Intrade XML API

Version 0.9
Last Update: 11 Feb 2008

Change Log

11 Feb	Aggregated several discrete documents into one master file

1 Introduction

The Intrade API consists of two parts:

- Data Retrieval API – This is used for determining the unique IDs of the listed markets and for checking the current and historical market data for those markets. This API is a simple set of URLs with various parameters that is requested over HTTP.

- Trading API – This is used to log in and enter orders and to check individual data. E.g. account balance, trading history, open orders. This is a simple XML over HTTP based API. To avoid unnecessary overheads, there are no wrappers around the XML.

This document contains instructions for both parts of the API. There is also some example code in various languages to help you get started.

2 Terms and Conditions of Usage

Users of the API must agree to the terms of usage which are available online at the following location:

http://www.intrade.com/jsp/intrade/misc/ToS/Intrade_API_ToS.html

If a user fails to abide by the Terms and Conditions or contravenes the Acceptable usage policy in this document, Intrade reserves the right to restrict access to the API at any time and without notice.

3 Data Retrieval API

Retrieving market data for intrade is a two step process.
1. Find the IDs of the contracts (markets) that you are interested in.
2. Using the contract IDs, get the current prices.

The XML documents and how to use them are described in this chapter.

3.1 Active contract listing

This is a list of all active contract and contracts that have expired(settled) within the last 24hrs.
url:

```
http://api.intrade.com/jsp/XML/MarketData/xml.jsp
```

The xml document is arranged in a hierarchical structure as follows:

```
Event Class
  --Event Group
    --Event
      ---Contract
```

To find the contract you are interested in, move down the tree searching for the relevant branch. Once you find a contract, you can find its ID and use this get market data using the market data interface.

Note: If you only need a list of contracts in a single class (e.g. Financials) then a subset of the contracts can be requested using the XMLForClass.jsp (see below).

Example response:

```
<MarketData intrade.timestamp="1096377463359">
<EventClass id="19">
<name> <![CDATA[ Politics ]]>   </name>
<EventGroup id="307">
<name>  <![CDATA[ 2008 Elections  ]]>   </name>
<Event EndDate="1099419900000" StartDate="1099386600000" groupID="290"
id="19658">
<Description />
<name> <![CDATA[ US Election Combinations  ]]>   </name>
<contract ccy="USD" id="145192" inRunning="false" state="O" tickSize="0.1"
tickValue="0.01" type="PX">
<name> <![CDATA[ Republicans to retain control of Presidency, Senate & House
of Representatives  ]]>   </name>
<symbol> <![CDATA[ GOP.SWEEP.2008  ]]>   </symbol>
<totalVolume>1347</totalVolume>
</contract>
</Event>
</EventGroup>
</EventClass>
</MarketData>
```

3.2 All Contracts By Event Class

To get a list of contracts in a specific class use the following url:
A subset of this data based on EventClass(e.g. Politics, Financial etc.) can be requested with the following url:

```
http://api.intrade.com/jsp/XML/MarketData/XMLForClass.jsp?classID=X
```

where X is the id of the EventClass

e.g.

```
http://api.intrade.com/jsp/XML/MarketData/XMLForClass.jsp?classID=19
```

This will bring back a document containing Political contracts only.

3.3 Price Information

Current price information is retrieved using the ContractBookXML.jsp.

url:

```
http://api.intrade.com/jsp/XML/MarketData/ContractBookXML.jsp
```

Parameters:

id - the id of the contract for which marketdata is to be returned. Multiple ids may be specified. (e.g. id=23423&id=34566&...). The id of a contract can be got from the active contract listing (xml.jsp)

timestamp - a timestamp to indicate a cutoff period for marketdata. Contracts whose marketdata has not changed since this timestamp will not be displayed. This is useful for an application that are maintaining current information and only need information about updates. Timestamps are represented by the number of milliseconds since January 1, 1970, 00:00:00 GMT.

depth – the depth of orders that are returned. This defaults to 5. i.e. a maximum of 5 bid/ offer prices are returned. If you just need the best price you should pass in “depth=1”

example:

```
http://api.intrade.com/jsp/XML/MarketData/ContractBookXML.jsp?
id=21433&id=11738&timestamp=0
```

result:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContractBookInfo lastUpdateTime="1096381561235">
<contractInfo conID="11738" lstTrdPrc="68.0" lstTrdTme="1096380611680"
state="O" vol="546074">
<symbol><![CDATA[PRESIDENT.GWBUSH2004]]></symbol>
<orderBook>
<bids>
<bid price="67.7" quantity="1"/>
<bid price="67.6" quantity="1"/>
<bid price="67.5" quantity="8"/>
<bid price="67.4" quantity="1"/>
<bid price="67.3" quantity="4"/>
</bids>
<offers>
<offer price="69.0" quantity="1000"/>
<offer price="69.6" quantity="2"/>
<offer price="69.7" quantity="90"/>
<offer price="69.9" quantity="50"/>
<offer price="70.0" quantity="225"/>
</offers>
</orderBook>
</contractInfo>
</ContractBookInfo>
```

3.4 Contract Information

Other contract data can be retrieved using ConInfo.jsp.

url:

