- **Default** – indicates that the default value defined in the template should be used as the decoded value when a data value is not present. If a data value is present, use that value.
- **Copy** – indicates that the data value in the prior occurrence of this field should be used if a data value is not present for this occurrence.
- **Delta for integers** – when used with an integer, it indicates that the data value represents the arithmetic difference between the current and prior values.
- **Delta for strings** – when used with a string, it indicates that the data value is either pre-pended or appended to the prior value of this field after removing the specified number of bytes from the beginning or end of the string.
  - A negative subtraction length means the operation takes place on the front of the string.
  - A zero or positive subtraction length means the operation takes place on the end of the string.
- **Increment –** indicates that the data value for the prior occurrence of this field should be incremented by 1 if a data value is not present for the current occurrence. This operator works with integers only.
- **Tail –** this operator works with strings and byte vectors and specifies the number of characters to remove and append to the base value. The length of this value must be constant.

3.1.5 FAST Template

A FAST template corresponds to a FIX message type and uniquely identifies an ordered collection of fields. The template also includes syntax indicating the type of field and transfer decoding to apply. A template is communicated between CQG and client systems in XML syntax using the FAST v1.1 Template Definition Schema maintained by FIX. The XML format is human- and machine-readable and can be used for authoring and storing FAST templates. Session Control Protocol (SCP) will not be used.

3.1.6 Presence Map and Stop Bit

The Presence Map (Pmap) indicates which fields in the template have data present and which fields have data implied. A Pmap is a sequence of the encoded bits with each bit representing a template field according to sequence. Fields with data present have the Pmap bit set to '1'. Fields with data implied have the Pmap bit set to '0' (Exception: fields with a Constant operator in which the bit is set to '1' for an implied state).

3.1.6.1 Presence Map Rules

Pmap rules determine when a FAST template field required a corresponding Pmap bit. A field does not require a bit in the Pmap when it meets any of the following criteria:

1. The field is defined as mandatory without a field operator –
   a value will always be present.

2. The field is defined as mandatory with a constant operator –
   a value should always be instantiated in the decoded message based on the value in the template.
3. The field is defined as mandatory with a delta operator –
   a delta value is always present.
4. The field is defined as optional without a field operator –
   either a value or NULL will always be present.
5. The field is defined as optional with a delta field operator –
   a delta value or NULL will always be present.

See the FAST v1.1 specification for the complete information on presence map rules.

Rules for Determining if a Presence Map Bit is Required

Operation	Mandatory	Optional
None	No	No
Constant	No	Yes
Copy	Yes	Yes
Default	Yes	Yes
Delta	No	No
Increment	Yes	Yes
Tail	Yes	Yes

3.1.6.2 Message Structure

Note:

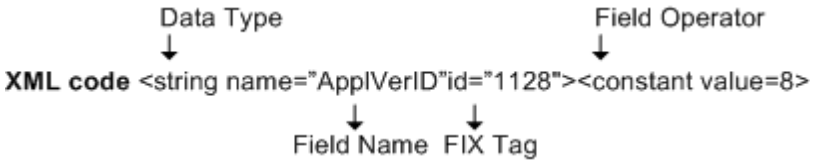
The Market Data Incremental Refresh (tag 35-MsgType = X) message is used for example purposes throughout this document; not all FAST messages follow this format.

The FIX Market Data Incremental Refresh (tag 35-MsgType = X) message is made up of three components: a Header, a Body and a set of one or more Market Data Entries as shown in the diagram below. The **Header** carries transmission details. The **Body** carries information pertinent to all entries in the message such as TradeDate. The **Market Data Entry** carries specific instructions for updating the book or recording trades. The templates provided by CQG conform to this general structure and use only fields that are part of the Market Data Incremental Refresh (tag 35-MsgType = X) message.

Header Message Type - ApplVerID - MsgSecNo - Sending Time – SenderCompID
Body MarketDataEntryNo
MarketDataEntry Update Action - EntryType - SecurityID - EntryPx

Example: How to Read a Template Field

Field Operators and Data Types appear within the Template as follows:

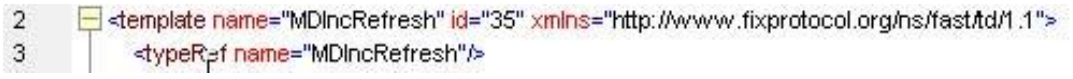


3.1.6.3 Template ID Usage

Each template is assigned a Template ID to uniquely describe the format of an encoded message. A Template ID is carried in every encoded message to provide a reference to the correct template for decoding purposes.

The Template ID is an unsigned integer carried as the first data field following the first Pmap of every message allowing the decoding system to apply the correct template to the message upon receiving it.

Example: Template ID



CQG will not send the same Template ID more than once in a UDP packet. When the packed message is received, the sequence to begin decoding a message is:

1. Locate and read the first Pmap field, which begins in the first byte of the application data preceding the FIX/FAST Direct message Header.
2. Treat the next field as the Template ID (the first logical field of the message).
3. Retrieve the template specified by the Template ID.
4. Decode the remainder of the message.

3.1.6.4 Optional vs. Mandatory Fields

A field is defined as optional when it is possible for a value to not be present in the decoded message. When a field is optional, the wire representation "null" indicates the field is not

present. A field is defined as mandatory when that field must be present in the decoded message. Note that a field may not be present ("coded away") in the wire formatted message but can be present in the decoded message as the result of an encoding operation (e.g. default) that removes data for efficiency but continues to imply data is present through the Pmap.

Optional fields are useful when a generic template is used to provide multiple market data types such as book updates and trades. In this situation, there may be fields which are present in one occurrence of the repeating group but not in another within a given message.

For example, the MDEntries repeating group that is present in the MDIncRefresh template shown above can be used to express a trade, bid, ask, high, low, etc. within a single message. A trade entry will not use tag 346, NumberOfOrders, which will then be defined as optional. However, FAST requires that this field be accounted for in the encoded message if specified in the template. This is done through the use of a reserved value of NULL to indicate that the field is not present in the decoded data. In the serialization layer, FAST reserves binary zeros to indicate a NULL value which tells the decoder that no data is present for this Template Distribution.

Rules for Determining if a Field is Nullable

Operation	Mandatory	Optional
None	No	Yes
<constant/>	No	No
<copy/>	No	Yes
<default/>	No	Yes
<delta/>	No	Yes
<increment/>	No	Yes
<tail/>	No	Yes

3.1.6.5 XML Template Example

A template consists of Field Instructions that define the fields contained in the message. Field Instructions specify the field name, tag number, data type, field operator, and presence attribute that indicates if a field is optional or mandatory.

A sample market data template is shown below. The syntax is standard XML and can be parsed using a variety of open source tools. Valid template syntax is determined by the FAST Template Schema, which is available in the FAST v1.1 specification.

The real-time feed templates are based on the Market Data Incremental Refresh message type.

The information contained in a template is passed to the client FAST decoder at run-time such that the decoder recognizes how each field is encoded in terms of data type representation (Transfer) and data redundancy removal (Field).

The template is constructed of several sections including Template Identification, Header, Body and Sequence.

- Template Identification provides the template name and identifier.
- The Header includes FIX header fields such as ApplVerID (tag 1128), MsgType (tag 35), and SendingTime (tag 52).
- The Body provides information common across all repeating groups.

- Sequence represents a repeating group with a corresponding length field and a set of repeating group fields which carry the detailed entry information.

Please refer to the Appendix for a decomposition of the template above with an explanation for each instruction.

```
<?xml version="1.0" encoding="UTF-8" ?>
<templates xmlns="http://www.fixprotocol.org/ns/fast/td/1.1">
   <template name="MsgHeader">
      <string id="1128" name="ApplVerID"> <constant value="1"/> </string>
      <string id="49" name="SenderCompID"> <constant value="CQG"/> </string>
      <uInt32 id="34" name="MsgSeqNum"/>
      <uInt64 id="52" name="SendingTime"/>
   </template>
   <template dictionary="1" id="1" name="MDIncRefresh">
      <string id="35" name="MessageType"> <constant value="X"/> </string>
      <templateRef name="MsgHeader"/>
      <sequence name="MDEntries">
         <length id="268" name="NoMDEntries"/>
         <uInt32 id="279" name="MDUpdateAction" presence="optional"> <copy value="0"/> </uInt32>
         <uInt32 id="1023" name="MDPriceLevel" presence="optional"> <default value="1"/> </uInt32>
         <string id="269" name="MDEntryType"/>
         <uInt32 id="48" name="SecurityID" presence="optional"> <copy/> </uInt32>
         <uInt32 id="83" name="RptSeq" presence="optional"> <increment/> </uInt32>
         <decimal id="270" name="MDEntryPx" presence="optional">
            <exponent> <default value="0"/> </exponent>
            <mantissa> <delta/> </mantissa> </decimal>
         <uInt32 id="273" name="MDEntryTime"> <copy/> </uInt32>
         <int32 id="271" name="MDEntrySize" presence="optional"> <delta/> </int32>
         <string id="276" name="QuoteCondition" presence="optional"> <default value="K"/> </string>
         <uInt32 id="1070" name="MDQuoteType" presence="optional"> <constant value="0"/> </uInt32>
      </sequence>
   </template>

   <template dictionary="2" id="2" name="MDSecurityDefinition">
      <string id="35" name="MessageType"> <constant value="d"/> </string>
      <templateRef name="MsgHeader"/>
      <uInt32 id="911" name="TotNumReports"/>
```

Please refer to the Appendix for a decomposition of the template above with an explanation for each instruction.

Note:

CQG requires that client systems use an API layer to load templates rather than implement hard coded templates. Since templates are subject to change, this will facilitate template modification in production environments.

Templates are available from the FTP site at: ftp://develop.cqg.com/.

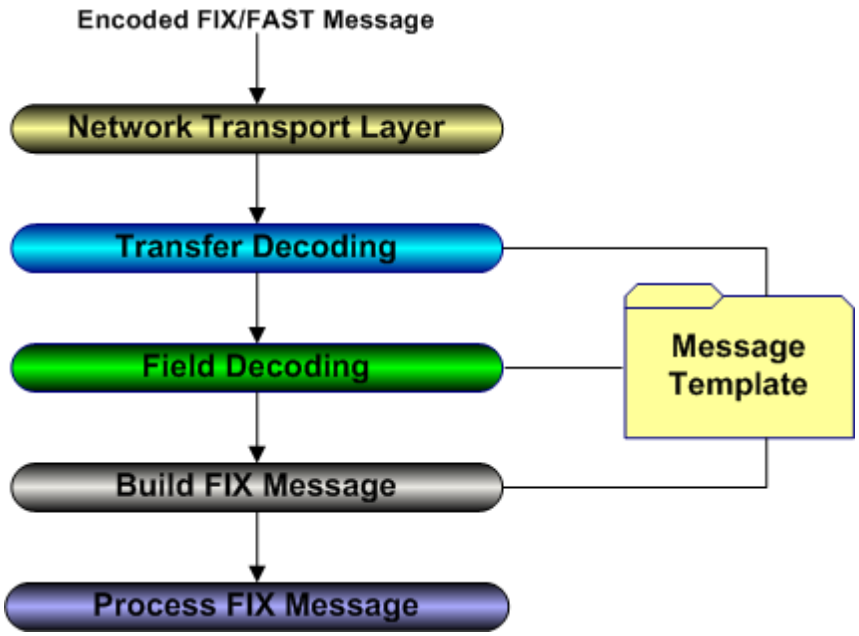
3.2 FAST Decoding

This section presents one possible approach to decoding CQG FAST messages to demonstrate the fundamental concepts involved in the process. This approach is provided for example purposes and is not definitive.

FAST processing decodes a FIX/FAST message by means of FAST templates provided by CQG. The FAST template contains the instructions to decode and reconstruct compressed message

data into the FIX format and also supports repeating groups (sequences) that allow a single message to convey multiple instructions (i.e., book update, trade, high/low, etc.).

A decoding sequence can follow the steps diagramed below; this document describes Transfer Decoding, Field Decoding, and Building the FIX message only.



Step 1. Transport –Client system receives encoded FAST message.

Step 2. Transfer Decoding

- Identify template
- Extract binary encoded bits
- Map bits to fields per template field

Step 3. Field Decoding – Apply operators to determine values per template field.

Step 4. Build FIX message.

Step 5. Process FIX message

3.3 Transfer Decoding Overview

Transfer decoding is the initial step that converts data from the FAST 7-bit binary format.

Note:

The decoding process can take place on a field-by-field basis, de-serializing and rebuilding the decoded value for each field. Alternately, the decoding process may de-serialize the entire message and then make a second iteration over the message to rebuild the decoded values.

The FAST transfer format is a binary representation enhanced with the following attributes:

- **Stop Bit** - Each data byte contains a stop bit to indicate whether this is the last byte of a field. The stop bit is the seventh bit of the most significant byte. A stop bit set to 1 indicates the last byte of a field. A stop bit set to zero indicates this is not last byte of the field.
- **Presence Map** - The Presence Map occurs at the start of every message or repeating group and indicates the presence or absence of individual fields. The template specifies the type of transfer decoding to employ for each field in the message (i.e. string, integer, decimal, etc.).

