AquaQ Training

HTML5 WebSockets and kdb+

email:
support@aquaq.co.uk

web:
www.aquaq.co.uk

AquaQ Analytics

Revision History

Revision	Date	Author(s)	Description
0.1	February 25, 2014	Glen Smith	First version created

Contents

Chapter 1

Company Overview

AquaQ Analytics Limited is a provider of specialist data management, data analytics and data mining services. We also provide strategic advice, training and consulting services in the area of market-data collection to clients within the capital markets sector. Our domain knowledge, combined with advanced analytical techniques and expertise in best-of-breed technologies, helps our clients get the most out of their data.

The company is currently focussed on four key areas, all of which are conducted either on client site or near-shore:

- Kdb+ Consulting Services: Development, Training and Support;
- Real Time GUI Development Services;
- SAS Analytics Services;
- Providing IT consultants to investment banks with Java, .NET and Oracle experience.

The company currently has a headcount of 30 consisting of both full time employees and contractors and is actively hiring additional resources. Some of these resources are based full-time on client site while others are involved in remote/near-shore development and support work from our Belfast headquarters. To date we have MSAs in place with 6 major institutions across the UK and the US.

Chapter 2

What are HTML5 and WebSockets?

2.1 Brief overview

By the end of this training document you should be able to fully utilize WebSockets and kdb+ together. You should also be familiar with using the web console in Chrome, setting up kdb+ to work with WebSockets, and writing basic JavaScript and HTML5. This will be an introduction to very basic web development, so let us explore some commonly used terminology before continuing.

HTML5 - Hypertext Markup Language is a markup language used to build web pages. HTML5 is the latest version of HTML and comes with many new features, one of which is WebSockets.

WebSockets - WebSockets allows web applications to maintain bidirectional communications with server side processes over one TCP socket. In this case we can create a WebSocket connection between the client (browser) and a server (kdb+ process).

JavaScript - This is a scripting language that is used by all modern browsers to add interactivity to web pages. It is the language we will use to work with WebSockets.

HTTP - Hypertext Transfer Protocol is the protocol that enables a web browser to communicate with a server with the aim of displaying web documents.

DOM - Document Object Model allows programs and scripts to dynamically access and update the content, structure and style of documents. Before this was introduced HTML documents were not changeable inside the browser.

2.2 What are WebSockets?

WebSockets allows web applications to maintain bidirectional communications with server side processes over one TCP socket[1]. The old method of creating a connection and retrieving data between a client and server is called AJAX. The AJAX method consisted of continually polling a connection to receive new data, opening a new connection, downloading data and then closing the connection. With WebSockets, once the connection between the client and server is established it persists and data is sent down to the client whenever new data is created. This method is far more efficient as the client is not required to ask if the server has new data as the server will send the new data when it is updated. HTTP wasn't designed for real-time, full-duplex communication so as WebSockets operates over TCP there could be a 500:1 reduction in unnecessary HTTP header traffic and 3:1 reduction in latency. You can now see why this is so exciting for building new web applications.

2.3 WebSocket protocol

The protocol has two parts: the handshake and data transfer. In order to establish a WebSocket connection the client and server must upgrade from the HTTP protocol to the WebSocket protocol. This is done during their initial handshake via a HTTP request shown below. Once the handshake is completed, data transfer is started and this occurs on a two way communication channel where both parties can send data independently of each other over TCP[2] .

```
GET /text HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: some.host.com:port
Origin: http://some.client.com/app

HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://some.client.com/app
WebSocket-Location: ws://some.host.com:port
```

2.4 When to use WebSockets

- Whenever you need real time updates in your web application
- You need a cross platform front end for a kdb+ server that can be accessed through the internet
- You want to keep network resource usage low whilst providing the best functionality

[1] W3 WebSockets specification: http://www.w3.org/TR/websockets

[2] The WebSocket Protocol: http://datatracker.ietf.org/doc/rfc6455/?include_text=1

Chapter 3

HTML

HTML is a markup language used to create web pages. It uses tags to describe the structure of a document and its content. This will be a brief overview as more insightful tutorials can be found online.