```
http://api.intrade.com/jsp/XML/MarketData/ConInfo.jsp
```
Parameters:
id - the id of the contract for which information is to be returned. Multiple ids may be specified. (e.g. id=23423&id=34566&...). The id of a contract can be got from the active contract listing (xml.jsp)

example:
```
http://api.intrade.com/jsp/XML/MarketData/ConInfo.jsp?id=11738
```
result:

```
<?xml version="1.0" encoding="UTF-8"?>
<conInfo>
<contract ccy="USD" close="67.5" conID="11738" dayhi="69.6" daylo="68.0"
dayvol="1237" lifehi="75.0" lifelo="49.0" lstTrdPrc="68.0"
lstTrdTme="1096380611680" maxMarginPrice="100.0" minMarginPrice="0.0"
state="O" tickSize="0.1" tickValue="0.01" totalvol="546.1k" type="PX">
<symbol><![CDATA[PRESIDENT.GWBUSH2004]]></symbol>
</contract>
</conInfo>
```

3.5 Historical Trading Info (Closing Prices)

Historical closing price and session hi/lo data can be retrieved using the following ClosingPrice.jsp page:

url:

```
http://api.intrade.com/jsp/XML/MarketData/ ClosingPrice.jsp
```

Parameters:

conID - the id of the contract for which information is to be returned. Single id only. The id of a contract can be got from the active contract listing (xml.jsp)

http://api.intrade.com/jsp/XML/MarketData/ClosingPrice.jsp?conID=11738

result:

```
<?xml version="1.0" encoding="UTF-8"?>
<ClosingPrice timestamp="1096381661106">
…..
<cp date="7:26AM 09/22/04 BST" dt="1095834388000" price="70.0"
sessionHi="70.5" sessionLo="68.5"/>
<cp date="7:26AM 09/23/04 BST" dt="1095920796000" price="68.7"
sessionHi="70.0" sessionLo="66.6"/>
<cp date="7:26AM 09/24/04 BST" dt="1096007201000" price="66.8"
sessionHi="69.0" sessionLo="66.6"/>
<cp date="7:26AM 09/25/04 BST" dt="1096093587000" price="67.2"
sessionHi="68.4" sessionLo="65.1"/>
<cp date="7:26AM 09/26/04 BST" dt="1096179980000" price="67.5"
sessionHi="68.5" sessionLo="66.5"/>
<cp date="7:26AM 09/27/04 BST" dt="1096266370000" price="68.3"
sessionHi="69.7" sessionLo="67.0"/>
<cp date="7:26AM 09/28/04 BST" dt="1096352774000" price="67.5"
sessionHi="69.6" sessionLo="67.5"/>
</ClosingPrice>
```

3.6 Daily Time and Sales

Time and Sales data information is retrieved using the TimeAndSales.jsp.
url:

```
https://api.intrade.com/jsp/XML/TradeData/TimeAndSales.jsp?conID=XXX
```

Parameters:
conID - the id of the contract for which marketdata is to be returned. Single id only. The id of a contract can be got from the active contract listing (xml.jsp)

The data is returned in CSV format.

The format is:
UTC Timestamp, BST Datetime, Price, volume.

example:

```
https://api.intrade.com/jsp/XML/TradeData/TimeAndSales.jsp?conID=11738
```

result:

```
1095880647289,      20:17:27 09/22/04 BST,     69.0,  67
1095880647289,      20:17:27 09/22/04 BST,     68.9,  30
1095880647289,      20:17:27 09/22/04 BST,     68.8,  10
1095882472743,      20:47:52 09/22/04 BST,     68.6,  10
```

4 Trading API

4.1 Introduction

This document explains the usage of the xml interface to the intrade system.
It does not explain the working of intrade exchange, so users should be familiar with normal usage (using html interface) before attempting to use this interface.

Note: In order to aid the monitoring of applications, all requests should now use an additional “appID” parameter. The value of the parameter will be supplied by the exchange to users when applying to the exchange to use the API. This supplied id can be appended with a version number if wished.