WARNING

FAST does not support timestamps. CQG will convert the timestamp to an integer by removing punctuation from the UTC timestamp prior to encoding. The decoding application should convert the integer to the FIX UTC format after decoding. In addition, the timestamp header on the message packet should not be used for latency comparisons. The message needs to be decoded to determine the timestamp for updates.

3.3.1 Field Decoding Overview

Field decoding is the second part of the decompression process that reconstructs data values according to template-specified operations. Field decoding operations are assigned per field within the template; decoding reinstates data as indicated by the template.

3.3.1.1 When to Reset Decoder State

The state of the decoder will need to be reset for each received UDP packet and the dictionary applied. Resetting state means that all fields are set to their initial pre-processing state. This is because UDP transport is not reliable and data in a packet cannot be dependent on data in a previous packet since it is possible for packets to be lost or arrive out of sequence.

3.3.2 Receiving Data over FIX/FAST Direct

The basic operation requires the decoding of data within a discreet packet as well as the ability to determine the end of one message and beginning of another within that packet.

3.3.2.1 Decoding a Packet

Decoding of the datagram should be conducted as a unit of work in which only the data in a given packet participates in the scope of the decoding process. To determine when the process has fully decoded a discreet message, it is important to use the template to step through a message field-by-field.

3.3.2.2 Decoding Messages in a Packet

When the decoder determines that it has processed all repeating groups in the message, and that there are no further fields specified by the template, then it can conclude that the end of the message has been reached. In this way, it is necessary to use the presence maps, NoMDEntries, and the template to determine when end of message has been reached. To recap, the steps involved are:

- Decode fields in the non-repeating body of the message using the high-level presence map.
- If the last field in the template has been reached the end of message has been encountered.
- If the NoMDEntries field is present, then process each repeating group using the repeating group presence map and template.

- When number of repeating groups processed equals the count specified in NoMDEntries, then the end of the repeating groups has been encountered.

3.3.2.3 Error Handling in a Broadcast Environment

If FAST encounters an error during the process of decoding the contents of a UDP packet, the process should stop, log the problem, and decode the next packet. In all likelihood, the error is due to an inconsistency between the encoded data and the template being used to decode the data. At this point, it is advised that parties synchronize templates in order to ensure that the same data definitions are being used.

3.4 Decoding Sequence

This section provides a detailed example of the process for decoding a FAST message. You should thoroughly test your application to determine that you have properly decoded a message.

Note:

This decoding example is a generic description of the basic decoding process; it does not reflect the structure of the reference code.

FAST message decoding consists of three major steps:

Step 1. **Transfer Decoding**

- Identify Template ID and Pmap bit value for each given field. Pmap values are 0, 1.
- Decode FAST 7-bit binary values to identify data present in the message.

Step 2. **Field Decoding**

- Apply field operators to extracted binary values.
- Determine state of field to be decoded.

Step 3. **Build FIX Message**

- Apply FIX or internal structure to the decoded message.

The decoding sequence in the example used throughout this section uses a FAST message containing a header, trade, and two new bids.

3.4.1 Transfer Decoding

Upon receipt of the FAST message, the client decoder loads the template according to the Template ID. From this template the client decoder identifies how to decode each field, whether a field will have a Pmap bit, and if a field can carry a null value. In this example the MDIncRefresh template contains a header and repeating group structure (italics).

Note:

The following template is an EXAMPLE TEMPLATE.

```
<template name="MDIncRefresh" id="30">
  <typeRef name="MDIncRefresh"/>
<string name="MessageType" id="35"> <constant value="X"/> </string>
<sequence name="MDEntries"><length name="NoMDEntries" id="268"></length>
 <uInt32 name="MDUpdateAction" id="279"> <copy value="0"/> </uInt32>
 <uInt32 name="MDEntryType" id="269"> <copy value="0"/> </uInt32 >
 <decimal name="MDEntryPx" id="270">
   <exponent><copy value="-2"/></exponent>
   <mantissa><delta/></mantissa> </decimal>
 <int32 name="MDEntrySize" id="271"> <delta/> </int32>
 <uInt32 name="MDPriceLevel" id="1023" presence="optional"> <increment/>
 <uInt32 name="SecurityID" id="48"> <copy/> </uInt32>
 <uInt name="SecurityIDSource" ID="22"><constant value="8"/></uInt32>
 <int32 name="NumberOfOrders" id="346" presence="optional"> <delta/> </int32>
   </sequence>
</template>
```

3.4.2 Decoding Process

The client system reads the Pmap to identify required field-level decoding.

Pmap 1	Header	Pmap 2	Trade	Pmap 3	New Bid 1	Pmap 4	New Bid 2
11000000	Fields	10101100	Fields	10101000	Fields	10000000	Fields

In this example light grey values indicate an unused bit.

Step 1. **Extract Pmap1 from data**. Pmap1 is the top-level Pmap that contains the Pmap bits for all fields requiring a Pmap bit in the message.

- The first bit in Pmap1 after the stop bit (in bold) is assigned to the Template ID, which always has a bit. The bit is turned on indicating that the Template ID is present in the encoded message and that the corresponding template must be loaded.
- The first field in the template is the MessageType, defined as a mandatory field using a Constant operator with a value of “X” retrieved from the template and therefore does not have a bit.
- The next field in the template is NoMDEntries. The length specified in the sequence determines the number of repeating groups in the message. This field does not have a bit since the field is mandatory and does not have an operator.
- **The remaining bits in Pmap1 are defaulted to 0 but are not referenced by the decoding process; at this point, the Header fields have been decoded.**

Step 2. **Extract Pmap2 from data**. Pmap2 represents the first repeating group and contains trade-related fields.

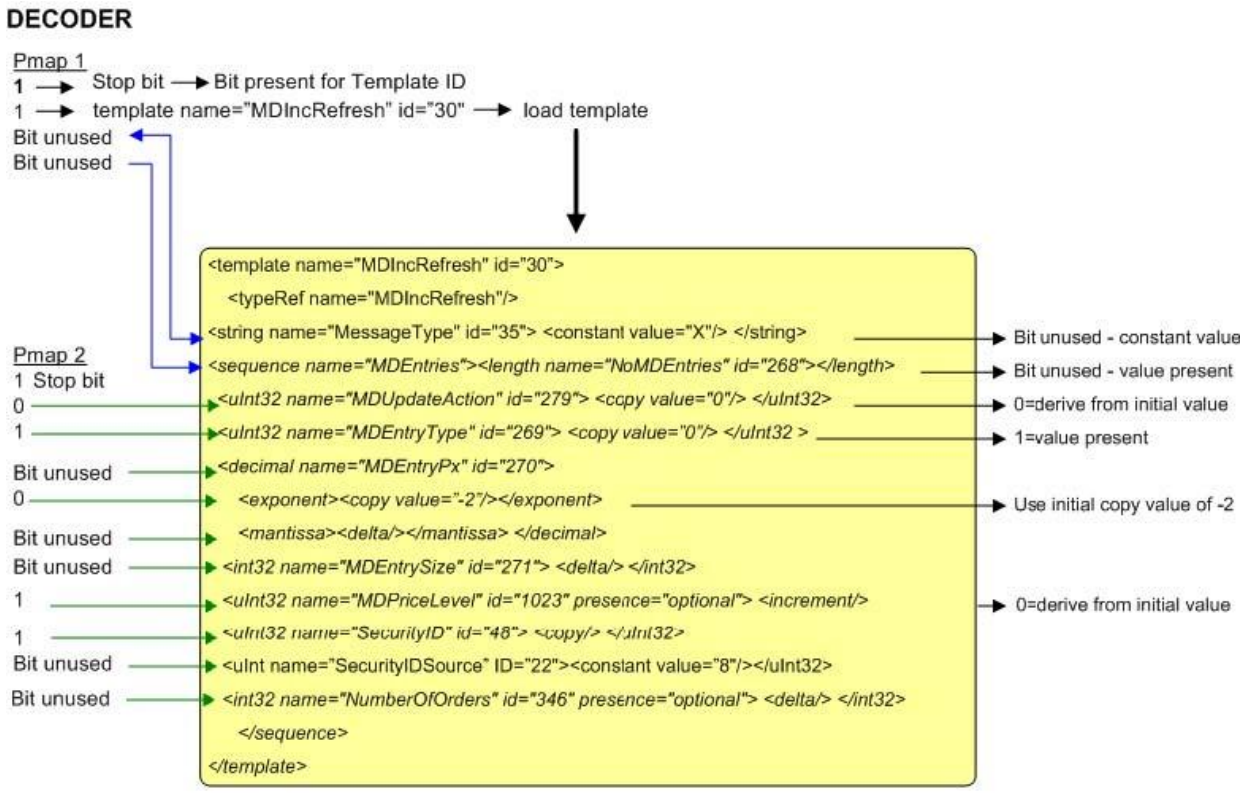
- The next field in the template is MDUpdateAction. By definition, this field requires a bit in the Pmap since it uses the Copy field operator. In this example, the bit is set to 0 indicating that no data is present and that the value must be derived from the initial value.
- The next field in the template is MDEntryType, which is a Copy field. The Pmap set to '1' indicates an encoded value present in the field.
- The decoding process continues using the template and Pmap2 to traverse the data.

Step 3. **Extract Pmap3 from data**. Pmap3 represents the presence map for repeating group2 followed by the New Bid1 fields.

- By the time the decoding process reaches Pmap3, prior values have been established for several fields based on the content of the prior repeating group.
- The decoding process uses the prior values in combination with the encoded data to generate the proper decoded values.

Step 4. **Extract Pmap4 from data**. Pmap4 is the final presence map and the decoder recognizes that all bits are set to off indicating that either prior values should be used or that fields do not use a bit.

The following diagram illustrates the decoding process flow:



Bit unused = no Pmap bit required for delta or constant values

3.4.3 Build the FIX Message

The table in this example shows the step-by-step conversion for each field of the FAST message into the FIX format.

The following steps describe the actions taken to obtain the data shown in the Message Decoding Process table.

Step 1. Decoding begins with the identification of the Pmap bit for each field.

Step 2. The encoded FAST 7-bit binary values are obtained as shown in the "Encoded FAST 7-Bit Binary Value" column.

Step 3. The encoded FAST 7-bit binary values from step 2 are deserialized based on the data type specified in the template.

Step 4. The decoder maintains the state of prior values for each field throughout decoding and applies them for fields having operators of Delta, Copy, or Increment.

Step 5. Obtain fully decoded values.

Message Decoding Process Table									
				Step 1	Step 2	Step 3	Step 4		Step 5
#	Field Name	Presence*	Data Type/Field Operator	Pmap Bit	Encoded FAST Hex/Binary Value	Deserialized Encoded Value	Prior Value	Initial Value	Decoded Value
BIT STREAM = 11000000 10011110 100000011									
1	Template ID	M	uint32/copy	1	0x9e – 10011110	30			TID = 30
2	MsgType	M	string/constant	No bit	None	None		X	35=X
3	NoMDEnt	M	uint32/no operator	No bit	0x83 - 10000011	3			268=3
Repeating Group 1 - Trade									
BIT STREAM = 10101100 10000010 00111001 01100000 11001010 10000101 10000000 00110000 01101010 11111011 10000000									
4	UpdateAction	M	uint32/copy	0	None	None		0	279=0
5	EntryType	M	uint/copy	1	0x82 - 10000010	2		0	269=2
6	EntryPrice	M	Exp: int32/copy Mant: int64/delta	Exp – 0 Mant – No bit	Exp: None Mant: 0x39 0x60 0xca – 0111001 01100000 11001010	Exp = None Mantissa = 946250		-2	270=9462.50
7	EntrySize	M	int32/delta	No bit	0x85 - 10000101	5			271=5
8	PriceLevel	O	uint32/increment	1	0x80 – 10000000	NULL		1	1023=No Value
9	SecurityID	M	uint32/copy	1	0x30 0x6a 0xfb - 00110000 01101010 11111011	800123			48=800123
10	NoOrders	O	int32/delta	No bit	0x80 - 10000000	NULL		0	346=No Value
Repeating Group 2 – New Bid 1									
BIT STREAM = 10101000 10000000 11001110 00000001 10101010 10000010 10010000									
11	UpdateAction	M	uint32/copy	0	None	None	0		279=0
12	EntryType	M	uint/copy	1	0x80 - 10000000	0	2		269=0
13	EntryPrice	M	Exp: int32/copy Mant: int64/delta	0 No bit	Exp: None Mant: 0xce - 11001110	Exp = None Mantissa = -50	-2946250		270=9462.00
14	EntrySize	M	int32/delta	No bit	0x01 0xaa –00000001 10101010	170	5		271=175
15	PriceLevel	O	uint32/increment	1	0x02 – 10000010	2*	NULL		1023=1
16	SecurityID	M	uint32/copy	0	None	None	800123		48=800123

Message Decoding Process Table									
				Step 1	Step 2	Step 3	Step 4		Step 5
#	Field Name	Presence*	Data Type/Field Operator	Pmap Bit	Encoded FAST Hex/Binary Value	Deserialized Encoded Value	Prior Value	Initial Value	Decoded Value
17	NoOrders	O	uint32/delta	No bit	0x90 - 10010000	16*	0**		346=15
Repeating Group 3 – New Bid 2									
BIT STREAM = 10000000 11001110 11010110 11111101									
18	UpdateAction	M	uint32/copy	0	None	None	0		279=0
19	EntryType	M	uint/copy	0	None	None	0		269=0
20	EntryPrice	M	Exp: int32/copy Mant: int64/delta	0 No bit	Exp: None Mant: 0xce-11001110	Exp: None Mantissa=-50	-2946200		270=9461.50
21	EntrySize	M	int32/delta	No bit	0xd6 – 11010110	-42	175		271=133
22	PriceLevel	O	uint32/increment	0	None	1	1		1023=2
23	SecurityID	M	uint32/copy	0	None	None	800123		48=800123
24	NoOrders	O	uint32/delta	No bit	0xfd - 11111101	-3	15		346=12

*M=mandatory O=optional

**Subtract 1 from non-negative optional integer fields

***For delta fields, if a null is received, the prior value in the dictionary is not changed. In this case, the initial value was not specified in the template, so the initial value is set to '0'. The first encoded field received has a value of null, so the prior value is '0' when decoding the second instance of this field.

