ELEKTRON DIRECT FEED

USER GUIDE

Document Version 1.6
Date of issue: *06 October 2015*~~30 September 2015~~

THOMSON REUTERS™

REVISION HISTORY

Name	Date	version	Summary of changes
Robert Cranston	31/03/2014	1.0	Initial Version
Robert Cranston	24/04/2014	1.1	Minor updates. Corrected error in gap recovery multicast address. Added section on ITRS Geneos Integration
Robert Cranston	07/05/2014	1.2	Added updates to the operations section, added additional details on the License Server integration.
Robert Cranston	11/07/2014	1.3	Details on how to capture configuration settings for support. Licensing information
Michael Granoski	05/08/2014	1.4	Updated license details Relocated ITRS config to CI doc Various formatting updates Updated exchange recovery section
Laurent Brenot	22/06/2015	1.5	Added flat wire format support
Laurent Brenot	30/09/2015	1.6	Added TREP Bridge service

REVIEW HISTORY

Reviewer Name	Doc Version Reviewed	Date Sent for review	Date Review Filed	Approved/Rejected (with Reasons)
Cranston, Robert	1.4	05/08/2014		
Deborah, Jaffa				
Herfeld, Gilles				
Planchon, Philippe	1.4	05/08/2014		
Edmonds, John				
Laria, Greg				
Melson, Chris				

CONTENTS

ABOUT THIS DOCUMENT

INTENDED READERSHIP

This document is intended for anyone looking for a detailed overview of the Elektron Direct Feed product.
In this guide

IN THIS GUIDE

This guide breaks out the components involved in the Elektron Direct Feed product. The primary goals of the document are to give users a high level overview of the functionality of the product as well as the details of how the solution operates.

CONVENTIONS

Code, command line commands and variables are shown in Lucida Console font, for example:

```
$ sudo netstat -a
```

File names and paths are shown in Arial bold italics, for example:

celoxica-clx-ng-XXX.rpm

PRODUCT OVERVIEW

The Elektron Direct Feed is a direct feed offering from Thomson Reuters using FPGA based technology to create an extremely low latency feed handler for clients looking for a standardized view of market data from many different global trading venues in the lowest possible latency.

The solution has been modelled using an industry standard multicast distribution mechanism with full recovery and resiliency capabilities built in. This ensures that clients will be able to receive the lowest possible latency on their data while still making sure that in the case of an issue, recovery and getting back to a stable order book is possible as quickly as possible.

PRODUCT POSITIONING

Thomson Reuters Elektron has a number of different products available to clients looking for real time, and static data. In addition to Elektron Direct Feed these include:

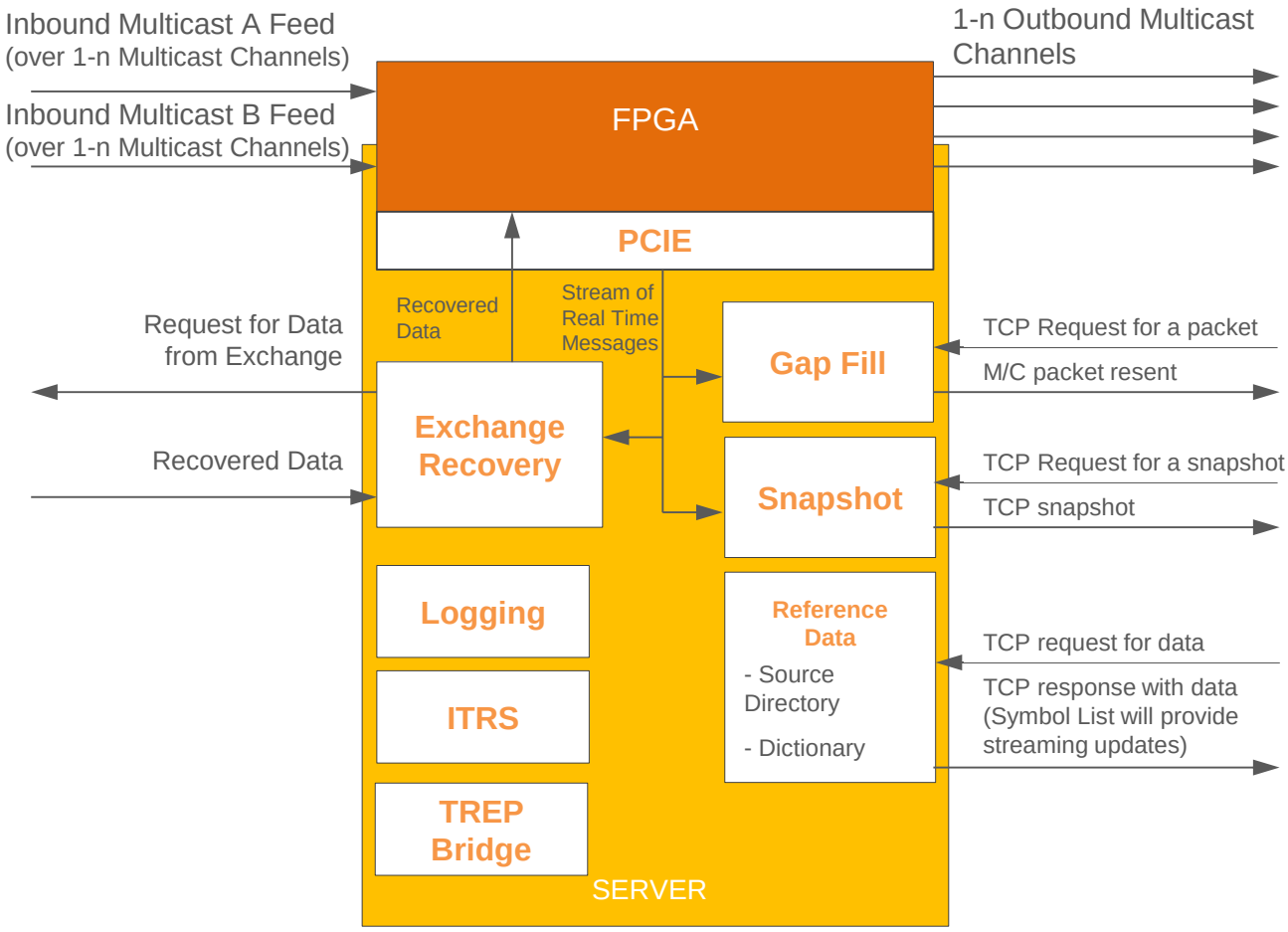
- **Elektron Real Time** – the industry leading consolidated feed offering from Thomson Reuters. Elektron Real Time is focussed on providing a large range of direct and value added data from all over the globe to clients in a single point of presence. It provides low latency access to information from anywhere in the world.

- **Elektron Tick History** – provides access to historical tick services for clients looking to do in depth historical analysis of the markets as well as clients looking for back testing data for algorithms or other trading strategies.

The Elektron Direct Feed is focussed on providing clients looking for an extremely low latency direct feed offering a solution that minimizes the work needed between venues while still providing the needed consistent low latency. The Elektron Direct Feed focuses on standardizing the data into Thomson Reuters Elektron data models and presenting the data to client in the lowest possible latency.

PRODUCT DETAILS

The Elektron Direct Feed is a combination of FPGA accelerated feed handling and a number of software based components to facilitate client and exchange side recovery as well as providing static data. Below is a high level diagram of the overall solution.



FPGA ACCELERATOR CARD

The FPGA card is responsible for processing all of the data from the exchange and pushing it through to clients using UDP Multicast distribution. The hardware itself is actually composed of a number of different components that deal with the data coming in from the exchange on multiple sources and needing to be published to clients.

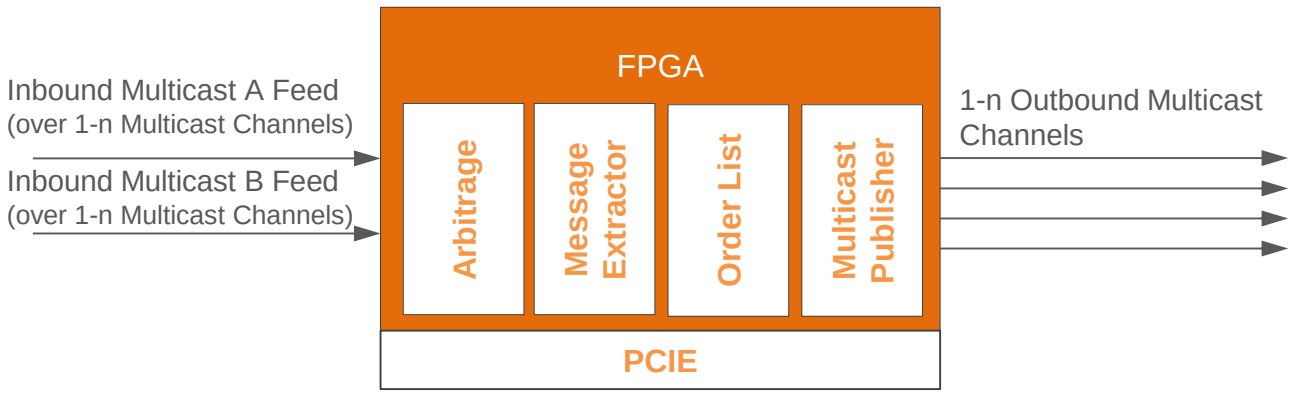
The normalized data is published using either the standard Thomson Reuters data formats and models or a flat wire format which does not need an API to be consumed. The availability of either the RWF format or the flat wire format depends on the exchange feed.