```
 E.g. For an appID of XXX, the request parameter may be supplied as “XXX.v1”
<appID>XXX.v1</appID>
```

4.2 Request Format

Requests should be posted over http using SSL. Requests to test system need not use SSL.
The url that will handle requests will be https://SERVER_NAME/xml/handler.jsp where SERVER_NAME will be api.intrade.com for the live system. Information on available test systems is given in the appendix.

Requests are posted to the intrade site as xml with the following format:

```
 E.g. For an appID of XXX, the request parameter may be supplied as “XXX.v1”
<appID>XXX.v1</appID>
```

The sessionData element must contain a valid key obtained using the “getLogin” operation. The username element is used by intrade for logging. This element should also use the data returned by the “getLogin” operation.

4.3 Response Format

The response is an XML document with the following format.

```
<tsResponse resultCode=RESULTCODE requestOp=OPERATION timestamp=".." >
    ...result info        <!—elements depend on operation -->
  <faildesc>...</faildesc>
  <errorcode>...</errorcode>
  <sessionData>...</sessionData>
</tsResponse>
```

The resultCode is “0” for success, otherwise “-1”.
The sessionData is an updated passport that can be used in subsequent requests. Each sessionData can be used multiple times and have a lifetime of approximately 5 hours. SessionData elements should only be sent over SSL. (Note that it is not necessary to use SSL on the test system)

4.4 Supported Operations

Operation	**Use**
getLogin	login to get valid sessionData
getBalance	get frozen/available cash balance
updateMultiOrder	enter orders
cancelMultipleOrdersForUser	Cancel multiple orders by order ID
getCancelAllInContract	Cancel all orders in Contract
getCancelAllBids	Cancel all buy orders in Contract
getCancelAllOffers	Cancel all sell orders in Contract
getCancelAllOrdersInEvent	Cancel all sell orders in Event
cancelAllOrdersForUser	Cancel all orders for a user
getPosForUser	get positions for user
getOpenOrders	get open orders for user
getOrdersForUser	get orders by order ID
getUserMessages	get trading messages for users
setAsRead	delete trading messages for users

Operation getLogin

Description: Use to login to get valid sessionData and username. sessionData element is valid for 5hrs or until endOfDay (2am EST approx)

```
<xmlrequest requestOp="getLogin">                 <!-- NOTE: SSL only>
<membershipNumber>...</membershipNumber>      <!-- required -->
    <password>....</password>                 <!-- required -->
</xmlrequest>
```

Response Example

```
 <?xml version="1.0" encoding="UTF-8"?>
<tsResponse requestOp="getLogin" timestamp="1071059184925" resultCode="0">
<username>ed</username>
<faildesc><![CDATA[Ok]]></faildesc>
<sessionData>….</sessionData>
</tsResponse>
```

Operation getBalance

Description: Use frozen and available cash balance

Example:

```
  <xmlrequest requestOp="getBalance">
    <sessionData>...</sessionData>
  </xmlrequest>
```

Response Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<tsResponse requestOp="getBalance" timestamp="1071059326592" resultCode="0">
<ccy>USD</ccy>
<available>4172.68</available>
<frozen>95.67</frozen>
<faildesc><![CDATA[Ok]]></faildesc>
<sessionData>…</sessionData>
</tsResponse>
```

Operation updateMultiOrder

Description: Use to enter new orders into the exchange.

Example:

```
  <xmlrequest requestOp="updateMultiOrder">
    <timeInForce>GTC</timeInForce>            <!-- required. can be GTC,
GFS, GTT  -->
    <timeToExpire>1064602760453</timeToExpire><!-- timestamp required if
timeInForce set to GTT  -->
    <username>myname</username>        <!-- required -->
    <sessionData>...</sessionData>
    <cancelPrevious>...</ cancelPrevious> >  <!—optional – "true" if all
previous orders in contracts for which orders are being entered should be
cancelled-->
    <quickCancel>...</ quickCancel  >  <!—optional – "true"  to  give
enhanced order cancel speed when used with the "cancelPrevious option. NB if
quick cancel is used the list of cancelled order ids will not be returned→

    <order conID="19" limitprice="1" quantity="10" side="B" />           <!--
each order must have conID, limitprice, quantity and side attributes. Side
must be "B" (Buy) or "S" (Sell)-->
    <order conID="19" limitprice="99" quantity="10" side="S"  />  <!-- can
be any number of orders in single request-->
  </xmlrequest>
```