3.4.3.1 Result – Decoded Values

```
35=x|268=3 (message header)
279=0|269=2|270=9462.50|271=5|48=8001234 (trade)
279=0|269=0|270=9462.00|271=175|1023=1|48=8001234|346=15 (new bid 1)
279=0|269=0|270=9461.50|271=133|1023=2|48=8001234|346=12 (new bid 2)
```

3.5 Sample Template

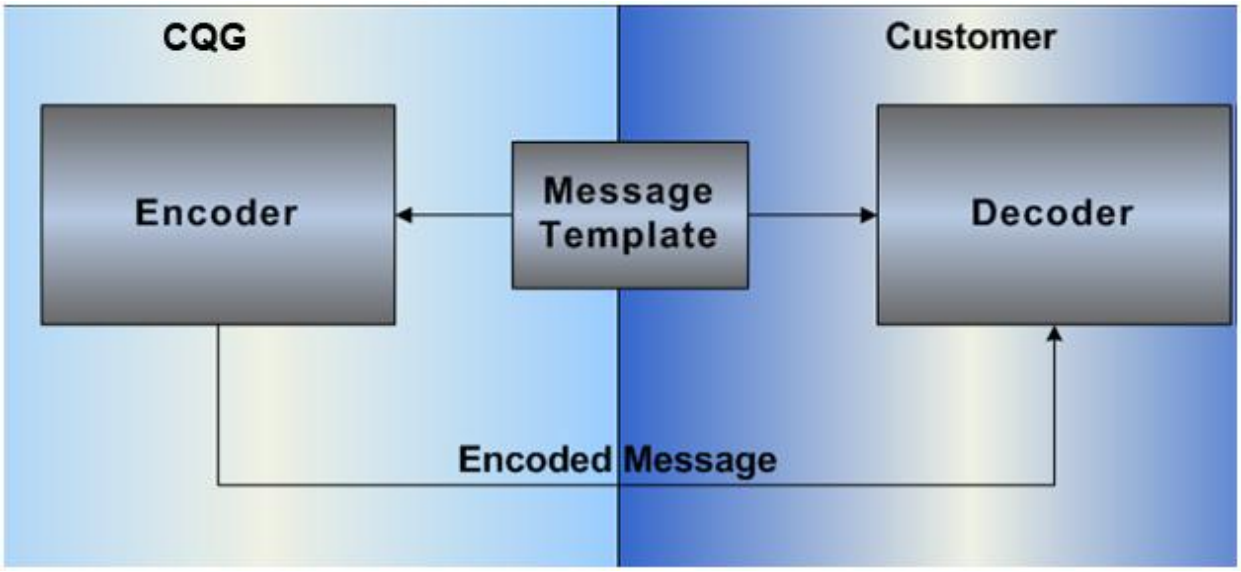
Line #	Template Syntax	Use and Description
2	<template name="MDIncRefresh" id="35">	Provides the template name (MDIncRefresh) and template identifier (35).
3	<typeRef name="MDIncRefresh"/>	Specifies the type reference name that allows the template to be referenced in other templates.
4	<string name="ApplVerID" id="1128"> <constant value="FIX.5.0" /> </string>	Field instruction for ApplVerID defined as a string with an identifier of 1128 corresponding to the FIX tag number. ApplVerID has a constant field operator with a value of FIX.5.0 indicating the FIX version.
5	<string name="MessageType" id="35"> <constant value="X" /> </string>	Field instruction for MessageType defined as a string with identifier = 35 corresponding to the FIX tag number. MessageType has a constant field operator with a value of X which indicates the FIX message type—in this case Market Data Incremental Refresh.
6	<string name="SenderCompID" id="49"> <constant value="CQG" /> </string>	Field instruction for SenderCompID defined as a string with identifier = 49 corresponding to the FIX tag number. SenderCompID has a constant field operator with a value of 'CQG' which indicates the originator of the data.

7	<uInt32 name="MsgSeqNum" id="34"> <increment /> </uInt32>	Field instruction for MsgSeqNum defined as an unsigned integer with identifier = 34 corresponding to the FIX tag number. MsgSeqNum has an increment field operator.
8	<uInt64 name="SendingTime" id="52"> <delta /> </uInt64>	Field instruction for SendingTime defined as an unsigned integer and with identifier = 52 corresponding to the FIX tag number. SendingTime has a delta field operator.
9	<sequence name="MDEntries"> <length name="NoMDEntries" id="268"> <copy /> </length>	Sequence instruction demarks the beginning of the MDEntries repeating group. The sequence includes a length field called 'NoMDEntries' that specifies the number of repeating groups present in the message. NoMDEntries has a copy field operator.
10	<int32 name="MDEntryPx" id="270"> <delta /> </int32>	Field instruction for MDEntryPx (first field instruction in repeating group) defined as a signed integer with identifier = 270 corresponding to the FIX tag number. MDEntryPx has a delta field operator.
11	<int32 name="MDEntrySize" id="271"> <delta /> </int32>	Field instruction for MDEntrySize defined as a signed integer with identifier = 271 corresponding to the FIX tag number. MDEntrySize has a delta field operator.
12	<uInt32 name="MDEntry-Time" id="273"> <delta /> </uInt32>	Field instruction for MDEntryTime which is defined as an unsigned integer with identifier = 273 corresponding to the FIX tag number. MDEntryTime has a delta field operator.
13	<uInt32 name="MDPriceLevel" id="1023" presence="optional"> <increment /> </uInt32>	Field instruction for MDPriceLevel defined as an unsigned integer with identifier = 1023 corresponding to the FIX tag number. MDPriceLevel has a delta field operator.
14	<uInt32 name="MDUpdateAction" id="279"> <copy value="0" /> </uInt32>	Field instruction for MDUpdateAction defined as an unsigned integer and identifier = 279 corresponding to the FIX tag number. MDUpdateAction has a copy field operator with a value of 0.
15	<string name="MDEntryType" id="269"> <default value="1" /></string>	Field instruction for MDEntryType which is defined as a string and has an identifier of 269 which corresponds to the FIX tag number. MDEntryType has a default field operator with a value of 1.
16	<int32 name="SecurityID" id="48"><delta/></int32>	Field instruction for SecurityID defined as an unsigned integer and has an identifier = 48 corresponding to the FIX tag number. SecurityID has a delta field operator.
17	<int32 name="NumberOfOrders" id="346" presence="optional"> <delta /> </int32>	Field instruction for NumberOfOrders defined as a signed integer and with identifier = 346 corresponding to the FIX tag number. NumberOfOrders has a delta field operator.
18	<string name="QuoteCondition" id="276" presence="optional"> <copy /> </string>	Field instruction for QuoteCondition defined as a string with identifier = 276 corresponding to the FIX tag number. QuoteCondition has a copy field operator.
19	<string name="TickDirection" id="274" presence="optional"> <default /> </string>	Field instruction for TickDirection is defined as a string with identifier = 274 corresponding to the FIX tag number. TickDirection has a default field operator.
20	<uint32 name="NetChgPrevDay" id="451" presence="optional"> <default /> </uint32>	Field instruction for NetChgPrevDay defined as an unsigned integer with identifier = 451 corresponding to the FIX tag number. NetChgPrevDay has a default field operator.
21	<string name="TradeCondition" id="277" presence="optional"> <default /> </string>	Field instruction for TradeCondition defined as a string with identifier = 277 corresponding to the FIX tag number. TradeCondition has a default field operator.
22	<int32 name="TradeVolume" id="1020" presence="optional"> <default /> </int32>	Field instruction for TradeVolume defined as a signed integer with identifier = 1020 corresponding to the FIX tag number. TradeVolume has a default field operator.
23	<string name="TradingSessID" id="336"> <default value="1" /> </string>	Field instruction for TradingSessID defined as a string with identifier = 336 corresponding to the FIX tag number. TradingSessID has a default field operator. TradingSessID is the last field in the Sequence

4. Template Overview

CQG provides a single xml file that contains a collection of templates and is shared across all market data channels between the encoder (CQG side) and decoder (customer side). Each template has a unique template ID that describes the format of an encoded message. A template ID is carried in every encoded message to provide a reference to the correct template for decoding purposes.

A template selection process should be carried out on the customer side. The decoder should dynamically identify and retrieve the correct template indicated in the encoded message, then use that template to decode the message.



4.1 XML Template Example

A template consists of Field Instructions that define the fields contained in the message. Field Instructions specify the field name, tag number, data type, field operator, and presence attribute that indicates if a field is optional or mandatory.

A sample market data template is shown below. The syntax is standard XML and can be parsed using a variety of open source tools. Valid template syntax is determined by the FAST Template Schema which is available in the FAST v1.1 specification.

The real-time feed templates are based on the Market Data Incremental Refresh message type.

The information contained in a template is passed to the client FAST decoder at run-time such that the decoder recognizes how each field is encoded in terms of data type representation (Transfer) and data redundancy removal (Field).

The template is constructed of several sections including Template Identification, Header, Body and Sequence. Template Identification provides the template name and identifier. The Header includes FIX header fields such as ApplVerID (tag 1128), MsgType (tag 35), and SendingTime (tag 52). The Body provides information common across all repeating groups. Sequence

represents a repeating group with a corresponding length field and a set of repeating group fields which carry the detailed entry information.

```
<?xml version="1.0" encoding="UTF-8" ?>
<templates xmlns="http://www.fixprotocol.org/ns/fast/td/1.1">
   <template name="MsgHeader">
      <string id="1128" name="ApplVerID"> <constant value="1"/> </string>
      <string id="49" name="SenderCompID"> <constant value="CQG"/> </string>
      <uInt32 id="34" name="MsgSeqNum"/>
      <uInt64 id="52" name="SendingTime"/>
   </template>
   <template dictionary="1" id="1" name="MDIncRefresh">
      <string id="35" name="MessageType"> <constant value="X"/> </string>
      <templateRef name="MsgHeader"/>
      <sequence name="MDEntries">
         <length id="268" name="NoMDEntries"/>
         <uInt32 id="279" name="MDUpdateAction" presence="optional"> <copy value="0"/> </uInt32>
         <uInt32 id="1023" name="MDPriceLevel" presence="optional"> <default value="1"/> </uInt32>
         <string id="269" name="MDEntryType"/>
         <uInt32 id="48" name="SecurityID" presence="optional"> <copy/> </uInt32>
         <uInt32 id="83" name="RptSeq" presence="optional"> <increment/> </uInt32>
         <decimal id="270" name="MDEntryPx" presence="optional">
            <exponent> <default value="0"/> </exponent>
            <mantissa> <delta/> </mantissa> </decimal>
         <uInt32 id="273" name="MDEntryTime"> <copy/> </uInt32>
         <int32 id="271" name="MDEntrySize" presence="optional"> <delta/> </int32>
         <string id="276" name="QuoteCondition" presence="optional"> <default value="K"/> </string>
         <uInt32 id="1070" name="MDQuoteType" presence="optional"> <constant value="0"/> </uInt32>
      </sequence>
   </template>

   <template dictionary="2" id="2" name="MDSecurityDefinition">
      <string id="35" name="MessageType"> <constant value="d"/> </string>
      <templateRef name="MsgHeader"/>
      <uInt32 id="911" name="TotNumReports"/>
```

4.2 Template Implementation Considerations

The following items should be considered before implementing template functionality:

- Any change to the template will result in an update to the template ID.
- The following template changes should be handled by the client without any changes to their decoder.

Note:

Customers will be notified 2-4 weeks prior to a template change.

Template Change	Decoder Impact	Template Release Plan	Additional Information
New Tags (CQG originated)	none	A template that contains a new tag (CQG originated) will be released in the New Release Certification environment approximately 4 weeks prior to Production environment (and Certification environment) roll-out.	A new tag may be defined as a result of a change in the business logic (which client systems can choose to implement or not). This change should not require a modification to the client system decoder, however, there may be changes required to the clients back-end systems if they choose to implement this change.