3.1 Tags

Tags are keywords that determine how their content will be interpreted by the browser. An example of a tag is `<div>`, it also has a corresponding closing tag `</div>`. For structure `<div>` tags are used, where the content inside it will belong to that `<div>`. There are tags that are used for type and an example of one is the headings tag `<h1>`. HTML5 is pushing towards using semantic tags such as `<header>`, `<article>` and `<time>` which describe its meaning to the web browser and developer. Tags also have attributes such as class and id which are used in both JavaScript and CSS. An element is a tag and its content.

```
<div class="container">
  <h1 id="header">Basic HTML5 Template</h1>
</div>
```

3.2 CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language. CSS allows for properties of an HTML tag to be changed, such as height, width, background colour and font size. You can assign styles to elements by using its tag name or you can include a class or id attribute. The class attribute allows for styles to be applied to many elements whereas the id attribute only allows styles to apply to one element. There are three ways to include CSS in a HTML document: using inline style, putting code inside `<style>` tags and lastly include the code by linking to a CSS file.

```
<h1 style="font-size:100px:">Basic HTML5 Template</h1>
<style> h1{font-size:100px;} </style>
```

```
<link rel="stylesheet" href="css/main.css" type="text/css" />
```

3.3 Integrating JavaScript

To get JavaScript to work in a HTML document there are three options: using inline JavaScript, putting the JavaScript code inside the `<script>` tags and lastly include the code by linking to a JavaScript file. There is more information on JavaScript in Chapter 4.

```
<button onclick="window.alert('HELLO');">Click me</button>
<script>window.alert("HELLO");</script>
<script src="http://kx.com/q/c/c.js"></script>
```

3.4 HTML5 Template

This is a basic HTML5 template that shows how all the tags are used in one document.

```
<!doctype html>
<html lang="en"><head><meta charset="utf-8">
<title>Basic HTML5 Template</title>
<!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
</script><![endif]-->
<style>
.container { margin:0 auto; width:500px;} /* . denotes class */
#header { font-family:Georgia; } /* # denotes id */
h1{font-size:100px;}
</style>
</head>
<body>
<div class="container">
  <h1 id="header">Basic HTML5 Template</h1>
</div>
<script src="http://code.jquery.com/jquery-latest.min.js" type="text/javascript"></
    script>
<script src="http://kx.com/q/c/c.js"></script> <!-- Include Kx's c.js file -->
<script>
// Enter JavaScript here
</script>
</body></html>
```


3.5 Directory structure

As there are different components required to build a web page it is good practice to work around a logical directory structure. The files below are just used as an example and not all of them are required to build a web page.

```
html
├── css
│   ├── bootstrap-min.css
│   ├── bootstrap-theme.min.css
│   └── main.css
├── img
│   └── ...
├── js
│   ├── vendor
│   │   ├── bootstrap.min.js
│   │   ├── c.js
│   │   ├── d3.v3.min.js
│   │   ├── jquery-1.11.0.min.js
│   │   └── modernizr-2.6.2-respond-1.1.0.min.js
│   └── monitor.js
└── index.html
```

The root directory `html` contains the index.html and all the directories. The `css` directory holds the CSS style sheets that determine the layout of the page. The `img` directory holds any images that you will use on your web page. The `js` directory holds the local JavaScript files and the sub directory `vendor` holds JavaScript libraries. These JavaScript files are stored locally but you could directly link to them in another location such as on a CDN but storing them locally cuts out the dependency on the uptime of other services.

Chapter 4

JavaScript

JavaScript is a scripting language that is used in all modern browsers in order to add interactivity to web pages. In recent years there has been a lot of framework developments using JavaScript. Popular developments include jQuery which is a framework that makes it easier to work with JavaScript and Node.js which is platform to create real time applications based on JavaScript. This will be a brief introduction to JavaScript.

4.1 Data Types

Data types in JavaScript are dynamic, meaning that one variable could be used as a string on one line and then a number on the next. You do not have to declare a type to it. There are only a few data types and these are String, Number, Boolean, Array, Object, Null and Undefined. There is one surprising and equally confusing feature in JavaScript, everything except null and undefined is an object. To create them you must instantiate the data type's object. To see what data type a variable is use `typeof variable`. For a table showing kdb+ data types and their corresponding JavaScript data type see table 6.2.

```
var example = "Hello World";  // String
example = 'Hello World';        // String with single quotes
example = new String("Hello World");  // String created by instantiating String
    object
example = 123.0945;  // Number
example = true;  // Boolean
example = [99,"problems"]  // Array using literal notion
example = new Array(99,"problems");  // Array created by instantiating Array Object
example = {name:"Glen",iq:423};  // Object
example = new Object();  // Empty object created by instantiating Object
example.name = "Glen"; // Adding properties to object
example.iq = 423;
example = null;  // Null
example = undefined; // Undefined
```

