



Gateway Development Kit (Native)

14 Mar 2013

0 Gateway Development Kit (Native)	4
1. Introduction	5
2. Document Overview	6
3. Service Architecture Concepts	8
3.1.0 Orc Software Architecture Overview	9
3.2.0 Client and Server Components, Roles, and High-Level Data Flow	10
3.3.0 Custom SD Gateway Development Overview	13
3.4.0 Services for a Single Market Provided by Multiple Processes	17
4. Service Definitions	18
4.1.0 XML Files	19
4.2.0 XML File Layout and Details	20
4.3.0 Message Behavior	29
4.4.0 SD Message Header	30
5. SDOFF	31
5.1.0 DOFF Wire Protocol	32
5.2.0 SDOFF - SD protocol over DOFF	36
5.3.0 Encode Decode Example Code	40
5.4.0 FIX Fields	46
6. Initial Handshakes and Login	47
6.1.0 Server Registration with Environment Manager (EM)	48
6.2.0 Client Service Discovery	50
6.3.0 Establishing the TCP IP Connection	51
6.4.0 Login and Logout	52
6.5.0 Disabling DOFF Compression in Orc Trader	53
7. Instrument Service	54
7.1.0 Instrument Service Behavior	55
7.2.0 Creating CFICode Using ISO 10962	63
8. Market Data Service	84
8.1.0 Market Data Service Behavior	85
9. Order Service	89
9.1.0 Order Service Behavior	90
10. Trade Report Service	98
10.1.0 Trade Report Service Behavior	99
11. Quote Service	102
12. MassQuote Service	103
13. Orc Trader Custom Order Panel	104
14. Development Tools	115
14.1.0 MGF Test Gateway	116
14.2.0 SD Command Line Based Test Client	117

14.3.0 Orc Trader and Liquidator Debug Logs	121
14.4.0 BufferDumper Hex to SD Parser	122
15. End2End Metrics	125
15.1.0 End2End Metrics Concepts	126
15.2.0 End2End Reporter	129
16. Orc System Integration	143
17. Support	145

Gateway Development Kit (Native)

© 1999-2013 Orc Group, all rights reserved. Orc® and the Orc logo are trademarks of Orc Group. Orc Group is a trade name of Orc Group.

Documentation is provided "as is" and all express or implied conditions, representations and warranties, including any implied warranty of merchantability, fitness for a particular purpose or non-infringement, are disclaimed, except to the extent that such disclaimers are held to be legally invalid.

Third-party software components:

ASM

Project License

Copyright (c) 2000-2005 INRIA, France Telecom

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Introduction

Purpose

To explain the SDOFF protocol for the purpose of writing client or server processes that operate in the Orc Services environment.

Intended audience

Developers - both internal to Orc and external customers.

History of SDOFF

SDOFF development started around 2008. It has been in production use since 2009 (starting from Orc TS-9.0.8, but TS-9.0.13 or later is highly recommended).

It is the proprietary internal protocol used between Orc clients (Orc Trader, Orc Liquidator, Orc Market Maker, Orc Spreader, etc.) and Orc servers (market gateways). It is intended for low latency high throughput usage.

SDOFF is replacing DOFF. DOFF simply prescribed encoding/decoding and transport, leaving the message structure and behavior to the individual market gateway developer. SDOFF extends DOFF by adding a data model, specific message structures, and behavior.

Currently, there are about 40 Orc market gateways that are provided as services. However, all new Orc market gateways are rolled out this way. As time progresses, more and more existing markets are service-enabled.

Customer application integration with SDOFF

Orc is offering SDOFF integration to its customers to allow for tightly coupled native connectivity of custom trading applications and specialized market gateways.

Other ways of integrating with Orc

Other ways of integrating applications also exist. A complete list is included below:

- Market Access (transactions and market data): GDK-Native, GDK-FIX, Orc Trade Access
- System Integration (execution report, risk, theoretical pricing): Orc Protocol
- Parameter Control (offsets, vol, interest rates, dividends)
- Calculations: PMAPI, VMAPI
- Trading Strategies: Liquidator
- Liquidator Control: JMX, Liquidator Protocol

[Document Overview](#) >

Document Overview

This document contains the following sections:

- [Service Architecture Concepts](#): This section provides an overview of the Orc software architecture. It describes the various client and server components, their roles, and a high-level data flow. It also outlines the approach for a custom SD gateway development project.
- [Service Definitions](#): The Orc Service Definitions are defined in a set of XML files. This section provides an explanation for the general layout and structure of these files. A quick introduction to the behavior of request/response and subscription models is also provided
- [SDOFF](#): This section describes SDOFF, which is the SD protocol over the classic Orc DOFF wire protocol. SDOFF is used by all the services described in later sections.
- [Initial Handshakes and Login](#): Regardless of the services being provided, each gateway must provide basic login/logout functionality to support client connectivity.
- [Instrument Service](#): The Instrument Service is used to provide clients with instruments for an exchange. It supports start-of-day snapshots as well as intraday updates to instruments. This service is typically provided along with the Market Data Service by the same gateway, since instruments and price feeds are typically provided by the same interface from an exchange.
- [Market Data Service](#): The Market Data Service is used to provide clients with BBOs (best bid & offer), depth of market, public trades, and news for an exchange. This service is typically provided along with the Instrument Service by the same gateway, since instruments and price feeds are typically provided by the same interface from an exchange.
- [Order Service](#): The Order Service is used to enter, modify, and cancel orders, as well as to receive trades from those orders. It is also used for Request For Quote (RFQ) and instrument creation functionality.
- [Trade Report Service](#): The Trade Reporting Service is used to send and receive off-exchange trades. (It is NOT used to report trades from orders entered through the Order Service. Trades for those orders are reported on the Order Service.)
- [Quote Service](#): The Quote Service is used for market making functionality. The client sends the gateway one or more double-sided quote messages, followed by a flush message. The gateway then packs these quotes into a single mass quote message to send to the exchange.
- [MassQuote Service](#): The MassQuote Service is used to send a single MassQuote message from a client to a gateway. It can be used in place of the normal Quote Service if the gateway supports the MassQuote message and functionality. Performance is improved by reducing the number of messages sent and eliminating the flush delay. The MassQuote Service is also assumed to be asynchronous, meaning that a new quote can be sent before the ack for the previous quote has been received
- [Development Tools](#): This section provides details of the various development tools which can be used in creating a custom SD gateway. The tools include a test gateway/simulator, a test client, debug logs, and TCP dump parser.
- [End2End Metrics Concepts](#): End2end metrics is Orc's method of analyzing transaction latency and throughput through the system. The header for each message contains a TransactionID, InTimestamp, and OutTimestamp. Messages are written to files and run through a file analyzer to produce an HTML report with latency statistics and histograms.
- [Support](#): For support during the process of developing a custom SD gateway, please call or email your local Orc support desk. They will be able to assist you directly or forward your request to the appropriate Orc engineering personnel.

In addition to this document, there is also a [GDK-Native Getting Started Guide](#) which walks through some basic functionality like logging into a service, subscribing for TopOfBook, and entering an order.

< [Introduction](#) / [Service Architecture Concepts](#) >

Service Architecture Concepts

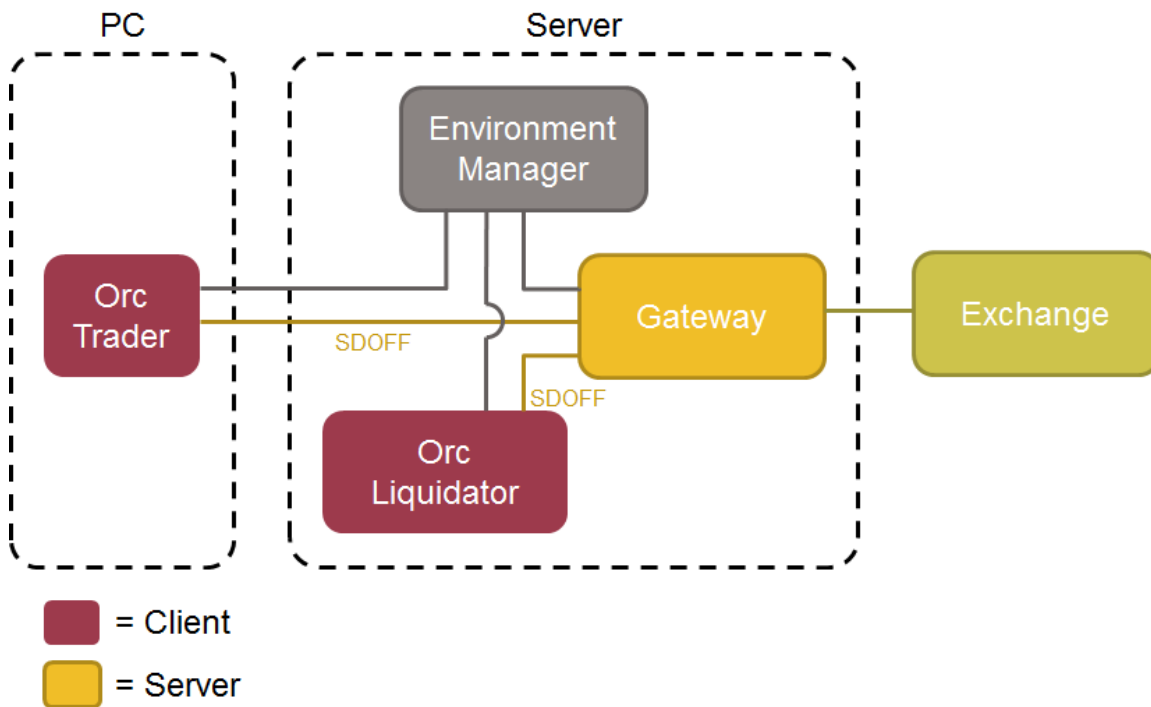
This section provides an overview of the Orc software architecture. It describes the various client and server components, their roles, and a high-level data flow. It also outlines the approach for a custom SD gateway development project.

- [Orc Software Architecture Overview](#)
 - [Client and Server Components, Roles, and High-Level Data Flow](#)
 - [Custom SD Gateway Development Overview](#)
 - [Services for a Single Market Provided by Multiple Processes](#)
-

< [Document Overview](#) | [Service Definitions](#) >

Orc Software Architecture Overview

This is a simplified view of the Orc software architecture, from the point of view of a gateway. Other important server-side processes like CDS and Storage are not shown, because a gateway does not communicate with them.



The gateway connects to the exchange over the exchange's proprietary protocol. The gateway connects to the Orc Environment Manager (EM) to make itself available to all clients in the environment, Orc Trader and Orc Liquidator.

Orc Market Maker and Orc Spreader are products based on Orc Liquidator. For the purpose of this documentation, all products based on Liquidator will just be generally referred to as Liquidator.

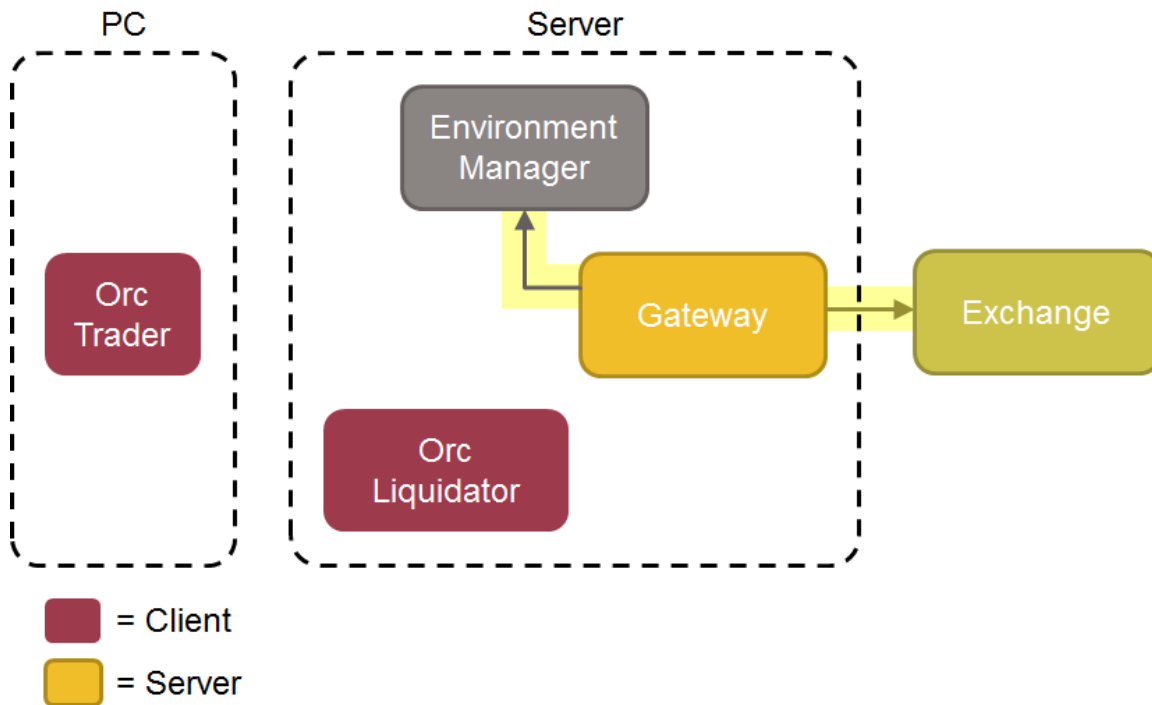
The gateway connects to the clients in the environment via SDOFF. This documentation focuses on the SDOFF protocol, providing the information necessary to write an SDOFF-compatible gateway that can run in the Orc environment.

[Service Architecture Concepts](#)

Client and Server Components, Roles, and High-Level Data Flow

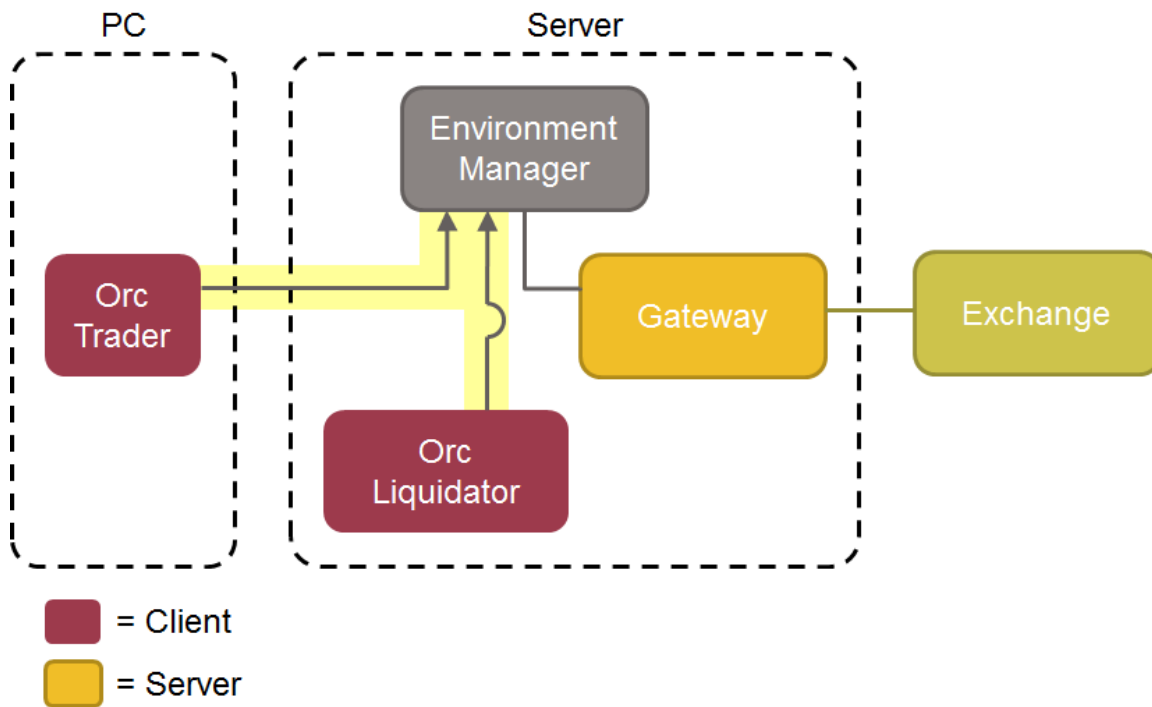
Gateway startup

When a gateway starts up, first it establishes a connection to the exchange. Once that connection is established and running (instruments are downloaded, market data snapshots have been loaded and incremental updates are flowing, and the trade sessions are connected), it registers with the Environment Manager (EM) to let all the clients in the environment know that it is available for client connections.



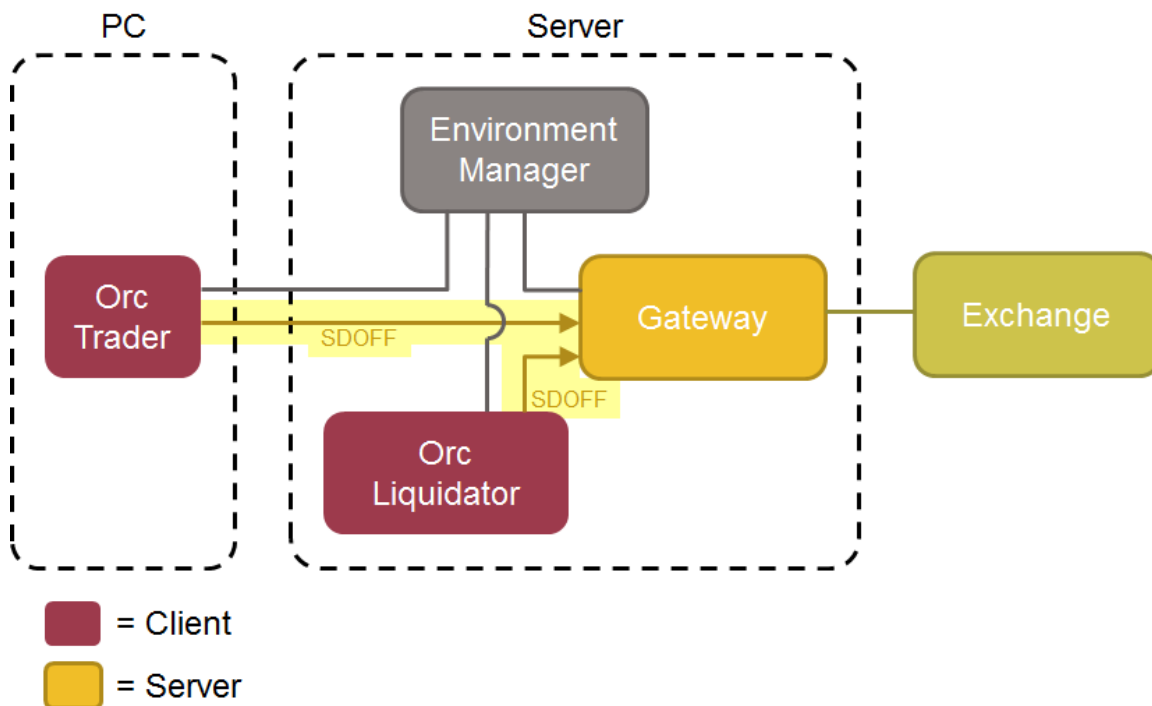
Orc client startup

When an Orc Client (Orc Trader or Orc Liquidator) starts up, it registers with the EM to get a list of all the available gateways.



Communicating via SDOFF

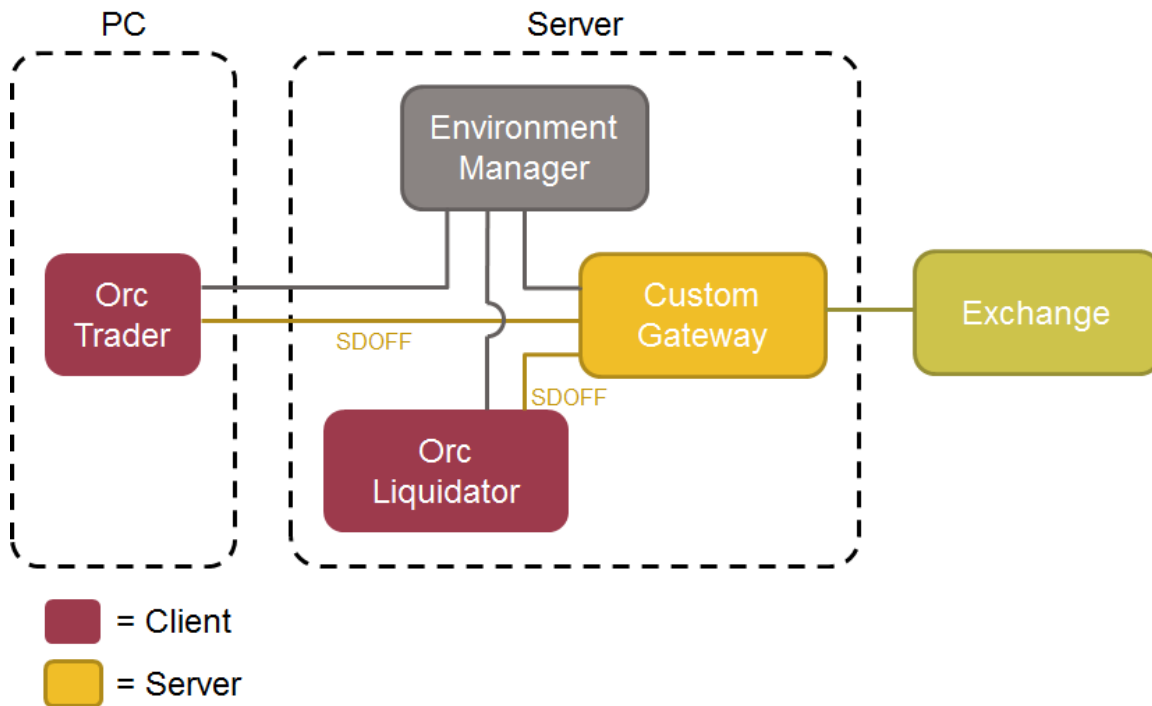
Assuming the gateway has already registered with the EM, the clients will be able to see it and connect to it. The connections to the gateway are established through the SDOFF protocol. And from that point forward, all transactions (orders, trades, mass quotes, etc.) and market data (BBOs, depth, public trades, etc) are communicated via SDOFF between the client and the gateway.



Custom SD Gateway Development Overview

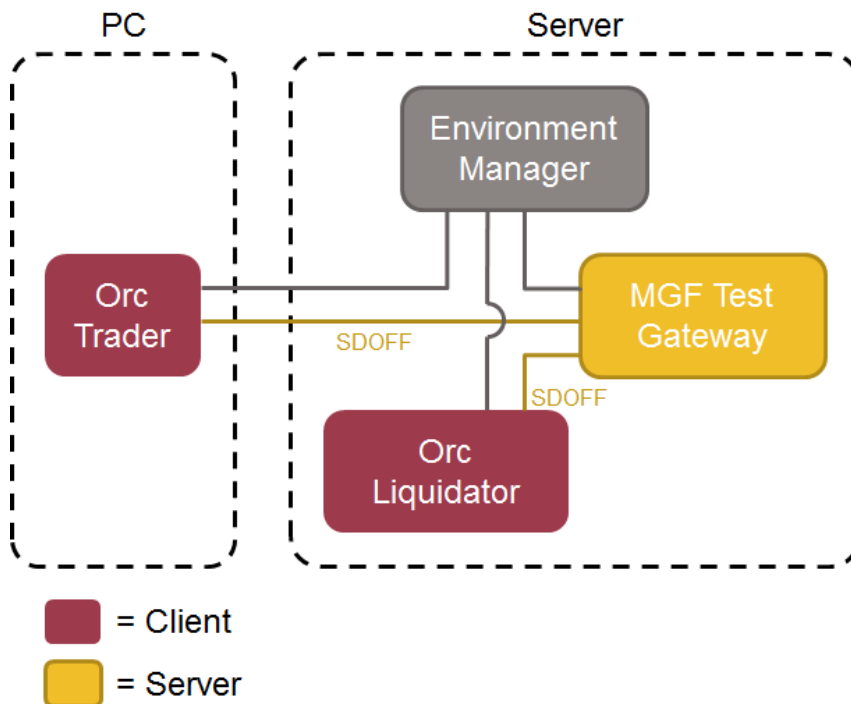
Custom SD gateways

Orc customers can write their own custom SD gateways that connect to an exchange (or broker, dark pool, internal order management system, etc). This custom gateway can provide one or many services to the Orc clients for that exchange/destination: instruments, market data, orders & trades, quotes, or off-exchange trade reporting.



MGF Test Gateway

To assist customers with their gateway development projects, Orc provides a test gateway/simulator called MGF Test.



This gateway does not connect to an external exchange/destination, but instead has a simple internal matching engine.

