Celoxica
pure acceleration

UltraBook UB

Product Overview

About This Document

This document describes the UB product overview.

Copyright Information

Sales	sales@celoxica.com
Customer Support	support@celoxica.com
Website	http://www.celoxica.com

UK Head Office
Celoxica Limited
34 Porchester Road
London
W2 6ES, UK
Phone: +44 (0) 20 7313 3180

US Head Office
Celoxica Inc.
275 Madison Avenue, Suite 404
New York, NY
10016, USA
Phone: +1 (0) 212 880 2075

US Chicago Office
Celoxica Inc.
141 W Jackson Blvd, Suite 2350
Chicago, IL
60604, USA
Phone: +1 (0) 312 893 1204

Content

Table of Figures

Revisions

Revision	Date	Description of Changes
R2013-1.0	28 DEC 2012	Initial version

1. Functional Description

1.1 Principles

Celoxica's generic and data-normalized UltraBook UB is an FPGA-based solution providing processed market data stored in a book and high-performance communication with a client software application.

UB normalizes order-driven and book-driven feeds and provides a unified interface for reading orders for different instruments. It supports multiple markets and multiple levels depth.

See the UB user guide for more details.

1.2 Order Book Management

1.2.1 Symbology

UB provides a flexible symbology, where every instrument can be addressed in multiple ways.

Currently UB needs to determine the mapping between symbol names and indexes from the reference data received directly from the feed.

UB supports several identifier schemes to uniquely identify an instrument. The availability of the schemes per instrument depends on the market.

1.2.2 Instrument Static Information

UB provides instrument industry standard static and normalized information:

- The instrument identifiers depending on the available identifier schemes
- The ISO 10962 CFI code
- The instrument type as provided by the exchange

UB also provides raw referential data as provided by the exchange.

UB automatically registers all referential data received from the exchange. The user subscribes to an instrument to get levels, trades, prices and imbalances. UB will only process market data related to a symbol if the user subscribes to that symbol.

1.2.3 Levels

The order book is maintained from level 0 (top of book) to level n where n <=4 and provides the bid and offer quantity per level.

If an exchange provides different quantity types such as customer or all or none quantity type, UB will maintain as many order books as there are quantity types available.

1.2.4 Trades

UB provides the trades received from the exchange when available.

1.2.5 Prices

UB provides the order book OHLC (Open / High / Low / Close) and settlement prices information when available.

1.2.6 Imbalances

UB provides the order book imbalance information when available.

1.2.7 Trading Status

The security and market trading statuses generic forms are provided when available. Note that such statuses often vary significantly between feeds, so it is not always possible to provide a full and generic mapping. UB therefore provides market specific statuses for the user's reference.

UB automatically registers all instrument trading statuses received from the exchange even for non-subscribed instruments; as a consequence, if the user subscribes to an instrument intra-day, he will get the correct instrument trading status.

1.2.8 Integrity

The instrument and market integrity is also made available to the client application. This integrity status notifies the state of the instrument book and the market:

- **Clean**

There is no error, no data loss.

- **Best-effort**

  The instrument book / market information is probably not accurate because of gaps and data loss; a recovery may be in progress to update the order book / market integrity.

- **Stalled**

  The instrument book / market information flow has stalled e.g. one of the multicasts on the market has timed out.

This feature is partially supported:

- The instrument integrity is always clean
- The market integrity is set clean at startup, becomes best-effort in case of a sequence gap; however, no recovery process is implemented yet.

1.3 Configuration

UB is configured using a configuration file which implements a configuration tree upon UB initialization.

The configuration tree allows UB users to set two configuration parameters types:

- **Global settings**

  UB process settings …

- **Market settings**

  Specific market parameters, live multicasts …

1.4 Filtering

Software filtering is natively implemented in UB for all feeds as UB automatically filters the symbols based on the subscription.

Note:

UB does not support hardware symbol filtering yet.

1.5 Recovery

Recovery is handled by UB automatically. Books are always initiated with best-effort status, so UB detects this and triggers the snapshot recovery process. During the snapshot process, the user is presented with all updates received from the market even if the book is in best-effort status. Once the snapshot has been processed successfully, the book switches to clean status. If a gap is detected the book switches back to best-effort status and the snapshot process is triggered again.