Each FPGA card is responsible for processing the data from one feed, so a separate card is required for each individual feed from the exchange (for example, two cards are required to process BATS BZX and BATS BYX in the US). One server can run multiple cards based on the number of PCIE slots available on the server.

There is one exception to this standard for OPRA. For OPRA, 2 FPGA cards are used, and the solution must run on a single dedicated server. This is due to the exceptional data volumes from this feed.

FPGA Components (Full Hardware Implementation)

The diagram below shows a detailed view of the FPGA card composition.



Arbitrage

For exchanges that give us a multicast stream of data, the normal mode of distribution is to have two separate multicast streams that are used to try and minimize lost packets on any one stream. The process used to combine the two different streams varies slightly depending on the exchange, but this is all taken care of by the FPGA card. Using both streams, the Arbitrage component will pass on a single instance of each message from the exchange to the next component, the ***Message Extractor***.

In the case where the exchange only provides TCP connectivity, the Arbitrage component is not used.

Message Extractor

When we receive the data from the exchange, it immediately goes through a parsing and normalization phase to bring it into an internal format to the card. This logic is specific to each protocol supported. The goal of this piece of logic is to bring every format from the exchanges into a standard representation for the ***Order List***.

Order List

The order list is responsible for keeping track of information on every individual order from the exchange in order to ensure we can process updates and deletions independent of the information provided from the venue on these different messages. As different protocols from the venues provide different levels of information on the update messages, we cache critical information in the order list in order to make sure we can present a standard interface to clients for all of the critical common data on orders. This list also ensures that the card has a full view of the active orders currently on the exchange.

In the case where the exchange publishes a ***Market by Price*** feed of data instead of a ***Market by Order*** stream, the Order List actually stores information on each price level from the book.

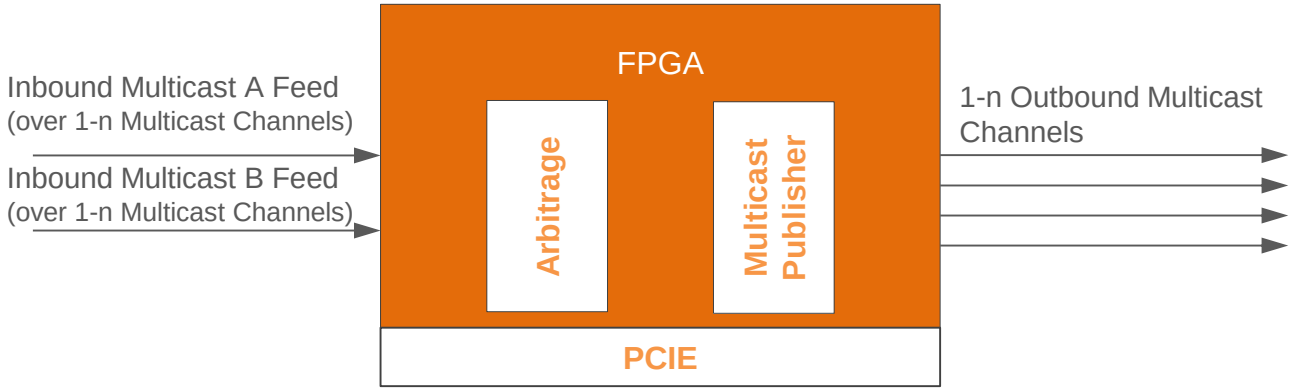
Multicast Publisher

Once we get the order information into the ***Order List***, we publish out the updates to clients in standard RWF wire format. All the formatting into the RWF wire format is done natively in the hardware allowing us to get the low deterministic latency while still publishing standard FID values in a common format.