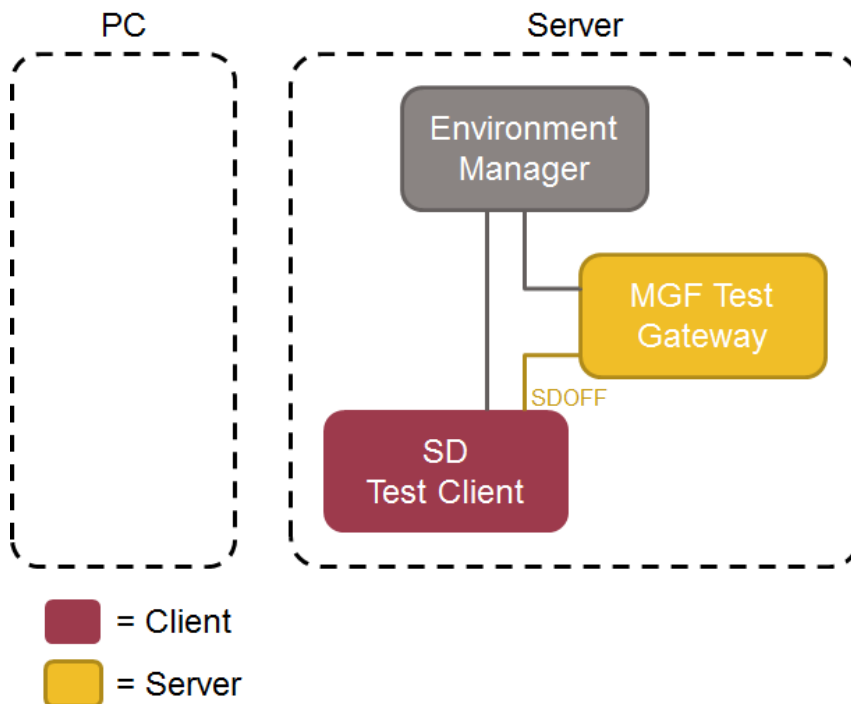
- Sending an order will result in an acknowledgment and a market data message.
- Sending an order on the opposite side will result in an acknowledgment, 2 trades (one for each order submitted), and the appropriate market data messages
- etc.

This gateway is useful for developers to see examples of proper SD gateway behavior without having the added complication of connecting to an exchange. Debug logs can be examined to see the sequence and content of the various SDOFF messages between the clients and the gateway in various scenarios.

See the [MGF Test Gateway](#) section for more details.

SD Test Client

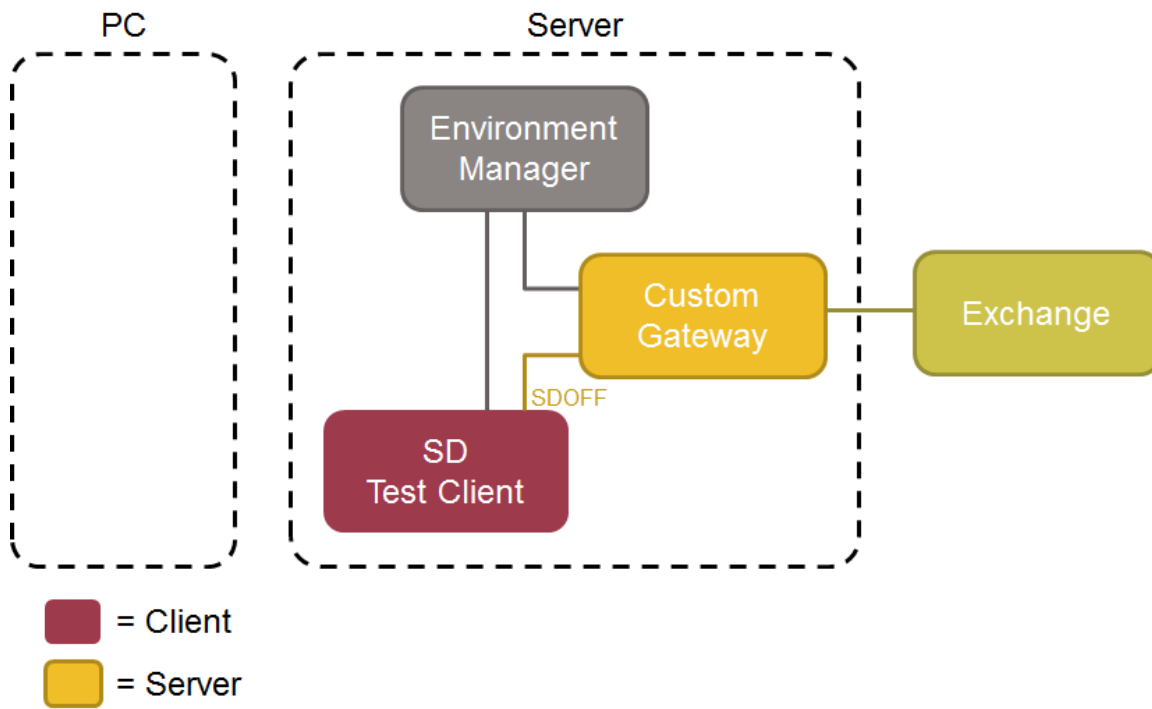
In addition to the MGF Test Gateway, the SD Test Client is a command line based application that can send and receive SDOFF messages. It can communicate with the MGF Test Gateway or any other SD gateway.



It is useful for developers to step through scenarios slowly, with complete control over each message being sent to the gateway. It can be preferable to using Orc Trader and Liquidator during initial development, since those clients sometimes send many messages in quick succession and can be difficult to work with as your development is just getting started.

Debug logs can be examined to see the sequence and content of the various SDOFF messages being sent and received by the SD Test Client.

If a scenario works correctly with the SD Test Client and the MGF Test Gateway, it is relatively easy to shut down the MGF Test Gateway and start up your custom gateway in its place (with the same market constant registered to the EM), and you should be able to walk through the same scenario with your gateway, comparing the messages sent and received.



See the [SD Command Line Based Test Client](#) section for more details.

[Service Architecture Concepts](#)

Services for a Single Market Provided by Multiple Processes

Different services can either be in the same process or in different processes, on the same host or on different hosts; the client neither knows nor has to know exactly where a particular service runs.

[Service Architecture Concepts](#)

Service Definitions

The Orc Service Definitions are defined in a set of XML files. This section provides an explanation for the general layout and structure of these files. A quick introduction to the behavior of request/response and subscription models is also provided.

- [XML Files](#)
- [XML File Layout and Details](#)
- [Message Behavior](#)
- [SD Message Header](#)

< [Service Architecture Concepts](#) | [Document Overview](#) | [SDOFF](#) >

XML Files

The services are defined by a set of XML files:

File	Description
fix-fields-definitions.xml	FIX 5.0 fields
infrastructure-message-definitions.xml	base message types
service-definitions.xml	definitions of the services, fields and structures used by the services

A current copy of these files can be obtained from your local Orc support desk.

[Service Definitions](#)

XML File Layout and Details

File Structure

The services files are organized in the following hierarchy:

- **namespace:** Similar to a C++ namespace or Java package, defines the container of the subsequent fields, types, etc...
 - **fields:** A container for field definitions
 - **field:** The definition of an individual field that may be referenced in a struct, group, request, response, group or view.
 - **types:** A container for types referenced by fields
 - **enums:** A container for individual enumerations
 - **enum:** Definition of a list of legal values
 - **structs:** A container for structures
 - **struct:** A container containing a list of field references
 - **group:** A grouping of field references. This differs from a struct in one important way. When a group is referenced, its fields are "in-lined" within the object that is referencing them.
 - **messages:** A container for messages used by services
 - **request:** A message sent to a service
 - field reference
 - group reference
 - view (struct) reference: A view is a way for a message to contain a subset of a struct. It contains a list of fields from the struct to include in the message.
 - field reference
 - **response:** A message sent from a service in response to a request
 - **feed:** An unsolicited message sent from a service
 - **services:** A container holding service definitions
 - **service:** A slice of gateway functionality (e.g. Order Service)
 - **operation:** A transaction to a gateway
 - request reference
 - response reference
 - **query:** A request to a gateway for information (e.g. FindInstruments)
 - request reference
 - feed reference(s)
 - response reference
 - **subscription:** A request to a gateway for subsequent unsolicited updates from the gateway (e.g. TradesSubscription).
 - request reference
 - feed reference(s)
 - response reference

Basic Types

The service definitions support the following basic types:

Type	Description
int8	Signed 8-bit integer
int16	Signed 16-bit integer

int32	Signed 32-bit integer
int64	Signed 64-bit integer
uint8	Unsigned 8-bit integer
uint16	Unsigned 16-bit integer
uint32	Unsigned 32-bit integer
uint64	Unsigned 64-bit integer
real32	32-bit floating point (a.k.a. float)
real64	64-bit floating point (a.k.a. double)
boolean	T/F
date	Date type (same as uint32) in format YYYYMMDD
time	Time type (same as int64) in format HHMMSSssssuuu
string	Character string
char	Single character
datetime	uint64, Microseconds since Unix Epoch (midnight, Jan 1, 1970 UTC+0000)

field

```
<field name="Instrument" tag="orc.fields.1" type="Instrument" array="false"
comment="An instrument"/>
```

Each field entry defines a field that may be used in a message to/from a service.

- **name:** The name of the field. This is used wherever the field is referenced.
- **tag:** The ID of the field. This is used when encoding/decoding the message.
- **type:** The data type of the field. This references a basic type, enum or struct.
- **array:** Indicates whether or not this field is an array of the type specified.
- **comment:** Arbitrary text describing the field.

enum

```
<enum name="AdvSideEnum" type="int8">
  <description>Broker`s side of advertised trade</description>
  <value name="Buy" value="B" label="Buy"/>
  <value name="Sell" value="S" label="Sell"/>
  <value name="Trade" value="T" label="Trade"/>
  <value name="Cross" value="X" label="Cross"/>
</enum>
```

Each enumeration specifies a list of possible values for a field.

- **name:** The name of the enumeration. This is the name referenced by fields.
- **type:** The basic type of the enumeration. Note that "int8" is used instead of "char".
- **extends:** This indicates an enumeration that augments another enumeration. See below for example.
- **restricts:** This indicates an enumeration that limits the values of another enumeration. See below for example.

 The extends and restricts attributes are exclusive.

The value elements have the following attributes:

- **name:** The name of the value. This may not be unique.
- **value:** The specific value.
- **label:** Human-readable form of the value.

Extends

When the extends attribute is specified, it indicates the name of the enumeration that is being augmented. The extends tag has the format "<namespace>.<base_enum>" where:

1. namespace is the XML namespace of the augmented enumeration.
2. base_enum is the name of the augmented enumeration.

Restricts

In order to limit the values of an enumeration the restricts attribute is specified. The format is the same as the extends attribute but instead of augmenting the enumeration, it limits the values of the enumeration to the values specified in enumeration.

group

```
<group name="OrderBookFields">
  <field name="OrderBookID" required="true"/>
  <field name="TickRulesTableID"/>
  <field name="Currency"/>
  <field name="RoundLot"/>
  <field name="Factor"/>
  <field name="MassQuoteID" comment="Key to combine quotes into a single
massquote transaction"/>
  <field name="PriceDisplay"/> <!--\- For price display in a client \-->
</group>
```

A group contains a list of field references.

struct

```

<struct name="Instrument" tag="orc.types.1">

<!--\-\- Common fields \-->
<group name="OrderBookFields"/>
<group name="InstrumentIDs"/>

<!--\-\- Instrument classification \-->
<field name="CFICode" required="true" comment=
"http://wiki.orcsoftware.com:8100/display/ea/Instrument+Classification"/>
<field name="Product"/>
<field name="SecurityType" comment="Used only for combinations (MLEG)"/>
<field name="SecuritySubType" comment="For Combinations to reflect
combination type"/>
<field name="ExerciseStyle" comment="Used only for warrants. For options
style should be defined in
CFICode"/>

<!--\-\- For information purposes \-->
<field name="SecurityExchange" required="true" comment="MIC - Market
Identification Code (ISO 10383)"/>
<field name="MarketSegmentID" comment="Market specific identifier of
market segment"/>
<field name="SecurityGroup" comment="Market specific name for group of
related securities"/>
<field name="Issuer"/>
<field name="IssueDate"/>
<field name="SecurityDesc"/>

<!--\-\- Bonds, Derivatives: Futures, Options, Warrants and so on \-->
<field name="MaturityDate" comment="Last trading date, used for pricing,
required for applicable product types"/>
<field name="ValidityDate" comment="date after which instrument is not
valid any more, used for sorting and for cleanup of expired contracts,
optional"/>
<field name="ContractMultiplier"/>

<!--\-\- Common derivatives fields \-->
<field name="UnderlyingSymbol" />
<field name="UnderlyingOrderBookID" />
<field name="SettlType"/>
<field name="SettlDate"/>
<field name="SettlMethod" comment="Used only for warrants. For options
style should be defined in CFICode"/>

<!--\-\- Options, Warrants .. \-->
<field name="StrikePrice"/>
<field name="StrikeCurrency"/>
</struct>

```


A struct contains a list of fields. It specifies the list of fields by:

- field reference (by name) to the field
- group reference to the group of fields

For the above example, a corresponding C++ class may look like:

```
class Instrument
{
    String iOrderBookID;
    String iTickRulesTableID;
    String iCurrency;
    real64_t iRoundLot;
    real32_t iFactor;
    String iMassQuoteID;
    int32_t iPriceDisplay;
    String iSymbol;
    String iCUSIP;
    String iSEDOL;
    String iISIN;
    String iValoren;
    String iSecondaryInstrumentID;
    String iCFIcode;
    int32_t iProduct;
    int32_t iSecurityType;
    int32_t iSecuritySubType;
    int32_t iExerciseStyle;
    String iSecurityExchange;
    String iMarketSegmentID;
    String iSecurityGroup;
    String iIssuer;
    uint32_t iIssueDate;
    String iSecurityDesc;
    uint32_t iMaturityDate;
    uint64_t iValidityDate;
    real32_t iContractMultiplier;
    String iUnderlyingSymbol;
    String iUnderlyingOrderBookID;
    char iSettlType;
    uint32_t iSettlDate;
    char iSettlMethod;
    real64_t iStrikePrice;
    String iStrikeCurrency;
}
```

requests, responses, feeds (messages)

```

<request name="CreateOrderReq" tag="orc.messages.76"
extends="orc.BulkableReq"><!-- 'L' -->
  <view name="Order">
    <field name="OrderBookID" required="true"/>
    <field name="OrderID" required="true"/>
    <field name="Side" required="true"/>
    <field name="OrderQty" required="true"/>
    <field name="OrdType" required="true"/>
    <field name="TimeInForce" required="true"/>
    <field name="Price"/>
    <field name="Originator"/>
    <field name="Account"/>
    <field name="PositionEffect"/>
    <field name="ExpireDate" comment="Mandatory if TimeInForce is
GoodTillDate"/>
    <field name="Text"/>
    <field name="RawData" />
    <field name="RetainFlag"/>
    <field name="Portfolio"/>
    <field name="StopPx"/>
    <field name="StopPxType"/>
    <field name="StopPriceCondition"/>
    <field name="MaxFloor"/>
    <field name="MinQty"/>
    <field name="CumQty" comment="To support activation of partially filled
orders"/>
  </view>
  <field name="TargetOrdStatus" />
</request>

```

There are three types of messages defined in service definitions:

1. Requests are messages that are sent to a service
2. Responses are sent by a service in response to a request
3. Feed messages are generated asynchronously-- without request.

All messages have the following attributes:

- **name:** Name of the message. Used anywhere the message is referenced.
- **tag:** ID of the message. Used in encoding/decoding the message.
- **extends:** Specifies the message that this message augments. This is like C++ or Java inheritance. Fields, groups, views from the message being augmented are included.

Each message may contain combinations of field references, group references, struct reference and views.

- **field reference:** The name of the field to include in the message.
- **group reference:** The name of the group to include in the message. Group fields are substituted in-place.
- **struct reference:** The name of the struct to include in the message.
- **view:** Views specify a subset of struct fields to include in the message. View attributes include only a name.

The logical class layout of the CreateOrderReq specified above is:

```

class Order
{
    ... order fields
};

class CreateOrderReq
{
    // Fields from "Header" group specified in BaseReq
    int64_t iRequestID;
    uint32_t iServiceTypeID;
    uint32_t iServiceOperationID;
    uint64_t iTransactionID;
    uint64_t iInTimestamp;
    uint64_t iOutTimestamp;
    // Fields from BulkableReq
    bool iPartOfBulkTransaction;
    bool iFlushFlag;
    // Fields from CreateOrderReq
    Order iOrder;
    char iTargetOrdStatus;
};

```

operation

```

<operation name="CreateOrderOperation" id="3">
    <request name="CreateOrderReq"/>
    <response name="CreateOrderRsp"/>
</operation>

```

An operation is a transaction that may be requested of a service (e.g. CreateOrder):

- **name:** The name of the operation.
- **id:** The identifier of the operation. This is used when encoding/decoding the message.

Operations specify a request and response message that make up the transaction.

query

```
<query name="FindInstrumentsQuery" id="3">
  <request name="FindInstrumentsReq"/>
  <feed name="InstrumentFeed"/>
  <feed name="InstrumentLegFeed"/>
  <feed name="OrderBookFeed"/>
  <response name="SnapshotEndRsp" />
</query>
```

subscription

```
<subscription name="InstrumentsSubscription" id="4" type="STATEFUL">
  <control name="InstrumentsSubscriptionControlReq"/>
  <feed name="InstrumentFeed" />
  <feed name="InstrumentLegFeed" />
  <feed name="OrderBookFeed" />
  <response name="SnapshotEndRsp" />
</subscription>
```

Service Definitions

Message Behavior

Orc Service behaves like a typical exchange. In general, services are either requests and response or subscriptions.

- Request and response: CreateOrderReq and CreateOrderRsp
- Subscription: Prices, trades, etc.

Clients (e.g. Orc Trader or Liquidator) open sessions to the individual service and sends and receives messages to them. There are two basic types of message patterns; a request and response pattern and a subscription pattern. In the request and response pattern, the client sends a request and the server provides a response. For the subscription patterns, the client sends a subscription control request and the server provides a stream of feed messages for the subscription until the client sends a subscription control request to stop the subscription.

Detailed behavior per service is described in later sections.

[Service Definitions](#)

SD Message Header

All messages have a header, the fields in the header can be part of any message.

- **RequestID**
 - All **request** messages can have a RequestID set. The service is required to include the same RequestID in the response.
 - For **subscriptions**, if the ControlRequest contains a RequestID, the snapshot (if any) will include the RequestID, but not the subsequent updates.
- **End2End metrics** – End2End metrics are used to be able to analyze single transactions through the system and in which component that particular transaction spends most of its time (e.g. user strategy, Liquidator, gateways and exchange). There are three fields in the header associated with End2End metrics (TransactionID, InTimestamp & OutTimestamp).
 - **TransactionID** is a field that can be passed in any message, TransactionID will be used to analyze round-trip behavior. TransactionID is an optional field that need not be there, that can be there all the time for a given session, or can be there once in a while for a session.
 - **InTimestamp** is a field that can be passed in any message which tells when the sending component in its turn received the message.
 - **OutTimestamp** can be passed in any message and tells when the sending component sent the message (using its local time in UTC (ie non-localized timezone, but not easy to compare with the receiving component if the receiving component is on another server due to clock differences between servers).

[SDOFF](#)

SDOFF

This section describes SDOFF, which is the SD message protocol over the Orc DOFF wire protocol. SDOFF is used by all the services described in later sections.

- [DOFF Wire Protocol](#)
 - [SDOFF - SD protocol over DOFF](#)
 - [Encode Decode Example Code](#)
 - [FIX Fields](#)
-

< [Service Definitions](#) / [Document Overview](#) / [Initial Handshakes and Login](#) >

DOFF Wire Protocol

Introduction

The DOFF Wire Protocol is a proprietary layer on top of TCP. A standard TCP connection is used to transfer data between client and server. DOFF defines the message structure on the wire. DOFF has two different segment structures:

- Connection Segment Structure - Only used during the initial connection between client and server, or handshake.
- Data Transfer Segment Structure - Used for all other client/server communication after the initial handshake.

We encode application messages, such as an OrderFeed, using the Service Definitions (SD) and write them to the wire using the DOFF Data Transfer Segment Structure. We refer to this as SD over DOFF, or SDOFF.

Segment Structure

Connection Segment Structure

Bytes	0	1	2	3	4	5
0 - 5	MessageType	Protocol	Compression	Encryption	Len	
6 - ...	Payload(of length "Len")					

Message type

1 byte ASCII encoded character. Identifies the types of message being sent. Below are the list of supported types

MessageType Constants	
Data only	5

Protocol

1 byte ASCII encoded character. Identifies the type of protocol SD client will use during data transfer. Different possible values for protocol are

Protocol Constants	
Binary DOFF	5
Zlib binary	6

There is no difference in implementation on the gateway side between Binary DOFF and Zlib binary so your application just needs to sanity check that it is 5 or 6.

Compression

1 byte ASCII encoded character. Identifies the type of compression being used during message transfer. During data transfer everything except **Message Type** and **DOFF Message Length** is compressed using the specified compression technique. Different possible compression techniques to be supported are:

Compression Constants	
No compression	0
Zlib	2

If Zlib compression is enabled then the section of Data Transfer Message Segment after the Doff Message Length is Zlib compressed. Standard Java libraries exist for performing zlib compression: `java.util.zip.Inflater` and `java.util.zip.Deflater`.

Encryption

1 byte ASCII encoded character. Identifies if messages are encrypted during data transfer or not.

Encryption Constants	
No	0

Encryption is not currently used by any of the Orc clients.

Len

2 bytes. Identifies the length of number of bytes in the payload. This field is big-endian encoded.

Payload

Length of Len bytes. Client usually sends the session key as the payload. Server can ignore this payload.

Data Transfer Segment Structure

Bytes	0	1	2	3	4	5	6	7	8
0 - 8	Message Type	DOFF Message Length				SD Message Length			
9 - 17	SD-Header + SD Body								
18 -									

The Data Transfer Segment Structure consists of a DOFF header followed by one or more SDOFF messages. The DOFF header is made up of a MessageType and a DOFF Message Length. Each SDOFF Message is made up of a SD Message Length and SD Header + SD Body.

Message type

1 byte ASCII encoded character. Identifies the type of message being sent. Below is the list of different possible values

DOFF Message Type Constants		desc
Compressed	C	Data transfer mode using compression

Plain data	D	Ordinary data transfer mode, no compression
Final	F	Signals connection is closing
HeartBeat	H	Heartbeat packet - contains no payload. Total absence of any message for a longer period of time indicates some network problem.

DOFF message length

4 bytes. Identifies the total length of the following SDOFF message(s). It doesn't include its own length. Length is big-endian encoded. If compression is enabled this represents the length after the following SDOFF message(s) have been compressed.

Note: While the DOFF Message Length is BigEndian all the remaining SDOFF fields are LittleEndian encoded

SD message length

4 bytes. Identifies the length of SD header and SD Body. It doesn't include its own size. "SD Message Tag", "SD Service Tag" and "SD Operation ID" constitute the SD header.

SD-Header

Bytes		Max 4 bytes		Max 4 bytes		1 byte
SD - Header	=	SD MessageTag	+	SD Service Tag	+	SD OperationID

This header is different from the SD Message header described here: [SD Message Header](#). This header describes SD protocol information while the [SD Message Header](#) is encoded as part of the SD Body.

SD message tag and SD Service Tag

The SD message tag and SD service tag from the [Service Definitions](#) are both variable in length with a max of 4 bytes. They are of type uint32_t. Descriptions of SDOFF data types are here: [SDOFF - SD protocol over DOFF](#) and encoding/decoding examples can be found here: [Encode Decode Example Code](#)

SD operationID

1 byte. The operation-id from the service definitions (where both request and response have the same operation ID).

SD body

See [SDOFF - SD protocol over DOFF](#) for how to encode/decode the SD body.

Protocol Operation

SDOFF operations may be divided into three phases. Connection must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After completion of data transfer, connection termination releases all allocated resources.

Connection Establishment

On top of TCP handshake, to establish a connection, DOFF employs a two-way handshake.

1. After TCP connection is established, SD-DOFF client sends *connection-segment structure* and waits for a single byte response.
2. After parsing *connection-segment structure* SD-DOFF server replies with one byte:

Connect Response	Description
"B"	If the request MessageType byte is DataOnly and the protocol is ZLIB BINARY DOFF or BINARY DOFF
"F"	If you received unexpected data in the connection message and you are closing the connection.

Data transfer mode begins when client receives a begin message byte ("B"). On final ("F") client closes the connection.

NOTE: A typical handshake bytes from client start with 5,5,0,0 or 5,6,2,0.

- 5, 5, 0, 0 = MsgTyep::Data only, Protocol::Binary DOFF, Compression::None, Encryption:: NO
- 5, 6, 2, 0 = MsgTyep::Data only, Protocol::Zlib Binary DOFF, Compression::Zlib, Encryption:: NO

Data Transfer

In data transfer mode data transfer segment structure is used. "*Message Type*" and "*DOFF Message Length*" constitute the DOFF header. Multiple SD message can be sent under a single DOFF header. In this case *DOFF Message Length* encompasses length of all the included SD messages. In some cases the DOFF message can grow quite large so it is recommended to start with a sufficiently large receive buffer size or have the ability to dynamically resize your buffer.