The following types of recovery are or will be available:

- **Natural refresh**

  The order book is built using only live data. This provides a quick and easy way of order book refresh.

  This feature is supported.

- **Snapshot recovery**

  The picture of the order book is refreshed using market snapshot (when available). This is useful for mid-day starts and for non-liquid instruments.

  This feature is not supported yet.

- **Gap skip recovery**

  The order book may be crossed when some data are missed. UB is able to cope with such cases by using the uncrossing mechanism: this is a scheme whereby crossing entries are deleted from the book automatically.

  This feature is not supported yet.

1.6 Shared Memory API

1.6.1 Introduction

UB handles the market data from the exchanges and writes the appropriate market events to shared memory. The shared memory is made accessible to the client application which can therefore read it via a shared memory API.

UB has been designed so that the shared memory order book can be accessed by multiple user readers – multiple threads - as illustrated below.

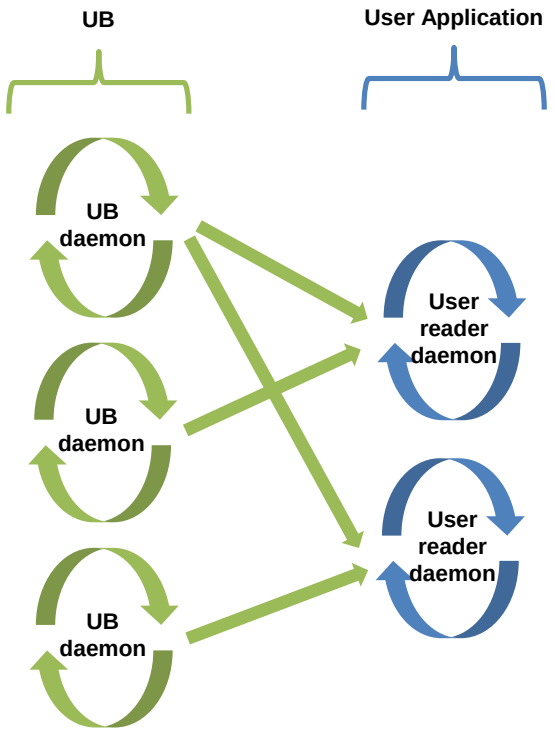

Figure 1 – Multiple user readers

The client application gets access to:

- **Order book tables**

  A complete order book per symbol is made available through shared memory. This API is an easy and quick way for the client application to get access to order books.

  The user gets the most recent live information updates by using this API.

- **Update queue per reader**

  The client application defines a set of reader slots from which it is notified if there was a change in the order book. These notifications do not accumulate and are cleared when the user reads the change list.

  The change list contains flags about what has changed in the book. Change flags can stack up over time in the notification list however the order of the changes is not maintained.

  Several types of changes in the order book are flagged:

  - Top of book (level 0) change per quantity type
  - Levels other than top of book change per quantity type
  - Trades change
  - Prices change
  - Imbalances change
  - Instrument Status change
  - Market Status change
  - Instrument integrity
  - Market integrity

  The client application creates views which provide the last live information updates on an order book in shared memory.

- **Symbol directory**

The client application reads the symbol directory to get static information of instruments.

The client application is able to retrieve the most up-to-date exchange specific referential data as well as standard and normalized instrument characteristics.

Order books for regulatory and consolidated feeds have the same client API presentation as primary feeds i.e. a separate order book per venue.

1.6.2 Usage

UB automatically processes the data coming from the market feeds. It builds order books for instruments for which there is at least one subscription initiated. The use of the API does not affect the UB internal operation.

1.6.2.1 Instrument Pre-Subscription and Subscription

When an instrument is created, the API tries to find it in the instruments already referenced by UB (instruments for which referential data have been received from the exchange) to subscribe to it.

The user has the ability of anticipating an instrument creation by forcing the subscription to that instrument: a forced subscription is a pre-subscription.