4.2 Arrays

Arrays are analogous to lists and they operate in a similar way. You can have elements with mixed data types as shown below. To access an element you must use its index number. Arrays also come with methods and properties.

```
var example = ["that",69,true, {how:"now",brown:"cow"},null,undefined];  // Array
example[0] = "that"; // Accessing element with index 0
example.length; // Returns 6. Length property analogous to count function
example.indexOf(69); // Returns 1. IndexOf method analogous to find (?)
```

4.3 Objects

As mentioned almost everything in JavaScript is an object. They have properties and methods (functions) which are accessed using . notation.

```
Var car = {};  // Using literal notion
car.brand = "Reliant Robin";  // Property
car.wheels = 3;
car.print = function (){ console.log("My car is a " + this.brand + " and it has " +
    this.wheels + " wheels!")}; // Method - Notice how properties are accessed using
    this
car.brand;  // Returns "Reliant Robin";
car.print();  // Returns "My car is a Reliant Robin and it has 3 wheels!"
```

4.4 Functions

A function is a block of code that is executed once it is called. It is good practice to split code into functions that do one thing, this way any errors or unexpected results can be found or changed easily.

```
function hello(name){
  console.log("Hello " + name + "!");
}
hello("Glen"); // Returns "Hello Glen!"
```

4.5 Manipulating HTML content

JavaScript can interact with HTML and change the content of elements, add new elements or delete them.

```
var header = document.getElementById("header"); // Select element
header.innerHTML = "My content has been changed";  // Change content
header.parentNode.removeChild(header); // Delete it
```


Chapter 5

jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. It is the default library used by almost all web developers and is used by 80% of the top 10,000 websites.

5.1 Install

To add the jQuery library in your web document, include the link below just before the `</body>` tag. To check if it is installed you can enter `jQuery` into the console and it should return a result. The code included here will link you to the latest jQuery library, this may not be the best choice in the case that you may have some deprecated jQuery functions included in your code.

```
<script src="http://code.jquery.com/jquery-latest.min.js" type="text/javascript"></
    script>
```

5.2 Manipulating HTML content

jQuery makes it much easier to select HTML elements. It accepts a selector based off CSS rules and some of jQuery's own custom selectors too.

```
var a = document.querySelectorAll('.post__content h2');// HTML method
a[a.length-1].innerHTML = "CCCCChanges";
$('.post__content h2:last-child').html("CCCCCChanges");// jQuery method
```

5.3 Event Handlers

If you would like to add some JavaScript that will run after the user clicks on something or hovers over an element you can use event handlers. They are quite powerful as they allow the developer full control over user interaction.

```
// Makes every link displaying an alert message and prevents user from going to that
    url
$('a').click(function(e){
  alert("No Escape!");
  e.preventDefault();
});
// Once you submit a form you will get an alert.
$('form').submit(function(){
  alert("You have submitted the form");
});
```

5.4 Effects

When creating a web page you want to have full control over what the user sees and how they interact with it. Sometimes HTML and CSS are not enough and you need something more. Sometimes effects are needed and in this case they can be very useful. Take for example the show/hide effect, this could be used for multiple things such as to display messages and then dismiss them. The effect included below is the animate effect, it allows you to set final CSS rules that the element will be animated to from its original properties. This idea could go further and include more advanced techniques such as draggable/droppable elements, autocomplete input forms, slider for changing values etc. With these packages it is easy to see why more desktop applications are being carried over to work in the browser. In the example below there is an event handler "click" attached to the element with a "header" id that will hide this element once it is clicked.

```
$("#header").click(function(){
  $(this).css({"position":"relative"}).animate({left:'250px'});
});
```

5.5 Traversing the DOM

In jQuery, traversing means to move through the DOM and find elements based on their relation to other elements. For example, if a click handler function has been applied to a `button` element and when it is clicked it will hide its parent `div` element. For more detailed information about traversing visit `http://www.w3schools.com/jquery/jquery_traversing.asp`.

```
$("button").click(function(){
  $(this).parent().hide();
});
```


Chapter 6

Using WebSockets

6.1 Getting started

In order to proceed you need the following:

1. Set up a port in your q console using `\p 4321`
2. Web browser - Google Chrome

To become familiar with using the developer console that comes with Chrome, press F12 or go to Tools > JavaScript console. See figure 6.1 for a screenshot. This JavaScript console is similar to your q console, you can run commands and expect a return. Try the basic stuff such as `1+1`, `var date = new Date()` or `typeof date`.