[SDOFF](#)

SDOFF - SD protocol over DOFF

General overview

SDOFF message messages are encoded using the SDOFF wire format described in the section below this.

- message-length, message-tag, service-tag, and operation-id are all described in more detail in [DOFF Wire Protocol](#) where they are referred to under the SDOFF Message Length and SD Header sections.
- The <message-body> is what we referred to as the SD-Body in [DOFF Wire Protocol](#). The message body consists of fields which are tag-value pairs.
- The tag is of type uint32 and uniquely identifies the field from the service definitions.
- A value may consist of a struct, an array, or simply a flat-value which may be numeric-value (int16, int32, etc..), string-value, octets-value (byte), or bool-value. No matter the type of the value it is always preceded by its ASCII code (Ex. struct='S', array='A', int32='i').
- For structs the ASCII code of 'S' is followed by a length which is of type le-uint32, which is simply 4 bytes little-endian, and indicates the total length of the fields that are set in the struct.
- For arrays length also follows the ASCII code defining the total length of the fields in the array. This is followed by count which is uint32 and defines the number of elements "n" in the array. This will be followed by a field for each element up to "n".
- Strings are encoded with ASCII code 'S' followed by the string length, v-length. v-length is encoded as uint32 which differs from the length of structs and arrays. Following the v-length is the bytes comprising the string. The string encoding to use is not defined SDOFF protocol but instead you should use the same encoding as the exchange. We typically default our gateways to use ISO/IEC 8859-1 string encoding but allow the ability to override this. String fields will always end in ASCII '\0' or byte 0x00.
- All SDOFF data types are little-endian encoded.

SDOFF wire format

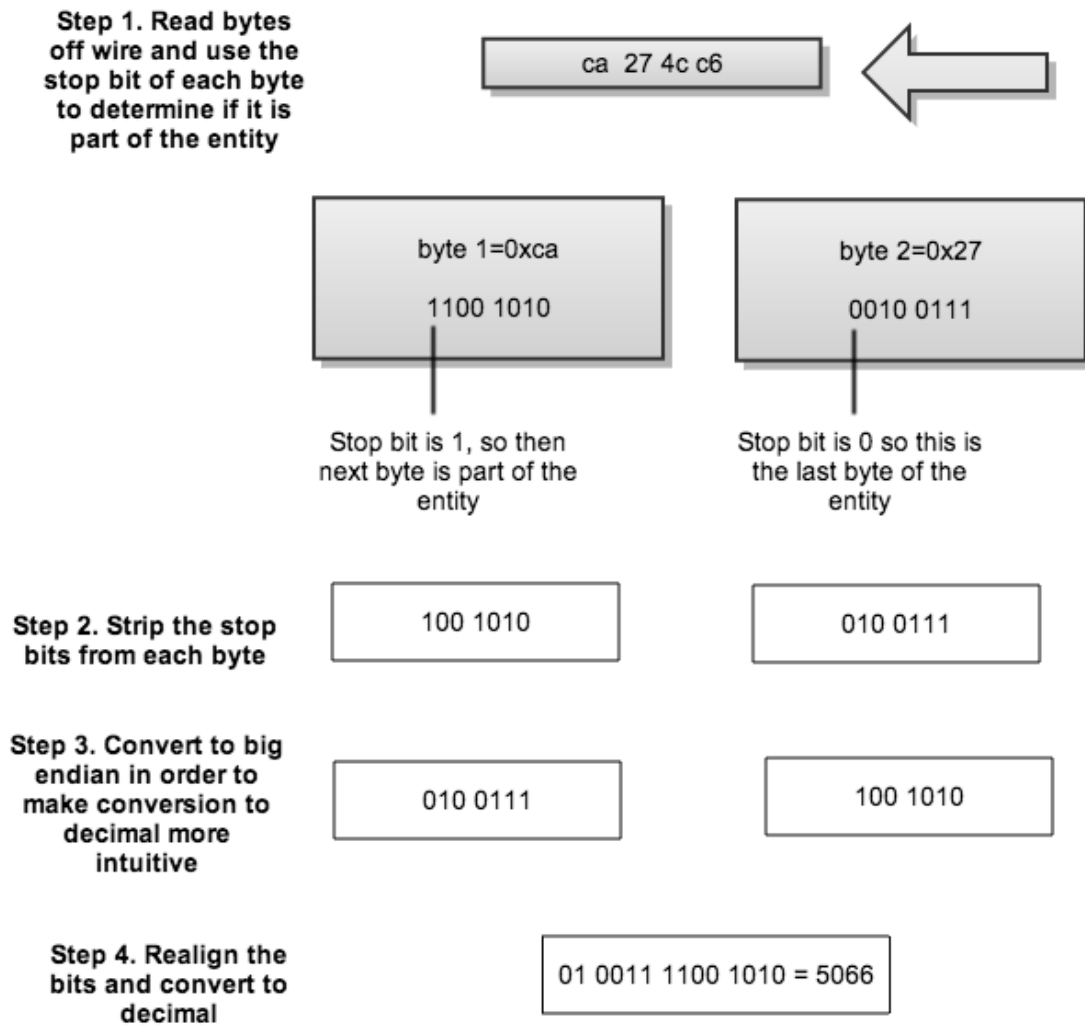
```

<message> ::=
<message-length><message-tag><service-tag><operation-id><message-body>
  <message-length> ::= <length>
  <message-tag> ::= <uint32>
  <service-tag> ::= <uint32>
  <operation-id> ::= <octet>
  <message-body> ::= (<field>)*
    <length> ::= <le-uint32>
    <field> ::= <field-tag> <value>
    <value> ::= (<struct> | <array> | <flat-value>)
    <flat-value> ::= <numeric-value> | <string-value> | <octets-value> |
<bool-value>
      <struct> ::= 'S'<length><fields>
      <array> ::= 'A'<length><count>(<value>){count}
  <numeric-value> ::= 'w'<int16> | 'W'<uint16> | 'i'<int32> | 'I'<uint32>
| 'l'<int64> | 'L'<uint64> | 'r'<real32> | 'R'<real64> | 'b'<octet>
  <string-value> ::= 's' <v-length> (<octet>)* '\0'
  <octets-value> ::= 'o' <v-length> (<octet>)*
  <bool-value> ::= <true-value> | <false-value>
  <true-value> ::= 't'
  <false-value> ::= 'f'
  <v-length> ::= <uint32>
  <field-tag> ::= <uint32>
  <count> ::= <uint32>

```

1. **<uint16><uint32><uint64>** are encoded with run-length-encoding using stop bits. An important property of this type of encoding is the use of variable length stop bit encoded integers. A stop bit encoded integer is a sequence of bytes where the most significant bit (X*****) in each byte indicates whether the next byte is part of the entity. If the bit is set, the next byte belongs to the entity, otherwise it is the last byte. The seven bits following the stop bit are significant data bits in each byte while the stop bit of each byte should be stripped out of the entity. Values up to 127 will fit in one byte, up to 16383 in two bytes. Numbers greater than 16383 will require a third byte, etc.
2. **<int16><int32><int64>** types are encoded in the same way except the bit before the most significant (X*****) of the first byte: this bit is a sign-bit. So the six bits following the stop and sign bits of the first byte and the seven bits following the stop bit of the others are significant data bits for signed integer types. Non-zero sign-bit means negative number.
3. **<real32><real64>** are put in unsigned integer variable of corresponding size (with direct memory copying for little-endian platforms). Example for real64 on java: `buffer.writeLong (Double.doubleToLongBits(value))` and make sure that the buffer is in little-endian order before you write the value.
4. **<le-uint32>** encoded as 32-bit little-endian numbers (natively used e.g. on Intel architectures)
5. The **<v-length>** in a **<string-value>** is the number of octets in the string encoding which is not necessarily the same as the number of characters in the string.
6. The string encoding to use is not currently specified by SDOFF. A market gateway will use the same string encoding as the exchange.
7. **<octet>** means a byte passed as is on the wire

Example decoding stop bit encoded uint32



Encoding/Decoding Code Examples

- Code examples in java for encoding and decoding the various types: [Encode Decode Example Code](#)

Computation of tags

There are service-tags, message-tags, and field-tags. All tags are of type uint32 so they use run-length-encoding as described above in the description of <uint16><uint32><uint64>.

Fields

Each field is defined within a namespace, e.g. "orc" or "fix", and has a tag attribute which will look like this tag="orc.XXXXX.###". We compute the tag value as the namespace value plus ###. As an example let's look at the Instrument field in service-definitions.xml:

```
<namespace name="orc" value="5000">
  <fields>
    <field name="Instrument"
tag="orc.fields.1"    type="Instrument"/>
```

Instrument is in the "orc" namespace which has a value 5000 and it's tag attribute = tag="orc.fields.1", so the tag value = 5000 + 1 = 5001.

The fix namespace value is 0, so fix tag values will just be ###. Market specific fields have a namespace of 8000 so their tag value will be 8000 + ###.

Services and Messages

Similar to fields, services and messages are defined in a namespace and have a tag attribute. However since all services and messages reside in the "orc" namespace our clients will typically just use the tag, ###, and not add it to the namespace value. For consistency in our gateways if we receive a service or message tag less than 5000 we will add 5000 to account for it being in the "orc" namespace. As of yet there are no market-specific messages but should they be added in the future they would be computed the same as 8000 + ### similarly to fields.

service or message tag decode example:

```
private int fastDecodeInt(ByteBuffer buffer) {
    int result = 0;
    byte shift = 0;
    byte b = -1;
    while (b < 0) {
        b = buffer.get();
        result += ((b & 0x7F)) << shift;
        shift += 7;
    }

    //Only the tag value and not the orc namespace value are used for
    services and messages
    //so we add 5000 to account for the orc namespace since we know no
    services or messages are defined in the fix namespace.
    if(result < 5000){
        result += 5000;
    }

    return result;
}
```

Getting Started Guide

- For additional examples, please see the **GDK-Native Getting Started Guide**.

[SDOFF](#)

Encode Decode Example Code

- **Example Java method to decode stop bit encoded entity for unsigned types: uint16_t, uint32_t, or uint64_t.**

```
private int fastDecodeInt(ByteBuffer buffer) {  
    int result = 0;  
    byte shift = 0;  
    byte b = -1;  
    while (b < 0) {  
        b = buffer.get();  
        result += ((b & 0x7F)) << shift;  
        shift += 7;  
    }  
  
    return result;  
}
```

- **Example Java method of how to decode a stop bit encoded entity of a signed or unsigned field type. If the field type is signed (int16_t, int32_t, or int64_t) then boolean the parameter signed should be true. Otherwise for type uint16_t, uint32_t, or uint64_t signed should be false.**


```

private long fastDecodeLong(ByteBuffer buffer, boolean signed) {
    long result = 0;
    byte shift = 0;
    byte b = -1;
    boolean neg = false;
    while (b < 0) {
        b = buffer.get();
        if (signed && shift == 0) {
            neg = ((b & 0x40) == 0x40);
            result += (b & 0x3F);
            shift = 6;
        }
        else {
            if (signed && shift == 62) {
                long lastPart = (b & 0x7F);
                if (lastPart == 0L || lastPart == 1L) {
                    result += lastPart << shift;
                }
                else if (neg && lastPart == 2L && result == 0L) {
                    return Long.MIN_VALUE;
                }
                else {
                    throw new RuntimeException("Value overflow, value outside
range of long");
                }
            }
            else {
                result += ((long) (b & 0x7F)) << shift;
                shift += 7;
            }
        }
    }
    if (neg && result < 0) {
        throw new RuntimeException("Value overflow, value > " +
Long.MAX_VALUE);
    }
    return neg ? -result : result;
}

```

- Java method for encoding a stop-bit encoded entity. If signed (int16_t, int32_t, int64_t) then parameter signed is true and for unsigned (uint16_t, uint32_t, uint64_t) it is false.

```

private final long mask8 = 0x00000000000000ffL;
private final long mask16 = 0x000000000000ffffL;
private final long mask32 = 0x00000000ffffffffL;
private final long mask64 = -1L;

/**

```

```

* Copy the value into the buffer at its current position.  Param
signed indicates whether the type is signed or unsigned.
* Param mask is one of the masks above corresponding to the data type
you are encoding.  So for int8 or uint8 pass in mask8, int16 ->
mask16, etc...
*/
private void fastEncodeLong(ByteBuffer buffer, long value, boolean
signed, final long mask) {
    rangeCheck(value, mask);

    long tmp = value;
    if (tmp < 0 && !signed) {
        if ((tmp & ~mask) != ~mask) {
            throw new RuntimeException("Value is negative but written as
unsigned.");
        }
        else {
            tmp = value & mask;
        }
    }
    if (signed) {
        byte sign = 0x00;
        if (signed && tmp < 0) {
            sign = (byte) 0x40;
            tmp = -tmp;
        }
        if (tmp > 0x3F) {
            buffer.put((byte) (0x80 | sign | (tmp & 0x3F)));
            tmp = tmp >> 6;
            while (tmp > 0x7F) {
                buffer.put((byte) (0x80 | (tmp & 0x7F)));
                tmp = tmp >> 7;
            }
            buffer.put((byte) tmp);
        }
        else {
            buffer.put((byte) (sign | tmp));
        }
    }
    else {
        while ((tmp & 0xffffffffffffffff80L) != 0) {
            // while (tmp > 0x7F) {
            buffer.put((byte) (0x80 | (tmp & 0x7F)));
            tmp = tmp >> 7;
            tmp &= 0x01ffffffffffffffL;
        }
        buffer.put((byte) (tmp & 0x7F));
    }
}

/**
 * Checks that the long value is in the range implied by the
provided mask. The method is agnostic with respect to

```

```

    * signed/unsigned. For instance, to check that a long value is a
    * valid uint8, set mask to 0xff. Any bit pattern in
    * the ranges 0xffffffff00-0xfffffffffff and 0x000000000-0x0000000ff will
    * be accepted. Note that this implies that there are
    * two alternative representations of any uint8 value. In other
    * words, 0xffffffff00 == 0x000000000 when seen as a uint8.
    *
    * @param value
    *           The value to be checked
    * @param mask
    *           Bit mask (2**N)-1 (N=8,16,32 or 64)
    * @param contextDescription
    *           Used for error reporting
    */
private void rangeCheck(final long value, final long mask) {
    long extraBits = (value & ~mask);
    if (extraBits == 0L || extraBits == ~mask) {
        return;
    }
}
```

```
        throw new IllegalArgumentException("Value out of range");
    }
```

- **Java method for decode real64_t**

```
private double fastDecodeDouble(ByteBuffer buffer, String
contextDescription, SerializationContext context) {
    ByteOrder savedOrder = buffer.order();
    buffer.order(ByteOrder.LITTLE_ENDIAN);
    long representation = buffer.getLong();
    buffer.order(savedOrder);
    return Double.longBitsToDouble(representation);
}
```

- **Java method for decode real32_t**

```
private float fastDecodeFloat(ByteBuffer buffer, String
contextDescription, SerializationContext context) {
    ByteOrder savedOrder = buffer.order();
    buffer.order(ByteOrder.LITTLE_ENDIAN);
    int representation = buffer.getInt();
    buffer.order(savedOrder);
    return Float.intBitsToFloat(representation);
}
```

- **Java method for encode real64_t**

```
private void fastEncodeDouble(ByteBuffer buffer, double value) {
    long representation = Double.doubleToLongBits(value);
    ByteOrder savedOrder = buffer.order();
    buffer.order(ByteOrder.LITTLE_ENDIAN);
    buffer.putLong(representation);
    buffer.order(savedOrder);
}
```

- **Java method for encode real32_t**

```
private void fastEncodeFloat(ByteBuffer buffer, float value) {  
    int representation = Float.floatToIntBits(value);  
    ByteOrder savedOrder = buffer.order();  
    buffer.order(ByteOrder.LITTLE_ENDIAN);  
    buffer.putInt(representation);  
    buffer.order(savedOrder);  
}
```

- **C method for encode real64_t**

```
real64_t value=15.1;  
uint64_t v=0;  
memcpy(&v, &value, sizeof(v));  
encode(output, v, sizeof(v));
```

[SDOFF](#)

FIX Fields

As far as possible, the fields in our messages should be standard FIX fields, if applicable.

[SDOFF](#)

Initial Handshakes and Login

Regardless of the services being provided, each gateway must provide basic login/logout functionality to support client connectivity.

- [Server Registration with Environment Manager \(EM\)](#)
 - [Client Service Discovery](#)
 - [Establishing the TCP IP Connection](#)
 - [Login and Logout](#)
 - [Disabling DOFF Compression in Orc Trader](#)
-

< [SDOFF](#) | [Document Overview](#) | [Instrument Service](#) >

Server Registration with Environment Manager (EM)

The service URL needs to be published in a service registry where the clients can find it. The current service registry is the Orc Environment Manager (EM). URLs can be statically published by adding them as command line option to the em process in orc.conf.

EM-registration

The Environment Manager service can be set to have a static external process registration entry using the -U switch when starting emd. See **Orc server core processes** in the *Server Administration Manual* for details.

EM registration

```
emd <all-your-normal-options> -U
<market-id>,<service-name>,<processname>,<service-group>,<host>,<port>[,...]
```

Example of an em process in orc.conf configured to statically register mgftest MarketDataService and InstrumentService at localhost:10000

```
PROC:em:0300:0200:0,1,2,3,4,5,6:em:TSRELEASE:emd::EM:em:-C cds -S storage -U
471,InstrumentService,mgftest,mgftest,localhost,10000,471,MarketDataService,mgftest,
mgftest,localhost,10000 ::
```

Verification of the EM dynamic or static registration

Orc provides a tool called "eminfo" in the bin directory of a Trading Systems release. This utility will list the current registrations in the EM.

To run the eminfo command:

- cd <Orc Trading System Release>/bin
- eminfo <em name>

For example to run eminfo against a EM named "em" in the orc.conf

- cd /usr/orcts9/bin
- ./eminfo em

You should see something like this, depending on what services are being provided by the gateways configured in orc.conf:

```
MGF Test SD 471 mgftestjava_ctt mgftestjava_ctt InstrumentService quagmire(192.168.153.32) 1
sdoff://192.168.153.32:49367 mgftestjava_ctt Gateway
MGF Test SD 471 mgftestjava_ctt mgftestjava_ctt MarketDataService quagmire(192.168.153.32) 2
sdoff://192.168.153.32:49367 mgftestjava_ctt Gateway
MGF Test SD 471 mgftestjava_ctt mgftestjava_ctt OrderService quagmire(192.168.153.32) 3
sdoff://192.168.153.32:49367 mgftestjava_ctt Gateway
MGF Test SD 471 mgftestjava_ctt mgftestjava_ctt QuoteService quagmire(192.168.153.32) 4
```



```
sdoff://192.168.153.32:49367 mgftestjava_ctt Gateway
MGF Test SD 471 mgftestjava_ctt mgftestjava_ctt TradeReportService quagmire(192.168.153.32) 6
sdoff://192.168.153.32:49367 mgftestjava_ctt Gateway
```

You should see your gateway's dynamic or static registration in this listing.

[Initial Handshakes and Login](#)

Client Service Discovery

- Ask the EM or hardcode it

Clients (Orc Trader and Liquidator) attach to services by locating their service registration record. The service registration record is a URL in the standard URL format (i.e. protocol://resourcelocation-information-specific-to-the-protocol). There is one protocol commonly used in existing clients which is sdo://host:port.

[*Initial Handshakes and Login*](#)

Establishing the TCP IP Connection

Establishing a connection to the service involves the following steps:

1. Determine the endpoint (IP address, port) of the required service. This may be performed statically via configuration or dynamically by querying the Environment Manager.
 2. Create a TCP/IP socket connection to the service.
 3. Perform the DOFF handshake described in [DOFF Wire Protocol](#).
 4. Login to the service described in the [Login and Logout](#) section.
-

[Initial Handshakes and Login](#)

Login and Logout

Login/Logout

When a client wishes to use a service, it starts by sending a LoginRequest to the service. The LoginRequest includes Orc username, Orc group and some version information. When a client wishes to stop using a service it will send a LogoutRequest. In the current Orc systems the same user can only have one simultaneous login to any server or service. A user from an Orc perspective is uniquely identified with username+group.

LoginReq

When a client decides to use a service, the first message that is sent to the service must be a LoginReq. The LoginReq must include Username, Group, ServiceVersion, ServiceImplVersion and ApplicationVersion. Username combined with Group is a unique identifier of the user. The service should prohibit the same user from being logged in multiple times. ServiceVersion is the service version the client was compiled against MAJOR.MINOR. ServiceImplVersion is a market gateways specific service version that can get updated if a market requires new mandatory fields, normally just 1.0. ApplicationVersion is an identifier of the application (e.g. "Orc Trader 9.0.11").

LoginRsp

The service will respond with its own ServiceVersion, ServiceImplVersion and ApplicationVersion in the LoginRsp. The ApplicationVersion identifies the particular gateway version.

If the ServiceVersion or ServiceImplVersion MAJOR numbers (i.e. before the first dot) are different the Service must reject the Login attempt.

[Initial Handshakes and Login](#)

Disabling DOFF Compression in Orc Trader

By default, Orc Liquidator does not use DOFF compression, but Orc Trader does. To turn off compression in Orc Trader:

1. In the Orc Trader login window, click the "Configure..." button
2. In the configuration window, click "Doff" in the left navigation panel
3. In the Doff configuration, uncheck "Use compression"

[*Initial Handshakes and Login*](#)

Instrument Service

The Instrument Service is used to provide clients with instruments for an exchange. It supports start-of-day snapshots as well as intraday updates to instruments. This service is typically provided along with the Market Data Service by the same gateway, since instruments and price feeds are typically provided by the same interface from an exchange.

- [Instrument Service Behavior](#)
 - [Creating CFICode Using ISO 10962](#)
-

< [Initial Handshakes and Login](#) | [Document Overview](#) | [Market Data Service](#) >

Instrument Service Behavior

- [Normal Client Operations](#)
- [Instrument Service Messages](#)
 - [InstrumentFeed Message](#)
 - [All Security Types](#)
 - [Futures and Debts \(Interest Rates, Bonds, etc.\)](#)
 - [Common Derivative Fields](#)
 - [Options and Warrants Fields](#)
 - [Combination Fields](#)
 - [InstrumentLegFeed Message](#)
 - [InstrumentLeg Structure](#)
 - [TickRulesTableFeed Message](#)
 - [OrderBookFeed](#)
 - [OrderBooks Structure](#)
- [CFICode creation](#)
 - [Examples of creating a CFICode:](#)
 - [Equites - first 2 characters are required](#)
 - [Debts \(Interest Rates, Bonds etc\) - the first 3 and 5th characters are required \(except for multilegs\)](#)
 - [Indice](#)
 - [Warrant - first 5 characters are required.](#)
 - [Option - first 5 characters of CFICode are required.](#)
 - [Future - first 3 characters of CFICode are required.](#)
 - [Forwards](#)
 - [Combinations\(Combo, Multileg Instruments\)](#)
- [The standard operations of the Instrument Service](#)
 - [Login, Logout Operations](#)
 - [FindInstrumentsQuery](#)
 - [InstrumentsSubscription](#)
 - [Opening a Subscription](#)
 - [Closing a Subscription](#)
 - [TickRuleTablesQuery](#)
 - [TickRuleTablesSubscription](#)
 - [Opening a Subscription](#)
 - [Closing a Subscription](#)
 - [OrderBooksSubscription](#)

The Instrument Service keeps the reference data for all the instruments you can work with in the other gateway services as well as keep track of the tick-rules associated with said instruments. There are basically three ways to get an instrument:

1. Query the instrument directly using a FindInstrumentsQuery
2. Get a snapshot of all instruments using the InstrumentsSubscription
3. Get an intra-day update with a new instrument using the InstrumentsSubscription

The Instrument Service is used to provide clients with reference data for instruments that are traded on this particular market. The instrument service is designed to support start-of-day snapshot as well as intraday updates to

instruments.

Normal Client Operations

A client of the Instrument service will normally request the following after logon:

- InstrumentsSubscription
- TickRuleTablesSubscription

This will send an instrument and tick rule snapshot to the client. The client will normally leave these subscriptions open for the continuation of the session.

Instrument Service Messages

InstrumentFeed Message

Defines a security definition. In the InstrumentFeed message there are fields that are required by all security types (Future, Equity, etc..) and fields that are only required for specific security types. Other fields are not required but should be set if the values are available from the exchange. If there is a direct mapping between a field in the exchange definition of the instrument and a field in the InstrumentFeed (even if it's not in the table below) then you should set it. In the tables below OT means that a field is required if the instrument is to be used in Orc Trader.