Depending on the feed available from the exchange, the data published will be either ***Market By Order*** (if the exchange provides an order based feed) or ***Market By Price***, as well as the ***Market Price*** domain being supported for trades and non-order book data.

FPGA Components (Hybrid Hardware/Software Implementation)

The diagram below shows a detailed view of the FPGA card composition.



For solutions that take advantage of a hybrid hardware/software solution for message translation, the FPGA card is responsible for the following pieces of logic:

- Arbitrage
- Multicast Publisher

For the hybrid decoding solution, the message decoding takes place in software.

PCIe Bridge

This is a critical component that passes data to and from the software components from the FPGA card. It passes a real time stream of all data published to clients over multicast also being passed to the software components on the server, as well as vital statistics from the card for monitoring. The software can also pass information to the card via the PCIe for initial install of firmware, configuration of the card for publication, or when we need to be able to recover data from the exchange in the case of dropped packets (see Exchange Recovery).

GAP RECOVERY SERVER

As all data published by the FPGA card is in raw UDP multicast, there is an issue of loss of packets. Clients are welcome to set up multiple cards to re-publish the data in order to try and minimize this, or they can rely on the provided Gap Recovery server to request any missed packets from the real time stream.

The Gap Recovery server listens to all of the data being published by the card either via the PCIe bridge from the card, or via the multicast stream. Using this data the server accepts TCP connections from clients who need to request packets to be resent based on gaps they have detected in the stream. Once the TCP request is received, if the Gap Server is able to resend the packet(s), the needed packets will be resent on a separate dedicated multicast channel for each outbound channel from the FPGA card. The multicast channel used for the Gap Recovery will always be the same as the one used by the FPGA publisher shifted by a configurable address policy.

The packet(s) resent are identical to the original packets sent out and can be processed in exactly the same way: the Gap Recovery server provides the same service whatever the format used to publish the normalized data (either the RWF format or the flat wire format).

The Gap Recovery server works on a concept of a sliding window, the size of which can be configured by the client. What this means is that it will keep in a buffer a number of packets from each multicast channel published by the FPGA card in order for clients to be able to request these messages. The default size of this window is the previous 1,000 packets, but this size can be configured by clients.

The reason we need to work with a small subset of the data for the Gap Server is in order to ensure the response times and performance of the server when we are dealing with millions of messages across many different multicast channels over the day. In the event that the Gap Server is overloaded, or is not responsive as quick as needed, the first place to look is to reduce the size of the buffered window for recovery.

The interaction with the Gap Server is as follows.

1. The client detects a gap in the Packet Level sequence number from the real time stream on a specific multicast group.
2. The client establishes a TCP connection to the Gap Recovery Server and logs into the service. This can also be done at start up and left as an active connection open if needed.
3. The client then formats a request for the missing data. This request must contain the details of the multicast group the gap is needed from and the sequence numbers of the missing packets.
4. The Gap Recovery Server will respond to the request from the client with one of the following responses:
    a. Request Successful
    b. Request Failed – if the messages requested are not available, or if the request is invalid.
5. If the request was successful, the requested packet(s) are sent out on the matching multicast channel for the requested channel.

NOTE:

The Gap Recovery Server uses a custom domain for all publication of data for recovery. This is defined in the UPA API as "Gap Request". A detailed example of how the Gap Recovery process works from the UPA API side can be found in the examples included with the API.

SNAPSHOT SERVER

The Snapshot Server allows clients at any point in the day to get an immediate picture of the current state of the order book for an instrument.

The snapshot server works based on clients establishing a TCP connection to the server and requesting a snapshot using the instrument identifier from the real time stream to identify the instrument. There is no streaming service available from this server. It is a pure refresh based on the request. The request for the snapshot must contain the Service ID from the Source Directory. The snapshot server provides the same service whatever the format used to publish the normalized data (either the RWF format or the flat wire format).

A snapshot for the market by order/market by price domain will return a full list of orders/price levels on the book for the instrument.

The snapshot server will respond over the same TCP connection with one of these responses.

1. ***Error*** – in the case of an unknown instrument, or an issue with the server itself.
2. ***Empty Snapshot*** – this will be an OMM Map, but with no entries to signify that the system is up and running, but that the requested instrument has no open orders/price levels on the book. This will be a single message which ends the snapshot.
3. ***Successful Snapshot*** – this can be split over multiple messages if the book is large. The final message of the refresh is marked to indicate the end of the refresh.