The pre-subscription and subscription mechanism works as follows:

1. If the instrument is referenced by UB, the subscription will be activated and the user will be able to create views with that instrument.
2. If the instrument is not referenced by UB yet and the instrument creation is forced, the user will pre-subscribe to the instrument. All views created with a pre-subscribed instrument will also be considered as pre-subscribed. As new instruments may be added intra-day, the user is responsible for changing the instruments and views pre-subscriptions into subscriptions.

1.6.2.2 View Changes Notifications

The API gives the user the ability to access the selected order books. It is possible to iteratively monitor the views by using the access methods … The output of these methods is a current snapshot.

Order book changes are flagged using the change flags provided by UB. Changes for a view are monitored using the change flags on a per view basis. UB notifies the user when the change flags are first set.

Then the user can obtain the change flags by calling a specific method. This method returns the change flags and then clears them. No further notifications occur until the change flags have been cleared using this method.

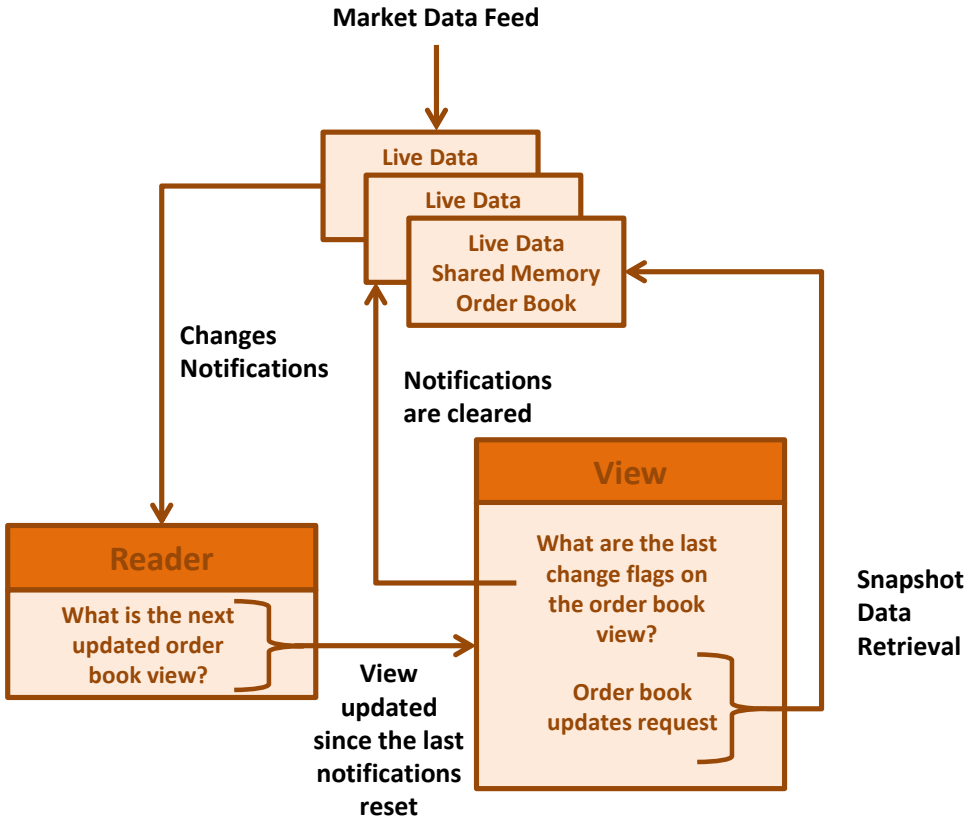

Figure 2 – View changes notifications

1.6.2.3 Market Changes Notifications

The API gives the user the ability to monitor the markets. The output of these methods is a current snapshot.

Market changes are flagged using the change flags provided by UB. Changes for a market are monitored using the change flags on a per market basis. UB notifies the user when the change flags are first set.

Then the user can obtain the change flags by calling a specific method. This method returns the change flags and then clears them. No further notifications occur until the change flags have been cleared using this method.