All Security Types

Field	Description	Required
OrderBookID	Market gateway-provided unique OrderBook ID	yes
CFICode	Standard FIX field	yes
SecurityExchange	Standard FIX field	yes
Symbol	Standard FIX field	OT
SecurityDesc	Standard FIX field	OT
UnderlyingSymbol	Standard FIX field. If there is no UnderlyingSymbol can be equal to Symbol	OT
Currency	Standard FIX field	OT
ContractMultiplier	Factor by which price must be adjusted to determine the true nominal value of one futures/options contract. (Qty * Price) * Factor = Nominal. Multiplier in OT/CDS. Can default to 1 if not applicable for instrument	recommended

Factor	Specifies the ratio or multiply factor to convert from "nominal" units (e.g. contracts) to total units (e.g. shares). Price Multiplier in OT/CDS. Can default to 1 if not applicable for instrument	recommended
TickRulesTableID	Identifier of the tick rule applicable to an instrument.	recommended
PriceDisplay	Similar to FIX PriceType. See PriceDisplayEnum in service-definitions.xml	recommended

Futures and Debts (Interest Rates, Bonds, etc.)

Field	Description	Required
MaturityDate	last trading date. Maps to ExpiryDate in OT. type is date (uint32 YYYYMMDD)	yes
MaturityMonthYear	Optional field that can be used if maturity is different from expiry. Maps to MaturityDate in OT. Type is string (YYYYMMDD)	no
ValidityDate	Date after which instrument is no longer valid. Optionally used for cleanup of expired contracts. Type is datetime (uint64, Microseconds since Unix Epoch)	no

Common Derivative Fields

Field	Description	Required
UnderlyingOrderBookID	OrderBookID of the underlying instrument	no
MassQuoteID	Key to combine quotes into a single massquote transaction	only if you are using QuoteService or MassQuoteService

Options and Warrants Fields

Field	Description	Required
StrikePrice	Standard FIX field	yes

MaturityDate	last trading date. Maps to ExpiryDate in OT. type is date (uint32 YYYYMMDD)	yes
MaturityMonthYear	Optional field that can be used if maturity is different from expiry. Maps to MaturityDate in OT. Type is string (YYYYMMDD)	no
ValidityDate	Date after which instrument is no longer valid. Optionally used for cleanup of expired contracts. Type is datetime (uint64, Microseconds since Unix Epoch)	no
StrikeCurrency	Standard FIX field	no

Combination Fields

Field	Description	Required
SecurityType	Standard FIX field. Always set to MultiLeg	yes

InstrumentLegFeed Message

Defines a security leg definition of a combination (exchange spread or user defined spread). The following fields are important:

Field	Description	Required
OrderBookID	OrderBook ID of combination instrument	yes
InstrumentLegs	A collection of Instrument Leg Structures for this combination.	yes

InstrumentLeg Structure

Field	Description	Required
OrderBookID	OrderBook ID of Leg Instrument	yes
LegRatioQty	The ratio of quantity for this individual leg relative to the entire multileg security. May be negative.	yes

TickRulesTableFeed Message

Field	Description	Required
TickRulesTableID	Identifier of the tick rule applicable to an instrument	yes
TickRules	A collection of TickRule Structures	yes

OrderBookFeed

Dynamically change security information, such as lot size. Also, defines the security available in different sub-markets.

Field	Description	Required
OrderBookID	Market gateway-provided unique OrderBook ID	yes

OrderBooks Structure

Field	Description	Required
OrderBookID	Market gateway-provided unique OrderBook ID	yes
SubMarketID	see SubMarketEnum in service-definitions.xml	no
RoundLot	The trading lot size of a security	recommended
TickRulesTableID	Identifier of the tick rule applicable to an instrument	recommended

CFI Code creation

If the market exchange doesn't provide CFI code or provides an incomplete CFI code we need to attempt to create it ourselves: [Creating CFI Code Using ISO 10962](#).

What if there is no correct CFI code in the standard? Then we define it as much as possible in CFI code and then try and set these fields on the instrument as best you can: SecurityType, SecuritySubType, and Product.

Examples of creating a CFI Code:

Equities - first 2 characters are required

- Equity Common, ordinary shares:
CFI Code=ESXXXX
- Equity Preferred shares:
CFI Code=EPXXXX
- Equity Mixed units, Baskets:
CFI Code=EMXXXX, SecurityType=MLEG
- Common share, each share has one vote, ownership could be transferred freely, nil paid and the owner is not registered in the books of the issuer or of the registrar:
CFI Code=ESVUOB

Debts (Interest Rates, Bonds etc) - the first 3 and 5th characters are required (except for multilegs)

- Bond with fixed interest rate and fixed maturity:
CFIcode=DBFXFX
- Convertible bond with fixed interest rate and fixed maturity:
CFIcode=DCFXXF
- Mixed units as a number of debts or debts and others:
CFIcode=DMXXXX, SecurityType=MLEG

Indice

- general case:
CFIcode=MRXXXX

Warrant - first 5 characters are required.

- Traditional Call Warrant on Debt:
CFIcode=RWDTCX
- Traditional Put Warrant on Debt:
CFIcode=RWSTPX
- Covered Call Warrant on Equity:
CFIcode=RWSCCX

Option - first 5 characters of CFIcode are required.

- Call Option on Futures, European exercise and Cash settlement:
CFIcode=OCEFCX
- Put Option on Equities, European exercise and Cash settlement:
CFIcode=OPESCX
- Put Option on Equities, American exercise and Physical delivery:
CFIcode=OPASPX
- Multileg instrument consists of Options
CFIcode=OMXXXX, SecurityType=MLEG

Future - first 3 characters of CFIcode are required.

- Future on Equity and Physical delivery:
CFIcode=FFSPXX
- Future on Oil and Physical delivery:
CFIcode=FCEPXX
- Future on Currency and Cash settlement:
CFIcode=FFCCXX
- Multileg instrument consists of Futures
CFIcode=FMXXSX, SecurityType=MLEG, SecuritySubType={Calendar},{Vertical},{Butterfly} etc

Forwards

- general case
CFIcode=MMFXXX

Combinations(Combo, Multileg Instruments)

- Multileg instrument consists of Futures
CFIcode=FMXXSX, SecurityType=MLEG, SecuritySubType={Calendar},{Vertical},{Butterfly} etc
- Multileg instrument consists of Options
CFIcode=OMXXXX, SecurityType=MLEG
- These are general that aren't fallen into multileg above.
CFIcode=MMMXXX, SecurityType=MLEG, SecuritySubType={Calendar},{Vertical},{Butterfly} etc

The standard operations of the Instrument Service

- **Login** - Login
- **Logout** - Logout
- **FindInstrumentsQuery** - Query for Instruments
- **InstrumentsSubscription** - Start/Stop instrument subscription
- **TickRuleTablesQuery** - Query for a tick rule
- **TickRuleTablesSubscription** - Start/Stop tick rule subscription
- **OrderBooksSubscription** - Start/Stop an OrderBooks Subscription.

Login, Logout Operations

Are the same as the other services.

FindInstrumentsQuery

The FindInstrumentQuery is currently not used by either Orc Trader or Orc Liquidator. This query has an identically reply to the snapshot part of opening an InstrumentsSubscription.

InstrumentsSubscription

Opens a subscription for instrument, instrument legs, and OrderBooks. When subscribing to this subscription you will first receive a snapshot of all available instruments, instrument legs and OrderBooks. If the market supports it you will then receive further updates when new instruments are added.

The instruments in the snapshot should be sent in the following order:

1. spots, index
2. futures, warrants
3. options
4. combos

Opening a Subscription

Send an InstrumentsSubscriptionControlReq with the SubscriptionRequestType set appropriately. The gateway will respond with a snapshot of InstrumentFeed, InstrumentLegFeed, and OrderBookFeed messages. The snapshot is finished when a SnapshotEndRsp is sent. This subscription will send intra-day instrument definitions asynchronously to its clients.

Closing a Subscription

Send an InstrumentsSubscriptionControlReq with the SubscriptionRequestType equal to Unsubscribe or log off of the service.

TickRuleTablesQuery

The TickRuleTablesQuery is not currently used by either Orc Trader or Orc Liquidator. This query has an identically reply to the snapshot part of opening a TickRuleTablesSubscription.

TickRuleTablesSubscription

This subscription provides a list of all available TickRules. At least one tickrule table is required. Multiple different instruments can refer to the same tickrule table. A tick rule table describes what the valid tick-sizes for different price-levels. The last tick rule in a tick rule table is valid for all price-levels equal to and above that price. This subscription will send newly defined tick rule definitions asynchronously to its clients.

Opening a Subscription

Send a TickRuleTablesSubscriptionControlReq with the SubscriptionRequestType set appropriately. The gateway will respond with a snapshot of TickRulesTableFeedmessages. The snapshot is finished when a SnapshotEndRsp is sent.

Closing a Subscription

Send an TickRuleTablesSubscriptionControlReq with the SubscriptoinRequestType equal to Unsubscribe or log off of the service.

OrderBooksSubscription

This subscription will asynchronously send OrderBookFeeds for information on existing contracts that can dynamically change, such as lot size.

Instrument Service

Creating CFICode Using ISO 10962

- [ISO 10962 Classification of Financial Instruments \(CFI code\)](#)
 - [Categories](#)
 - [E – Equities](#)
 - [D – Debt Instruments](#)
 - [R – Entitlements\(Rights\)](#)
 - [O – Options](#)
 - [F – Futures](#)
 - [M – Others\(Miscellaneous\)](#)
 - [Details](#)
 - [Equities description](#)
 - [Debt Instruments description](#)
 - [Entitlements\(Rights\) description](#)
 - [Options description](#)
 - [Futures description](#)

If the market exchange does not provide CFI code or provides incomplete CFI code we need to attempt to create it ourselves using the ISO 10962 standard.

ISO 10962 Classification of Financial Instruments (CFI code)

The CFI code consists of six alphabetic characters. The first character indicates the highest level of classification and differentiates between six generic categories: Equities, Debt instruments, Entitlements (Rights), Options, Futures, and Others. The second character indicates specific groups within each category: Equities, for example, are broken down into Shares, i.e. common/ordinary, Preferred shares, Convertible shares, Preferred convertible shares, Preference convertible shares, Units, i.e. unit trusts/mutual funds/OPCVM/OICVM, and Others. Within the category Debt instruments, the groups are Bonds, Convertible bonds, Bonds with warrants attached, Medium term notes, Money market instruments, and Others.

The four last characters indicate the most important attributes applicable to each group: whereas voting rights, restrictions, payment status and form are useful information in Equities, these features do not exist for Options, which have other attributes (underlying instruments, type of scheme, delivery, standardized/non-standardized).

Categories

- E – [#Equities](#)
- D – [#Debt Instruments](#)
- R – [Entitlements\(Rights\)](#)
- O – [#Options](#)
- F – [#Futures](#)
- M – [Others\(Miscellaneous\)](#)


E – Equities


Financial instruments representing an ownership interest in an entity or pool of assets.

Category	#Group	1st attribute	2nd attribute	3rd attribute	4th attribute
E – Equities					

	S – Shares, i.e. common/ordinary	<u>#Voting right</u>	<u>#Ownership/transfer restrictions</u>	Payment status	<u>#Form</u>
		V – Voting N – Non-voting R – Restricted voting E – Enhanced voting	T – Restrictions U – Free	O – Nil paid P – Partly paid F – Fully paid	B – Bearer R – Registered N – Bearer/Registered Z – Bearer depository receipt A – Registered depository receipt
	C – Convertible shares			<u>#Income</u>	
		See above	See above	F – Fixed Rate Income C – Cumulative, Fixed Rate Income P – Participating Income Q – Cumulative, Participating Income A – Adjustable Rate Income N – Normal Rate Income	See above
	P – Preferred shares R – Preference shares F – Preferred convertible shares V – Preference convertibles shares		<u>#Redemption</u>		

		See above	R – Redeemable E – Extendible T – Redeemable/ext endible	See above	See above
	U – Units, i.e. unit trusts/mutual funds/OPCVM/OICVM	Closed/open-end	#Distribution policy	#Assets	
		C – Closed-end O – Open-end	I – Income funds G – Growth funds M – Mixed funds	R – Real estate S – Securities M – Mixed-general C – Commodities D – Derivatives	See above
	M – Others (Miscellaneous)	1st attribute	2nd attribute	3rd attribute	
		X – Not applicable/Undefined	X – Not applicable/Undefined	X – Not applicable/Undefined	See above

 Mixed Units consisting of share(s), bond(s) and warrant(s), share(s) and bond(s), share(s) and warrant(s), a number of shares are classified under the category "Equity", group "Other".


 Baskets are classified under the category "Equity", group "Other".

D – Debt Instruments

Financial instruments evidencing moneys owed by the issuer to the holder on terms as specified.

Category	Group	1st attribute	2nd attribute	3rd attribute	4th attribute
D – Debt Instruments					

	B – Bonds C – Convertible bonds W – Bonds with warrants attached T – Medium-term notes Y – Money market instruments M – Others (Miscellaneous)	#Type of interest	#Guarantee	#Redemption/Reimbursement	#Form
		F – Fixed rate Z – Zero rate/Discounted V – Variable	T – Government/Treasury guarantee G – Guaranteed S – Secured U – Unsecured/unguaranteed	F – Fixed maturity G – Fixed maturity with call feature C – Fixed maturity with put D – Fixed maturity with put and call A – Amortization plan B – Amortization plan with call feature T – Amortization plan with put L – Amortization plan with put and call P – Perpetual Q – Perpetual with call feature	See Equities Form

 Mixed units consisting of shares and debt instruments are classified under the category "Equity", group "Other" and bond with warrants attached build their own group within the category "Debt instruments", mixed units consisting of

- a number of debt instruments or
- debt instrument(s) and other(s) (e.g. insurance policies)

are classified under the category "Debt instruments", group "Other".

R – Entitlements(Rights)

Financial instruments providing the holder the privilege to subscribe to or to receive specific assets on terms specified.

Category	Group	1st attribute	2nd attribute	3rd attribute	4th attribute
R – Entitlements(Rights)					
	A – Allotment rights M – Others (Miscellaneous)				#Form
		x – Not applicable/Undefined	x – Not applicable/Undefined	x – Not applicable/Undefined	See Equities Form
	S – Subscription rights P – Purchase rights	Underlying assets			
		S – Ordinary shares P – Preferred shares R – Preference shares C – Convertible shares F – Preferred convertible shares V – Preference convertible shares B – Bonds O – Others	x – Not applicable/Undefined	x – Not applicable/Undefined	See Equities Form
	w – Warrants	Underlying assets	Type	Call/Put	

		B – Basket S – Stock-Equities D – Debt Instruments/Interest Rates T – Commodities C – Currencies I – Indices M – Others	T – Traditional warrants N – Naked warrants C – Covered warrants	C – Call P – Put B – Call and Put	See Equities Form
--	--	--	---	---	-----------------------------------

o – Options

Contracts which grant to the holder either the privilege to purchase or the privilege to sell the assets specified at a predetermined price or formula at or within a time in the future.

Category	Group (PutOrCall(201))	1st attribute (ExerciseStyle(1194))	2nd attribute	3rd attribute(SettlementMethod(1193))	4th attribute
o – Options					
	M – Others				
		X – Not applicable/Undefined	X – Not applicable/Undefined	X – Not applicable/Undefined	X – Not applicable/Undefined
	C – Call options P – Put options	#Type of scheme	Underlying assets	#Delivery	Standardized/non-standardized
		A – American E – European	B – Basket S – Stock-Equities D – Interest rate/notional debt securities T – Commodities C – Currencies I – Indices O – Options F – Futures W – Swaps M – Others	P – Physical C – Cash	S – Standardized N – Non-standardized

F – Futures

Contracts which obligate the buyer to receive and the seller to deliver in the future the assets specified at an agreed price.

Category	Group	1st attribute	2nd attribute(Settlement methodEnum(1193))	3rd attribute	4th attribute
F – Futures					
	F – Financial Futures	Underlying assets	#Delivery	Standardized/non-standardized	
		B – Basket S – Stock-Equities D – Interest rate/notional debt securities C – Currencies I – Indices O – Options F – Futures W – Swaps M – Others	See Options Delivery	See Options Standardized/non-standardized	x – Not applicable/Undefined
	C – Commodities Futures	Underlying assets			
		E – Extraction Resources A – Agriculture, forestry and fishing I – Industrial Products S – Services	See Options Delivery	See Options Standardized/non-standardized	x – Not applicable/Undefined

M – Others(Miscellaneous)

Financial instruments which do not meet categories as defined.

Category	Group	1st attribute	2nd attribute	3rd attribute	4th attribute
----------	-----------------------	---------------	---------------	---------------	---------------

M – Others(Miscellaneous)					
	R – Referential Instruments	Further grouping			
		C – Currencies T – Commodities R – Interest Rates I – Indices	X – Not applicable/Undefined	X – Not applicable/Undefined	X – Not applicable/Undefined
	M – Other assets (Miscellaneous)	Further grouping			
		R – Real Estate Deeds I – Insurance Policies E – Escrow Receipts F – Forwards P – Precious Metal Receipts M – Others (Miscellaneous)	X – Not applicable/Undefined	X – Not applicable/Undefined	X – Not applicable/Undefined

Details

Equities description

Letter	Name	Comment
	Group	
S	Shares, i.e. common/ordinary	Holders typically being entitled to vote and receive dividends. In the event of liquidation, holders of shares usually rank behind the entity's creditors and holders of preferred shares.

P	Preferred shares	Payment of dividends to holders normally takes preference over the payment of dividends to other classes of shares. In the event of liquidation, preferred shares normally rank above ordinary shares but behind creditors of the company.
R	Preference shares	Similar to the preferred shares, preference shares have a prior claim on dividends, and on assets in an event of corporate liquidation or dissolution. But preferred stock would take precedence over preference stock in respect of dividends and assets that may be available for distribution.
C	Convertible shares	Shares (common/ordinary) that, at the option of the holder, are convertible into other securities, at a designated rate. The conversion privilege may be perpetual or limited to a specific period.
F	Preferred convertible shares	Preferred shares (common/ordinary or preferred) that, at the option of the holder, are convertible into other securities, usually common shares, at a designated rate. The conversion privilege may be perpetual or limited to a specified period.
V	Preference convertibles shares	Preference shares (common/ordinary or preferred) that, at the option of the holder, are convertible into other securities, usually common shares, at a designated rate. The conversion privilege may be perpetual or limited to a specified period.
U	Units, i.e. unit trusts/mutual funds/OPCVM/OICVM	Securities representing a portion of assets pooled by investors: run by a management company whose share capital remains separate from such assets.

M	Others (Miscellaneous)	Equities which do not fit into any of the above Groups.
	Voting right	
V	Voting	Each share has one vote
N	Non-voting	Share has no voting right
R	Restricted voting	The shareholder may be entitled to less than one vote per share
E	Enhanced voting	The shareholder is entitled to more than one vote per share
	Ownership/transfer restrictions	
T	Restrictions	The ownership or transfer of the security is subject to special conditions)
U	Free	Unrestricted; the ownership or transfer of the security is not subject to special conditions)
	Form	
B	Bearer	The owner is not registered in the books of the issuer or of the registrar.
R	Registered	Securities are recorded in the name of the owner on the books of the issuer or the issuer's registrar and can only be transferred to another owner when endorsed by the registered owner.
N	Bearer/Registered	Securities issued in both bearer and registered form but with the same identification number.
Z	Bearer depository receipt	Receipt - in bearer form - for securities issued in a foreign market to promote trading outside the home country of the underlying securities.

A	Registered depository receipt	e.g. ADR; Receipt - in registered form – for securities issued in a foreign market to promote trading outside the home country of the underlying securities.
	Income	Indicates the kind of dividend income the shareholders are entitled to
F	Fixed Rate Income	The shareholder periodically receives a stated income.
C	Cumulative, Fixed Rate Income	The shareholder periodically receives a stated amount. Dividends not paid in any year accumulate and must be paid at a later date before dividends can be paid on the common/ordinary shares.
P	Participating Income	Preferred shareholders, in addition to receiving their fixed rate of prior dividend, share with the common shareholders in further dividend distributions and in capital distributions.
Q	Cumulative, Participating Income	Shareholders are entitled to dividends in excess of the stipulated preferential rate under specified conditions. Dividends not paid in any year accumulate and must be paid at a later date before dividends can be paid on the common/ordinary shares.
A	Adjustable Rate Income	The dividend rate is set periodically, usually based on a certain yield.
N	Normal Rate Income	Shareholders are entitled to the same dividends as common/ordinary shareholders but have other privileges, e.g. as regards distribution of assets upon dissolution.

	Closed/open-end	Indicates whether units are traded or whether funds continually stand ready to sell new units and to redeem the outstanding units on demand.
C	Closed-end	Units are sold on either an organized exchange or in the over-the-counter market and are usually not redeemed.
O	Open-end	Funds permanently sell new units to the public and redeem outstanding units on demand, resulting in an increase or decrease of outstanding capital.
	Distribution policy	Indicates the fund's normal distribution policy
I	Income funds	The fund regularly distributes its investment profits.
G	Growth funds	The fund normally reinvests its investment profits.
M	Mixed funds	Investment profits are partly distributed, partly reinvested.
	Assets	Indicates the investment policy/objective of the fund as set forth in its prospectus
R	Real estate	Fund invests exclusively in real estate.
S	Securities	Fund invests in securities/financial instruments.
M	Mixed-general	Fund invests in different assets.
C	Commodities	Fund invests exclusively in commodities.
D	Derivatives	Fund invests in derivatives.

Debt Instruments description

Letter	Name	Comment
	Group	
B	Bonds	Any interest-bearing or discounted security that normally obliges the issuer to pay the bondholder a contracted sum of money and to repay the principal amount of the debt.
C	Convertible bonds	A bond that can be converted into other securities.
W	Bonds with warrants attached	A bond that is issued together with one or more warrant(s) attached as part of the offer, the warrant(s) granting the holder the right to purchase a designated security, often the common stock of the issuer of the debt, at a specified price.
T	Medium-term notes	Negotiable debt instruments offered under a program agreement through one or more dealers upon request of the issuer. The program defines the terms and conditions of the notes.
Y	Money market instruments	Financial instruments designated at issuance as such with a short-term life, usually twelve months or less, e.g. treasury bills, commercial paper.
M	Others (Miscellaneous)	Debt instruments which do not fit into any of above Groups.
	Type of interest	
F	Fixed rate	All interest payments are known at issuance and remain constant for the life of the issue.
Z	Zero rate/Discounted	No periodical interest payments are made; the interest charge (discount) is the difference between maturity value and proceeds at time of acquisition.

V	Variable	The interest rate is subject to adjustment through the life of the issue; includes graduated, i.e. step-up/step-down, floating and indexed interest rates.
	Guarantee	Indicates, in the case of the issuer's insolvency, whether the debt issue is additionally secured
T	Government/Treasury guarantee	The debt instrument is guaranteed by a federal or state government.
G	Guaranteed	The debt instrument is guaranteed by an entity other than the issuer; not a federal or state government
S	Secured	A debt issue against which specific assets are pledged to secure the obligation e.g. mortgage, receivables
U	Unsecured/unguaranteed	The direct obligations of the issuer rest solely on its general credit.
	Redemption/Reimbursement	Indicates the retirement provisions made for the debt issue
F	Fixed maturity	The principal amount is repaid in full at maturity.
G	Fixed maturity with call feature	The issue may be called for redemption prior to the fixed maturity date.
C	Fixed maturity with put	The holder may request the reimbursement of his bonds prior to the maturity date.
D	Fixed maturity with put and call	
A	Amortization plan	Reduction of principal by regular payments.
B	Amortization plan with call feature	The redemption of principal may occur as the result of the outstanding portion of the bond being called.

T	Amortization plan with put	
L	Amortization plan with put and call	
P	Perpetual	The debt instrument has no fixed maturity date and is only due for redemption in the case of the issuer's liquidation.
Q	Perpetual with call feature	The issue may be called for redemption at some time in the future.

Entitlements(Rights) description

Letter	Name	Comment
	Group	
A	Allotment rights	Privileges allotted to existing security holders, entitling them to receive new securities free of charge.
S	Subscription rights	Privileges allotted to existing security holders, entitling them to subscribe to new securities at a price normally lower than the prevailing market price.
P	Purchase rights	Anti-takeover device that gives a prospective acquires shareholders the right to buy shares of the firm or shares of anyone who acquires the firm at a deep discount to their fair market value.
W	Warrants	Financial instruments which permit the holder to purchase a specified amount of a financial instrument, commodity, currency or other during a specified period at a specified price.
M	Others (Miscellaneous)	Entitlements (Rights) which do not fit into any of above Groups.

	Warrants Underlying assets	Indicates the type of underlying assets that the warrant holder is entitled to acquire
B	Basket	The warrant holder is entitled to acquire a package or group of assets.
S	Stock-Equities	The warrant holder is entitled to acquire equity.
D	Debt Instruments/Interest Rates	The warrant holder is entitled to acquire debt instruments.
T	Commodities	The warrant holder is entitled to acquire a specific commodity.
C	Currencies	The warrant holder is entitled to acquire a specified amount in a certain currency at a specified exchange rate.
I	Indices	The warrant holder is entitled to acquire a specified amount based on the performance of an index.
M	Others	Miscellaneous; the warrant holder is entitled to acquire other assets not mentioned above.
	Warrant Type	Indicates whether the warrant is issued by the issuer of the underlying instrument or by a third party
T	Traditional warrants	issued by the issuer of the underlying instrument
N	Naked warrants	issued by a third party which is not the issuer of the underlying securities to which the warrant refers. The warrant issuer does not hold as many securities as would be required if all the warrants are exercised.