The Snapshot message received from the server will also contain the instrument level sequence number from the point at which the snapshot was taken in order to ensure clients can sync the snapshot up to the real time stream of traffic. Clients requesting snapshots should buffer the real time stream and start processing the stream after the snapshot from the sequence number immediately following the one sent in the snapshot.

For example, if the snapshot contained sequence number 10, the client should start processing the real time stream of data starting with sequence number 11 and on as all updates up to and including sequence number 10 are contained in the snapshot.

The Snapshot server is also responsible for delivering clients a list of all available instruments from the EDF server.

Details of how this works can be found in the UPA example programs.

REFERENCE DATA SERVER

The reference data server is provided to clients in order to deliver all static and configuration data from the Elektron Direct Feed. There are two services provided from this server, the ***Source Directory / Multicast List*** and the ***Dictionaries***.

Source Directory / Multicast List

As described in the RDM Usage Guide contained in the UPA distribution, the Source Directory is intended to give clients all the information they need about the services available from the server. The implementation of this in the Elektron Direct Feed is intended to give clients a single point of connection in order to discover all the other services available from the solution.

The Source Directory is requested from the reference data server, and will give consumers information on all of the following:

- Which feeds are available from the server (depending on the number of FPGA cards configured). Each feed are assigned a Service ID in the source directory used to identify the specific feed in the other services.
- The multicast channels published for each of the different feeds, including the gap recovery channels.
- The port and IP of the Gap Recovery server
- The port and IP of the Snapshot Server

As any additional services or functionality if added to the Elektron Direct Feed, the intention is that this will always be discoverable via the Source Directory.

Dictionaries

There are multiple dictionaries available from the Elektron Direct Feed, these include the standard ***RDMFieldDictionary*** including the ***enumtype.def***, but also includes all the ***Set Definitions*** needed to be able to interpret the data from the real time stream.

The request for the dictionary data requires a Service ID in order to identify which feed the dictionaries are being requested for. Every service will return the RDMFieldDictionary and the Enum Dictionary as well as the Set Definitions for the specific feed in question.

Dictionaries are used only when normalized data is published using the RWF format and consumed via the API UPA.

EXCHANGE RECOVERY

The exchange recovery component of the solution is important for clients that want to make sure the solution is fully resilient in the case of dropped messages from the exchange market data feed. In the event that the FPGA card detects a gap from the exchange that it cannot recover itself (for example via the A/B multicast arbitrage), it will send a message to the exchange recovery server to indicate there has been a gap in the data.

The edf-recovery service is a software application, processing the same feed as the card. The raw exchange feed received by the card goes through the line arbitration logic, and then is piped through the PCIe bus to the edf-recovery service in parallel of being processed by the EDF business logic running on the FPGA. The edf-recovery maintains a 'golden' copy of the book state. If and when a gap occurs in the feed, it buffers data, requests the exchange for the gap, then processes all packets in order. After a gap has been successfully recovered, edf-recovery queries the snapshot service for all impacted symbols applies the following business logic, computes the delta between its own golden' copy of the book state and the state retrieved through snapshot service, which mirrors the state of the card and applies correction messages (add order / delete order / modify order to the card via the PCIe. The resulting correction messages are then processed by the card and published via RWF to the wire.

The recovery logic is differential: whenever multiple consecutive order modifications have been lost, only one correction message is triggered by edf-recovery. The intermediate state of that order are not recovered.

At this point a few things happen:

- Depending on the venue details, the exchange recovery server will send information to the FPGA card marking a set of instruments as ***Stale*** while the data from the exchange is recovered. The range of instruments set stale when information is lost is dependent on the venue and will be covered in the Venue Addendum for the venue.
- The Exchange Recovery component will then use the mechanisms provided by the venue to attempt recovery of the information from the exchange.
- Once this recovery completes, the Exchange Recovery component will compare the updates it has recovered from the exchange with the current state of the order book for any affected instruments. Any updates needed to the book will be sent to the FPGA card in order for these to be sent out as part of the real time stream. There are three different updates that can be sent out as part of recovery.
    - Addition of a new order/level - will be added with the current price/quantity at the time of it be re-inserted. Any modifications prior to this will be rolled into the one update.
    - Deletion of an order/level
    - Modification of an order/level - as with the order addition, the current state of the book at the time of the update being sent is what is communicated. Any previous states of the order between the data being lost, and the current state of the book are not sent.
- Once all updates are sent, and the book is in the correct state again, stale flags will be cleared.

