Market tracking with the
Bittrex WebSocket API

# Description

The WebSocket API is implemented with SignalR to provide Bittrex users a means in which to query data and receive updates from the Bittrex backend. The implementation can be found at <https://www.bittrex.com/signalr/hubs>.

Once connected to the Hub, applications can query for the current state of a market and register for messages from the server when new data is available for subscribed markets.

If you have any questions, feedback or recommendations for the API support you can post a question in our support center or send mail to support@bittrex.com.

# Topics

* Connecting to the Bittrex Hub
* Bittrex Markets
* Following Market changes
* Handling changes to the market
* Code Samples
* Sample Data

Connecting to the Bittrex Hub

The path to the socket is <https://www.bittrex.com/signalR>. To connect to the generated proxy include the following references in your code:

<script src="Scripts/jquery-1.10.2.min.js"></script>

<script src="Scripts/jquery.signalR-2.1.0.min.js"></script>

<script src="signalr/hubs"></script>

# Bittrex Markets

To get a list of the markets listed on Bittrex use the REST API GetMarketSummaries. For more information on this API please visit <http://www.bittrex.com/home/api>

# Following Market changes

Once you’ve determined which markets you wish to follow and verified they are listed by Bittrex you can retrieve the current state of the market as well as register for updates to the specific market.

You’ll first want to register for updates to the market you’re interested in. To do this you’ll want to register a client callback with the server to be notified when updates are available.

//

// Subscribe for messages for the given market for updates

// You'll want to call this prior to queryExchange state else you

// could miss an update in between

var subscribeExchange = function subscribeExchange(marketName) {

 hub.server.subscribeToExchangeDeltas(marketName)

 .done(function (ret) {

 // Once registered for the callback you can make the initial call

 // to query the initial state of the exchange

 queryExchangeState();

 });

}

function queryExchangeState() {

 hub.server.queryExchangeState(marketName)

 .done(function (exchangeState) {

 // Returns the current orderbooks (buys, sells) and

 // the order history (fills), take note of the current nonce

 // as this is the starting point for updates

 });

}

*Note:* ***Calling queryExchangeState in a loop will result in a ban…***

*Take note of the current nonce as this is your starting point. Updates will come in successive order tracked by the nonce.*

# Handling changes to the market

Prior to registering for updates to current market conditions the client will need to register a callback with the server that will handle the incoming message/update.

This call back is used by the server after subscribeToExchangeDeltas is called. Important things to keep in mind when receiving an update from the server:

* Update returns the market name and a nonce
* Handle the nonces in sequence from the last full pull from queryExchangeState
* If you miss a nonce due to a dropped/delayed connection, you'll need to start fresh to get back in sync

hub.client.updateExchangeState = function updateExchangeState(exchangeStateDelta) {

 // Client code to handle new messages from the server for the market change

}

# Code Samples

## C#

This sample code is from a console implementation in C#. The code leverages the Microsoft.AspNet.SignalR.Client and Newtonsoft.Json client libraries for connecting and handling the JSon messages broadcast to the client. These can both be added to your solution via the NuGet Package manager and searching for ”Microsoft.AspNet.SignalR.Client”.



# Sample Data

{

 {

 "MarketName": "BTC-ETH",

 "Nounce": 277182,

 "Buys": [

 {

 "Type": 0,

 "Rate": 0.10729986,

 "Quantity": 0.02690255

 },

 {

 "Type": 0,

 "Rate": 0.10458714,

 "Quantity": 0.02096395

 }

 ],

 "Sells": [

 {

 "Type": 2,

 "Rate": 0.11041172,

 "Quantity": 574.91312031

 },

 {

 "Type": 1,

 "Rate": 0.1105025,

 "Quantity": 0.0

 },

 {

 "Type": 0,

 "Rate": 0.11050253,

 "Quantity": 0.0051

 }

 ],

 "Fills": [

 {

 "OrderType": "BUY",

 "Rate": 0.11041172,

 "Quantity": 0.21252785,

 "TimeStamp": "2017-06-25T23:26:46.943"

 }

 ]

 }

 }

NOTE: need to get the enum for the actual type of Sell for context