C	Covered warrants	issued by a third party which is not the issuer of the underlying securities to which the warrant refers. The warrant issuer holds as many securities as would be required if all the warrants are exercised.
	Warrant Call/Put	Indicates whether the warrant entitles the holder to acquire assets at specified terms or to acquire cash in exchange for specific underlying assets.
C	Call	As in most cases, the warrant entitles the holder to acquire specific underlying assets during a specified period at a specified price.
P	Put	The warrant entitles the holder to acquire cash in exchange for specific underlying assets.
B	Call and Put	warrants with neither call nor put feature, warrants with call and put feature

Options description

Letter	Name	Comment
	Group	
C	Call options	Contracts between a buyer and a seller giving the buyer (holder) the right, but not the obligation, to buy the assets specified at a fixed price or formula, on or before a specified date. The seller of the call option assumes the obligation of delivering the assets specified should the buyer exercise his option.

P	Put options	Contracts between a buyer and a seller giving the buyer (holder) the right, but not the obligation, to sell the assets specified at a fixed price or formula, on or before a specified date. The seller of the put option assumes the obligation of buying the assets specified should the buyer exercise his option.
M	Others (Miscellaneous)	Options which do not fit into any of the above Groups
	Type of scheme	Indicates whether an option can be exercised at a specific date or within a defined period
A	American	The option can be exercised at any time between its issuance and expiration date.
E	European	The option can be exercised on its expiration date.
	Underlying assets	Indicates the type of underlying assets that the option holder is entitled to buy, respectively to sell
B	Basket	The option gives the right to buy, respectively to sell a package or group of assets.
S	Stock-Equities	The option gives the right to buy, respectively to sell equity.
D	Interest rate/notional debt securities	The option gives the right to buy, respectively to sell existing or notional/fictitious debt instruments with a specific interest rate and maturity.
T	Commodities	The option gives the right to buy, respectively to sell a specific commodity.
C	Currencies	The option gives the right to buy, respectively to sell a specified amount in a certain currency at a specified exchange rate.

I	Indices	The option gives the right to buy, respectively to sell a specified amount based on the performance of an index.
O	Options	The option gives the right to buy, respectively to sell options.
F	Futures	The option gives the right to buy, respectively to sell futures.
W	Swaps	The option gives the right to buy, respectively to sell swaps.
M	Others	Miscellaneous; The option gives the right to buy, respectively to sell other instruments not mentioned above.
	Delivery	Indicates whether the settlement of the option when exercised is made in cash or whether the underlying instruments are delivered
P	Physical	The underlying instrument must be delivered when the option is exercised.
C	Cash	The settlement of the option is made in cash.
	Standardized/non-standardized	Indicates whether the terms of options (underlying instruments, strike price, expiration date, contract size) are standardized or not
S	Standardized	The underlying instruments, exercise price, expiration date and contract size of the options are standardized. These options are traded on special option exchanges.
N	Non-standardized	The options are custom-made instruments normally sold over the counter. Underlying instruments, strike price, expiration date and contract size of the options are not standardized.

Futures description

	Groups	
F	Financial Futures	Futures contracts based on a financial instrument
C	Commodities Futures	Futures contracts based on bulk goods
	Financial Underlying assets	Indicates the type of underlying assets that the futures buyer receives, respectively that the seller delivers
B	Basket	The buyer receives, respectively the seller delivers a package or group of assets.
S	Stock-Equities	The buyer receives, respectively the seller delivers equity.
D	Interest rate/notional debt securities	The buyer receives, respectively the seller delivers existing or notional debt instruments with a specific interest rate and maturity.
C	Currencies	The buyer receives, respectively the seller delivers a specified amount in a certain currency at a specified exchange rate.
I	Indices	The buyer receives, respectively the seller delivers a specified amount based on the performance of an index.
O	Options	The buyer receives, respectively the seller delivers options.
F	Futures	The buyer receives, respectively the seller delivers futures.
W	Swaps	The buyer receives, respectively the seller delivers swaps.
M	Others	Miscellaneous; The buyer receives, respectively the seller delivers other instruments not mentioned above.

	Commodity Underlying assets	Indicates the type of underlying assets that the futures buyer receives, respectively that the seller delivers
E	Extraction Resources	Metals, Precious Metals, Coal, Oil, Gas
A	Agriculture, forestry and fishing	Corn, Coffee, Soybeans, etc...
I	Industrial Products	Construction, Manufacturing
S	Services	Transportation, Communication, Trade

[Instrument Service](#)

Market Data Service

The Market Data Service is used to provide clients with BBOs (best bid & offer), depth of market, public trades, and news for an exchange. This service is typically provided along with the Instrument Service by the same gateway, since instruments and price feeds are typically provided by the same interface from an exchange.

- [Market Data Service Behavior](#)
-

< [Instrument Service](#) | [Document Overview](#) | [Order Service](#) >

Market Data Service Behavior

- [Login/Logout](#)
- [Market Data Subscriptions](#)
- [TCP/IP Communication - Not UDP Multicast](#)
- [TopOfBook](#)
- [Depth](#)
- [PublicTrade](#)
- [News](#)
- [Undocumented Subscriptions](#)

The Market Data Service is used to distribute market data information from the market to the Orc system. One of the requirements on the Market Data Service is that it uses the same OrderBookIDs to identify the instruments as the Order Service used to enter orders. (The OrderBookID is currently equivalent to the Orc Feedcode for the instrument). Most Market Data flow is from the gateway to the client. The clients open a subscription for something and then data flows from gateway to client.

For most exchanges, the implementation of the Market Data Service and the Instrument Service are provided by the same process. This is typically so because the reference data and the market data are often provided by the same system from the exchange.

Login/Logout

When a client wishes to use a Market Data Service, it starts by sending a **LoginRequest** to the service. The LoginRequest includes Orc username, Orc group and service version. When a client wishes to stop using a service it will send a **LogoutRequest**. In the current Orc systems the same user can only have one simultaneous login to any server or service. A user from an Orc perspective is uniquely identified with username+group.

Market Data Subscriptions

A client can subscribe to various types of streams of market data for different instruments. The four types are

- **TopOfBookFeed**: Includes best-bid-offer, as well as other statistics for the instrument (such as last price, turnover etc).
- **DepthFeed**: Is used to get the whole OrderBook. The DepthFeed is of the incremental-delta kind and each feed-message only describes one or more changes to the OrderBook.
- **PublicTradeFeed**
- **NewsFeed**

The TopOfBookFeed is a thin stateful feed, in that only updated fields are sent on each update, the client must assume the other fields to be the same as they were before.

To initiate a subscription of a certain type on one instrument the client first sends a **SubscriptionControlRequest** identifying what instrument to subscribe to:

- For TopOfBookFeed the client is guaranteed that an initial TopOfBookFeed message will be sent by the service followed by update messages (but the client cannot discern whether it receives the initial message or just an ordinary update). There is no SnapshotEndRsp necessary after the initial TopOfBookFeed snapshot is sent since there will always be just one message in the snapshot.
- For DepthFeed, the service responds by first sending a snapshot of the current state, followed by a SnapshotEndRsp. As soon as there is any update of the specified type to the instrument the service will send a message with the update information to the client.

- For PublicTradeFeed and NewsFeed there is no snapshot sent, but as soon as there is a new public trade a PublicTradeFeed message is sent by the service.

TCP/IP Communication - Not UDP Multicast

Currently, in the Orc software architecture, the gateways always communicate via TCP/IP to all connected clients, not UDP. You need to manage subscriptions in your gateway and send each message to whichever clients are interested, which may involve sending the message multiple times.

TopOfBook

TopOfBookFeed provides best-bid-offer (BBO) (i.e. the top-row of both sides of the OrderBook), statistics and some trading status for an instrument. OrderBookID is always required in this message. If sub-market is supported, SubMarketID should be used to identify the update is belong to which sub-market.

Basic BBO information is sent using BidPx, BidSize, OfferPx and OfferSize.

ImpliedBidSize and ImpliedOfferSize can be used for Implied-orders for markets that have those.

BidAtMarket and OfferAtMarket are used to indicate that the current best bid/offer is an AtMarket-order.

The basic statistics fields are LastPx, LastQty, OpeningPrice, ClosingPrice, HighPx, LowPx, TotalTurnover & TotalVolumeTraded.

SecurityTradingStatus and SecurityTradingStatusDesc should be provided and are used to describe the trading status of the instrument (the Desc field is typically how the trading status should be displayed in a client).

FeedStatus is used to indicate if there are any types of problems with the incoming market data feed from the exchange (e.g. if the feed is temporarily out-of-sync).

- BBO - Same as the top-most-level of the OrderBook Depth
- Statistics
- Status of the instrument
 - The SecurityTradingStatus representing the trading status of the instrument on the exchange
 - The FeedStatus representing our market data connectivity for the instrument (up/down/recovering)

TopOfBook updates are thin - an initial snapshot of the state is sent at the beginning of a subscription after that updates are thin, meaning that the omission of a field means that it is the same value as before.

To remove a side of the BBO, clear any of the BBO fields that are set. Example clearing the bid: Set BidPrice, BidSize, NumberOfOrder, ImpliedBidSize to 0 and BidAtMarket to false.

Depth

A DepthFeed contains one or more DepthLevels. DepthLevel entries are used to build up the OrderBook for an instrument.

There are three main types of DepthLevel entries: A new entry, a changed entry and a deleted entry. MDUpdateAction specifies which one it is. For all three types OrderBookID, MDEntryID and Side have to be included. For a deleted entry these fields are the only fields expected and required. MDEntryID is the unique identifier for the DepthLevel. After a DepthLevel has been deleted the same MDEntryID can be used for another DepthLevel entry.

A changed entry is identified by the MDEntryID. A new entry should include all relevant fields; this is typically at least Price and OrderQty in addition to the required fields. Additional fields can be added but are not strictly required. A

changed entry can include one or more fields that have changed (in addition to the required fields). The client is responsible for its own sorting of the DepthLevel entries. The Priority field can be used to indicate the order for entries that have the same price (if such an order exists). MDEntryPositionNo is not used by clients directly, but can be provided by the Market Data Service for debugging reasons so that a client can check if its sorting order is the same as the gateway's sorting order.

All DepthLevels in the same DepthFeed should be treated as an atomic unit. For example, if there are two DepthLevel entries in the DepthFeed one delete of an MDEntryID and one insert of the same MDEntryID. This should be the exact same thing as a change of the MDEntryID. Clients must ensure that they don't expose a half-processed DepthFeed to end-users. The DepthLevels within the DepthFeed must be processed by the client in the order they were received.

There are two types of markets

1. Those that send incremental deltas of the OrderBook (i.e. only changes)
2. Those that send snapshots of the OrderBook (either full-book or top-X)

There are two styles of depth:

1. Depth-by-order - where individual orders have own "rows" in the OrderBook
2. Depth-by-price - where multiple orders with the same price are coalesced by the market
Sometimes one market may change its style during the trading-day for example Depth-by-price during the opening phase and then switching to Depth-by-Order during ordinary trading.

Between the client and the gateway we always use incremental deltas regardless of the exchange model.

Between the client and the gateway we represent both depth-styles with the same messages.

The main concept to understand is the MDEntryID which uniquely identifies one level or "row" in an OrderBook. For Depth-by-Order, MDEntryID is often the Exchange OrderID. For Depth-by-Price MDEntryID is often constructed using "S+Price+I" where S is Side and I is set to I if it is an implied price (e.g. B47.50 or S18.30I), another normal MDEntryID for Depth-by-Price is "S+Position+I" (e.g. B1, S3) - the implication of this latter model is that Depth-updates become bigger (e.g. if you first have one order buy 1@1, then another buy 2@2, you would get Insert B1 1@1 followed by Modify B1 2@2, Insert B2 1@1). The important thing here is that the client does not or should not care how the MDEntryID is constructed, it is used to uniquely identify the row.

Any DepthFeed consists of one or more DepthLevels, each DepthLevel must have an MDEntryID.

One DepthFeed should be treated atomically by the client (i.e. all the changes described in the feed should be visible or none of the changes should be visible).

Two separate DepthFeeds need not be treated atomically by the client. A gateway has to ensure that updates that must belong together are kept in the same DepthFeed (for example if the exchange is of the snapshot-kind, a new snapshot from the exchange, may result in all existing DepthLevels being deleted).

If DepthFeed is updated per sub-market, DepthLevels should contain SubMarketID to identify the DepthLevel is updated to that sub-market. This sub-market depth update is only supported for those markets that send snapshots of the OrderBook (either full-book or top-X).

When a subscription is requested - the first message that is sent is a snapshot, after the snapshot there is a SnapshotEndRsp, after that you get the normal updates. If there is no snapshot there is just a SnapshotEndRsp.

PublicTrade

This subscription reports public trades. Currently, it is stateless so you only receive public trades that happens after

you subscribe, public trades that happened earlier in the day are not retrievable.

The public trade feed contains the fields you would expect (OrderBookID, LastPx, LastQty) answering the questions what instrument at what price and how many. In addition the gateway typically provides additional fields describing the trade: ExecID, TrdType, ExecType, ExchCreationTime, MDEntryBuyer, MDEntrySeller, SecondaryExecID.

If sub-market is supported, SubMarketID should be used to identify the update is belong to which sub-market.

News

The NewsFeed provides news for a particular instrument. News means some sort of human readable news relating to the instrument that has been sent out, typically intended to be displayed in some sort of client ticker.

Undocumented Subscriptions

The following subscriptions are not yet ready for public use but are Orc internal:

- RFQSubscription - public RFQs sent to a set of market participants publicly.

[*Market Data Service*](#)

Order Service

The Order Service is used to enter, modify, and cancel orders, as well as to receive trades from those orders. It is also used for Request For Quote (RFQ) and instrument creation functionality.

- [Order Service Behavior](#)

< [Market Data Service](#) / [Document Overview](#) / [Trade Report Service](#) >

Order Service Behavior

- [The standard operations of Order Service](#)
- [Additional operations of Order Service](#)
- [Common field semantics](#)
 - [Operation semantics](#)
- [CreateOrder](#)
- [Modify Order - Order modification](#)
 - [Message ordering](#)
 - [Defined rejections](#)
- [Order Cancellation](#)
 - [Client requesting Order Cancellation \(CancelOrderReq\)](#)
 - [Server responding to an Order Cancellation \(CancelOrderRsp\)](#)
 - [Cancel All Orders](#)
 - [Order RetainFlag handling](#)
- [Orders Subscription flow](#)
 - [OrdersSubscriptionControlReq](#)
 - [OrderFeed](#)
 - [OrdStatus State Diagram](#)
 - [OrdersSnapshotEndRsp](#)
 - [Updates should be provided when...](#)
- [Order inactivation/activation](#)
- [Trades subscription](#)
 - [TradesSubscriptionControlReq](#)
 - [TradeFeed](#)
 - [Unsolicited Trades](#)

The Order Service is used to enter orders and view resulting trades from those orders. In addition, Order and Trade flows of other team members can be viewed.

The standard operations of Order Service

- **CreateOrder**
- **ModifyOrder**
- **CancelOrder**
- **CancelAllOrders** - remove all active Orders in one operation.
- **OrdersSubscription** - to view the Order flow.
- **TradesSubscription** - to view the Trade flow for your orders

Note that when doing operations with Order (Create, Modify or Cancel) the Order flow (but not the Trade flow) must be primarily analyzed using the responses for these operations. Because when a new Order is created, the creation information is sent in the response but not on the OrderFeed for the user that entered the order, this is to avoid unnecessary duplication (to all other users new Order is reported via OrderFeed). OrderFeed flow is still important for the order Owner because it brings unsolicited changes from the exchange (cancellation, volume reduction, stop order triggering, price for Market to Limit order).

Additional operations of Order Service

- **EnterRFQ** is used to request a quote on an instrument to get prices, typically relevant for a less liquid instrument.

- **CreateInstrument** and **CreateCombo** are used to define new instruments and combos for trading, for example issued on-demand option series and user defined exchange traded combinations (strategies).

The reason why those operations belong to the Order Service is that actions above are normally done as preparation for sending an order. If instrument are already available from the Instrument Service and has prices from Market Data Service - above operations are not needed and should not be used.

Common field semantics

- **OrderID** - OrderID + group is a unique combination. This means that in two different groups the same OrderID can be used and refers to different orders. If you get an order insert from user B when there is an active (or inactive) order with the same OrderID and Group it should be rejected with the reason "Duplicate order". (Unlikely to happen in practice). Order OrderID is completely independent from Quote OrderID. In Orc Trader, OrderID in Order is order number allocated by CDS. In Liquidator OrderID is a successive odd counter which starts from OrderID returned with OrdersSnapshotEndRsp (by default start from 1). In OrdersSnapshotEndRsp you return max OrderID among active and inactive orders from users belonging to the same group as subscription Owner. As you can see the result depend on the cleanup procedure for inactive orders and quotes. We do not cleanup it straight away because delayed trades sometime occur.

Operation semantics

The semantics of the normal order operations is similar to the corresponding FIX operations. Semantically, there are a lot of small differences: For example in FIX any change of an order results in a new order (with new ClOrderID), in the Orc world, the gateway uses the same OrderID before and after the modification. If the underlying market does not support modification of the market order, the gateway takes care of this without exposing this to the end user. This means that one Orc order can be different market orders during different parts of the order's lifetime. The "OrdStatus" potential states are fewer than in FIX, in particular Expired and DoneForDay do not exist but the order is instead in Cancelled and Suspended respectively.

CreateOrder

CreateOrderReq is used to enter a new order. The following fields must always be present:

- **MarketInstrumentId**
- **Side**
- **OrderID**
- **OrderQty**
- **OrdType**: LIMIT price is required. MARKET price is NOT allowed to be set.
- **TimeInForce**: TimeInForce is handled differently than in the FIX spec, in FIX, the absence of TimeInForce means day-order, the rationale for this is that we would like to avoid default values at this high-level (in the actual encoding/decoding TimeInForce may be omitted but that is an encoding/decoding optimization). ExpireDate is required if TimeInForce is GoodTillDate and should never be used otherwise.

Optional fields:

- **RetainFlag** specifies whether the orders should be retained or removed from the market if the client disconnects.
- **RawData** is used to associate client specific data with an order. On any status feedback on the order the RawData will be passed back. Liquidator is a frequent user of this field to avoid keeping own state on orders.
- **Portfolio** is normally used to identify an Orc portfolio and on any status feedback on the order the Portfolio is passed along.
- **CumQty** is used for inactive order activation (see below).
- **SubMarketId** is used to tell gateway side the order belongs to which sub-market.

Some gateways require PositionEffect to be set. (See FIX for definition). This specifies whether the order should

open or close a position.

All gateways must report which of the `OrdTypeEnum` order types they support as part of their capabilities. A client can send multiple `CreateOrderReq` in sequence (with different `OrderID`) without waiting for `Rsp`.

Deprecated: **CumQty** - `CumQty` can be sent in an order. If `CumQty` set, the real `OrderQty` is `OrderQty - CumQty`. This comes from a preexisting behavior in Orc Trader and its handling of synthetic iceberg orders. This field in `CreateOrderReq` will be removed over time and Orc Trader will stop doing this. `CumQty` must not be sent by other clients than Orc Trader. Orc Trader will not send `CumQty` when ice-bergs are supported natively (`<open-volume-supported value="true"` in the dynamic order panel).

CreateOrderRsp is always sent in response to a `CreateOrderReq`. The following fields must always be present:

- **OrderID**
- **OrdStaus**
 - If there was some problem with the order or `CreateOrder` failed, `OrderStatus` is "Rejected" and **OrdRejReason** is required. `OrdRejReason` should not be present if `OrderStatus` is any other value than `Reject`. Reasons normally includes a free text description of what went wrong for `Reject`.
 - `OrdStatus` can never be in any of the Pending states in this reply.

Optional Fields:

- **SecondaryOrderID** is the unique identifier used by the market to identify the order.
- **Reason** is text field that is required if the order was rejected. Optional if the order is not rejected (the text is displayed in logs and to client).
- **ExchCreationTime** is a best effort attempt to tell when the exchange created the order in local time. If the market does not provide it, the gateway should use the timestamp when it received the info from the market.
- **RawData** Same as in the request. If this field is received in the request, always set the same value in the response.
- **Portfolio** Same as in request. If this field is received in the request, always set the same value in the response.
- **RetainFlag** Same as in request. If this field is received in the request, always set the same value in the response.

Modify Order - Order modification

ModifyOrderReq is sent to change an order and the `OrderID` is required. `OrderID`, `OrderStatus` and `Side` can never be modified (e.g. `Side`, `OrdType`, `TimeInForce` cannot be changed instead the order has be delete and reinserted). The only things that can be changed are the things that are associated with a certain `OrdType` and a certain `TimeInForce`. For GTD orders, `ExpireDate` can be changed. A gateway should try to support as much as possible of this. However, non-market-reaching fields can also be changed: The following are system local values that can be changed and that should not have any impact on the market: `PortfolioID`, `Retain`, `MetaData`, `Owner`.

To the largest extent possible gateways should support order modification even when the underlying exchange does not support it (emulating order modification with delete insert). Of course we cannot completely hide the effects of the emulation and a market gateway that has to emulate order modification with delete+insert should report this as a specific capability.

Empty SD string fields can be sent by the client, it is up to the gateway to determine if the field is sent to the exchange.

ModifyOrderRsp is always the response to a `ModifyOrderReq` no matter what the result is. `OrderID` and `OrderStatus` are required in the answer. If the `ModifyOrder` failed `CxlRejectReason` is required to be set.

Message ordering

A client is not allowed to send a `ModifyOrderReq` on a specific `OrderID` before a previous `ModifyOrderReq` has received a `Rsp`. In addition, no `ModifyOrderReq` may be sent on an order before `CreateOrderRsp` has been received. Nor is a client ever allowed to send a `ModifyOrderReq` after it has sent a `CancelOrderReq` (unless the `CancelOrderRsp` is a reject, when `Replace` is again possible).

To support external cancellations - it is possible that one client sends a `CancelOrderReq` and following that a `ModifyOrderReq` is received from another client. This needs to be handled by the gateway - and the right message needs to be sent to each client (the `ModifyOrderReq` should be rejected if it is received after the Cancellation etc.etc.).

Defined rejections

`ModifyOrderReq` received must always be rejected if a previous `CancelOrderReq` has been received.
`ModifyOrderReq` received must always be rejected if the order has been completely filled.

A gateway that cannot 100% guarantee avoiding overfills when a certain change is made to the quantity or price must reject the request rather than trying to complete it

Order Cancellation

Client requesting Order Cancellation (`CancelOrderReq`)

The primary model is to accept a **`CancelOrderReq`** as often as possible. The motivation for this is that for a client sending a cancellation it is often crucial that the cancellation executes as rapidly as possible - a **`CancelReject`** and a subsequent retry involves additional networking communication (potentially long-distance) potentially delaying the cancellation longer than necessary. The standard rule is if it can be determined right now that the cancellation is not possible send a reject instantly, otherwise wait. If it is a pending new order, waiting for ack from the market, the cancel should still be accepted and it is up to the gateway to request the cancellation as soon as the exchange allows you to cancel.

`CancelOrderReq` is sent to cancel a particular order. `OrderID` must be sent in. A `CancelOrderReq` can be sent while there still is an outstanding `CreateOrderReq` or `ModifyOrderReq` on the `OrderID`. The gateway is responsible for queuing the cancel if it has to and process it as soon as it can.

Server responding to an Order Cancellation (`CancelOrderRsp`)

`CancelOrderRsp` is always sent as a reply to `CancelOrderReq`. **`OrderID` must always be set** in response to the same ID as received in `CancelOrderReq`. `CancelOrderRsp` contains both status of the request `RejReason` and status of the original order `Order->OrdStatus`. **`OrderStatus` is a required field and it must be set to the current order status**. If the cancellation succeeded `OrdStatus=Canceled` must be set. The gateway should also provide `CxlQty` for a successful cancellation (inability to provide `CxlQty` is a metadata property the gateway must report). If a gateway supports `CxlQty` `CxlQty` should only be provided if the cancel was successful and `CxlQty` can never be zero (then the cancel was not successful). If cancellation failed **`RejReason=Error` must be set** and `OrdStatus` must be set to the current order status. `Reason` and `CxlRejReason` may provide additional explanation of the cause of failure. Cancellation of an order which does not exist e.g. was just completely filled or unknown to gateway **must fail** and return corresponding `OrdStatus`.

Example of the cancel reply for just filled order:

```
CancelOrderRsp={RejReason=Error|Reason=Too late to cancel|CxlRejReason=0|Order={OrderID=1073749729|OrdStatus=Filled}}, for unknown order: CancelOrderRsp={RejReason=Error|Reason=Cannot find original order|CxlRejReason=1|Order={OrderID=1073749730|OrdStatus=Rejected}}
```

ExchModificationTime is the time the exchange reported that the cancellation was done and is only provided if the cancellation succeeded.