The key goal of the recovery logic is to correct the order book for consumers. The recovery logic will **NOT** provide full recovery of every single tick from the exchange. It will recover all trades into the Market Price domain, but will **NOT** put all of these trades into the ***Market by Order/Market by Price*** domain. These domains will receive simple modifications to bring the books back into the correct state.

TREP BRIDGE

The Elektron Direct Feed is fully integrated with the Thomson Reuters Enterprise Platform TREP thanks to the TREP Bridge. Note that the TREP Bridge is available for OPRA only for the time being.

MONITORING

The Elektron Direct Feed comes complete with an ITRS Geneos probe built-in to the solution to allow easy monitoring of the solution using their Geneos monitoring application. Monitoring is possible using basic network monitoring statistics from the FPGA cards and the NIC's on the server, but there is also dedicated logic in the solution to provide detailed statistics on the FPGA and software components for easy monitoring by clients.

LICENSING

All instances of the Elektron Direct Feed require a license key from the Thomson Reuters Enterprise License Server in order to start. All of the software components are controlled by this key, and will not run without valid keys for each instance of the product running on the server. The default location and file name for the license key file downloaded from is the ***$EDF_HOME/config/TR_LICENSE***

If the license cannot be found in this directory, the reference data server will not publish any information for the unlicensed feeds, even if they are configured. This will impact the Source Directory as well as the Symbol List and the Dictionary process (for the Set Definitions).

There is a detailed description of how to acquire licenses, and install them on the server in the Configuration and Install Guide for the Elektron Direct Feed.

OPERATION

Manager daemon

All software services in Elektron Direct Feed are operated through a manager daemon. The manager daemon can start and stop processes, and act as a watchdog to ensure processes stay up and running. It can send an alert when any process unexpectedly stops, and optionally try to restart it.

Logging/Log Daemon

All software services in Elektron Direct Feed are configured to write application log messages through a logging daemon. Processes using the Logger Daemon write their logging information in shared memory. The Logger Daemon reads the shared memory, collects and formats the logs, and writes the formatted log messages to a pool of files. The size and number of log files in the pool is set at startup to avoid disk space issues during runtime.

The Logger Daemon infrastructure therefore provides a simple and homogeneous logging approach across applications. If the Logger Daemon is down for any reason, the processes will write their logs into default logging files. The Logger Daemon is automatically started when the manager Daemon starts.

DATA FORMATS

RWF FORMAT

The Elektron Direct Feed presents data from multiple global venues into a standard representation which uses a subset of the Elektron Real Time data models. The feed is consumable using the API UPA. There are two domains of data provided from every Elektron Direct Feed instance.

- ***Market by Order / Market by Price***
- ***Market Price***

The Market Price domain is always provided, Market by Order or Market by Price are provided based on the feed from the exchange. If the exchange provides order level data, the feed will provide Market by Order data; otherwise it will provide Market by Price data.

The content of the Market by Order / Market by Price feed is focused on providing all details needed to maintain an order book for trading. The Market Price domain will provide all non-order book related information.

General examples of how the data is divided:

- Auction updates/status information will be published on the Market Price domain.
- Order book addition/modification/deletion – only published on the MbO/MbP domain.
- Executions of orders in the order book – an order update is published on the MbO/MbP domain indicating a trade has occurred, and the remaining quantity of the order left on the book. The full details of the trade (flags, trade reference, other information provided by the venue) are published on the Market Price domain.
- Executions of orders not visible in the order book – only published on the Market Price domain.
- Order book status messages (symbol trade state for example) – published in both domains.

FLAT WIRE FORMAT

The content of normalized market data using the flat wire format is focused on providing all details needed to maintain an order book for trading. The feed is consumable directly from the wire without need for an API. Each message is built on the following principles:

- The necessary data to build the order book is provided by the core payload of the message
- An extension is appended to the core message to provide additional data

If the exchange provides order level data, the feed will provide Market by Order data; otherwise it will provide Market by Price data. The flat wire format provides:

- Order book updates – only published for order-based feeds
- Level updates – only published for level-based feeds
- Auctions data
- Trading status
- Trades
- Statistics
- Symbol reference – used to provide referential data

- Venue Specific data – used to provide specific data that cannot be normalized
- Service message – used particularly to inform data consumers when an exchange recovery is in progress

See the EDF flat wire format specifications for details.