The response contain information about all new orders entered, along with a list of orders that have been cancelled (if any)
Result Example:

```
<tsResponse requestOp="updateMultiOrder" timestamp="1068146606139"
resultCode="0">
   <order orderID="3486">
      <conID>17</conID>
      <side>BUY</side>
      <quantity>10</quantity>
      <limitprice>1.0</limitprice>
      <success>true</success>
      <timeInForce>0</timeInForce>
      <timeToExpire>1064602760453</timeToExpire>
   </order>

   <order orderID="null">
      <conID>50347</conID>
      <side>SELL</side>
      <quantity>10</quantity>
      <limitprice>99.0</limitprice>
      <success>false</success>
      <failreason>
         <![CDATA[The requested contract '50347' doesn't exist in the
ContractCache or the Database]]>
      </failreason>
      <timeInForce>0</timeInForce>
      <timeToExpire>1064602760453</timeToExpire>
   </order>


<orderCancelList>
< ordID>11111</ordID >
< ordID>11112</ordID >
< ordID>11113</ordID >
< ordID>11114</ordID >
</orderCancelList>



   <faildesc>
      <![CDATA[Ok]]>
   </faildesc>

   <sessionData></sessionData>
</tsResponse>
```

Operation multiOrderRequest

Description: Used for the same purpose as "updateMultiOrder". However this request send orders to the matching engine in parallel. This can be more efficient when entering multiple orders in non-linked contracts. However, users must be"Gold API" enabled to make for this to work most efficiently.

Example:

```
<xmlrequest requestOp="multiOrderRequest">

<username><![CDATA[username]]></username>
    <cancelPrevious>...</ cancelPrevious> >  <!—optional – "TRUE" if all
previous orders in contracts for which orders are being entered should be
cancelled-->

<quickCancel>...</ quickCancel >   <!—optional – "TRUE"  to  give enhanced
order cancel speed when used with the "cancelPrevious option. NB if quick
cancel is used the list of cancelled order ids will not be returned→

<order>conID=3,side=B,limitPrice=5,quantity=1,timeInForce=GFS</order>
<order>conID=4,side=B,limitPrice=5,quantity=1,timeInForce=GFS,orderType=F</or
der>

<sessionData>…</sessionData>
</xmlrequest>
```

<order> tag description:

The order tag contains a comma delimited set of attributes describing the order. The full set are described below:

Attribute	Required/Optional	Value range	Description
conID	Required	Any active contract id	Contract ID
side	Required	B or S	Buy or Sell indicator
limitPrice	Required	Any multiple of contract tick size within contract min max range	Limit Price for order
quantity	Required	Integer greater than zero and less than max order size for contract	Quantity of order
timeInForce	Required	GTC, GFS, GTT	Order lifetime:
timeToExpire	Required if timeInForce specified as GTT	Any time in the future. See appendix for format	The maximum time the order will be executed, if the timeInForce is GTT
orderType	Optional, defaults to 'L'	L, T, F	Limit, Touch or FillorKill order types
touchPrice	Required if orderType = 'T'	Any multiple of contract tick size within contract min max range	Touch price for touch orders
userReference	optional	String up to 20 characters	User reference

The response is the same format as that of "updateMultiOrder"

Operation cancelMultipleOrdersForUser
Example:
Description: used to cancel open orders. OrderID elements contain the orderID from the getOpenOrders request
Example:

```
<xmlrequest requestOp=" cancelMultipleOrdersForUser ">
<orderID>3377</orderID>
<orderID>…</orderID>
<orderID>…</orderID>
<sessionData>...</sessionData>
</xmlrequest>
```

Order Cancel Operations
getCancelAllInContract
getCancelAllBids
getCancelAllOffers

Description: Used to cancel all orders, bids or offers in individual contracts.
Example:

```
<xmlrequest requestOp="getCancelAllInContract">
<contractID>1234</contractID>          <!--  required. can only be one-->
<sessionData>...</sessionData>
</xmlrequest>
```

Operation cancelAllInEvent
Description: Used to cancel all orders in an event. The eventID can be retrieved from the allcontracts xml tree.
Example:

```
<xmlrequest requestOp="cancelAllInEvent">
<eventID>1234</eventID>                <!--  required. can only be one-->
<sessionData>...</sessionData>
</xmlrequest>
```

Operation cancelAllOrdersForUser
Description: Used to cancel all orders for a user. This cancels all open orders in all contracts.
Example:

```
<xmlrequest requestOp="cancelAllOrdersForUser">
<sessionData>...</sessionData>
</xmlrequest>
```

Operation getPosForUser

Description: Used get positions for a user

Example:

```
<xmlrequest requestOp="getPosForUser">
<contractID>1234</contractID> <!--  optional -->
<sessionData>...</sessionData>
</xmlrequest>
```