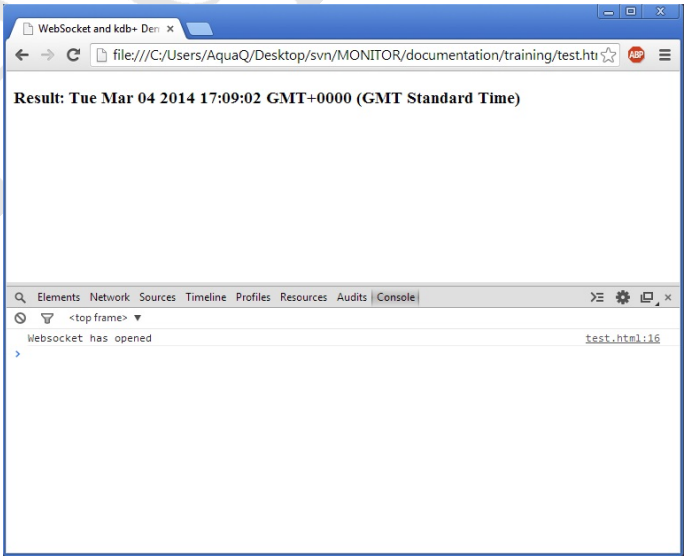

Figure 6.1: Screenshot of Chrome and its developer console

6.2 How to use WebSockets

To create a WebSocket connection, just create a new WebSocket instance including the URL to the server as an argument. It parses the URL argument string to obtain host, port, resource name and whether it is over a secure connection (wss:// instead of ws://). The port is optional as the default port used for ws is 80 and wss is 433. The following code snippet should be entered into the developer console.

```
var websocket = new WebSocket("ws://homer.aquaq.co.uk:4321")
```

In JavaScript `.` is used to access attributes (properties) and methods (functions) of an object. One of the attributes of a WebSocket is called readyState. The readyState attribute represents the state of the connection and these are shown by 4 possible numeric values:

- 0 - CONNECTING - The connection has not yet been established
- 1 - OPEN - The connection is established and communication is possible.
- 2 - CLOSING - The connection is closing.
- 3 - CLOSED - The connection has been closed or could not be opened.

```
websocket.readyState
> 1
```

The send(data) method transmits data using this connection. The method must return true if the connection is still established (and the data was queued and sent successfully) or false if the connection is closed. You will be sending a string or a variable in the send method.

```
websocket.send("1+1")
```

The close() method will close the WebSocket connection and change the readyState attribute to 3.

```
websocket.close()
websocket.readyState
> 3
```

An event occurs whenever something changes. These are handled by what are called Event handlers.

Event Handler name	When it's triggered
onopen	WebSocket opening
onclose	Closes
onmessage	A message is received from the server
onerror	An Error occurs

Table 6.1: Event handlers and their use

Here is how they are used. This will log a message in the console when the WebSocket opens.

```
websocket.onopen = function() { console.log("Websocket has opened"); }
websocket.onclose = function() { console.log("Websocket has closed"); }
websocket.onmessage = function(e) { console.log(deserialize(e.data)); }
websocket.onerror = function(err) { console.log("Error - " + err); }
```

Another attribute that we must change is the binaryType attribute. This is required by the c.js library and this attribute makes sure that the data comes as a typed array making it more efficient by minimizing network traffic. It provides the `deserialize` function shown inside the onmessage handler.

```
websocket.binaryType = 'arraybuffer';
```

In the q console you have to modify the .z.ws function which is the function that kdb+ uses once a WebSocket message is received. The following function will interpret the value of the string that was received and then send that value asynchronously back down the handle `.z.w`. To get a list of handles that are currently connected to your process use `.z.W`. *Warning* this isn't proof that each of the handles is a WebSocket connection as the handles could be other processes that are connected via IPC. Another way to do this would be to add handles that connect via WebSockets to a dictionary inside the `.z.ws` function but please use with caution.

```
q).z.ws:{neg[.z.w] -8!value -9!x}
q).z.po:{-1"connection opened!"; 0N!x}
q).z.pc:{-1"connection closed!"; 0N!x}
```


6.3 Data Types

Before we proceed there is something that must first be clarified. In q the data types are different compared to other languages and below is a table showing how they are interpreted by JavaScript. JavaScript tries to convert date data types to the native JavaScript date object but this can prove problematic as it is not always accurate.

kdb+	JavaScript
list	array
dictionary	object
table	array of objects
boolean	boolean
byte	number
short	number
int	number
long	number
real	number
float	number
char	string
symbol	string
month	object
date	object
datetime	object
minute	object
second	object
time	object