New Tags (FIX originated)	none	A template that contains a new tag (FIX originated) will be released in the New Release environment approximately 4 weeks prior to Production environment (and Certification environment) roll-out.	A new tag may be defined as a result of a change in the business logic (which client systems can choose to implement or not). This change should not require a modification to the client system decoder, however, there may be changes required to the clients back-end systems if they choose to implement this change.
Removed Tags	none	A template in which a tag is no longer available will be released in the New Release environment approximately 4 weeks prior to Production environment (and Certification environment) roll-out.	A tag may be removed, for example, if it is no longer needed.
Modified Tags	none	A template that contains a modified tag will be released in the New Release environment approximately 4 weeks prior to Production environment (and Certification environment) roll-out.	A modification to an existing tag may be defined as a result of a change in the business logic (which client systems can choose to implement or not). This tag modification should not require a modification to the client system decoder, however, there may be changes required to the clients backend systems if they choose to implement this change.
Rename or Reorder Tags	none	A template that contains a renamed or reordered tag will be released in the New Release environment approximately 2 weeks prior to Production environment (and Certification environment) roll-out.	A renamed or reordered tag may be defined to increased efficiency. This tag modification should not require a modification to the client system decoder, however, there may be changes required to the clients back-end systems if they choose to implement this change.
New Message Types	none	A template that contains a new message type will be released in the New Release environment approximately 4 weeks prior to Production environment roll-out.	A new message type may be defined as a result of an enhanced CQG functionality, or a change in business logic (which FIX/FAST client systems can choose to implement or not). This change should not require a modification to the FIX/FAST client system decoder, however, there may be changes required to the clients back-end systems if they choose to implement this change.
Modified Operators	none	A template that contains a modified operator will be released in the New Release environment approximately 2 weeks prior to Production environment roll-out.	
Modified Data Types	none	A template that contains a modified data type will be released in the New Release environment approximately 2 weeks prior to Production environment roll-out.	

4.3 Template Distribution

The current template for each environment is available for download from an FTP site. Refer to Services - Template Dissemination, Market Data Configuration for an outline of the process.

Note:

CQG strongly recommends that you download the current templates.xml file every Sunday prior to market open.

Historical templates.xml files will be maintained on the FTP site and stored in a directory indicating the dates they were effective. The historical templates will be moved to the "Archive" directory for the corresponding environment.

5. Incremental Book Management

5.1 Overview

CQG uses an electronic market data format based on the FIX standard which provides a sound model for reducing the overall content of data transmitted through an incremental book management approach. FIX provides a flexible protocol for market data messaging which is well suited for transmitting electronic books from the CQG trading platform to customers. Incremental book management provides significant efficiency and flexibility across the entire market data infrastructure. The use of incremental FIX market data messaging in combination with FAST compression produces a highly optimized feed which results in bandwidth savings as well as latency reductions.

CQG represents markets in executable orders and quotes using an order book that may grow indefinitely. This order book is constantly changing as market events cause orders to be added, modified, and cancelled. Updates are then sent out over a market data stream so client systems can then construct a copy of the book in order to track prices in the market and submit appropriately priced orders.

5.2 Incremental Book Management Applicable FIX Message Structures

The Market Data Incremental Refresh (tag 35-MsgType = X) message is used to apply instructions to a book. These instructions are incremental and update applicable parts of the book as necessary, as opposed to refreshing the entire book each time there is an update. This message is used to maintain the order book for contracts listed by CQG. This message, on a real-time basis, is also used to send statistics.

The FIX Market Data message has a FIX header followed by a number of data blocks. Each data block represents a single instruction such as a book update or trade. This section refers to the items as data blocks when describing how the book is maintained.

5.2.1 Common Book Update Tags

Common Tags

Tag	Field Name	Description
279	MDUpdateAction	Type of Market Data update action.
269	MDEntryType	Type of Market Data entry.
83	RptSeq	Sequence number per Instrument update.
276	QuoteCondition	Space-delimited list of conditions describing a quote.
48	SecurityID	Unique instrument ID as qualified by the exchange.
1023	MDPriceLevel	Position in the book.
273	MDEntryTime	Time of Market Data Entry.

Tag	Field Name	Description
270	MDEntryPx	Price of the Market Data Entry.
271	MDEntrySize	Quantity or volume represented by the Market Data Entry.

5.3 Central Limit Order Book

The central limit order book is available over a market data feed that provides an aggregate book. The term "aggregate book" is used to refer to the case where orders are summarized at each price level. The quantity associated with that price level is the aggregation of all individual order quantities for that price level. CQG offers the following:

- multiple-depth book
- top-of-book

Notes:

Client systems must be able to process all valid values in tag 279-MDUpdateAction.

5.4 Book Management Mechanics - Multiple-Depth Book

This section applies to CQG products that have a multiple-depth book. Client systems must determine the book-depth for an instrument from tag 264-MarketDepth in the Security Definition (tag 35-MsgType=d) message.

The examples in this section illustrate the mechanics of a 5-deep book. All books that are not top-of-book will use the same mechanics. For example, a 3-deep book will use the same mechanics as described in this section. For top-of-book only, refer to "Book Management Mechanics -Top-of-Book".

5.4.1 Overview

CQG provides a multiple-depth book for several products. Client systems must determine the book-depth for an instrument from tag 264-MarketDepth in the Security Definition (tag 35-MsgType=d) Message. The aggregate book reports summarized order quantities and order counts at a given price level. The depth represents the number of price levels that are supported in the feed, which may have the following values:

- unlimited
- size=1

The view can be visualized as a number of rows in a table for each of the bid and ask sides. On each side, there are a number of rows showing the quantity available at a number of price levels. An aggregate depth book is sequenced by price, descending for bid and ascending for ask.

CQG provides the best bid and ask in the market for each contract. Trade details and instrument status are provided in separate data blocks. The following table illustrates the types of information that can be displayed:

Top of Book - Best Bid/Ask

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	100	9427.50	9428.00	40	2
19	500	9427.00	9428.50	600	35
34	750	9426.50	9429.00	850	55
25	400	9426.00	9429.50	350	21
14	300	9425.50	9430.00	150	12

CQG maintains the Aggregate Depth view with the following data blocks:

- **Add** - to create/insert a new price at a specified price level (tag 279 MDUpdateAction=0)
- **Change** - change quantity for a price at a specified price level (tag 279 MDUpdateAction=1)
- **Delete** - remove a price at a specified price level (tag 279 MDUpdateAction=2)

An Aggregate book is built from a series of data blocks which indicate whether an entry is to be inserted (Add), changed (Change), or removed (Delete). All data blocks are issued for a specified entry type (tag 269), price (tag 270), and price level (tag 1023). The incremental instruction approach assumes the use of the Market Data Incremental Refresh (tag 35-MsgType =X) message. The Bid and Ask sides are updated independently with separate data blocks. The practice of sending separate data blocks provides efficiencies by allowing only the bid or ask to be sent, based on which side has changed, rather than both sides.

CQG sends an add data block if there is a new price level. Client systems should then shift price levels down. CQG will send explicit deletes when needed.

CQG sends a delete data block to remove a price level in the book. Client systems should shift prices below the data block up to the price level vacated by the deleted price level. If available, an add data block will be sent to fill in the last price level.

The change data block is sent to update characteristics of a price level without changing the price itself, or impacting any other prices on the book. The change data block is sent to update the order count and / or quantity for a price level. The change data block is not sent when the price changes at a given price level.

In general, if a trade occurs, CQG will send a delete or change data block to update the book. The trade data block itself is not used to update the order book.

5.4.1.1 Basic Book Update Data Block

A FIX message sent to update the top of book will update one side only. The tags normally sent for a book update data block are:

- tag 279-MDUpdateAction
- tag 269-MDEntryType
- tag 83-RptSeq
- tag 1023-MDPriceLevel
- tag 273-MDEntryTime
- tag 271-MDEntrySize
- tag 270-MDEntryPx
- tag 48-SecurityID

5.4.2 Examples

This example shows how a book is built and updated. The book generally consists of a set of bid prices and ask prices, in which bids are descending and asks are ascending. The quantity and order count is provided at each price level.

The following table illustrates an initial market book:

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	100	9427.50	9428.00	40	2
19	500	9427.00	9428.50	600	35
34	750	9426.50	9429.00	850	55
25	400	9426.00	9429.50	350	21
14	300	9425.50	9430.00	150	12

5.4.2.1 Quantity on Buy Side Modified

The quantity of an order can be modified. The book will show an update to the size displayed:

Buy 90 @ 9427.50

Book Update Instruction

Book Update - Data Block

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	1	1=change. Type Market Data update action.
269	MDEntryType	0	0=bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	90	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9427.50	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange per tag 22-SecurityIDSource.

Updated Book

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	90	9427.50	9428.00	40	2
19	500	9427.00	9428.50	600	35
34	750	9426.50	9429.00	850	55
25	400	9426.00	9429.50	350	21
14	300	9425.50	9430.00	150	12

5.4.2.2 Entire Order Canceled and New Order Entered

An entire order can be canceled, which removes the top of book, and a new order data block will be sent to fill the open price. The book will show the removal of price level 1, followed by an addition to price level 5. A delete data block will be sent for trades.

Note:

Client systems should shift prices below the data block up to the price level vacated by the deleted price level. If all levels in the book are full, an add data block will be sent to fill in the last price level.

Cancel 90 @ 9427.50

Book Update Instruction

Book Update - Data Block 1

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	2	2=delete. Type Market Data update action.
269	MDEntryType	0	0=bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	0	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	0	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange per tag 22-SecurityIDSource.

Add 400@ 9425.00

Book Update Instruction

Book Update – Data Block 2

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	0	0=add. Type Market Data update action.
269	MDEntryType	0	0=bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	5	Price level 5. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	400	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9425.00	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Updated Book

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
19	500	9427.00	9428.00	40	2
34	750	9426.50	9428.50	600	35
25	400	9426.00	9429.00	850	55
14	300	9425.50	9429.50	350	21
1	400	9425.00	9430.00	150	12

5.4.2.3 New Order Entered at Same Price

A new order can be entered to a book at the same price level. The book will show an update:

New Order 3 @ 9427.00

Book Update Instruction

Book Update - Data Block

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	1	1=change. Type Market Data update action.
269	MDEntryType	0	0=bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	503	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9427.00	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Updated Book

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
20	503	9427.00	9428.00	40	2
34	750	9426.50	9428.50	600	35
25	400	9426.00	9429.00	850	55
14	300	9425.50	9429.50	350	21
1	400	9425.00	9430.00	150	12

5.4.2.4 New Best Price Entered

A new order can be entered to an existing book as a new best price. The data block indicates that a new order should be inserted at price level 1 (new best price). As a result, all orders on the book should be shifted down accordingly.

New Order 200 @ 9427.50

Book Update Instruction

Book Update - Data Block

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	0	0=new. Type Market Data update action.
269	MDEntryType	0	0=bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	200	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9427.50	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange per tag 22-SecurityIDSource.

Updated Book

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	200	9427.50	9428.00	40	2
20	503	9427.00	9428.50	600	35
34	750	9426.50	9429.00	850	55
25	400	9426.00	9429.50	350	21
14	300	9425.50	9430.00	150	12

5.5 Book Management Mechanics – Implied Book

CQG provides a 1-deep best bid and ask in the market for each implied prices futures contract. **Implied book updates are denoted by the presence of tag 276-QuoteCondition = K (Implied)**. Trade details and instrument status are provided in separate data blocks. The following table illustrates the types of information that can be displayed:

Bid		Ask	
Quantity	**Price**	**Price**	**Quantity**
100	9427.50	9428.00	40

CQG maintains the Aggregate Depth view with the following data blocks:

- **Add** - to create/insert a new price at a specified price level (tag 279 MDUpdateAction=0)
- **Change** - change quantity of a price at a specified price level (tag 279 MDUpdateAction=1)
- **Delete** - remove a price at a specified price level (tag 279 MDUpdateAction=2)

An Aggregate book is built from a series of data blocks which indicate whether an entry is to be inserted (Add), changed (Change), or removed (Delete). All data blocks are issued for a specified entry type (tag 269), price (tag 270), and price level (tag 1023). The incremental instruction approach assumes the use of the Market Data Incremental Refresh message (tag 35-MsgType =X). The Bid and Ask sides are updated independently with separate data blocks. The practice of sending separate data blocks provides efficiencies by allowing only the bid or ask to be sent, based on which side has changed, rather than both sides.

CQG sends an add data block if there is a new price level. Client systems should then shift price levels down. An explicit delete will be sent by CQG when needed.

CQG sends a delete data block to remove a price level in the book. Client systems should shift prices below the data block up to the price level vacated by the deleted price level. If all levels in the book are full, an add data block will be sent to fill in the last price level.

The change data block is sent to update characteristics of a price level without changing the price itself, or impacting any other prices on the book. The change data block is sent to update the order quantity for a price level. The change data block is not sent when the price changes at a given price level.

If a trade occurs, CQG will send a delete or change data block to update the book. The trade data block itself is not used to update the order book.

5.5.1 Basic Book Update Data Block

A FIX message sent to update the top of book will update one side only. The tags normally sent for a book update data block are:

- tag 279-MDUpdateAction
- tag 269-MDEntryType
- tag 83-RptSeq
- tag 276-QuoteCondition
- tag 1023-MDPriceLevel
- tag 273-MDEntryTime
- tag 271-MDEntrySize
- tag 270-MDEntryPx
- tag 48-SecurityID

5.5.2 Examples

This example shows how a book is built and updated. The book generally consists of a set of bid prices and ask prices, in which bids are descending and asks are ascending. The quantity is provided at each price level. For implied prices futures, a two-deep book is available.

The following table illustrates an initial market book.