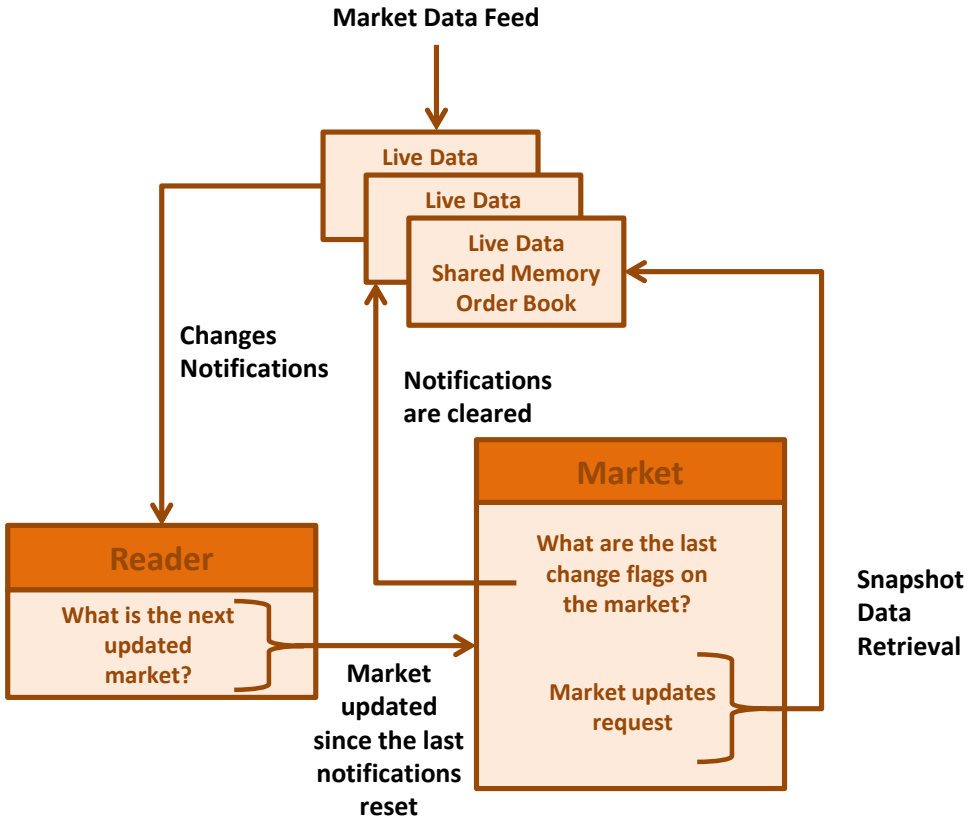

Figure 3 – Market changes notifications

2. Technical Architecture

2.1 Principles

The Celoxica application daemons, especially the UB daemon, are monitored by the Celoxica Manager.

The Celoxica Manager is a Python script which allows the user to monitor and parameterize the following processes:

1. **Master Daemon**

   The Master Daemon or the Manager Daemon is responsible for monitoring all the other Celoxica processes and switching to the backup node in case of failure on the master node.

2. **Logger Daemon**

   This process is responsible for recording application logs asynchronously.

3. **Celoxica Application Daemons**

   The following application daemons are monitored by the Manager:

   a) UltraBook (UB)

   b) FIX Market Access Gateway (MAG)

   c) Generic eXecution Accelerator (GXA Lite)

The Manager provides commands that allow the user to monitor all of these processes.