Cancel All Orders

CancelAllOrdersReq is sent to cancel all orders created by the specified Owner. If no Owner is specified it means the user in this session. If Orc at any future point in time will support multiple connections from the same user, CancelAll will still mean remove all orders owned by the specified user. All CreateOrderReq messages received before this message should be canceled by this request, CreateOrderReq messages received after the CancelAllOrdersReq must not be affected. Due to the effects of queue, message-transmission etc, the exact order of messages processed by the gateway can be hard for the client to predict.

CancelAllOrdersReq can also be used to cancel orders from set of users using wildcards in Owner field. The following wildcard forms of Owner field are supported:

- Owner=*<group>
to cancel all order submitted by users belonging to the <group>
- Owner=*.
to cancel all orders from all users

Order RetainFlag handling

Every Order has **RetainFlag** which specifies if the gateway should carry out cancellation of unattended orders on its own behalf. RetainFlag is False or not specified by default which means that orders are not retained. The gateway should avoid having any active non retained orders at exchange of which Owner is not connected to the gateway. This means gateway should cancel non retained client orders

- upon client logout from gateway
- upon client disconnect from gateway
- if gateway is disconnected from exchange and after re-connect the Owner of non retained order is not connected
- upon gateway shutdown
- upon gateway startup

If RetainFlag is set to True the gateway should never try to cancel the order without explicit client request.

For GTC/GTD orders the client should set the RetainFlag to True. Otherwise, the GTC/GTD order will be cancelled by the gateway upon disconnect.

Orders Subscription flow

OrdersSubscriptionControlReq

An OrdersSubscriptionControlReq is sent to start subscribing to orders. As part of the subscription the user can provide Owner. The following format of Owner field are supported:

- Owner=full username e.g. <user>:<group>
to receive updates of all order submitted by specified user
- Owner=*<group>
to receive updates of all order submitted by users belonging to the <group>
- Owner=*.
to receive updates of all orders from all users
- If Owner is not specified subscription is on orders from the user in this session.

OrderFeed

Upon receiving an OrderSubscription the gateway should send a snapshot consisting of an OrderFeed for each active order associated with the subscription and from then on send an OrderFeed for each update associated with the subscription.

In the snapshot every field associated with the OrderFeed should be sent.

The following fields of Order are always provided in updates:

- **OrderID.**

Order updates are provided for every new order, trade, replace and cancellation.

On a new order:

- **OrderQty** must be set
- The following should be set if they were received in the CreateOrderReq: **PositionEffect** and **PortfolioID**, **MetaData**, **Owner**.

On a trade:

- **CumQty** must be set
- **AvgPx** should be set if provided by exchange, optional otherwise.

For every successful cancel:

- **CxlQty** should be set (if a gateway does not support CxlQty this is a capability that must be true).

For every successful ModifyOrder all the modified fields must be provided.

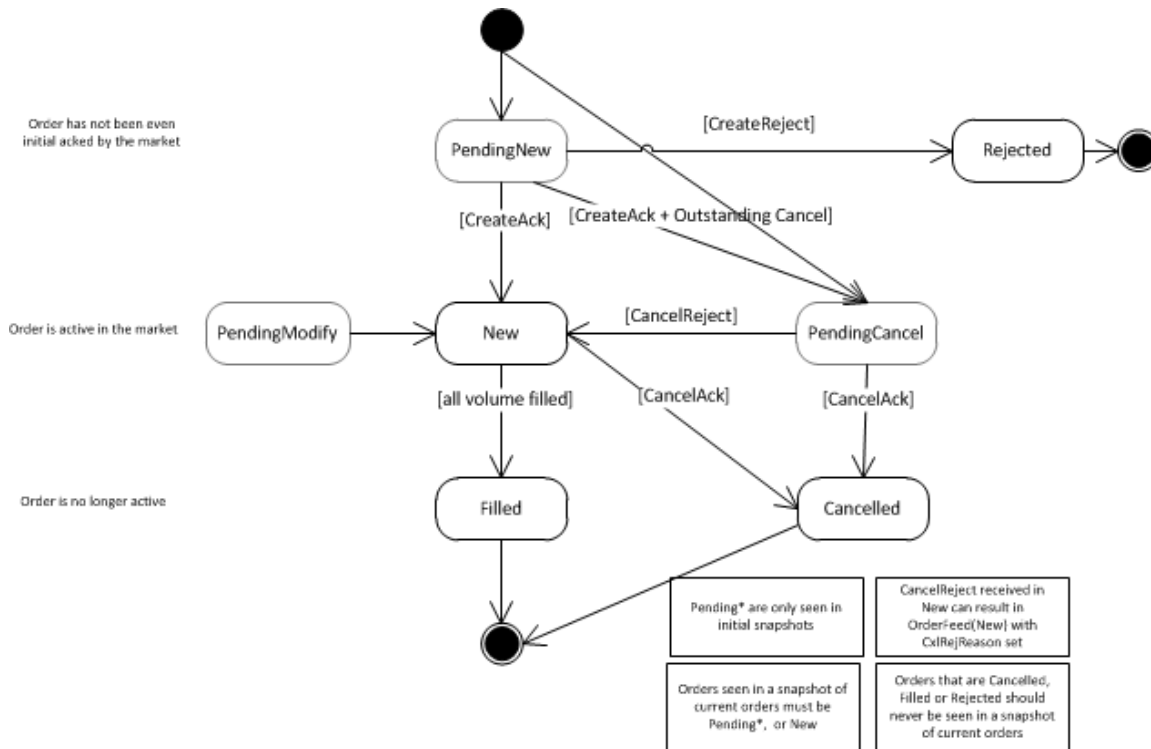
Only fields that have changed should be present in the OrderFeed updates.

$\text{OrderQty} = \text{open quantity} + \text{CumQty} + \text{CxlQty}$

Note that OrderFeed updates are **not** sent to the session that sent a Create/Modify/Cancel. Instead this information is provided in the response to those requests for that particular session. All other sessions however receive the OrderFeed on a Create/Modify/Cancel.

During the snapshot you do not need to send inactive orders (order that have been cancelled or completely filled) but it is recommended to keep filled orders in your cache for up to 3 days in case a trade bust is received and the OrderFeed needs to be updated. Cancelled orders that have not been filled can be removed from your cache almost immediately.

OrdStatus State Diagram



OrdersSnapshotEndRsp

After the snapshot of all active orders has been sent an OrdersSnapshotEndRsp message is sent indicating the end of the snapshot. One field in the OrdersSnapshotEndRsp is the (Max)OrderID this field indicates the highest seen OrderID for all orders for this user.

Updates should be provided when...

- A new order is accepted by the market (OrdStatus NEW)
- There is a trade (PARTIALLY FILLED, FILLED).
- An order is updated successfully with ReplaceOrder. (NEW, PARTIALLY FILLED, FILLED).
- An order is Cancelled successfully (CANCELLED).
- An order is suspended (e.g. end-of-day) (SUSPENDED)

Pending* are never used in the OrderFeed flow.

DoneForDay is not used instead the order is suspended.

Expired is not used instead the order is cancelled.

OrderFeed updates from different events can be coalesced.

Order inactivation/activation

Orc Trader has a feature that allows you to "inactivate" a working order. This will result in a cancel request being sent for the order but the order will remain in OT's list of working orders but will be marked as inactivated. If the order is then activated a new order request will be sent to the gateway with a completely new OrderID but the CumQty field will be set as the same CumQty as the previously inactivated order. This CumQty field needs to be set on any OrderFeed sent and any filled qty for the new order should be added to the existing CumQty.

Trades subscription

TradesSubscriptionControlReq

A TradesSubscriptionControlReq is sent to start subscribing to trades. Just like OrdersSubscriptionControlReq the user can provide Owner and the same format of Owner as in the OrdersSubscription is supported.

NumberOfDays='n' can also be set in which case for the snapshot the gateway is expected to publish a TradeFeed for each trade in the past 'n' days. For example: NumberOfDays=7, The gateway should publish snapshot of all the trades in the past 7 days.

It is recommended that the gateway cache trades for up to 7 days, but this should be easily configurable.

TradeFeed

A TradeFeed is published for each partially filled or completely filled order. Note that TradeFeeds may arrive/are allowed to arrive before CreateOrderRsp. In such cases OrderID or SecondaryOrderID must be set, so that the trade can be mapped to the CreateOrder.

The following fields are required:

- **ExecID:** ExecID should be eternally unique per market gateway.
- **OrderBookID**
- **LastPx**
- **LastQty**
- **Side**

Other common fields:

- **ExchCreationTime:** If not provided by exchange you may set to the time the gateway received the trade
- **MultiLegReportingType:** Required for multi-leg instruments
- **RawData:** Use the RawData field from the original OrderReq if set
- **Owner:** The full owner name <user:group> of the filled order

Unsolicited Trades

Sometimes a gateway is not able to match an incoming trade from the market with any order. Such trades must still be reported. In such case, for Orc gateways we use the configured defaultUser as Owner. These unsolicited trades need to have the required fields (as specified above set) as well as Owner set to defaultUser. For certain markets it is not possible to reconstruct the complete trade from the trade message itself (e.g. on INET OUCH you only receive price and quantity, not what instrument and not what side). In such cases, no TradeFeed message is sent.

For unsolicited trades a NotificationFeed should also be sent to defaultUser to inform about the unsolicited trade.

Order Service

Trade Report Service

The Trade Report Service is used to send and receive off-exchange trades. (It is NOT used to report trades from orders entered through the Order Service. Trades for those orders are reported on the Order Service.)

- [Trade Report Service Behavior](#)

< [Order Service](#) / [Document Overview](#) / [Quote Service](#) >

Trade Report Service Behavior

- [Description](#)
- [Prerequisites](#)
- [Negotiations](#)
 - [Counterparty Accept](#)
 - [Matching](#)
 - [Exchange Reject](#)
 - [Counterparty Decline](#)
 - [Cancellation](#)

Description

Trade reporting functionality in the Orc System utilizes negotiations between two parties about a trade outside of the exchange matching engine. This section describes example scenarios where two Orc clients, "userA" and "userB", perform such negotiations.

Since there are two parties involved in a trade reporting process the "request" terminology might seem confusing, but the expression "trade request" is a widely established concept for a transaction used in trade negotiations and it is customary to state that "userA sends a *trade request* and userB receives this *trade request*". However, in order to describe this process in service definition words, different names are required for the outgoing and incoming transactions, and the following terminology is used:

- **TradeReport** - outgoing from Orc system transaction.
- **TradeRequest** - incoming to Orc system transaction.

Orc users can send TradeReports and receive TradeRequests.

- Primary key for TradeReport is OrderID.
- Primary key for TradeRequest is TradeRequestID.
- TradeReportType is used to indicate the current stage of the trade reporting process (TradeReportTypeEnum=Submit/Accept/Decline/Cancel).
- OrdStatus is used to indicate if a TradeReport/TradeRequest is Active(OrdStatus=New) or Inactive(OrdStatus=Filled/Canceled/Rejected/Expired).

Prerequisites

- The trade reporting functionality is part of TradeReportService and both users need to login to this service in order to participate in negotiations.
- Next, both users should subscribe to the TradeReportFeed using the TradeReportsSubscription. The TradeReportFeed is a feed for the users own outgoing TradeReports. The TradeReportsSubscriptionControlReq is derived from TransactionsSubscriptionControlReq and requires the Owner parameter to be set. The NumberOfDays parameter will be ignored since TradeReports are only valid during the day. Initial snapshot for this subscription will return all active TradeReports with OrdStatus=New. Later this feed will broadcast changes in the TradeReport status, not related to own transactions, e.g. updates triggered by counterpart or exchange. Note that this means that TradeReportFeeds are **not** sent to the session that sent a EnterTradeReportReq/CancelTradeReportReq. Instead this information is provided in the response to those requests for that particular session. All other subscribed sessions however receive the TradeReportFeed on a Enter/Cancel.
- Additionally, both users should subscribe to the TradeRequestFeed using the TradeRequestsSubscription. This is a feed for incoming TradeRequests from the counterpart and the

TradeRequestsSubscriptionControlReq requires OrderBookID.

Negotiations

To initiate a negotiation, userA sends EnterTradeReportReq with TradeReportType=Submit and ContraBroker=userB. A unique identifier should be assigned for this TradeReport by the client, e.g. **OrderID=101**.

In case of a problem with this transaction, the gateway replies with EnterTradeReportRsp having RejReason==Error. An EnterTradeReportRsp without a RejReason field means that the transaction is successful and in the Orc Trader client, an active TradeReport will be displayed in the Transaction List. If there is an identifier assigned for the report by the exchange at this point, it will be returned as SecondaryOrderID=exchOrderX.

Counterparty Accept

userB will be notified about a new request from userA via the TradeRequestFeed with TradeReportType=Submit and ContraBroker=userA. In the Orc Trader client this request is displayed in the OTC window.


A unique identifier should be assigned for this TradeRequest by the gateway, e.g. **TradeRequestID=userA-101**. TradeRequestID should be a globally unique identifier since it is used by the client application as the primary key to manage all incoming requests. SecondaryOrderID, (not necessarily unique), may be set in the feed which likely equals the one returned to userA in previous step, e.g. **SecondaryOrderID=exchOrderX**.

To react on the request, userB needs to send his own TradeReport with **TradeRequestID=userA-exchOrderX** and **ContraBroker=userA**. The unique identifier should be picked up, e.g. **OrderID=202**. Note that the OrderID is unique per client, so it could be OrderID=101 again, but a different ID has been selected for clarity. The TradeReportType should be set to Accept or Decline, depending on the intention. userB will always get EnterTradeReportRsp which may have SecondaryOrderID=exchOrderY, not necessarily equal to the SecondaryOrderID assigned by the exchange to userA's request.

Similar to the notification to userB, userA will now be notified about the request added by userB via the TradeRequestFeed with **TradeReportType=Accept**, **TradeRequestID=userB-exchOrderY** and **ContraBroker=userA**. Additionally, the feed should have **OrderID=101** since it refers to the TradeReport already entered by userA.

At this point, both users have two active data entities: Incoming TradeRequest and outgoing TradeReport:

- userA has TradeReport with OrderID=101 and TradeRequest with TradeRequestID=userB-exchOrderY
- userB has TradeReport with OrderID=202 and TradeRequest with TradeRequestID=userA-exchOrderX.

 Note that userA's TradeReport and userB's TradeRequest still have TradeReportType=Submit which does not reflect the current state since it was accepted.

The correct status should be delivered via the TradeReportFeed for userA's TradeReport with OrderID=101 carrying updated TradeReportType=Accept and TradeRequestID=userB-exchOrderY.

And TradeRequestFeed for userB's TradeRequest with TradeRequestID=userA-exchOrderX carrying updated TradeReportType=Accept and OrderID=202. These updates may originate at exchange or at gateway.

Matching

Let us assume that exchange confirm negotiated trade. This will result in TradeFeed and TradeRequestFeed sent to each user. For userA it will be a TradeFeed with OrderID=101 matching TradeReport with same OrderID and TradeRequestFeed with TradeRequestID=userB-exchOrderY updating OrdStatus=Filled. For userB it will be a

TradeFeed with OrderID=202 and TradeRequestFeed with TradeRequestID=userA-exchOrderX respectively.

Exchange Reject

If the exchange does not confirm the negotiated trade, this should be reported by the TradeReportFeed and TradeRequestFeed sent to each client.

- For userA it should be a TradeReportFeed on OrderID=101 with OrdStatus=Rejected and TradeRequestFeed on TradeRequestID=userB-exchOrderY with OrdStatus=Rejected.
- For userB it should be a TradeReportFeed on OrderID=202 with OrdStatus=Rejected and TradeRequestFeed on TradeRequestID=userA-exchOrderX with OrdStatus=Rejected.

Counterparty Decline

If userB sends his TradeReport with TradeReportType=Decline, the following message flow is identical to that when accepting, but all feeds have TradeReportType=Decline and OrdStatus=Rejected, and no matching will be made. When both TradeReport and TradeRequest in the two clients have TradeReportType=Decline and OrdStatus=Rejected it is a final state.


Cancellation

At any time prior to a trade matching, the user may decide to cancel the TradeReport sending a CancelTradeReportReq. The gateway will reply with a CancelTradeReportRsp response and relevant set of feeds will be sent to one or both users.

[Trade Report Service](#)

Quote Service

The Quote Service is used for market making functionality. The client sends the gateway one or more double-sided quote messages, followed by a flush message. The gateway then packs these quotes into a single mass quote message to send to the exchange.

 This service is currently not supported for customer use.

[Trade Report Service](#) | [Document Overview](#) | [MassQuote Service](#) >

MassQuote Service

The MassQuote Service is used to send a single MassQuote message from a client to a gateway. It can be used in place of the normal Quote Service if the gateway supports the MassQuote message and functionality. Performance is improved by reducing the number of messages sent and eliminating the flush delay. The MassQuote Service is also assumed to be asynchronous, meaning that a new quote can be sent before the ack for the previous quote has been received.



This service is currently not supported for customer use.

< [Quote Service](#) / [Document Overview](#) / [Orc Trader Custom Order Panel](#) >

Orc Trader Custom Order Panel

Creating the Custom Order Panel

The custom order panel for a gateway consists of two xml files:

1. **market-capabilities.xml** which defines the order types included in the order panel
2. **dynamic-order-panel.xml** which defines the additional fields used in the order panel

These files will be located in the Orc Trader installation under <Orc Trader>\Contents\Resources\LocalStorage\markets\<short_name> (where short_name is the same as the entry in orcservers.xml).

Market Capabilities File Details

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<market-capabilities
xmlns:mc="http://www.orcsoftware.com/market-capabilities" country="US"
id="551" name="GDK FIX" timezone="EST"
xmlns="http://www.orcsoftware.com/market-capabilities">
  <attributes>
    <attribute name="skip-spots"/>
    <attribute name="skip-combos"/>
    <attribute name="submarkets-in-orders-supported"/>
    <attribute name="allow-synthetic-fill-and-kill"/>
  </attributes>
  <order-types>
    <order-type OrdType="Limit" TimeInForce="Day" has-gtc="true"
has-gtd="true" name="Limit"/>
    <order-type OrdType="Market" TimeInForce="Day" name="Market"/>
    <order-type OrdType="Limit" TimeInForce="ImmediateOrCancel" name="IOC -
Fill and Kill"/>
    <order-type OrdType="Limit" TimeInForce="FillOrKill" name="IOC - Fill
or Kill"/>
    <order-type OrdType="Custom" TimeInForce="Day" wire-value="X
name="DarkX"/>
  </order-types>
</market-capabilities>
```

The market-capabilities.xml file has three main parts:

- The market-capabilities tag which gives us a name, an ID and a time-zone for the market.
- The attributes lists market specific attributes. In the attribute element, the *name* attribute is required, the *value* attribute can be either a boolean or a number or a string and is optional (assumed to be boolean and defaults to true) . *Value* is optional and defaults to "true" if not set. This is attribute specific and clients that understands the attribute know the type of the attribute (i.e. the attribute at least always has the same type).
- The order-types lists the supported order-types of the market. The actual order type name used in the market may be identified differently here, i.e. an order-type called "Market on close(MOC)" may be described here as OrdType="Market" TimeInForce="AtClose". In addition order-type can have a name. This is the name of

this particular order-type combination that it should have in the market.

Common Attributes

Capability	Description	Default value
allow-synthetic-fill-and-kill	Disable on market where and an order may not be immediately followed by a cancel	True
at-best-information-available	If true adds this market to 'At best information' preference panel	False
disable-roundlot-rounding	If true, disables roundlot rounding functionality and thus disables rounding down inbound orders to the nearest multiple of the RoundLot size.	False
ignore-owner-check	By default, OT always will compare owner of incoming trade with current username and ignore trade if they are differ. This comparison can be ignored with this setting.	True
implied-price-for-combo-supported	Calculate implied price for combo	True
open-volume-supported	Needs to be changed if market does not support native iceberg	True
recreate-combo	Re-create combos every day	False
remove-all-combos-supported	Enable "Remove-all-combos" button in ServerLink panel. Should be set to true for most derivative markets with combinations.	False
retain-supported	Retain orders flag	False
reverse-download-supported	For GDK-FIX, set to the same value as the reverseDownloadEnabled attribute in ofc.xml. Otherwise, set to whether CreateInstrumentOperationRequest message is supported.	False
skip-combos	Skip combo flag for skipping combos during download	True

skip-spots	Skip spots flag for skipping spots during download	True
submarkets-in-orders-supported	Adds submarket pop up in order panel	False
trade-cancellation-supported	Trade cancellation support flag	False
trade-report-modify-supported	Trade reports native modification	False
trade-update-supported	Trade update support flag	False
use-marketdataservice-for-rfq	Set if Market Data Service should receive RFQs instead of Quote Service	False

Order-Type Details

Types are used to specify the execution style, validity, price condition (market, limit, pegged, etc.), volume condition (normal, all or none, etc.) of an order. Each order-type must have a unique name which will display in the Type list menu. Add <order-type> entries to the <order-types> list according to the following examples:

A fill and kill order:

```
<order-type OrdType="Limit" TimeInForce="ImmediateOrCancel" name="IOC - Fill and Kill"/>
```

A fill or kill order:

```
<order-type OrdType="Limit" TimeInForce="FillOrKill" name="IOC - Fill or Kill"/>
```

A standard market order:

Note that the validity is "Today" and not "Immediate" because on many exchanges, market orders may rest if there is no opposing side. Some markets however require that market orders are sent as IOC, in which case we would change this to validity="Immediate".

```
<order-type OrdType="Market" TimeInForce="Day" name="Market"/>
```

Valid OrdType Values


Limit


Market

MarketWithLeftOverAsLimit
StopLimit
StopOrStopLoss
Funari
Pegged
Custom

Valid TimeInForce Values

Day
AtOpening
AtClose
AtCrossing
ImmediateOrCancel
FillOrKill

 To implement GTC and/or GTC.TimeInForce cannot have the values GoodTillCancel or GoodTillDate. Instead use TimeInForce=Day and specify *has-gtd="true"* and/or *has-gtc="true"*.

 Cross-orders and interbank orders are expressed in the trade-reports section not in the order-types section.

Customized Order Types

It is possible to send a customized order type by specifying only **name** and the desired value as the **wire-value** attribute for a normal execution style order type, according to the following example:

```
<order-types>
    .....
    <order-type name="DarkX" OrdType="Custom" TimeInForce="Day"
wire-value="X" />
</order-types>
```

The example will send "OrdType=X". Some order types are considered to have a custom execution style. These order types should specify execution-style="Custom" and set the wire-value attribute to the value specified below.

Order Type	wire-value
Market to Limit	MtL

Market on Close	MoC
At the open	AtO
At the close	AtC
At market and kill	MaK
Pegged Order	PG
Nordic mid order	NM

Dynamic Order Panel File Details

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<dynamic-market-link xmlns="http://www.orcsoftware.com/dynamic-market-link"
xmlns:dml="http://www.orcsoftware.com/dynamic-market-link"
xmlns:xi="http://www.w3.org/2001/XInclude"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.orcsoftware.com/dynamic-market-link
../_schema/dynamic_market_link.xsd">
  <market core-version="1" id="551" name="GDK FIX" version="1"/>

  <xi:include href="../_common/shared.xml" parse="xml"/>
  <xi:include href="market-capabilities.xml" parse="xml"/>
  <additional-data-in-order-panel>
    <fields>
      <field id="A" name="Account" sd-tag="1" type="xsd:string"/>
    </fields>
    <xi:include href="../_common/type_definitions.xml" parse="xml"/>
    </field-types>
  </additional-data-in-order-panel>
</dynamic-market-link>
```

i For CBLS processes, create a folder <Orc Release>/arch/x86_64-unknown-linux/apps/OrcD.app/Contents/Resources/LocalStorage/market s/<short_name>.

i Note

Orc Trader does NOT need to be restarted each time this file is changed. See the [Caveats and Debugging](#) section for more information.

In this file, change <long_name> to the long_name specified in orcservers.xml and <server_id> to the server_id. This can be verified by starting Orc Trader and opening an order panel for any contract, which should look like:

Buy Order (short_na...

A

▼

Template

▼

Buy

Sell

@

0.00

Customer

▼

Portfolio

▼

Reference

Iceberg

Type

Limit

▼

Validity

Today

▼

Account

Comment

Active


Exchange order

▼

Send Bid

▼

Volume is zero

 Note that the Type list menu specifies Limit, Validity species Today and neither field is available for selection, while there is an Account field available. If the order panel does not have the Account field and does not show Type set to Limit as above, check all file and directory names for correct spellings (including file extensions which may be hidden in MS Windows).

The list of Types can be expanded by adding <order-type> entries to the <order-types> list. Additionally, you can add fields by adding <field> entries to the <fields> list, and then add the field id to the "fields" attribute of the appropriate <fieldset> entry. For more information, refer to the [Customized Order Types](#) section.

Adding Fields

Fields may be added to specify optional extra data which will be sent to the GDK-FIX gateway.