Response example:

```
- <tsResponse requestOp="getPosForUser" resultCode="0">
- <position conID="19">
  <quantity>0</quantity>
  <totalCost>0.0</totalCost>
  <trueTotalCost>0.0</trueTotalCost>
  <totalIM>0.0</totalIM>
  <openIM>0.0</openIM>
  <bidAmt>1.0</bidAmt>           <!--  current bid amt (all open bid
orders)-->
  <bidQty>10</bidQty>            <!--  total num. lots bid -->
  <offerAmt>-99.0</offerAmt>     <!--  current offer amt (all open offer
orders)-->
  <offerQty>-10</offerQty>             <!--  total num. lots offer -->
  <netPL>193.0</netPL>
  </position>
  <faildesc>Ok</faildesc>
  <sessionData>....</sessionData>
  </tsResponse>
```

Operation getOpenOrders
Description: Used get open orders for a user
Example:
```
<xmlrequest requestOp="getOpenOrders">
<contractID>1234</contractID>    <!--  optional -->
<sessionData>...</sessionData>
</xmlrequest>
```
Response example:
```
<?xml version="1.0" encoding="UTF-8"?>
<tsResponse requestOp="getOpenOrders" timestamp="1071057351905"
resultCode="0">
<order orderID="3513">
<conID>1</conID>
<quantity>1</quantity>
<limitprice>1.0</limitprice>
<type>Limit</type>
<side>B</side>
<quantity>1</quantity>
<originalQuantity>1</originalQuantity>
<timeInForce>GTC</timeInForce>
<visibleTime>1070475625464</visibleTime>
</order>
<order orderID="3510">
     …..
</order>
<faildesc><![CDATA[Ok]]></faildesc>
<sessionData>…</sessionData>
</tsResponse>
```

Operation getOrdersForUser

Description: Used orders for a user by order ID.

Example:

```
<xmlrequest requestOp=" getOrdersForUser ">
< orderID >1234</ orderID >      <!--  required, multiple allowed -->
<sessionData>...</sessionData>
</xmlrequest>
```

Response example:

```
 <tsResponse requestOp="getOrdersForUser" timestamp="1071057109475"
resultCode="0">
<order orderID="1234">
<conID>1</conID>
<side>Buy</side>
<quantity>1</quantity>
<orig_quantity>1</orig_quantity>
<limitprice>1.0</limitprice>
<timeInForce>0</timeInForce>
<numFills>0</numFills>
<endTime>1068739413429</endTime>
<status>CEDU</status>
</order>
<order orderID="3474">
       ……
</order>
<faildesc><![CDATA[Ok]]></faildesc>
<sessionData>…:</sessionData>
</tsResponse>
```

Operation getUserMessages

Description: Used get exchange notifications for a user.
This will return the last 50 messages on a users account.
Example:

```
<xmlrequest requestOp="getUserMessages">
< timestamp>...</ timestamp >          <!--  optional if supplied, returns
trades since timestamp -->
<sessionData>...</sessionData>
</xmlrequest>
```

Response:

```
<tsResponse requestOp="getUserMessages" resultCode="0"
timestamp="1097253563961">
<msg msgID="10378145">
<msgID>10378145</msgID>
<conID>169431</conID>
<symbol>DOW.08OCT.-25</symbol>
<readFlag>false</readFlag>
<type>X</type>
<msg><![CDATA[null]]></msg>   <!--  contains orderID for trade messages -->


<price>33.5</price>
<quantity>1</quantity>
<side>B</side>
<timestamp>1097253548000</timestamp>
</msg>
<faildesc><![CDATA[Ok]]></faildesc>
<sessionData>..<sessionData>
</tsResponse>
```

Messages can be any of the following types:

type	name
D	Cancelled By exchange
E	Contract Expiry
J	Rejected Cancel Request
M	Message
R	Rejected Order
S	Contract scratched
T	Execution
V	Stop Activated
X	Expired Order

Operation setAsRead

Description: Used to delete exchange notifications for a user

Example:

```
      <xmlrequest requestOp=" setAsRead">
      <userNotificationID>3456</userNotificationID>  <!--  required. can be
multiple-->
      <userNotificationID>5678</userNotificationID>
      <sessionData>...</sessionData>
      </xmlrequest>
```

Response

```
<tsResponse resultCode=0 requestOp= “setAsRead”  timestamp="123124213423" >

  <faildesc>...</faildesc>
  <errorcode>...</errorcode>
  <sessionData>...</sessionData>
</tsResponse>
```

Operation getTradesForUser
Description: Used to get information on trades that have happened on user account
This can take either a timestamp or a contract id. If a timestamp is passed in, then all trades on this account after the timestamp on all contracts are given.
If a contract is given then all trades on this contract are returned regardless of time.