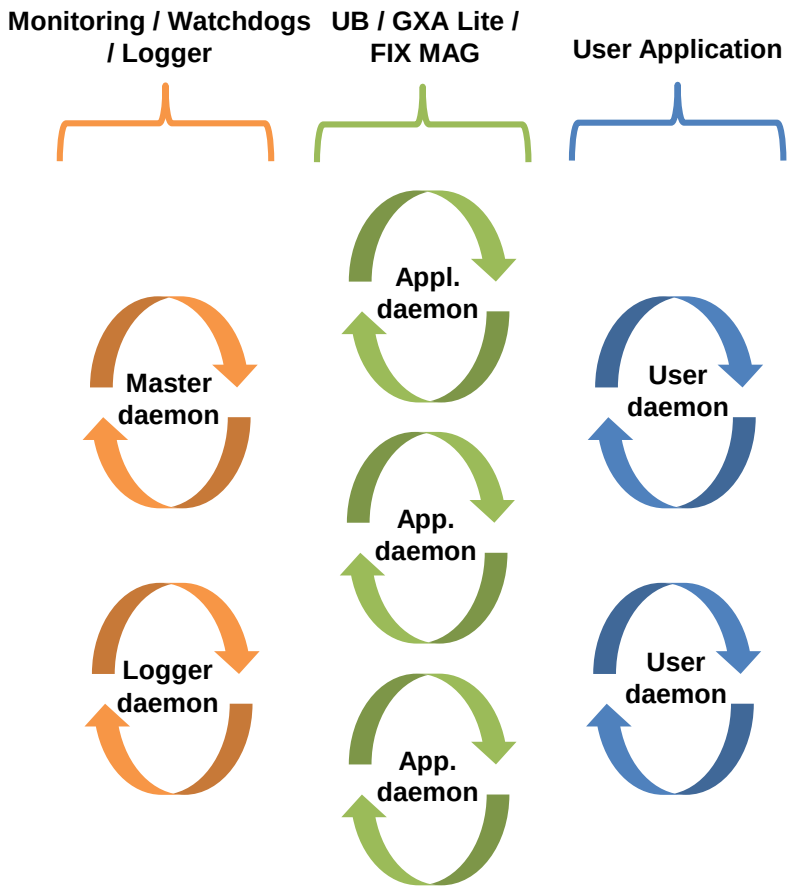

Figure 4 – Celoxica Manager daemon

See the Celoxica Manager user guide for more details.

2.2 Master Daemon

The Master Daemon can start and stop processes, acts as a watchdog and sends alerts in case of issues (e.g. process down or restarted).

It is responsible for ensuring and maintaining system availability, and is also able to manage failover to a backup node – note that this feature is not available yet.

The list of processes to watch and actions are fully configurable, for instance:

- Automatic process restart

  When a process is stopped, the Master Daemon tries to restart the process. The number of restart attempts and the timespan during which the Master Daemon tries to restart the process are configurable.

- Stop watching a process after n restart attempts

  If the Master Daemon fails to restart a process, the process will cease to be watched.

2.3 Logger Daemon

The processes which use 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 files. Log messages are coloured according to the log level. The log size is set at startup to avoid disk space issues during runtime. The number of log files is configurable and the logs can be written to specific files according to filtering expressions.

The Logger Daemon 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 Master Daemon starts.

2.4 UB Daemon Monitoring

UB is monitored by the Master Daemon.

UB uses the Logger Daemon.

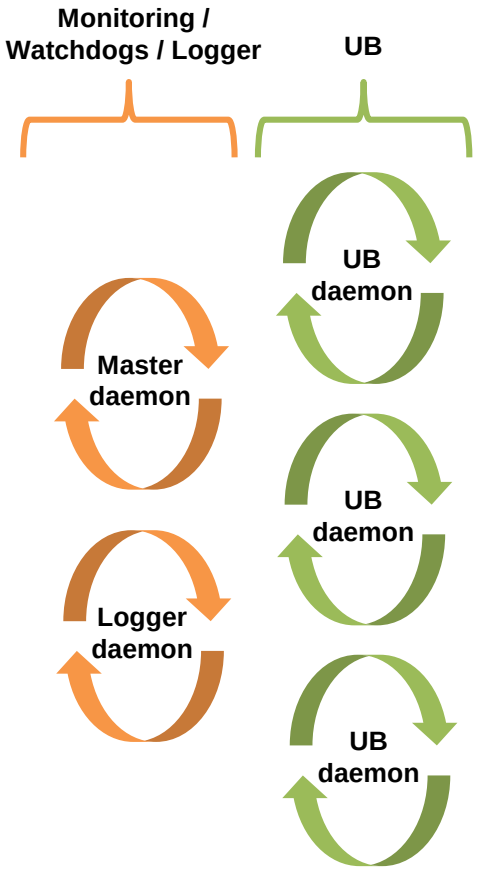

Figure 5 – UB daemon

2.5 Shared Memory Allocation

The shared memory is represented as a file. The size of this file naturally determines the maximum size of the shared memory segment(s) that can be allocated for UB.

The shared memory must be allocated before starting the UB daemon using the “ultrabook-manage-memory” tool.

This tool is monitored by the Master Daemon and uses the Logger Daemon.