PROCESSING THE DATA

In order to maximize the throughput of data from the FPGA card, deliberate decisions have been made as to the format of what is being published.

Global Set Definitions

In order to maximize the value of the data on the wire, and minimize the amount of data having to be passed on the network, we have used Global Set Definitions as defined in the RDM Usage Guide. What this means is that in order to process the messages from the Elektron Direct Feed, a set of Global Set Definitions for each feed must be loaded from the Dictionary domain and used to process the messages. Each message sent on the feed contains a clear Set Definition reference that is used to parse that data sent from the feed and assign FID's to the values sent. This is all taken care of by the standard RWF Decoders in UPA, but the Set Definitions need to be explicitly requested and loaded from the Reference Data server for each feed being consumed. Examples of how to do this are in the example programs included with the UPA distribution.

Global Set Definitions are used only when normalized data is published using the RWF format and consumed via the API UPA.

Symbol Mapping

In order to support the maximum number of symbol mappings and information available from the venues, all symbol related data can be retrieved from the Symbol List domain on the Reference Data Server, and the only instrument identification found on the real time stream is an index value which identifies which item in the symbol list for that venue the data applies to. The client should pull in this symbol list and map the information as needed depending on how the client needs to identify the instrument internally.

The Elektron Direct Feed aims to pull together a number of different sources of exchange static data into a single presentation through the Symbol List domain. In order to do this the system is set up to consume data from the exchange using a variety of methods.

- ***Real Time Messages*** – usually from a morning static data load, or intraday changes
- ***FTP/Static Files*** – provided by venues either from a dedicated FTP or the web.

The Elektron Direct Feed will pull these different sources together, depending on the venue, and create a Symbol List message to represent the data.

The specifics of how each venue will support this will be documented in the Venue Supplement for the venue.

Sequence Numbers

Every message sent from the Elektron Direct Feed has two different sequence numbers assigned by the solution as well as the exchange provided sequence number. Each of these has specific uses.

- ***Packet Level sequence number*** – this is used to detect gaps in the feed itself and is used in relation to the Gap Recovery server.
- ***Instrument and Domain specific sequence number*** – Each instrument on each domain has an incrementing sequence number for each message assigned. This sequence number is the one referenced by the Snapshot Server when it references the live feed at which point the snapshot was taken.

- ***Exchange sequence Number*** – this is valuable information for latency monitoring or other operations that rely on matching the exchange message with the update from the Elektron Direct Feed.

Timestamps

Currently, the solution provides a single timestamp based on the exchange. We will send nanosecond granularity of timestamps where this is supported by the venue. The expectation is that latency monitoring or other solutions can be used outside the solution itself to track a message through the solution.

This has been done to minimize the overhead of the additional space on every message to add additional timestamp information.

MONITORING

LATENCY MONITORING

In order to allow clients to monitor the latency of the solution it is critical that clients are able to match the updates from the Elektron Direct Feed up with the messages coming through from the exchange. As the wire format of the messages coming from the Elektron Direct Feed is not an open format, below we provide details on how clients can configure their applications to be able to match the incoming data from the exchange with the published updates from the Elektron Direct Feed.

The basic format of the messages out of the Elektron Direct Feed has a fixed length for each message, and we populate the Exchange Sequence Number directly into each message sent out which will give clients the ability to match an update from the exchange to an update from the Elektron Direct Feed.

For more details on how latency testing has been performed during the setup and testing of the Elektron Direct Feed please refer to any of the Elektron Direct Feed Performance Test documents for the different venues.

ITRS GENEOS MONITORING

The Elektron Direct Feed offers built in integration with the ITRS Geneos monitoring solution. This is done by including an ITRS Netprobe in the solution, as well as providing an implementation of the RMC component as used in the Thomson Reuters Enterprise Platform to expose information about all the pieces of the system.

There are a large number of statistics and data available through the standard ITRS integration, as well as through the custom components that are part of the Elektron Direct Feed solution.

For more information
Send us a sales enquiry at
thomsonreuters.com/about/contact_us
Read more about our products at
thomsonreuters.com/products_services
Find out how to contact your local office
thomsonreuters.com/about/locations

THOMSON REUTERS™

Document Version 1.4
Date of issue: *06 October 2015*~~30 September 2015~~