Example:
```
<xmlrequest requestOp=" getTradesForUser">
<tradeStartTimestamp>…</tradeStartTimestamp>  <!--  optional timestamp-->

<endDate>…</ endDate >   <!--  optional see appendix for known date patterns
-->
<contractID>…</ contractID > <!--  optional (one or none)-->


<sessionData>...</sessionData>
</xmlrequest>
```

Result Example:

```
<tsResponse requestOp="getTradesForUser" resultCode="0"
timestamp="1080575475804">
<trade conID="72388">
<conID>72388</conID>
<orderID>14160028</orderID>
<side>B</side>
<quantity>1</quantity>
<price>30.0</price>
<executionTime>1075460956711</executionTime>
</trade>
<trade conID="98695">
<conID>98695</conID>
<orderID>16095163</orderID>
<side>B</side>
<quantity>1</quantity>
<price>51.0</price>
<executionTime>1077816966226</executionTime>
</trade>
<faildesc><![CDATA[Ok]]></faildesc>
<sessionData>…:</sessionData>

</tsResponse>
```

Operation getGSXToday

Description: Used to as a dummy request to check if the user has any trading messages. If the “checkMessage” element is set to “true” the response will contain the attribute “hasMessages” IF the user has unread messages.

Example:

```
<xmlrequest requestOp="getGSXToday">
      <checkMessages>true</checkMessages>
      <sessionData>...</sessionData>
</xmlrequest>
```

Result Example:

```

<tsResponse requestOp=" getGSXToday" resultCode="0" timestamp="1080575475804"
hasMessages=”1”>
<faildesc><![CDATA[Ok]]></faildesc>
<sessionData>…:</sessionData>
</tsResponse>
```

5 Acceptable usage policy for Intrade API

5.1 Introduction

Misuse of the API provided by Intrade could result in performance problem on the exchange due to flooding the exchange's web-servers and/or database with unnecessary requests. Therefore the rules outlined in this document should be followed by anyone intending to use the API. Applications that do not conform to these rules will be blocked by the exchange, possibly without warning. Access to the exchange will only be re-allowed after the application has been demonstrated to conform using the Intrade test exchange. This will be a lengthy process, so it is in the interests of application writers to make sure their systems comply from the beginning. If there is any doubt, please contact the exchange before running code on the live system.

This document outlines the basic rules. However, if at any point an application is deemed to be putting unnecessary load on the Intrade servers then it may be blocked without notice.

There are many users of the API putting many millions of requests through the Intrade systems every day. No reasonable use of the API should result in loss of access.

In order to aid the monitoring of applications, all requests should now use an additional "appID" parameter. The value of the parameter will be supplied by the exchange to users when applying to the exchange to use the API. This supplied id can be appended with a version number if wished.

E.g. For an appID of XXX, the request parameter may be supplied as "XXX.v1"

```
/jsp/XML/MarketData/ContractBookXML.jsp?appID= XXX.v1&id=…&id=…&timestamp=…
```

5.2 Market Data retrieval

5.2.1 Contract Listing.

```
/jsp/XML/MarketData/xml.jsp
```

As this is a large file, this should only be retrieved at start-up and not more than one time per 15 minutes.

5.2.2 Market Data

1. The ContractBookXML.jsp interface has been designed to allow efficient retrieval of market data ***updates***. It should not be used to repeatedly get all market data for contracts (i.e. used without a “timestamp” parameter). Continual requests for market data for a contract or list of contracts without using a recent timestamp is inefficient and will result in a client being blocked.

2. It is more efficient to request market data for multiple contracts simultaneously, rather than for a single request for each contract.

3. You must only request market data for contracts that are listed in the xml.jsp contract listing. Multiple requests for contracts that do no exist will result in client being blocked.

5.3 Trading Interface

Applications should be designed to cache information regarding an account. Information on an account (i.e. balance, positions, orders, trades) will only change when an order is entered, cancelled, expired or a trade occurs. Therefore this information should only be requested at start-up or when either orders have been entered or when a user message is detected. Continuous unnecessary requests for positions, orders or trades will result in the client being blocked.

In order to detect that a trade has occurred on an account, the user's messages should be checked. This can be done two ways:

1. The "GetGSXToday" request can be called. If this responds with a "hasMessages" attribute, then the app should use the "getUserMessages" request to mark the messages as read and trades can be checked. This returns the last 50 messages on an account and mark all return messages as read.
2. Alternatively, you can poll for new messages by supplying a timestamp to the "getUserMessages" request. This returns any messages generated since the timestamp.