Bid		Ask	
Quantity	**Price**	**Price**	**Quantity**
100	9427.50	9428.00	40

5.5.2.1 Quantity on Buy Side Modified

The quantity of an order can be modified. The book will show an update to the size displayed:

Buy 90 @ 9427.50

Book Update Instruction

Book Update - Data Block

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	1	1=Change. Type Market Data update action.
269	MDEntryType	0	0=bid. Type of Market Data entry.

Tag Number	Tag Name	Value	Description
83	RptSeq		Sequence number per Instrument update.
276	QuoteCondition		Space-delimited list of conditions describing a quote.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	90	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9427.50	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Updated Book

Bid		Ask	
Quantity	**Price**	**Price**	**Quantity**
90	9427.50	9428.00	40

5.5.2.2 Entire Order Canceled and New Order Entered

An entire order can be canceled, which removes the top of book and a new order data block will be sent to fill the open price level 2. The book will show the removal of the first price level, followed by an addition to the second price level.

Note:

Client systems should shift prices below the data block up to the price level vacated by the deleted price level. If all levels in the book are full, an add data block will be sent to fill in the last price level.

Cancel 90 @ 9427.50

Book Update Instruction

Book Update -Data Block 1

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	2	2=Delete. Type Market Data update action.
269	MDEntryType	0	0=bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
276	QuoteCondition		Space-delimited list of conditions describing a quote.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	90	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9427.50	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange per tag 22-SecurityIDSource.

Add 80 @ 9426.50

Book Update Instruction

Book Update - Data Block 2

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	0	0=Add. Type Market Data update action.
269	MDEntryType	0	0=Bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
276	QuoteCondition		Space-delimited list of conditions describing a quote.
1023	MDPriceLevel	2	Price level 2. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	80	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9426.50	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Updated Book

Bid		Ask	
Quantity	**Price**	**Price**	**Quantity**
200	9427.00	9428.00	40
80	9426.50	9428.50	100

5.5.2.3 New Order Entered at Same Price

A new order can be entered to a book at the same price level. The book will show an update:

New Order 3 @ 9427.00

Book Update Instruction

Book Update - Data Block 1

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	1	1=Change. Type Market Data update action.
269	MDEntryType	0	0=Bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
276	QuoteCondition		Space-delimited list of conditions describing a quote.
1023	MDPriceLevel	1	Price level 2. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	203	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9427.00	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Updated Book

Bid		Ask	
Quantity	**Price**	**Price**	**Quantity**
203	9427.00	9428.00	40

5.6 Book Management Mechanics – Top-of-Book

This section applies to CQG products that are top-of-book. Client systems must determine the book-depth for an instrument from tag 264-MarketDepth in the Security Definition (tag 35-MsgType=d) message.

5.6.1 Overview

CQG provides a multiple-depth book for several products. Client systems must determine the book-depth for an instrument from tag 264-MarketDepth in the Security Definition (tag 35-MsgType=d) Message.

CQG provides the current best bid and ask (top-of-book) in the market for certain instruments. Client systems must determine the book-depth for an instrument from tag 264-MarketDepth in the Security Definition (tag 35-MsgType=d) Message. Trade details and instrument status are provided in separate data blocks. The following table illustrates the types of information that can be displayed:

Book -Best Bid/Ask

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	100	9427.50	9428.00	40	2

CQG maintains a top-of-book view with the following data block:

Overlay - add, modify, or delete an entry in the book. (tag 279 MDUpdateAction=5)

Note:

When the best price changes, CQG sends a single overlay instruction for price level 1.

The incremental instruction approach assumes the use of the Market Data Incremental Refresh message (tag 35-MsgType =X). The Bid and Ask sides are updated independently with separate data blocks. The practice of sending separate data blocks provides efficiencies by allowing only the bid or ask to be sent, based on which side has changed, rather than both sides.

5.6.1.1 Basic Book Update Data Block

A FIX message sent to update a top-of-book updates one side only. The tags normally sent for a top-of-book data block are:

- tag 279-MDUpdateAction
- tag 269-MDEntryType
- tag 83-RptSeq
- tag 1023-MDPriceLevel
- tag 273-MDEntryTime
- tag 271-MDEntrySize
- tag 270-MDEntryPx
- tag 48-SecurityID

5.6.1.2 Indexing the Entry

Top-of-book entries are referenced using a composite index which consists of the Security Description, Tag 269-MDEntryType (bid/ask) and Tag 270-MDEntryPx. This set of fields acts as a composite key that allows the entry (bid or ask) to be accessed and subsequently, updated or deleted.

5.6.2 Examples

This example shows how a book is built and updated. The book generally consists of a set of bid prices and ask prices, in which bids are descending and asks are ascending. The quantity and order count are provided.

The following table illustrates an initial market book.

Top-of-Book - Best Bid/Ask

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	100	9427.50	9428.50	40	2

5.6.2.1 Quantity on Buy Side Modified

The quantity of an order can be modified. The book will show an update to the size displayed:

Buy 90 @ 9427.50

Book Update Instruction

Book Update - Data Block 1

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	5	5=overlay. Type Market Data update action.
269	MDEntryType	0	0=Bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	90	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9427.50	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Top-of-Book - Best Bid/Ask

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	90	9427.50	9428.50	40	2

5.6.2.2 Entire Order Canceled

An entire order can be canceled, which removes the top of book. The book will show an update:

Cancel 90 @ 9427.50

Book Update Instruction

Book Update - Data Block 1

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	5	5=overlay. Type Market Data update action.
269	MDEntryType	0	0=Bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	0	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx		Price of the Market Data Entry.
346	NumberOfOrders	0	Number of orders in the market.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Top-of-Book - Best Bid/Ask

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
			9428.50	40	2

5.6.2.3 New Order Entered

A new order can be entered to a book. The book will show an update:

New Order 10 @ 9428.00

Book Update Instruction

Book Update -Data Block 1

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	5	5=overlay. Type Market Data update action.
269	MDEntryType	0	0=Bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	10	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9428.0	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Top-of-Book - Best Bid/Ask

Bid			Ask		
Order Count	**Quantity**	**Price**	**Price**	**Quantity**	**Order Count**
1	10	9428.00	9428.50	40	2

5.6.2.4 New Order Entered at Same Price

A new order can be entered to a book at the same price level. The book will show an update:

New Order 3 @ 9428.00

Book Update Instruction

Book Update -Data Block 1

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	5	5=overlay. Type Market Data update action.
269	MDEntryType	0	0=Bid. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
1023	MDPriceLevel	1	Price level 1. Position in the book.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	13	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9428.0	Price of the Market Data Entry.
346	NumberOfOrders	2	Number of orders in the market.
48	SecurityID		Unique instrument ID as qualified by the exchange.

Top-of-Book - Best Bid/Ask

Bid			Ask		
Order Count	Quantity	Price	Price	Quantity	Order Count
2	13	9428.00	9428.50	40	2

6. Real Time Statistics (Market Behavior Events)

There are a number of statistics (market data events) which are related to changes in a book, but are not used to update the book. The following artifacts fit this category: trade, total volume, open interest, and pre-opening statistics.

Notes:

Total volume is only sent if it differs from what can be estimated from trade updates (i.e., when there were implied trades they are not posted as trades but total volume will be updated).
Open interest (OI) is send depending on feed source.

These events describe the behavior of the market and allow a user to know when the market is moving in a certain direction and provide historical information on how the market has performed.

Note:

Multiple data blocks may be sent in the same message. To determine the number of data blocks you will receive in the message, refer to tag 268-NoMDEntries. Within the message, data blocks may be for different instruments or entry types (book update, statistics, or trades).

6.1 Pre-Opening Statistics

CQG provides indicative opening price.

FIX Syntax for Opening -Market Data Incremental Refresh (tag 35-MsgType = X)

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	0	0=new. Type Market Data update action.
269	MDEntryType	4	4=opening price. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
273	MDEntryTime		Time of Market Data Entry.
270	MDEntryPx	9550.00	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.
1070	MDQuoteType	0	0 = Indicative. Identifies the type of quote.

6.2 Trade

The trade data block is sent when a trade occurs to provide volume and trade statistics. The following sections detail various trade types and their respective data.

Tag 269-MDEntryType=2 (trade) is not used to update the view of the book.

A trade data block may be flagged as either the beginning or the end of a CQG event.

6.2.1 Example 1 – Two Outright Orders Trading

The following example illustrates the trade data block that is sent when a buy and sell match on GEZ2.

FIX Syntax for GEZ2 Outright Trade - Market Data Incremental Refresh (tag 35-MsgType=X)

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	0 or 2	0=new and 2=delete. Type Market Data update action.
269	MDEntryType	2	2=trade. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize	5	Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9550.00	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange per tag 22-SecurityIDSource.

6.3 Closing and Settlement Price

The price data block is sent to update settlement price. This data block is useful for obtaining the settlement price and is sent after the close of the trading session.

FIX Syntax for Previous Day Adjustment - Market Data Incremental Refresh (tag 35-MsgType = X)

Tag Number	Tag Name	Value	Description
279	MDUpdateAction	0	0=new. Type Market Data update action.
269	MDEntryType	5 or 6	5=closing price and 6= settlement price. Type of Market Data entry.
83	RptSeq		Sequence number per Instrument update.
273	MDEntryTime		Time of Market Data Entry.
271	MDEntrySize		Quantity or volume represented by the Market Data Entry.
270	MDEntryPx	9550.00	Price of the Market Data Entry.
48	SecurityID		Unique instrument ID as qualified by the exchange.

7. CQG Pricing

This section describes the pricing and tick convention used for instruments available on CQG. The CQG price is the format used in all price tags in order entry and market data interfaces. One method for obtaining the CQG price is from tag 270-MDEntryPX of the Market Data Incremental Refresh tag 35-MsgType=X message.

7.1 Tick Convention

The tick size is the minimum fluctuation in price allowed for a futures or options contract during a trading session as specified in the security definitions.

7.2 Fractional Pricing

The information in this section is for instruments that require a decimal-to-fractional price conversion as indicated in the following market data messages:

FIX/FAST Direct -Security Definition (tag 35-MsgType=d) message

- tag 871-InstAttributeType='24'
- tag 872-InstAttribValue='12'

For CBOT products with Fractional Pricing Flag = '0', customers can use the Display Factor for FIX/FAST to determine delimiter placement.

7.2.1 Tick Conversion Grid

Decimal to Formatted Fractional Tick

FPF	1	1	1	1	1	1	1	1	1	1
TDFT	01	02	04	08	16	32	64	Half 1/32 Tick	Quarter 1/32 Tick	Half 1/64 Tick
NDDP	00	01*	01	01	02	02	02	03	03	03
0			0	0	00	00	00	000	000	000
.0078125									002	005
.015625							01	005	005	010
.0234375									007	015
.03125						01	02	010	010	020
.0390625									012	025
.046875							03	015	015	030
.0546875									017	035
.0625					01	02	04	020	020	040
.0703125									022	045
.078125							05	025	025	050
.0859375									027	055
.09375						03	06	030	030	060
.1015625									032	065
.109375							07	035	035	070
.1171875									037	075
.125				1	02	04	08	040	040	080
.1328125									042	085
.140625							09	045	045	090
.1484375									047	095
.15625						05	10	050	050	100
.1640625									052	105
.171875							11	055	055	110
.1796875									057	115
.1875					03	06	12	060	060	120
.1953125									062	125
.203125							13	065	065	130
.2109375									067	135
.21875						07	14	070	070	140
.2265625									072	145
.234375							15	075	075	150
.2421875									077	155
.25			2	2	04	08	16	080	080	160
.2578125									082	165
.265625							17	085	085	170
.2734375									087	175
.28125						09	18	090	090	180
FPF	1	1	1	1	1	1	1	1	1	1
TDFT	01	02	04	08	16	32	64	Half 1/32 Tick	Quarter 1/32 Tick	Half 1/64 Tick
NDDP	00	01*	01	01	02	02	02	03	03	03
.2890625									092	185

Decimal to Formatted Fractional Tick

.296875							19	095	095	190
.3046875									097	195
.3125					05	10	20	100	100	200
.3203125									102	205
.328125							21	105	105	210
.3359375									107	215
.34375						11	22	110	110	220
.3515625									112	225
.359375							23	115	115	230
.3671875									117	235
.375				3	06	12	24	120	120	240
.3828125									122	245
.390625							25	125	125	250
.3984375									127	255
.40625						13	26	130	130	260
.4140625									132	265
.421875							27	135	135	270
.4296875									137	275
.4375					07	14	28	140	140	280
.4453125									142	285
.453125							29	145	145	290
.4609375									147	295
.46875						15	30	150	150	300
.4765625									152	305
.484375							31	155	155	310
.4921875									157	315
.5			5	4	08	16	32	160	160	320
.5078125									162	325
.515625							33	165	165	330
.5234375									167	335
.53125						17	34	170	170	340
.5390625									172	345
.546875							35	175	175	350
.5546875									177	355
.5625					09	18	36	180	180	360
.5703125									182	365
.578125							37	185	185	370
.5859375									187	375
.59375						19	38	190	190	380
.6015625									192	385
.609375							39	195	195	390
.6171875									197	395
.625				5	10	20	40	200	200	400
.6328125									202	405
.640625							41	205	205	410
FPF	1	1	1	1	1	1	1	1	1	1
TDFT	01	02	04	08	16	32	64	Half 1/32 Tick	Quarter 1/32 Tick	Half 1/64 Tick
NDDP	00	01*	01	01	02	02	02	03	03	03