1. Determine the type of your field. Available types are described in full below.

2. Determine the sd-tag for your field. See full description below.

3. Pick a unique identifier for your field

4. Determine which order types your field is available for. The field will only be visible for those order types, or may be added for all order-types.

The field can then be added to the fields element of the dynamic order panel xml. It must have the attributes: id (must be unique for all fields), name (this text will appear as a label for the field when it is visible on the order panel), sd-tag (see full description below), and type (see full description below). It may also have the attributes: default-value, true-string (if type=xsd:boolean), false-string (if type=xsd:boolean), multi-select (if type is an enum),

Field Type	Appearance on order panel
------------	---------------------------

xsd:string	editbox
xsd:boolean	checkbox
xsd:date	custom date-picker
xsd:time	time field
xsd:integer	text field (only allows integer)
xsd:double	text field (only allows double)
dml:price	text field (only allows price)
dml:volume	text field (only allows volume)
enum	list menu or multi-select box. Must be specified by enum name.

Full specifications for all sd-tags are packaged with the GDK-FIX release in <gdk>/doc/service-definitions. SD Fields are strongly typed, you must be sure to pick a field of the same type as the data allowed for the field. It is preferred, if possible, to use the sd-tag which is the same as the fix tag for the order type being sent. All fields in the order are populated into the fix message using the same tag, unless a different mapping is specified in the ofc.xml. A fields sd-tag attributed may specify a name or tag number. It is advisable to choose the tag whose description most closely matches the field being added so that strategies may be ported easily to and from this gateway.

As in the example above (where the Account field was added for type Limit), adding a field is as easy as adding a properly defined field entry to the <fields> element and adding the id of that field to each of the fieldset for each order-type for which it is necessary. A field may also be added to all fieldsets by adding a default-fieldset attribute to the fieldsets element. E.g. if the Account(id="A") field must be displayed for all order types and Capacity(id="OC") must be specified for Limit orders:

```
<fieldsets default-fieldset="A">
  <fieldset key="order-type" value="Limit" key-type="predefined-field"
  add-fields="OC" />
</fieldsets>
```



It is important to note that the "value" in the fieldset must be the NAME from the market-capabilities, and not the OrdType

Note that both default-fieldset and add-fields attributes are space (" ") separated lists of field id's.

Customizing enumerations

Enumerated fields have a name and a type. Check the sd field document for existing definitions of enumerated

fields. If the market definition of an enumeration does not match the generic definition:

- The enum name attribute must be unique, it is recommended that market specific enums use the name format "<Market short_name><field name>Enum" to prevent name conflicts.
- The type attribute of the enum is restricted and handled as follows:

sd-tag type	enum type	Handled
boolean	N/A	Do not use enum for boolean fields, use char, int8 or string
char	character or int8	ASCII character representation will be placed in fix message, if an integer representation is desired, use int32
datetime	N/A	Do not use enum for datetime fields, use string instead
int8	character or int8	ASCII character representation will be placed in fix message, if an integer representation is desired, use int32
int32	int32	Integer representation will be placed in fix message
int64	N/A	int64 fields are inappropriately large for use as enumerated fields, use int32 fields or strings instead
real64	N/A	real64 fields are inappropriately large for use as enumerated fields, use int32 fields or strings instead
string	string	String representation will be placed raw in fix message, string should not contain the fix delimiter (ASCII SOH)
uint32	N/A	uint32 fields are inappropriately large for use as enumerated fields, use int32 fields or strings instead
uint64	N/A	uint64 fields are inappropriately large for use as enumerated fields, use int32 fields or strings instead

Using an alternative enum for a field. Example, according to the sd field documentation, the OrderCapacity field is of type OrderCapacityEnum which has values of Agency(A), Proprietary(G), Individual(I), Principal(P), RisklessPrincipal(R), and AgentForOtherMember(W), but the market being created has the values Agency(A),

Principal(P), Riskless(R), and Other(O). To add a list menu field with the correct values, we add the field (to the fields section):

```
<field id="OC" name="Capacity" sd-tag="528" type="GdkFixOrderCapacityEnum" />
```

We then add the enum to the field-types/enums section (which may appear just below field-types). In this example we set the type of the enum to int8 (which is handled identically to char) because each of our possible values is a single ASCII character. Note that the sd-tag 528 is of type OrderCapacityEnum with has type int8. If one of the enum values had been "XX", a different sd-tag would have to be chosen (one of underlying type string) and the type of the enum changed to string.

```
<field-types>
  <enums>
    <enum name="GdkFixOrderCapacityEnum" type="int8">
      <value name="Agency" value="A" />
      <value name="Principal" value="P" />
      <value name="Riskless" value="R" />
      <value name="Other" value="O" />
    </enum>
  </enums>
</field-types>
```

Fields and Field Sets

Field sets are used to define which fields are available on the order panel. Fields which are not available, do not send to the gateway.

```
<fieldsets default-fieldset="A NP" />
```

The default-fieldset attribute is a space delimited list of the default set of field id's which will be visible by default. The the fieldsets element may contain additional fieldset elements which add, remove or redefine which fields are available based on a given key.


```

<fieldsets default-fieldset="A NP">
  <fieldset key="execution-style" value="Interbank"
key-type="predefined-field" fields="CT CU" />
  <fieldset key="order-type" value="Pegged" key-type="predefined-field"
add-fields="PT PM" />
  <fieldset key="order-type" value="Stop Limit" key-type="predefined-field"
remove-fields="NP" add-fields="ST" />
  <fieldset key="NP" value="Y" key-type="additional-field" add-fields="NQ"
/>
</fieldsets>

```

The valid values for key are execution-style, order-type, and field id's. For key values of execution-style and order-type, the key-type must be predefined-field, for keys which are field id's, it must be additional-field. The fields attribute, overrides the default-fieldset if the key has the specified value, but the add-fields and remove-fields attributes add and remove from the default-fieldset respectively. A fields attribute may not be used on the same fieldset element as an add-fields or remove-fields attribute.

In the example above, the default fields will be either A and NP or CT and CU depending on whether the selected order type has execution-style of Interbank. Selecting the Pegged order type adds the fields PT and PM, while selecting Stop Limit removes the field NP and adds the field ST to whatever fields are available. If the field NP has value Y, the field NQ will become available. Note that if the order type is Stop Limit, the field NP is removed which would also remove the field NQ if it was previously available.

Field and Enum Groups

Field groups allow for the values from several controls to send in the same SD tag separate by a specified delimiter.

```

<group id="MyFlagsID" name="Market segment" sd-tag="12345" delimiter=","
stick-together="NO">
  <field name="Allow odd lot integration" type="xsd:boolean"
default-value="true"/>
  <field name="Delayed Trade" type="xsd:boolean"
default-value="false"/>
  <field name="Trade Type" type="xsd:string"
default-value="example"/>
</group>

```

Groups can be referenced by id, the same way as a regular field, and participate in fieldsets. In the example above, the value of group would be stored under tag 12345, for instance, 12345=1,0,example using the default values. Fields will be shown in the same order as they are declared, but do not each have their own id and sd-tag. Because fields in groups do not have their own id, they may not be added or removed using fieldsets without the rest of the group.

Enum groups are used for multi-selection boxes based on a specified enum.

```

<enum-group id="RA" name="Rating" based-on="MyRating" sd-tag="10010"
delimiter=" " stick-together="NO" default-value="1"/>
...
<enum name="MyRating" type="string">
  <value name="Excellent" value="great"/>
  <value name="OK" value="3"/>
  <value name="Poor" value="1"/>
</enum>

```

If, say, 'Excellent' and 'Poor' were selected, you will have 10010="great 1".

Caveats and Debugging

Orc Trader debug log

Parsing errors are visible in the Orc Trader log by clicking Tools > Show Log. More detailed errors and warnings are available in My Documents\Orc.<date>.txt

Reloading dynamic order panel

The dynamic order panel can be reloaded without restarting Orc Trader by clicking Tools > Utilities > Command Line ... in the Orc Trader menu bar and entering "reload_markets local" without quotes. Note that any open order panel windows will not show the changes.

Common issues

The following are solutions to common issues with dynamic order panels. All issues assume that there were no errors logged parsing the xml file, the file has been loaded either by restarting Orc Trader or entering "reload_markets local" into the Orc Trader Command Line, and the order panel has been closed and opened since the xml file was reloaded.

A field is not displayed after adding it to the xml file.

1. Check that the field id has been added to the add-fields for the order-type selected or the default-fieldset
2. Check for typos
3. Check that there is not a type mismatch between the values allowed for the field and the values allowed for the sd-tag

An order-type does not stay selected in the Type list menu when other fields change.

- This indicates that two (or more) order-types have identical specifications (combination of validity, execution-style, price-condition, volume-condition, custom-data)

< [MassQuote Service](#) / [Document Overview](#) / [Development Tools](#) >

Development Tools

This section provides details of the various development tools which can be used in creating a custom SD gateway. The tools include a test gateway/simulator, a test client, debug logs, and TCP dump parser.

- [MGF Test Gateway](#)
- [SD Command Line Based Test Client](#)
- [Orc Trader and Liquidator Debug Logs](#)
- [BufferDumper Hex to SD Parser](#)

< [Orc Trader Custom Order Panel](#) | [Document Overview](#) | [End2End Metrics](#) >

MGF Test Gateway

Installation

- Extract the mgftest.tar.gz file into a local directory, normally /orcreleases. A directory called mgftest-X.YY-SNAPSHOT will be created.
- Install Oracle Java SE JRE or JDK (any build of version 6) and set the JAVA_HOME environment variable.
- In /etc/orc/orc.conf you will need to add a RELEASE and gateway entry for the MGF Test gateway:

- Add a RELEASE entry
 - | `RELEASE:MGFTEST:/orcreleases/mgftest_gateway-1.92-SNAPSHOT`

- Add the MGF Test gateway entry with static port configuration:
 - | `PROC:mgftest:::jserv:MGFTEST:mgftestmgfd::MGF Test:: -dmax -p <port> ::`

This will create the MGF Test gateway running in debug mode max and all it's services will listen for connections on port <port>

debug mode max will print the hex dumps of all the inbound and outbound messages. If you would like less logging you can try levels 1, 2, or 3 (-d1, -d2, -d3)

- You can also have your mgftest gateway use dynamic registration with the Environment Manager. You need to do this when connecting a gateway to Orc Trader and Liquidator. Trade Access clients will never use em registration so this section can be ignored.

- | `PROC:mgftest:::jserv:MGFTEST:mgftestmgfd::MGF Test:: -dmax -e em ::`

This instance of mgftest gateway will register with the Environment Manager named "em". The -p parameter is optional in this case and if it is not supplied a random available port will be chosen.

MGF Test Gateway Operations

- Log files

The log files are located in the "log" directory under release directory. In our example: /orcreleases/mgftest-1.61-SNAPSHOT/log.

Using the MGF Test Gateway in your development process

MGF Test Gateway has a built in exchange simulator so that orders entered will be acked and generate market data and crossed orders will result in trades. MGF Test can be used as a reference to make sure your custom gateway is encoding/decoding messages properly. Please see the [Custom SD Gateway Development Overview](#) section for a description of how to best use the MGF Test Gateway in your development process.

[Development Tools](#)

SD Command Line Based Test Client

The SD Test Client is a command line based application that can send and receive SDOFF messages. It can communicate with the MGF Test gateway or any other SD gateway. It is usually easiest to create a start script for the test client but it can also be started directly from the command line as documented below.

- [Start Script Using Gateway URL](#)
 - [UNIX Start Script](#)
 - [Windows Start Script](#)
 - [Using the Start Scripts](#)
- [Start Script Using EM](#)
 - [UNIX Start Script](#)
 - [Windows Start Script](#)
 - [Using the Start Scripts](#)
- [Starting Using the Command Line with a URL direct](#)
- [Starting Using the Command Line with a EM Registered Service](#)
- [Operation](#)
- [Using the SD Command Line Client in your development process](#)

Start Script Using Gateway URL

UNIX Start Script

```
# Service host and port
HOST=<host>
PORT=<port>
JAVA_OPTS="-Xms200m -Xmx400m -XX:+UseConcMarkSweepGC"
JAVA_OPTS="$JAVA_OPTS -Dlog4j.configuration=file:./log4j.xml"
JAVA_OPTS="$JAVA_OPTS -Dmarketdataservice.url=sdoff://$HOST:$PORT
-Dinstrumentservice.url=sdoff://$HOST:$PORT" -Dorderservice.url=sdoff://$HOST:$PORT"
JAVA_OPTS="$JAVA_OPTS -Denable.orders=false -Denable.quotes=false
-Denable.tradereports=false -Denable.instruments=true -Denable.marketdata=true"
#Make sure this is the correct version of your jar
java ${JAVA_OPTS} -jar gateway_cli_client-latest-SNAPSHOT-run.jar
```

Windows Start Script

```

set HOST=<host>
set NAME=<port>

set JAVA_OPTS=-Xms200m -Xmx400m -XX:+UseConcMarkSweepGC
set JAVA_OPTS=%JAVA_OPTS% -Dlog4j.configuration=file:./log4j.xml
set JAVA_OPTS=%JAVA_OPTS% -Dmarketdataservice.url=sdoff://%HOST%:%PORT%
-Dinstrumentservice.url=sdoff://%HOST%:%PORT%
-Dorderservice.url=sdoff://$HOST:$PORT"
set JAVA_OPTS=%JAVA_OPTS% -Denable.orders=false -Denable.quotes=false
-Denable.tradereports=false -Denable.instruments=true -Denable.marketdata=true

java %JAVA_OPTS% -jar gateway_cli_client-latest-SNAPSHOT-run.jar

```

Using the Start Scripts

- Create the script(s) above
- Edit the HOST and PORT parameters
- Edit the "-Denable*" options to enable or disable services as required.
- Execute the script.

Start Script Using EM

UNIX Start Script

```

# EM settings
EM_HOST=<host name>
EM_NAME=<em name>
PROVIDER=<gateway name>

JAVA_OPTS="-Xms200m -Xmx400m -XX:+UseConcMarkSweepGC"
JAVA_OPTS="$JAVA_OPTS -Dlog4j.configuration=file:./log4j.xml"
JAVA_OPTS="$JAVA_OPTS -Dem.host=$EM_HOST -Dem.name=$EM_NAME
-Dprovider.name=$PROVIDER"
JAVA_OPTS="$JAVA_OPTS -Denable.orders=false -Denable.quotes=false
-Denable.tradereports=false -Denable.instruments=true -Denable.marketdata=true"

java ${JAVA_OPTS} -jar gateway_cli_client-1.0-SNAPSHOT-run.jar

```

Windows Start Script

```

set EM_HOST=<host name>
set EM_NAME=<em name>
set PROVIDER=<gateway name>

set JAVA_OPTS=-Xms200m -Xmx400m -XX:+UseConcMarkSweepGC
set JAVA_OPTS=%JAVA_OPTS% -Dlog4j.configuration=file:./log4j.xml
set JAVA_OPTS=%JAVA_OPTS% -Dem.host=%EM_HOST% -Dem.name=%EM_NAME%
-Dprovider.name=%PROVIDER%
set JAVA_OPTS=%JAVA_OPTS% -Denable.orders=false -Denable.quotes=false
-Denable.tradereports=false -Denable.instruments=true -Denable.marketdata=true

java %JAVA_OPTS% -jar gateway_cli_client-1.0-SNAPSHOT-run.jar

```

Using the Start Scripts

- Create the script(s) above
- Edit the EM parameters
- Edit the "-Denable*" options to enable or disable services as required.
- Execute the script.

Starting Using the Command Line with a URL direct

```

java \-jar gateway_cli_client-latest-SNAPSHOT-run.jar
\ -Dmarketdataservice.url=sdoff://<host>:<port> \--Denable.instruments=false
\--Denable.orders=false

```

modify <host> and <port> above for the service you are connecting to

Starting Using the Command Line with a EM Registered Service

```

java \-jar gateway_cli_client-latest-SNAPSHOT-run.jar \-Dem.host=<em_host>
\--Dem.name=<em_name> \--Denable.instruments=false \--Denable.orders=false
com.orcsoftware.serviceapi.gatewayclient.Main

```

modify <em_host> and <em_name> for the em you are using

Operation

Once the tools starts, you will see following menu:

```
*****  
*** Service API Client ***  
*****  
*** c - Connect ***  
*****  
*** h - Print Help ***  
*** q - Quit ***  
*****
```

- To connect, enter "c"
 - The connect command will only connect to services you configured from the start script or from the command line.
- To get help, enter "h"

Using the SD Command Line Client in your development process

Please see the [Custom SD Gateway Development Overview](#) section for a description of how to best use the SD Command Line Client in your development process.

[Development Tools](#)

Orc Trader and Liquidator Debug Logs

It may be useful to enable full debug logging in Orc Trader and Liquidator so you can see what they are sending and receiving.

- To do this in Orc Trader:
 1. On the Orc menu bar click on Orc->Preferences
 2. Preferences window will pop up. On the left hand side under the Administrative heading select Log Levels.
 3. On the right side of the window under Server Link Logging click on Configuration Panel...
 4. Another pop up window. On the left find and select your gateway in the scrollable window.
 5. On the right side of the screen find the line that says Doff and select the radio button 3.

This should start logging debug to your Orc Trader log directory in Documents/Orc.log

- To do this in Liquidator:
Start Liquidator with this command line option: -dddd

If this is too much logging you can also try -d, -dd, and -ddd. -dd will log the decoded SD messages.

BufferDumper Hex to SD Parser

BufferDumper

BufferDumper is a tool included with the mgftest standalone distribution that will print a hex dump of a message from mgftest.debug (or any other gateway that logs the hex dump in a similar format).

- To run: <mgftest_root_dir>/bin/bufferDumper.sh <inputfile>
 - <inputfile> is a file containing the log output for a single SD message hexdump taken from mgftest.debug (or some other gateway).
 - Ex. /orcreleases/mgftest/bin/bufferDumper.sh order_req_dump.txt

order_req_dump.txt

```
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0001] - 0000:  44 00 00 00 54 50
00 00  00 4C 03 03 CC 27 6C A5
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0002] - 0010:  EB AC D5 C1 4D 90
27 53  39 00 00 00 B2 27 73 04
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0003] - 0020:  31 30 31 00 CA 27
4C C6  E5 D6 6A 36 62 31 26 52
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0004] - 0030:  00 00 00 00 00 00
14 40  28 62 32 3B 62 30 2C 52
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0005] - 0040:  00 00 00 00 00 00
34 40  B8 27 73 09 74 65 73 74
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0006] - 0050:  55 73 65 72 00 D0
27 62  30
```

/orcreleases/mgftest/bin/bufferDumper.sh order_req_dump.txt

Original hex dump

```

2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0001] - 0000:  44 00 00 00 54 50
00 00  00 4C 03 03 CC 27 6C A5
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0002] - 0010:  EB AC D5 C1 4D 90
27 53  39 00 00 00 B2 27 73 04
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0003] - 0020:  31 30 31 00 CA 27
4C C6  E5 D6 6A 36 62 31 26 52
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0004] - 0030:  00 00 00 00 00 00 00
14 40  28 62 32 3B 62 30 2C 52
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0005] - 0040:  00 00 00 00 00 00 00
34 40  B8 27 73 09 74 65 73 74
2012-03-13 13:33:12,818000 -0500 flashtaco mgftest
[1555:Channel_1_reader]/DEBUG:[cl/Cont'd_0006] - 0050:  55 73 65 72 00 D0
27 62  30

```

Cleaned up hex dump

```

0000:  44 00 00 00 54 50 00 00 00 4C 03 03 CC 27 6C A5
0010:  EB AC D5 C1 4D 90 27 53 39 00 00 00 B2 27 73 04
0020:  31 30 31 00 CA 27 4C C6 E5 D6 6A 36 62 31 26 52
0030:  00 00 00 00 00 00 14 40 28 62 32 3B 62 30 2C 52
0040:  00 00 00 00 00 00 34 40 B8 27 73 09 74 65 73 74
0050:  55 73 65 72 00 D0 27 62 30

```

Decoded message

```

00000: 44                                     : DOFF Message Type: 44 'D' (char)
00001: 00 00 00 54                           : DOFF message length: 84 bytes
(int32)
00005: 50 00 00 00                           : SDOFF message length: 80 bytes
(nbo-uint32)
00009: 4c 03 03                             : SDOFF Message Tag/Service
Tag/Operation Id: [1.76]/[1.3]/3
00012: cc 27 6c a5 eb ac d5 c1 4d           : [0.5068]=1331663575781 (int64)
00021: 90 27 53 39 00 00 00                 : [0.5008]=Struct. Length: 57 bytes
00028: b2 27 73 04 31 30 31 00             : [0.5042]="101" (string)
00036: ca 27 4c c6 e5 d6 6a                 : [0.5066]=223720134 (uint64)
00043: 36 62 31                             : [0.54]='1' (octet)
00046: 26 52 00 00 00 00 00 14 40          : [0.38]=5.0 (real64)
00056: 28 62 32                             : [0.40]='2' (octet)
00059: 3b 62 30                             : [0.59]='0' (octet)
00062: 2c 52 00 00 00 00 00 34 40          : [0.44]=20.0 (real64)
00072: b8 27 73 09 74 65 73 74 55 73      : [0.5048]="testUser" (string)
65 72 00
00085: d0 27 62 30                         : [0.5072]='0' (octet)

```

To run BufferDumper against output from your own gateway your hex dump must be similar to the output of mgftest. For example BufferDumper can interpret the hex dump below (Which is the same as above with different spacing and no line headers):

```
44 00 00 00 54 50 00 00 00 4C 03 03 CC 27 6C A5 EB AC D5 C1 4D 90 27 53 39
00 00 00 B2 27 73 04
31 30 31 00 CA 27 4C C6 E5 D6 6A 36 62 31 26 52 00 00 00 00 00 14 40 28
62 32 3B 62 30 2C 52
00 00 00 00 00 00 34 40 B8 27 73 09 74 65 73 74 55 73 65 72 00 D0 27 62 30
```

Here is a simple Java method that will append a hexdump to a StringBuffer in the format accepted by BufferDumper:

```
private void appendMessage(byte[] message, int offset, int length,
StringBuffer builder) {
    for (int i = 0; i < length; i += 16) {
        builder.append(String.format("\n%04X: ", i));
        for (int j = 0; j < 8 & i + j < length; j++) {
            builder.append(String.format(" %02X", message[offset + i + j]));
        }
        if (i + 8 < length) {
            builder.append(" ");
        }
        for (int j = 8; j < 16 & i + j < length; j++) {
            builder.append(String.format(" %02X", message[offset + i + j]));
        }
        builder.append("\n");
    }
}
```

End2End Metrics

- [End2End Metrics Concepts](#)
 - [End2End Reporter](#)
-

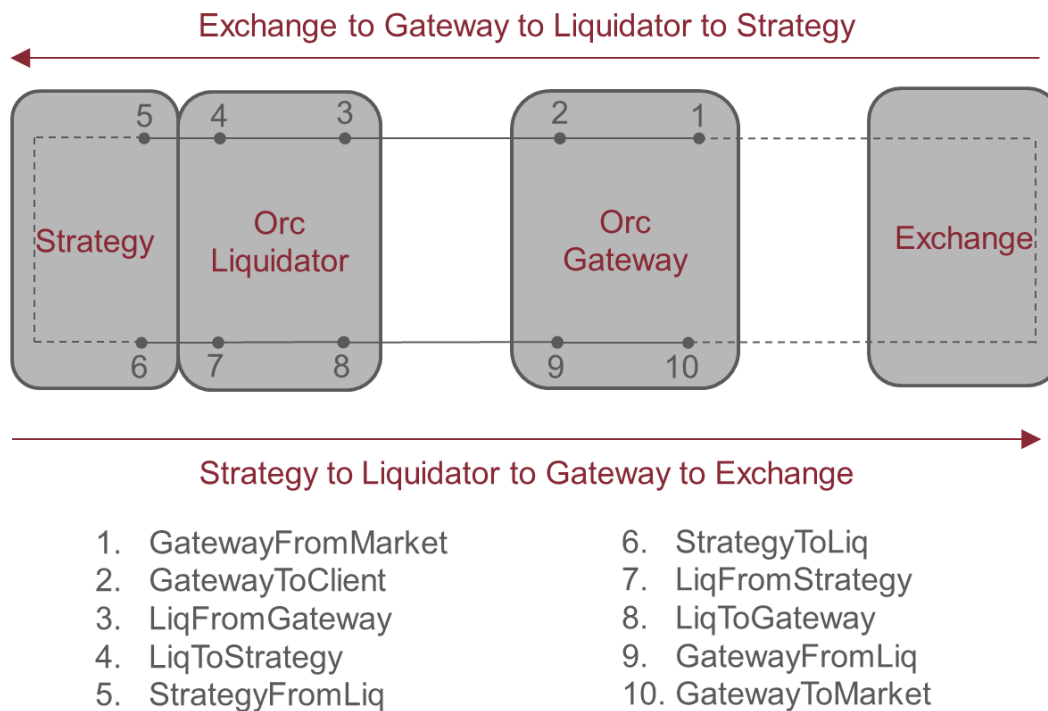
< [Development Tools](#) | [Document Overview](#) | [Support](#) >

End2End Metrics Concepts

End2End metrics is Orc's method of analyzing transaction latency and throughput through the system. The header for each message contains a TransactionID, InTimestamp, and OutTimestamp. For each message the TransactionID and timestamps at each point is written into log files. The data points are written to files by Liquidator and each gateway. The files can be post-processed and a report generated.