N.B. Orders are permitted via automated order entry practices [through the Exchanges application programmable interface (API) or otherwise]. The Exchange reserves the right to prohibit small-automated orders (9 lots and less) immediately ahead of non-automated orders.

6 Sample code

Here are some example code snippets in various languages. Note that these typically use non-SSL connections – for real money trading an SSL connection should always be used.

6.1 Java

Login example.

```
   public String send() {
      URL remoteURL = null;
      URLConnection connection = null;
      PrintWriter printwriter = null;
      BufferedReader bufferedreader = null;
        StringBuffer response = new StringBuffer();

        try {
            remoteURL = new URL("http://testexternal.tradesports.com:
83/xml/handler.jsp");
            connection = remoteURL.openConnection();
        } catch (Exception e) {
            ....
        }

        connection.setRequestProperty("USER_AGENT", "test [Java 1.3.0]");

        try {
            connection.setDoOutput(true);
            printwriter = new PrintWriter(connection.getOutputStream());
            printwriter.println("<xmlrequest requestOp=\"getLogin\"    >");
            printwriter.println("<membershipNumber>xxx</membershipNumber>");
            printwriter.println("<password>password123</password>");
          printwriter.println("</xmlrequest>");
            printwriter.close();
        } catch (IOException ioe) {
             ....
        }

         try {

            bufferedreader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
          String inputLine= null;
            while ((inputLine = bufferedreader.readLine()) != null) {
                response.append(inputLine);
                response.append("\r\n");
            }

            bufferedreader.close();

        } catch (IOException ioe) {
            return "Error";
        }

        return response.toString();
    }
```

6.2 C#

Login example

```
string request = string.Format("<xmlrequest 
requestOp=\"getLogin\"><membershipNumber>{0}</membershipNumber><password>{1}<
/password></xmlrequest>", "user", "password");
                    System.Text.ASCIIEncoding  encoding=new 
System.Text.ASCIIEncoding();
                    byte[] bytes = encoding.GetBytes(request);

                    //Try to log in
                    HttpWebRequest req = (HttpWebRequest)
HttpWebRequest.Create("http://testexternal.tradesports.com/xml/handler.jsp");
                    req.Method = "POST";
                    req.ContentLength = bytes.Length;
                    req.ContentType = "application/x-www-form-urlencoded";
                    System.IO.Stream reqStream = req.GetRequestStream();
                    reqStream.Write(bytes, 0, bytes.Length);
                    reqStream.Close();
                    HttpWebResponse response = 
(HttpWebResponse)req.GetResponse();

                    System.IO.Stream responseStream = 
response.GetResponseStream();
                    System.Xml.XmlReaderSettings settings = new 
System.Xml.XmlReaderSettings();
                    settings.IgnoreComments = true;
                    settings.IgnoreWhitespace = true;
                    System.Xml.XmlReader reader = 
System.Xml.XmlReader.Create(responseStream, settings);

                    reader.ReadToFollowing("tsResponse");
                    string result = reader.GetAttribute("resultCode");
                    if (result != "0")
                    {
                        //Get the faildesc
                        reader.ReadToDescendant("faildesc");
                        reader.ReadStartElement("faildesc");
                        string error = reader.Value;
                        MessageBox.Show(this, "Error logging in. " + error, 
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return;
                    }

                    reader.ReadToDescendant("sessionData");
                    reader.ReadStartElement("sessionData");
                    this._sessionData = reader.Value;


                    reader.Close();
                    responseStream.Close();
                    response.Close();
```

6.3 Perl

Contract listing

```
#!/usr/bin/perl -w

use Net::HTTP;
use XML::Simple;

$conn = Net::HTTP->new(Host => "testexternal.intrade.com") || die $@;
$page= "/jsp/XML/MarketData/XMLForClass.jsp?classID=19" ;

#print "Sending xml : $xmlToSend";

$conn->write_request(GET => $page, 'User-Agent' => "Mozilla/5.0");

#must strip off the headers before the xml data
($code, $message, %headers) = $conn->read_response_headers;

while (1) {
    my $buf;
    my $n = $conn->read_entity_body($buf, 1024);
    die "read failed: $!" unless defined $n;
    last unless $n;
    $responseXML .= $buf;
 }

print "Response xml: $responseXML";
```