Decimal to Formatted Fractional Tick

.6484375									207	415
.65625						21	42	210	210	420
.6640625									212	425
.671875							43	215	215	430
.6796875									217	435
.6875					11	22	44	220	220	440
.6953125									222	445
.703125							45	225	225	450
.7109375									227	455
.71875						23	46	230	230	460
.7265625									232	465
.734375							47	235	235	470
.7421875									237	475
.75			7	6	12	24	48	240	240	480
.7578125									242	485
.765625							49	245	245	490
.7734375									247	495
.78125						25	50	250	250	500
.7890625									252	505
.796875							51	255	255	510
.8046875									257	515
.8125					13	26	52	260	260	520
.8203125									262	525
.828125							53	265	265	530
.8359375									267	535
.84375						27	54	270	270	540
.8515625									272	545
.859375							55	275	275	550
.8671875									277	555
.875				7	14	28	56	280	280	560
.8828125									282	565
.890625							57	285	285	570
.8984375									287	575
.90625						29	58	290	290	580
.9140625									292	585
.921875							59	295	295	590
.9296875									297	595
.9375					15	30	60	300	300	600
.9453125									302	605
.953125							61	305	305	610
.9609375									307	615
.96875						31	62	310	310	620
.9765625									312	625
.984375							63	315	315	630
.9921875									317	635
1			0	0	00	00	00	000	000	000

Formatted Fractional Tick to Decimal

FPF	1
TDFT	01
NDDP	00
Ticks	Decimal

FPF	1
TDFT	02
NDDP	01*
Ticks	Decimal

FPF	1
TDFT	04
NDDP	01
Ticks	Decimal
0	.00
2	.25
5	.50
7	.75

FPF	1
TDFT	08
NDDP	01
Ticks	Decimal
0	.000
1	.125
2	.250
3	.375
4	.500
5	.625
6	.750
7	.875

Formatted Fractional Tick to Decimal

FPF	1
TDFT	16
NDDP	02
Ticks	Decimal
00	.0000
01	.0625
02	.1250
03	.1875
04	.2500
05	.3125
06	.3750
07	.4375
08	.5000
09	.5625
10	.6250
11	.6875
12	.7500
13	.8125
14	.8750
15	.9375

FPF	1
TDFT	32
NDDP	02
Ticks	Decimal
00	.00000
01	.03125
02	.06250
03	.09375
04	.12500
05	.15625
06	.18750
07	.21875
08	.25000
09	.28125
10	.31250
11	.34375
12	.37500
13	.40625
14	.43750
15	.46875

FPF	1
TDFT	32
NDDP	02
Ticks	Decimal
16	.50000
17	.53125
18	.56250
19	.59375
20	.62500
21	.65625
22	.68750
23	.71875
24	.75000
25	.78125
26	.81250
27	.84375
28	.87500
29	.90625
30	.93750
31	.96875

Formatted Fractional Tick to Decimal

FPF	1
TDFT	64
NDDP	02
Ticks	Decimal
00	.000000
01	.015625
02	.031250
03	.046875
04	.062500
05	.078125
06	.093750
07	.109375
08	.125000
09	.140625
10	.156250
11	.171875
12	.187500
13	.203125
14	.218750
15	.234375
16	.250000
17	.265625
18	.281250
19	.296875
20	.312500
21	.328125
22	.343750
23	.359375
24	.375000
25	.390625
26	.406250
27	.421875
28	.437500
29	.453125
30	.468750
31	.484375

FPF	1
TDFT	64
NDDP	02
Ticks	Decimal
32	.500000
33	.515625
34	.531250
35	.546875
36	.562500
37	.578125
38	.593750
39	.609375
40	.625000
41	.640625
42	.656250
43	.671875
44	.687500
45	.703125
46	.718750
47	.734375
48	.750000
49	.765625
50	.781250
51	.796875
52	.812500
53	.828125
54	.843750
55	.859375
56	.875000
57	.890625
58	.906250
59	.921875
60	.937500
61	.953125
62	.968750
63	.984375

FPF	1
TDFT	Half 1/32 Tick
NDDP	03
Ticks	Decimal
000	.000000
005	.015625
010	.031250
015	.046875
020	.062500
025	.078125
030	.093750
035	.109375
040	.125000
045	.140625
050	.156250
055	.171875
060	.187500
065	.203125
070	.218750
075	.234375
080	.250000
085	.265625
090	.281250
095	.296875
100	.312500
105	.328125
110	.343750
115	.359375
120	.375000
125	.390625
130	.406250
135	.421875
140	.437500
145	.453125
150	.468750
155	.484375

FPF	1
TDFT	Half 1/32 Tick
NDDP	03
Ticks	Decimal
160	.500000
165	.515625
170	.531250
175	.546875
180	.562500
185	.578125
190	.593750
195	.609375
200	.625000
205	.640625
210	.656250
215	.671875
220	.687500
225	.703125
230	.718750
235	.734375
240	.750000
245	.765625
250	.781250
255	.796875
260	.812500
265	.828125
270	.843750
275	.859375
280	.875000
285	.890625
290	.906250
295	.921875
300	.937500
305	.953125
310	.968750
315	.984375

Formatted Fractional Tick to Decimal

FPF	1	FPF	1	FPF	1	FPF	1
TDFT	Quarter 1/32 Tick	TDFT	Quarter 1/32 Tick	TDFT	Quarter 1/32 Tick	TDFT	Quarter 1/32 Tick
NDDP	03	NDDP	03	NDDP	03	NDDP	03
Ticks	Decimal	Ticks	Decimal	Ticks	Decimal	Ticks	Decimal
000	.0000000	080	.2500000	160	.5000000	240	.7500000
002	.0078125	082	.2578125	162	.5078125	242	.7578125
005	.0156250	085	.2656250	165	.5156250	245	.7656250
007	.0234375	087	.2734375	167	.5234375	247	.7734375
010	.0312500	090	.2812500	170	.5312500	250	.7812500
012	.0390625	092	.2890625	172	.5390625	252	.7890625
015	.0468750	095	.2968750	175	.5468750	255	.7968750
017	.0546875	097	.3046875	177	.5546875	257	.8046875
020	.0625000	100	.3125000	180	.5625000	260	.8125000
022	.0703125	102	.3203125	182	.5703125	262	.8203125
025	.0781250	105	.3281250	185	.5781250	265	.8281250
027	.0859375	107	.3359375	187	.5859375	267	.8359375
030	.0937500	110	.3437500	190	.5937500	270	.8437500
032	.1015625	112	.3515625	192	.6015625	272	.8515625
035	.1093750	115	.3593750	195	.6093750	275	.8593750
037	.1171875	117	.3671875	197	.6171875	277	.8671875
040	.1250000	120	.3750000	200	.6250000	280	.8750000
042	.1328125	122	.3828125	202	.6328125	282	.8828125
045	.1406250	125	.3906250	205	.6406250	285	.8906250
047	.1484375	127	.3984375	207	.6484375	287	.8984375
050	.1562500	130	.4062500	210	.6562500	290	.9062500
052	.1640625	132	.4140625	212	.6640625	292	.9140625
055	.1718750	135	.4218750	215	.6718750	295	.9218750
057	.1796875	137	.4296875	217	.6796875	297	.9296875
060	.1875000	140	.4375000	220	.6875000	300	.9375000
062	.1953125	142	.4453125	222	.6953125	302	.9453125
065	.2031250	145	.4531250	225	.7031250	305	.9531250
067	.2109375	147	.4609375	227	.7109375	307	.9609375
070	.2187500	150	.4687500	230	.7187500	310	.9687500
072	.2265625	152	.4765625	232	.7265625	312	.9765625
075	.2343750	155	.4843750	235	.7343750	315	.9843750
077	.2421875	157	.4921875	237	.7421875	317	.9921875

Formatted Fractional Tick to Decimal

FPF	1
TDFT	Half 1/64 Tick
NDDP	03
Ticks	Decimal
000	.0000000
005	.0078125
010	.0156250
015	.0234375
020	.0312500
025	.0390625
030	.0468750
035	.0546875
040	.0625000
045	.0703125
050	.0781250
055	.0859375
060	.0937500
065	.1015625
070	.1093750
075	.1171875
080	.1250000
085	.1328125
090	.1406250
095	.1484375
100	.1562500
105	.1640625
110	.1718750
115	.1796875
120	.1875000
125	.1953125
130	.2031250
135	.2109375
140	.2187500
145	.2265625
150	.2343750
155	.2421875

FPF	1
TDFT	Half 1/64 Tick
NDDP	03
Ticks	Decimal
160	.2500000
165	.2578125
170	.2656250
175	.2734375
180	.2812500
185	.2890625
190	.2968750
195	.3046875
200	.3125000
205	.3203125
210	.3281250
215	.3359375
220	.3437500
225	.3515625
230	.3593750
235	.3671875
240	.3750000
245	.3828125
250	.3906250
255	.3984375
260	.4062500
265	.4140625
270	.4218750
275	.4296875
280	.4375000
285	.4453125
290	.4531250
295	.4609375
300	.4687500
305	.4765625
310	.4843750
315	.4921875

FPF	1
TDFT	Half 1/64 Tick
NDDP	03
Ticks	Decimal
320	.5000000
325	.5078125
330	.5156250
335	.5234375
340	.5312500
345	.5390625
350	.5468750
355	.5546875
360	.5625000
365	.5703125
370	.5781250
375	.5859375
380	.5937500
385	.6015625
390	.6093750
395	.6171875
400	.6250000
405	.6328125
410	.6406250
415	.6484375
420	.6562500
425	.6640625
430	.6718750
435	.6796875
440	.6875000
445	.6953125
450	.7031250
455	.7109375
460	.7187500
465	.7265625
470	.7343750
475	.7421875

FPF	1
TDFT	Half 1/64 Tick
NDDP	03
Ticks	Decimal
480	.7500000
485	.7578125
490	.7656250
495	.7734375
500	.7812500
505	.7890625
510	.7968750
515	.8046875
520	.8125000
525	.8203125
530	.8281250
535	.8359375
540	.8437500
545	.8515625
550	.8593750
555	.8671875
560	.8750000
565	.8828125
570	.8906250
575	.8984375
580	.9062500
585	.9140625
590	.9218750
595	.9296875
600	.9375000
605	.9453125
610	.9531250
615	.9609375
620	.9687500
625	.9765625
630	.9843750
635	.9921875

8. Recovery

CQG offers several options for recovering missed messages or synchronizing FIX/FAST client systems to the latest state: TCP Replay and Market Recovery. Instrument level sequencing and natural refresh can be utilized to supplement the recovery process.

This section contains an overview of the various recovery methods and supplemental methods.

- Recovery Feeds - this section describes the following: TCP Replay, Market Recovery, and Instrument Replay
- Using the Incremental Market Data Feed to Determine State - the methods in this section provide supplemental methods to the recovery process, however they are not guaranteed to completely recover missed messages or synchronize client systems to the latest state.

There is also a section that describes at a high level the process to recover in various situations based on the recovery and supplemental methods.

- Recovering Data - Process

8.1 Recovery Feeds

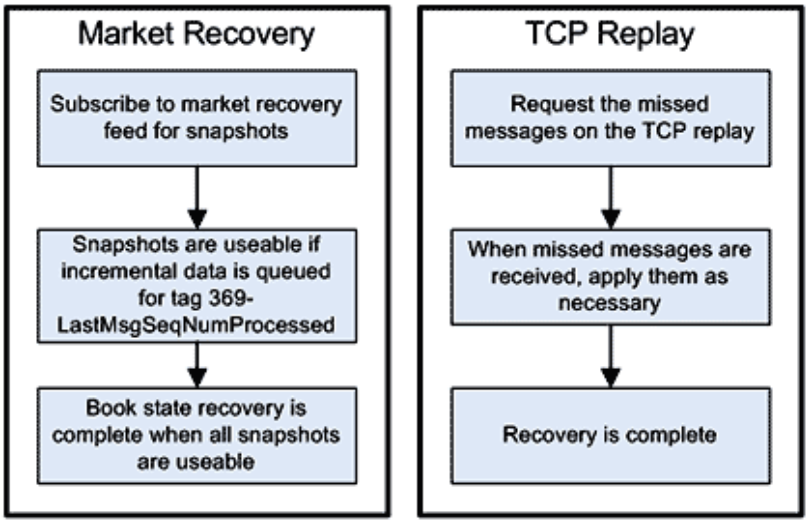
Message loss is detected using the decoded FIX message sequence numbers (tag 34-MsgSeqNum). The message sequence number is an incrementing number. If a gap is detected between messages in tag 34-MsgSeqNum, this indicates a message has been missed. It should be assumed at this point that all books maintained in the FIX/FAST client system may no longer have the correct, latest state maintained by CQG. Client systems must resynchronize all books to the latest state maintained by CQG, and determine whether any new instrument definitions were published.

Note:

During this synchronization process, all books are initially assumed to be in an incorrect state and are recovered during the synchronization process.