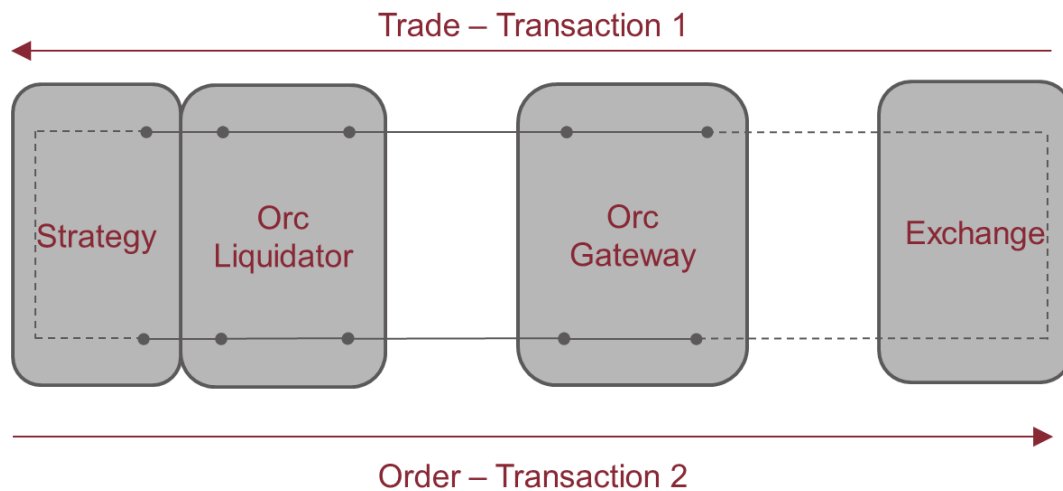
End2End Metrics Measurement Points

Points 1 through 10 in the diagram below are the standard measurement points for Orc's End2End metrics. Each point represents the "in time" and "out time" of each component in the data flow.



End2End Metrics Transactions

A series of these measurement points make up a transaction. Several transactions can be linked together to form a measurement sequence. E.g. when a Trade occurs which triggers a new Order being sent to the market - this can be implemented as two transactions, one for the incoming trade and one for the outgoing order. These two transactions can then be linked.



Measurement Sequence = Transaction 1 + Transaction 2

End2End Metrics Performance impacts

End2End is designed to be cheap to use to minimize effects on the running system. There are three modes, *Off*, *Production* and *Debug*. *Production* is meant to be used in live systems while *Debug* is only suitable for non production use. To not increase latencies, processes do not write to log files on the "hot path".

Mode	Action
Off	Nothing is written to log file
Production	Many of the gateway's points are not written to log file by the gateway but rather sent to Liquidator for processing. Liquidator will write to log file only if the messages "actually leads to an action" in a strategy otherwise it will log nothing.
Debug	Process writes all End2End data to log file

End2End Metrics Limitations

⚠ There are some limitations of the current implementation.

- Files from multiple hosts can be processed together but the End2End Reporter does not currently adjust for time differences between these hosts.
- Although not limited by End2End as such, currently all "measurement points" resides *within* the Orc processes.

End2End Metrics File Analyzer

The files are run through the End2End Reporter to produce a HTML report with latency and throughput statistics and charts. The End2End Reporter can also dump the contents of log files and output raw data to .csv files.



End2End Metrics

End2End Reporter

- [Introduction](#)
- [Usage](#)
 - [Set max memory](#)
- [Configuration](#)
 - [Default configuration](#)
 - [HtmlReport - Change report name, title and omit report section\(s\)](#)
 - [Histogram - Set chart axes of latency histogram to fixed values](#)
 - [LatencyByTime - Enable latency by time chart](#)
 - [HtmlReport - Treat start of data as time zero](#)
 - [Changing the unit of displayed time](#)
 - [View Time spent on Market](#)
 - [LinkTransactions - View sequences of transactions as one chain](#)
 - [Max length of Measurement Sequences](#)
 - [Samples Threshold - Omit transaction types with small volume](#)
 - [Filters](#)
 - [WarmupFilter - Skip warmup](#)
 - [TransactionFilter - Filter transactions based on type and points included](#)
 - [OrderFilter - Remove unwanted transactions](#)
 - [TimeFilter - Filter transactions based on start and end time](#)
 - [TrimFilter - Reduce \(trim\) transactions](#)
 - [SplitFilter](#)
 - [Warmup Statistics](#)
- [View raw content of a End2End log file using FileDumper](#)
- [Known issues](#)
 - [End2End on Multiple Hosts](#)
 - [Too many open files \(java.io.FileNotFoundException\)](#)

Introduction

The End2End Reporter is a command line tool that produces a report from one or more End2End performance log files. The report aggregates the end2end traces and displays the latency distributions of the traces, both for the complete traces and from parts of the traces, so it is possible to determine where in what component of the Orc System time is spent. The report is an HTML file, by default called performance_report.html.

The reporter is delivered as a jar file part of the Orc Performance Tools.

This chapter provides configuration examples on how to configure the data included in the performance log files.

Usage

Basic example:

```
java -jar end2endreporter.jar liquidator1_20120402_0900.end2end.log
gateway1_20120402_0901.end2end.log
```

More generically:

```
java -jar end2endreporter.jar [ea:OPTION]... [ea:POINTS_FILE]...
-config <arg>    Configuration file
-D,--define <arg>  Define an override for configuration file
```

Set max memory

When processing large files the default max memory limit of the Java VM may not be enough. The memory can be set by the `-Xmx` switch. For example, the following command gives the reporter 512 megabytes memory:

```
java -Xmx512m -jar end2endreporter.jar ...
```

Configuration

It is possible to configure multiple things in the report, for what is viewed in the report, name of the file etc. It is also possible to filter the data which the report is based on, for example on time or transaction types in order to focus on what is interesting.

Configuration is done, either on the command line, or in an XML configuration file. There is a default configuration to start with. Any setting in the configuration file can be overridden from the command line.

For example, to enable `Latency` you can use this syntax:

```
... -DHtmlReport.Latency ...
```

To set a value of an attribute like `filename` you can use:

```
... -DHtmlReport.filename=my_report.html ...
```

Default configuration

If no configuration file is specified on the command line the following configuration will be used:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <HtmlReport filename="performance_report.html">
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>
</Configuration>
```

HtmlReport - Change report name, title and omit report section(s)

The name of the report can be set by the **filename** attribute on the **HtmlReport** element. If you want each report have a unique name, set **filenameTimestamp** attribute to *true* and a timestamp will be appended to the report filename. If you want to add to the title in the report you use the **title** attribute. A subtitle can also be added using the attribute **subTitle**. By default, both latency and throughput are included, you can omit one by removing it from the configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <HtmlReport filename="latency.htm" filenameTimestamp="true" title="Project XYZ"
    subTitle="Run #12">
    <Latency />
  </HtmlReport>
</Configuration>
```

The configuration above omits throughput data and only produces latency data in the file **latency.htm**.

Histogram - Set chart axes of latency histogram to fixed values

The automatic scaling of the latency histogram can mask differences when comparing output from multiple runs of the performance reporter. You set the min and max values on each axes to make the diagrams comparable.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
    ..\schemas\config.xsd">
  <HtmlReport>
    <Latency>
      <Histogram latencyMinNanos="1000" latencyMaxNanos="5000" samplesMax="10000"/>
    </Latency>
  </HtmlReport>
</Configuration>
```

In the example configuration above we set:

latencyMinNanos	X-axis minimum value, i.e. the lower threshold in nanos of the leftmost bar of the diagram
latencyMaxNanos	X-axis maximum value, i.e. the upper threshold in nanos of the rightmost bar of the diagram
samplesMax	Y-axis maximum value, i.e. any bar with number of samples exceeding this will be truncated

- By setting the `latencyMinNanos` and/or `latencyMaxNanos`, samples may be omitted from the histogram which may be misleading. The samples falling outside the configured range will not be added to the min/max bars but just omitted.

LatencyByTime - Enable latency by time chart

This is an experimental chart that shows the median and max latency for each time interval. The time interval is 1 second for test runs lasting less than 5 minutes but the interval will be increased for longer runs to keep memory consumption bounded.

The chart shows two lines, one for the median latency for each interval and one for max latency. Because these two may differ by one or more magnitudes they are shown with different scales, the median line uses the scale to the left of the chart and the max line uses the scale to the right of the chart.

This chart is generated by including the **LatencyByTime** element in the Latency section:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">
  <HtmlReport>
    <Latency>
      <LatencyByTime/>
    </Latency>
  </HtmlReport>
</Configuration>
```

It is also possible to fix the axes of the chart by setting the following attributes in the **LatencyByTime** element:

<code>timeMin</code>	Lower bound for time axis, measured in seconds since the start of run
<code>timeMax</code>	Upper bound for time axis, measured in seconds since the start of run
<code>medianLatencyMaxNanos</code>	Upper bound for left hand (median) latency axes
<code>maxLatencyMaxNanos</code>	Upper bound for right hand (max) latency axes

- The LatencyByTime chart requires a lot of memory, so it is recommended to increase VM memory when using this chart. Roughly it will require 50 megabytes per each type of transaction times the number of measurements points in the transaction. Setting the WarmUp filter to 10 (or greater) can help prevent running out of memory: `<WarmupFilter count="10"/>`

HtmlReport - Treat start of data as time zero

Human readable time is shown in charts, which is a good fit for production data where the actual time may have

significance. If you run tests in a lab environment it may be useful to see time elapsed instead. This can be altered using the **readableTime** attribute as seen in this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <HtmlReport filename="performance_report.html" readableTime="false" >
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>
</Configuration>
```

Changing the unit of displayed time

Timestamps are collected as nanoseconds, by default, times in reports are displayed in micro seconds. The displayed time unit can be changed for **HtmlReport** and **TransactionDumper** by setting the attribute **timeUnit** to 'nanos', 'micros' or 'millis':

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <HtmlReport filename="performance_report.html" timeUnit="nanos">
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>
  <TransactionDump timeUnit="millis"/>
</Configuration>
```

View Time spent on Market

It is possible to view the time spent outside the Orc System, the time from when a request is sent to the exchange until the response is received. It is actually the case that such a path is treated as the same transaction under the hood in the end2end system. But by default the end2end reporter will break the transactions when the request (for example an order) is sent to the market, which simplifies the primary analysis. In order to show the time spent, enable the configuration **KeepTransactionsCrossMarket**, example:

```
java -jar end2endreporter.jar -DKeepTransactionsCrossMarket ...
```

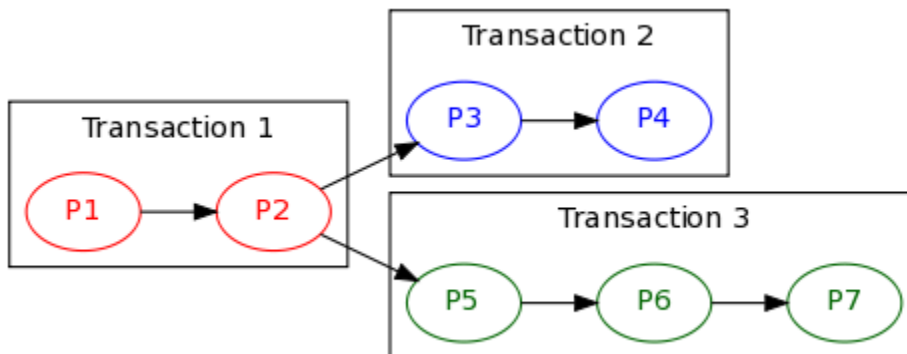
```

<?xml version="1.0" encoding="UTF-8"?>
<Configuration
  xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <HtmlReport filename="performance_report.html">
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>
  <KeepTransactionsCrossMarket enabled="true"/>
</Configuration>

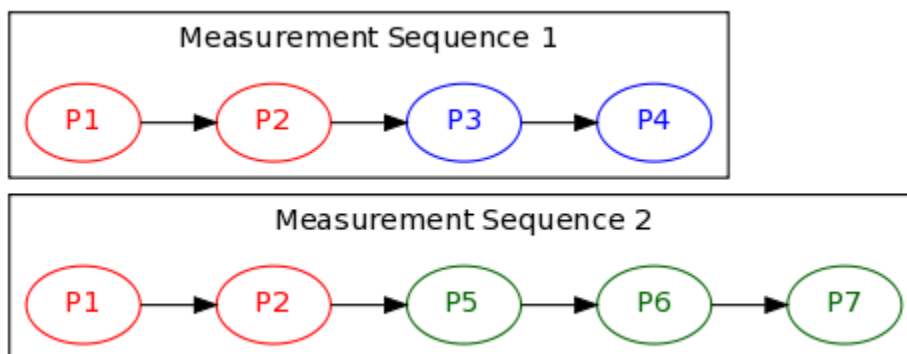
```

LinkTransactions - View sequences of transactions as one chain

When a transaction is triggered by another they will be linked by an association entry in the log file. For example if an order is placed in a Liquidator strategy when it is triggered by a market data update, such an association will be created. It is sometimes desirable to link or "unfold" these transactions into one measurement sequence. Here is an example which contains three transactions, where transaction 2 and 3 are triggered by transaction 1, i.e. transaction 2 and 3 are children of transaction 1:



The default configuration lists the three transaction types P1-P2, P3-P4 and P5-P6-P7 but when linking transactions, latencies and throughput will be computed for the following two sequences:



This is accomplished by adding the **LinkTransactions** element to the config:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">
  <HtmlReport filename="performance_report.html">
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>

  <LinkTransactions/>

</Configuration>
```

Max length of Measurement Sequences

The transaction chains can become very long. In order to not produce unusable report, or fail to generate them, there is a limit to how long measurement sequences can become. This limit is 10 by default, and possible to configure. Example:

```
<LinkTransactions maxCount="4"/>
```

Samples Threshold - Omit transaction types with small volume

By default, samples are omitted from the report if the transaction type has less samples than 1/1000 of the transaction type having the most number of samples. This is to avoid cluttering the performance file with transactions of the type only used during start, shutdown or error situations, these transactions would make it hard to find the analysis of the relevant transactions.

Therefore it is possible to configure how many samples of each transaction type to be included in the report. This configuration can be absolute or relative to the most number of samples of any transaction type, eg. **samplesThreshold=100** or **samplesThreshold=5%**.

The default value for samplesThreshold is **0.1%**. The configuration file below turns off this feature by setting the samplesThreshold to 0, i.e. all transaction types will be present in the report - even if there is only one sample.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd"
  samplesThreshold="0">
  <HtmlReport filename="performance_report.html">
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>
</Configuration>
```

Filters

WarmupFilter - Skip warmup

Quite often the initial measurements are a lot worse than the "normal" case because the VM has not compiled or optimized the code yet. Therefore it is useful to skip the first transactions of each type:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">
  <WarmupFilter count="10000"/>
  <HtmlReport>
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>
</Configuration>
```

The configuration above will ignore the first 10000 transactions of each type when computing latency and throughput.



The transaction type list at the top of the report will list all transactions even the warmup ones

TransactionFilter - Filter transactions based on type and points included

If you want to only include some specific transactions in the report you can setup filtering on those transactions:


```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">
  <TransactionFilter type="BBO">
    <Point>GatewayFromMarket</Point>
    <Point>GatewayToClient</Point>
    <Point>LiqFromGateway</Point>
  </TransactionFilter>
  <TransactionFilter type="CreateOrder"/>
  <HtmlReport filename="performance_report.html">
    <TransactionTypeList/>
    <Throughput/>
    <Latency/>
  </HtmlReport>
</Configuration>
```

Only transactions with the specified type and points (in order) will be included in the report. Either the **type** or **points** may be omitted when specifying a transaction filter, but not both.

Using the example configuration above the report will only include transactions which:

- Either have the type **BBO** and exactly the points GatewayFromMarket, GatewayToClient, LiqFromGateway.
- Or has the type **CreateOrder**

OrderFilter - Remove unwanted transactions

Use the **OrderFilter** to remove transactions where the points don't comply with the requested order.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">

  <OrderFilter type="BBO" >
    <Point>GatewayFromMarket</Point>
    <Point>GatewayToClient</Point>
    <Point>LiqFromGateway</Point>
    <Point>LiqToStrategy</Point>
  </OrderFilter>

  <HtmlReport filename="performance_report.html">
    <TransactionTypeList/>
    <Throughput/>
    <Latency/>
  </HtmlReport>
</Configuration>
```

TimeFilter - Filter transactions based on start and end time

Use **TimeFilter** to create a report within a certain timeframe of the run, based on nanos since January 1st 1970.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">

  <TimeFilter startTimestampNanos="1314778315147000"
    endTimestampNanos="1314779038350000" />

  <HtmlReport filename="performance_report.html">
    <TransactionTypeList/>
    <Throughput/>
    <Latency/>
  </HtmlReport>
</Configuration>
```

TrimFilter - Reduce (trim) transactions

The **TrimFilter** can be used to remove start and/or end points from a transaction. This can for example be used to remove the points of an order ack which consist of the reply from the market.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">

  <TrimFilter type="BBO"
    startPoint="GatewayFromMarket"
    endPoint="LiqToStrategy" />
  <TrimFilter type="CancelOrder"
    startPoint="LiqFromStrategy"
    endPoint="GatewayToMarket" />

  <HtmlReport filename="performance_report.html">
    <TransactionTypeList/>
    <Throughput/>
    <Latency/>
  </HtmlReport>
</Configuration>
```

SplitFilter

Split transactions between 2 adjacent points, for example between *GatewayToClient* and *LiquidatorFromGateway*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ..\schemas\config.xsd">
  <SplitFilter beforePoint="GatewayToMarket" afterPoint="GatewayFromMarket"/>
  <HtmlReport filename="performance_report.html">
    <TransactionTypeList />
    <Throughput />
    <Latency />
  </HtmlReport>

  <LinkTransactions/>

</Configuration>
```

Warmup Statistics

Use the **WarmupStatistics** collector to gather the time for different operations at operation X, Y and Z in order to see how long the first operation takes with regards to code warmup and initial work tasks. The statistics is by default printed in a CSV file in the folder `performance_report_files_<date>/warmup_statistics.csv`. File name and location can be changed, and the statistics can also be logged on stdout.

Example 1, which enables the gathering and prints on standard out, using command line arguments:

```
java -jar end2end-reporter.jar -DWarmupStatistics
-DWarmupStatistics.stdout="true" <end2end file(s)>
```

Example 2, which enables the gathering, configure the samples to print and the file location ans on stdout, using an xml file. It defines the samples to print (for each operation type) to 1, 2, 3, 4, 5, 10, 100, 500, 1000, 5000 and 10000. It also combines with trim and transaction filters in order to only show enter and cancel order operation times through the gateway.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ../schemas/config.xsd">

  <TrimFilter type="CancelOrder"
    startPoint="GatewayFromClient"
    endPoint="GatewayToMarket" />

  <TrimFilter type="CreateOrder"
    startPoint="GatewayFromClient"
    endPoint="GatewayToMarket" />

  <WarmupStatistics stdout="true"
    operations="1,2,3,4,5,10,100,500,1000,5000,10000"
    directory="my_performance_report_files"
    directoryTimestamp="true"
    filename="my_warmuptransactions.csv"/>

</Configuration>
```

TransactionDump - Dump transaction to CSV files

Use **TransactionDump** to dump all transactions to CSV files, one file for each type of transaction, i.e. one file for each row in the transaction type list. In case transaction linking is enabled the dumped transactions will be the linked ones.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://schemas.com.orcsoftware.end2endreporter.config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.com.orcsoftware.end2endreporter.config
  ../schemas/config.xsd">
  <TransactionDump directory="/temp/tx" directoryTimestamp="true"
  maxLinesPerFile="10000"/>
</Configuration>
```

CSV files are written to the current directory unless the **directory** attribute has been set to something else. If you want each report to write to a separate directory, set **directoryTimestamp** attribute to *true* and a timestamp will be appended to the directory name. Some applications have a problem handling large amount of lines in CSV files. This can be addressed by limiting the number of lines written to each file using the **maxLinesPerFile** attribute. When the limit is reached the file will be rotated and a new file will be used. If the limit is set to 0 or omitted no limit is used.

Each file has a header row which describes each field, then there is one row per transaction. The first column is the TransactionID and then one latency delta in each column.

Example file:

```
Transaction id,GatewayFromMarket->GatewayToClient,GatewayToClient->LiqFromGateway
11849253220808673047,367,110
11849253220808673049,176,90
11849253220808673051,297,101
11849253220808673053,412,93
```



By default, all latencies in CSV files are measured in microseconds, this can however be overridden using the **timeUnit** attribute, see below.

View raw content of a End2End log file using FileDumper

Use **FileDumper** to view the points in one file (print all on screen).

Example:

```
java -cp end2endreporter.jar com.orcsoftware.end2endreporter.FileDumper
myfile.end2end.log
```

Example printout:

```

Analyzing file "myfile.end2end.log"
Number of records 30305
  1 Header PERF_LOG
    Version 1
      2 Define point    0 as GatewayFromMarket
      3 Define point    2 as GatewayToClient
        - snip -
      10 Define transaction type    2 as BBO
      11 Define transaction type    4 as Depth
        - snip -
      21 Measurement point  GatewayFromMarket(  0) tx 4 type          Own
Trade(  8) time 2010-10-19 14:33:24.834(1287491604834838000) cMap.size: 0
      22 Measurement point  GatewayToClient(  2) tx 4 type          Own
Trade(  8) time 2010-10-19 14:33:24.835(1287491604835256000) cMap.size: 0
      23 Measurement point  LiqFromGateway(  8) tx 4 type          Own
Trade(  8) time 2010-10-19 14:33:24.835(1287491604835369000) cMap.size: 0
      24 Measurement point  LiqToStrategy( 10) tx 4 type          Own
Trade(  8) time 2010-10-19 14:33:24.837(1287491604837610000) cMap.size: 0
      25 Association parent 4 child 3
      26 Measurement point  StrategyFromLiq( 44) tx 3 type          Create
Order( 10) time 2010-10-19 14:33:24.837(1287491604837623000) cMap.size: 0
      27 Measurement point  StrategyToLiq( 46) tx 3 type          Create
Order( 10) time 2010-10-19 14:33:24.839(1287491604839101000) cMap.size: 0
      28 Measurement point  LiqFromStrategy( 12) tx 3 type          Create
Order( 10) time 2010-10-19 14:33:24.839(1287491604839101000) cMap.size: 0
        - snip -

```

Known issues

End2End on Multiple Hosts

The time on two different hosts is very likely not exactly the same at the microsecond level, even if you run a time-synchronization system like NTP. The End2End Reporter does not take this difference into account and End2End reports based on log files from more than one host may therefore be confusing and unreliable.

Too many open files (java.io.FileNotFoundException)

When analyzing (large) files the process may run out of file handles. This is due to the fact that a set of intermediate files are created and accessed in parallel. The number of files used is proportional to the size of the input file. Currently this problem can only be resolved by increasing the operating system limit of maximum open files (per process).

[End2End Metrics](#)

Orc System Integration

If desired a custom gateway can follow some internal Orc standards.

Command Line Parameters

Options	Value	Comment
-p	<port_number>	Statically sets the client listening port
-n	<server name>	Server name. Used in log files, EM registration, etc..
-config	<path_custom_configuration_file>	Path to custom configuration file.
-l	<file_path/log_name>	Log file path and name, defaults to <release dir>/log.
-dX		<p>Set debug level at gateway start in the form -dX, where X can be 1,2,3,4 or max</p> <p>-d1 - Transaction logging (i.e. FIX orders, exchange orders)</p> <p>-d2 - Orc client message logging</p> <p>-d3 - Internal gateway logging</p> <p>-d4 - Exchange message logging</p> <p>-dmax - Everything</p>
-sr	<directory>	

Intra-day Gateway Restarts

During the trading day, it is desired that the gateway restart quickly. Usually a instrument contract download is not performed on an intra-day restart or the contract refresh is performed in the background.

Logging Format

Column	Description
date time	YYYY-MM-DD HH:MM:ss,mmm - GMT offset

server host name	
server name	-n <name> parameter
process PID / thread	
logging category	ERROR, WARN, INFO, DEBUG
internal component	
log message	

```
2013-03-12 20:11:40,253000 -0500 DESK-davidl mgftest  
[8872:main]/INFO:[mgf.configuration] - <Log Message>
```

< [End2End Metrics](#) | [Document Overview](#) | [Support](#) >

Support

For support during the process of developing a custom SD gateway, please call or email your local Orc support desk. They will be able to assist you directly or forward your request to the appropriate Orc engineering personnel.

<http://www.orc-group.com/About-Orc/Contact/>

< [Orc System Integration](#) / [Document Overview](#)