Login example

```
#!/usr/bin/perl -w

use Net::HTTP;
use XML::Simple;
$membershipNumber = "49034";
$password = "934053";
$conn = Net::HTTP->new(Host => "testexternal.intrade.com") || die $@;
$xmlToSend = "<xmlrequest requestOp=\"getLogin\" >
 <membershipNumber>$membershipNumber</membershipNumber>
 <password>$password</password>
 </xmlrequest>";

#print "Sending xml : $xmlToSend";

$conn->write_request(POST => "/xml/handler.jsp", 'User-Agent' =>
"Mozilla/5.0", $xmlToSend);

#must strip off the headers before the xml data
($code, $message, %headers) = $conn->read_response_headers;

while (1) {
    my $buf;
    my $n = $conn->read_entity_body($buf, 1024);
    die "read failed: $!" unless defined $n;
    last unless $n;
    $responseXML .= $buf;
 }

#print "Response xml: $responseXML";

print $responseXML;
$xml = new XML::Simple;
$xmlData = $xml->XMLin($responseXML);
$sessionID = "Not Specified";
$sessionID = $xmlData->{sessionData} || die $@;
print $sessionID;
```

6.4 Ruby

Login example

```
require 'net/http'
require 'xmlsimple'
require 'pp'

membershipNumber = "username goes here";
password = "password goes here";
host = “api..intrade.com";

xmlToSend = "<xmlrequest requestOp=\"getLogin\" >
 <membershipNumber>#{membershipNumber}</membershipNumber>
 <password>#{password}</password>
 </xmlrequest>";

Net::HTTP.start(host)  do |http|
  response =http.post("/xml/handler.jsp", xmlToSend)
  doc =XmlSimple.xml_in response.body.to_s
# uncomment to show xml response
#  pp doc
  sessionData = doc["sessionData"][0]
  puts sessionData
end
```

6.5 Python

Login example

```
import urllib2
import pprint
import xml.dom.minidom
from xml.dom.minidom import Node

message =  '<xmlrequest requestOp="getLogin" > \n\
 <membershipNumber>xxxx</membershipNumber> \n\
 <password>yyyy</password> \n\
 </xmlrequest>'

res=""
url='http://api.intrade.com/xml/handler.jsp'

for line in urllib2.urlopen(url, message):
      res+=line

doc = xml.dom.minidom.parseString(res)
print doc.getElementsByTagName("sessionData")[0].childNodes[0].data
```

7 Appendix

7.1 Test System:

To test your system you can use the test system:

```
http://testexternal..intrade.com/
```

Open an account using a browser in the normal way.

7.2 Timestamps

Timestamps are milliseconds since Jan 01 00:00GMT

7.3 Known Date Patterns

The following date patterns may be used where a date is input to the system.
Times are in GMT (London time) unless specified.

```
        "dd/MM/yy HH:mm:ss",
        "dd/MM/yy HH:mm",
        "dd/MM/yy",
        "yyyy-MM-dd",
        "yyyy-MM-dd HH:mm:ss",
        "HH:mm",
        "EEE dd MMM HH:mm:ss zzz yyyy",
        "EEE MMM dd hh:mm:ss zzzz yyyy"
```

A timestamp can also be used in place of a date format.

7.4 Order Status Codes

Code	Description	Comments
LCO	Limit Checked	Limit checked prior to being added to incoming for book.
CEN	Crossing Engine – New	Order entered book.
CEMX	Crossing Engine – execution	This means that a **partial** trade was made.
CEMT	Crossing Engine – Order Touched	Touch order vizualised.
CEDU	Crossing Engine – Deleted by User Request	User would trigger this by canceling the order.
CEDA	Crossing Engine – Deleted by Admin User Request	Admin User would trigger this by canceling the order. Identified by submitterID.
CEDX	Crossing Engine – Deleted by execution (Fill)	This means that a complete trade was made and the

		order was removed from the book.
CEDE	Crossing Engine – Deleted (Expired)	This means the Crossing Engine has removed the order on expiry.
CEDM	Crossing Engine mass delete	Either deleteAllOrders (request) or CE deletes book queue as a result of contract cancellation.
CEDS	Settlement mass delete on expiry of contract	or CE deletes book queue as a result of contract settlement
CENR	Crossing Engine – rejects New Order	In circumstances where the OrderBook will receive a New Order AFTER the Book has changed state where it cannot add the New Order to the book i.e. PAUSE, CANCEL, SETTLE. This situation arises because of the asynchronous write to input queue for Book.
CERC	Crossing Engine – rejects cancellation	In rare circumstance where CE rejects an order e.g. 1) user requests order cancellation, but the original order has been executed prior to receipt.

7.5 Contract States

State Code	Description
I	Initialized
O	Open
P	Paused
C	Session Closed
E	Closed for Expiry
S	Settled (Expired)
X	Cancelled
R	Reversed