The following options are offered by CQG to recover missed messages or synchronize FIX/FAST client systems to the latest state:

- TCP Replay Overview
- Market Recovery Overview



8.1.1 TCP Replay Overview

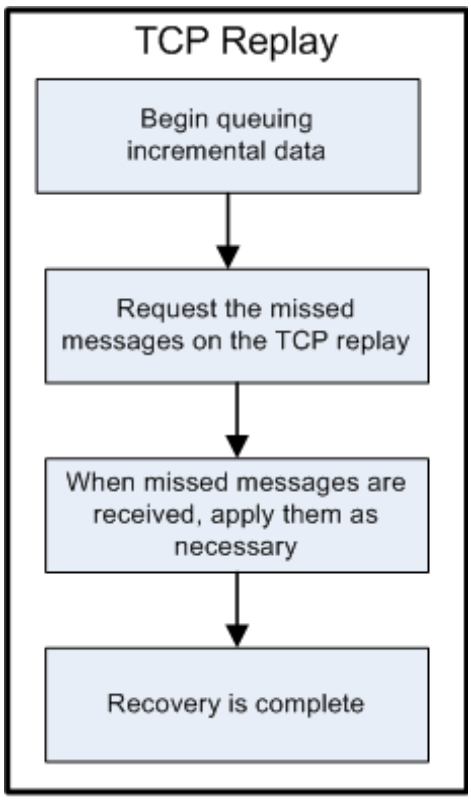
TCP replay should be used for small-scale data recovery (refer to "Implementation Considerations" on Page 84 for additional information). Client systems can recover specific messages that were missed using the sequence number and the TCP historical replay component. This method recovers all missed messages. **CQG logs IP addresses, ports, and passwords of the originator and the Channel ID and range of the requested sequence number.**

The TCP historical replay component allows you to request a replay of a set of messages already published on the UDP Incremental Market Data Channel. The request specifies messages to be replayed. The request is submitted with the FIX Market Data Request message (35=V). In response, CQG sends the requested messages and a Logout (35=5) message. A Logout (35=5) message is also sent in case of rejection of the initial request, tag 58-Text will list the reason for the logout.

This type of request is sent through a new TCP connection established by the customer. When making the request, the channel and sequence number are specified. The responses are sent by CQG through this same connection and the connection is then closed by CQG once the replay is complete. All requests from the client system to CQG are in FIX format. All responses from CQG to the client system are FIX/FAST encoded (including the reject response). The TCP Replay contains a stop-bit delimited block size represented as an unsigned integer in FAST. This stop-bit is used instead of FIX's current 2 byte separator. Replay is limited in the number of messages that can be requested.

Note:

Client systems should queue real-time data until all missed data is recovered. The recovered data should then be applied prior to applying the queued data.



8.1.1.1 Implementation Considerations

The following are limitations to consider prior to implementing TCP replay functionality:

- TCP requests and responses are via TCP.
- Resend request messages and other inbound messages (i.e. heartbeat) are in FIX format, outbound response messages are FAST encoded.
- There is a 24 hour time limit on the number of messages available via the TCP replay functionality.
- The maximum number of messages that can be requested in one resend request message is 2000.

Note:

Only one Market Data Request can be processed during the current established session. If multiple Market Data Requests are sent when the session is established, only the first Market Data Request is processed and subsequent Market Data Requests are ignored.
You must re-establish a new session with the TCP Replay server when the current Market Data request is processed (Login then Logout) and send a new Market Data Request (tag 35-MsgType = V) message.

8.1.1.2 Process

1. Customer establishes a TCP connection.

   Refer to the configuration file for TCP IP and port information. For additional information on the configuration file, refer to Template Dissemination.

2. Customer sends Logon (tag 35-MsgType=A) message to CQG.

Note:

Customer Username and Password are verified. If the Username and Password are incorrect, a Logout (tag 35-MsgType=5) will be sent.

3. CQG sends Logon (tag 35-MsgType=A) message to the customer.

Note:

CQG will send a Logout (tag35-MsgType=5) message if a Market Data Request (tag35-MsgType=V) is not received in 5 seconds.

4. Customer sends Market Data Request (tag 35-MsgType=V) message to CQG.

Note:

Client systems must indicate the channel ID (tag 1180-ApplFeedID) and sequence numbers (tag 1182-ApplBeginSeqNo and tag 1183-ApplEndSeqNo) in the Market Data Request (tag 35-MsgType=V) message.

5. CQG sends Heartbeat (tag35-MsgType=0) messages to customer every 2 seconds until the first recovery message is sent.

Note:

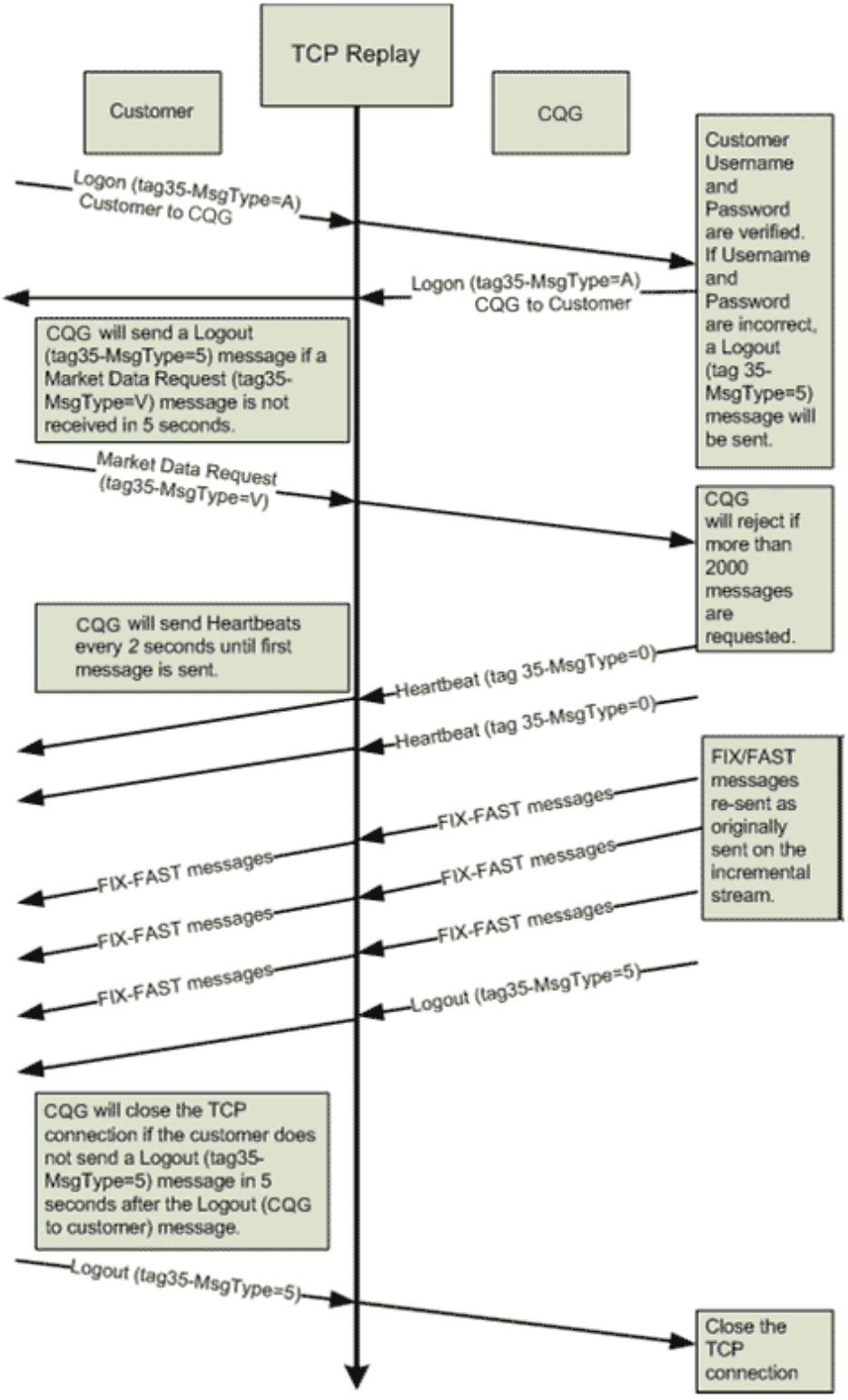
The heartbeat feature is not implemented yet.

6. CQG sends FIX/FAST recovery messages that were requested by the customer in the Market Data Request (tag35-MsgType=V) message.
7. CQG sends a Logout (tag35-MsgType=5) message to the customer.

Note:

CQG will close the TCP connection if the customer does not send a Logout (tag 35-MsgType=5) message within 5 seconds from the time CQG sends a Logout (tag35-MsgType=5) message.

8. Customer sends a Logout (tag 35-MsgType=5) message to CQG.
9. CQG closes the TCP connection.



8.1.2 Market Recovery Overview

This recovery method should be used for large-scale data recovery (i.e. major outage or late joiners) to synchronize client systems to the latest state maintained by CQG. FIX/FAST client systems can use the Market Recovery feed on each channel to determine the state of each book in every channel. Each Market Recovery feed constantly loops and sends the Market Data

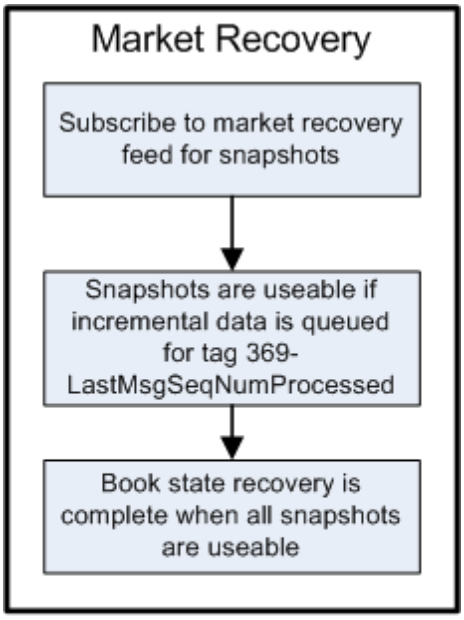
Snapshot Full Refresh (tag 35-MsgType = W) message. The Market Recovery feed is known to be valid as of a sequence number on the Incremental Market Data feed, which is found in tag 369-LastMsgSeqNumProcessed. This sequence number (tag 369-LastMsgSeqNumProcessed) is found on each Market Data Snapshot Full Refresh (tag 35=W) message. If the Market Recovery method is used, client systems also need to subscribe to the Instrument Definition feed to determine whether any new instruments were defined. Client systems will recover the most recent statistics on the Market Recovery feed. Any intermediary statistics will not be recovered.

Note:

FIX/FAST client systems should queue real-time data until all snapshot data is retrieved. The queued data should then be applied as necessary.

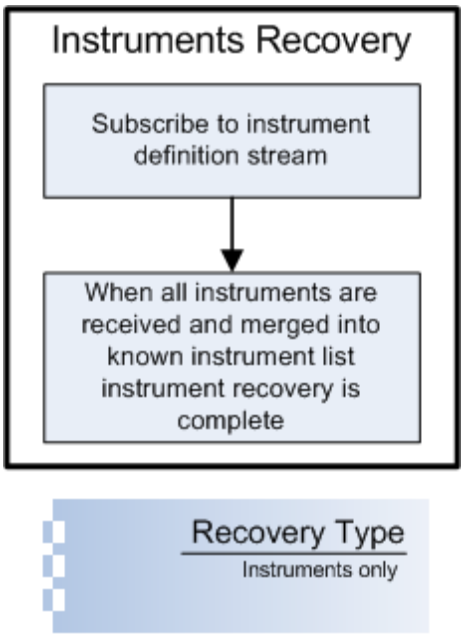
Note:

CQG strongly recommends that the Market Recovery feeds be used for recovery purposes only. Once client systems have retrieved recovery data, client systems should stop listening to the Market Recovery feeds.



8.1.3 SDS Communication Overview

FIX/FAST client systems may maintain their connection with SDS, in which case, they will receive new and updated security definitions as soon as they are published by CQG through the Security Definition (tag 35-MsgType=d) message.



8.2 Using the Incremental Market Data Feed to Determine State

In addition to the recovery methods that are offered by CQG, there are additional mechanisms that client systems can utilize to enhance the recovery process.

- Instrument Level Sequencing

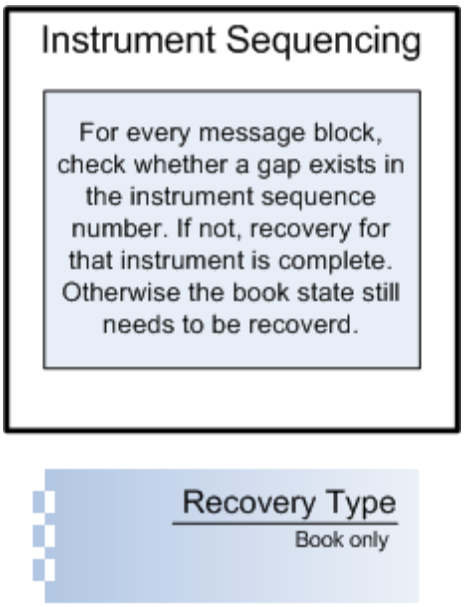
8.2.1 Instrument Level Sequencing

Market Data Incremental Refresh messages (tag 35=X) contain instrument sequence numbers (tag 83-RptSeq), in addition to message sequence numbers (tag 34-MsgSeqNum). Every repeating group instance of a market data entry contains an incrementing sequence number (tag 83-RptSeq) that is associated with the instrument for which data is present in the block. Instrument level sequencing can be used to identify which instruments you have not missed messages for, and apply during the TCP Replay method.

Client systems can keep track of the instrument sequence number (tag 83-RptSeq) for every instrument by inspecting incoming data and determining whether there is a gap in the instrument sequence number (tag 83-RptSeq).

- If there is a gap in the instrument sequence number (tag 83-RptSeq), it indicates that data was missed for the instrument when message loss occurred.
- If there is no gap, the data can be used immediately, and it also indicates that the book for this instrument still has a correct, current state.

It is likely that if only a small number of messages have been missed, there will be data in subsequent messages which are not affected by the missing data. If there are 10 instruments in a channel, for example, and the missed messages contain data for 2 of these instruments, any subsequent messages containing data about the other instruments are still valid.

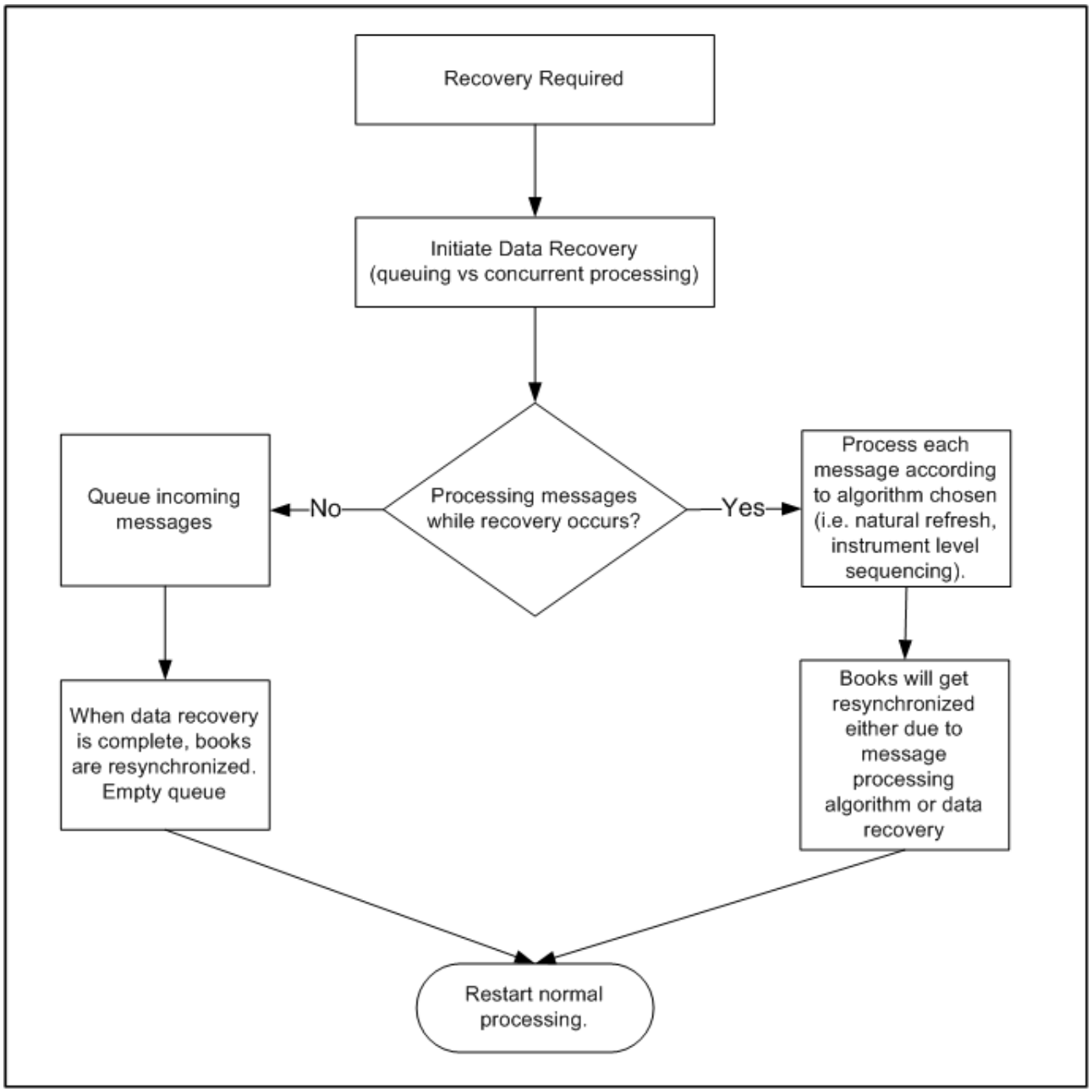


8.3 Recovering Data - Process

This section describes CQG recommended recovery processes. Based on the recovery and supplemental methods, client systems can choose to implement those methods as applicable. In general, two paths can be followed: processing while recovering and queuing while recovering.

Note:

The recovering data process applies to affected channels only. Unaffected channels can continue normal processing.



The following sections provide a high level process to recover in various situations:

- Large Scale Outage Using Market Recovery - Queuing
- Large Scale Outage Using Market Recovery - Concurrent Processing
- Small Scale Data Recovery Using TCP Replay -Queuing
- Small Scale Data Recovery Using TCP Replay -Concurrent Processing

Note:

TCP replay should be used for small-scale data recovery. The Market Recovery feed should be used for large-scale data recovery (i.e. major outage or late joiners).

8.3.1 Large Scale Outage Using Market Recovery - Queuing

This section describes at a high level the process to follow for a large scale outage in which the client system is out of sync. This process uses Market Recovery - queuing of the Incremental Market Data feed and Market Recovery feed until the client system is synchronized to the latest state advertised by CQG. In order to avoid an excessive number of queued messages, process snapshots and apply the applicable incremental feed as the snapshots arrive.

Note:

If the Market Recovery method is used, client systems also need to subscribe to the Instrument Definition feed to determine whether any new instruments were defined. Client systems will not recover any missed statistics on the Market Recovery feed.

1. Identify channel(s) in which the client system is out of sync.
2. Listen to and queue the Incremental Market Data feed for the affected channel(s).
3. Listen to the Market Recovery feed for the affected channel(s).
4. Verify that all snapshots have been received for a given Market Recovery feed.
5. Apply **all** incremental data in sequence, **where tag 34-MsgSeqNum is greater than the lowest value for tag 369-LastMsgSeqNumProcessed.**
6. Using queued data, restart normal processing.

8.3.2 Large Scale Outage Using Market Recovery - Concurrent Processing

This section describes at a high level the process to follow for a large scale outage using Market Recovery while continuing to process the Incremental Market Data feed and obtaining snapshots from the Market Recovery feed. Once books are recovered, they can resume normal processing even if other books are still being recovered.

Note:

If the Market Recovery method is used, client systems also need to subscribe to the Instrument Definition feed to determine whether any new instruments were defined. Client systems will not recover any missed statistics on the Market Recovery feed.

1. Identify channel(s) in which the client system is out of sync.
2. Listen to the Incremental Market Data feed for the affected channel(s) and optionally attempt a natural refresh.
3. Listen to the Market Recovery feed for the affected channel(s).
4. For each book, compare tag 369-LastMsgSeqNumProcessed on the Market Recovery feed to tag 34-MsgSeqNum on the Incremental Market Data feed and verify that the value for tag34-MsgSeqNum is not lower.
5. Restart normal processing.

8.3.3 Small Scale Data Recovery Using TCP Replay - Queuing

This section describes at a high level the process to follow for small scale data recovery using TCP Replay and queuing the Incremental Market Data feed until all missed messages are

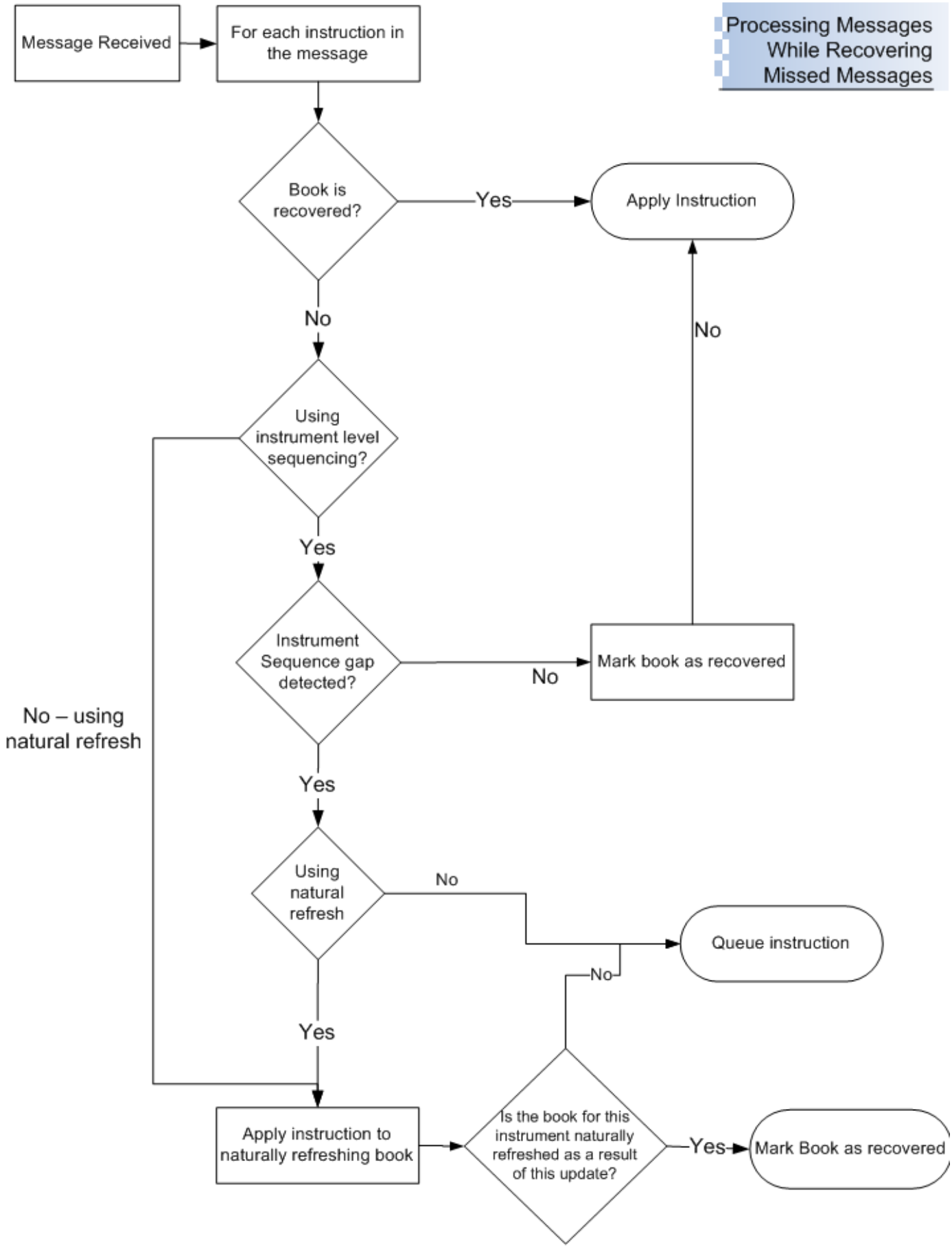
recovered. Client systems can recover specific messages that were missed using the sequence number and the TCP historical replay component.

1. Identify which messages have been missed (tag 34-MsgSeqNum).
2. Stop normal processing and initiate data recovery for affected channel(s).
3. Listen to and queue the Incremental Market Data feed for the affected channel(s).
4. Login to the TCP Replay component and request a replay of missed messages.
5. Apply resent messages in proper sequence.
6. Apply queued messages once replayed messages have been processed.
7. Stop queuing messages and restart normal processing.

8.3.4 Small Scale Data Recovery Using TCP Replay - Concurrent Processing

This section describes at a high level the process to follow for small scale data recovery using TCP Replay while continuing to process the Incremental Market Data feed. Client systems can recover specific messages that were missed using the sequence number and the TCP historical replay component.

1. Identify which messages have been missed (tag 34-MsgSeqNum).
2. Listen to and process the Incremental Market Data feed.
3. Login to the TCP Replay component and request a replay of missed messages.
4. If using Natural Refresh or Instrument Level Sequencing on the Incremental Market Data feed, refer to the following flowchart and apply to process.
5. For queued messages (see diagram):
   - Apply resent messages in proper sequence.
   - Apply queued messages once replayed messages have been processed.
6. Process missed messages from the TCP Replay component.
7. Books will get resynchronized either due to message processing algorithm or data recovery.
8. Restart normal processing.


Processing Messages While Recovering Missed Messages
Message Received
For each instruction in the message
Book is recovered?
Yes
Apply Instruction
No
No
Using instrument level sequencing?
Yes
Instrument Sequence gap detected?
No
Mark book as recovered
No – using natural refresh
Yes
Using natural refresh
No
Queue instruction
No
Yes
Apply instruction to naturally refreshing book
Is the book for this instrument naturally refreshed as a result of this update?
Yes
Mark Book as recovered