Table 6.2: kdb+ data types and their equivalents in JavaScript

6.4 Formatting data in kdb+

When sending data from kdb+ to the web browser, you have to know what format it is going to be in order to parse it correctly. You can only send strings or binary data [1] but there are multiple ways of formatting data and they include:

- Binary - Serialized raw data
- JSON - Formatted JSON string
- CSV - Converts table values to strings
- Formatted string - Format a table into HTML and then send it

[1] Section 5.6: `https://tools.ietf.org/html/rfc6455`

Binary allows for the largest volume and precision of data to be sent. Even though unserialized strings can be sent through WebSockets, all of the data should be serialized before it is sent to the client. This is done by using the `-8!` function which returns a IPC byte representation of the data. On the JavaScript side when the data is deserialized it will interpret the data types as shown in table 6.2 and so it is not required to be formatted in any particular way.

JSON is a lightweight data-interchange format for passing around objects that contain name/value pairs, arrays and other objects. It is commonly used in web development when passing data from server to client. It is sent as a string from the server and parsed into an object on the client side[2]. After it is parsed using the `JSON.parse` function it is used in the same way as the binary data type above and for this reason the terms are often used synonymously when talking about the client side. On the kdb+ side, the data must be formatted into a JSON string which requires additional overhead and adds more code. Data must be parsed into JSON on the q side.

It is possible to send a CSV formatted string of a table which can then be formatted into a HTML table by JavaScript. If you convert a table to CSV by using the `.h.cd` function and compare its size to a normal table it will show that the CSV string is much smaller than the table itself, which may come into consideration if you are updating often and have to conserve network resources. An example of sending and parsing CSV data is shown in section 7.3.

You could format a table into HTML inside kdb+ and once it is sent the JavaScript could just print the table instead of parsing it like the previous options. The downside to this is that it has a bigger overhead as q must format each table into HTML using extra HTML tag strings and there is also more network usage.

6.5 Serialization

Serialization is the process of taking objects and converting their state information into a form that can be stored or transported. When sending messages from the kdb+ server to your client you must serialize them by using the `-8!` function and deserialize incoming messages using the `-9!` function. Kx have released a serialization JavaScript library in order to serialize data using JavaScript. JavaScript natively does not have these functions which is why they must be included. As seen in section 7.1 a JavaScript library has been included using `<script src="http://kx.com/q/c/c.js"></script>`. By doing this you will be able to use the `serialize` and `deserialize` functions throughout your JavaScript as long as your script appears below the include link. Please note that it would be better if you downloaded and stored the c.js file locally as this would make your application independent from the uptime of kx.com and any changes in the location of the file.

[2] More information about JSON `http://www.json.org/`

6.6 Display tables

To display a table on a web page as it appears in your q console, you must format it to HTML. In order to do so you must understand the data that is sent from kdb+. In this case a table is sent as an object which JavaScript can easily work with. The code snippet below along with the code found in section 7.4 show how JavaScript will format an unkeyed table to HTML. To do the same for a keyed table it is a little more difficult, this code is included in section 7.5. This code must be included inside your script tags and called with the deserialized data as its argument. If you want to get familiar with the structure of a table visit `http://www.w3schools.com/html/html_tables.asp`

```
function jsonTbl(data){
  var table,colheaders,index,row,col;
  table = '<table><thead><tr>';
  for(colheaders in data[0]){ // Set up column headers
    table+= '<th>' + colheaders + '</th>';
  }
  table+= '</tr></thead><tbody>';
  for(index in data){ // Construct table body
    row = data[index];
    table+= '<tr>';
    for(col in row){
      table+= '<td>' + row[col] + '</td>';
    }
    table+= '</tr>';
  }
  table+= '</tbody></table>';
  return table;
}
```

6.7 Data Handling

When sending data, you want JavaScript to interpret it properly. For simple data such as the result of `"1+1"` it will be enough just to print that. Things get more complicated when you begin sending complex data objects such as a table. If you send a unkeyed table how would your JavaScript code know what to do with it? To do this you should use the dictionary format `(`type`data)!(type t;t)`. You then interpret it on the JavaScript side using a data handling function such as the one below. This is a basic data handling function that decides what to do with the data based upon its type. It tries to match cases for unkeyed and keyed tables or defaults if it is of another type. The `jsonTbl` function used below is found in section 7.5.

```
function dataHandler(data){
  switch(data.type){
    case 98:  // Unkeyed Table
      return jsonTbl(data.data,false);
    break;
    case 99:  // Keyed Table
      return jsonTbl(data.data,true);
    break;
    default:
      return data.data;
    break;
```



```
  }
}
```

6.8 Interpreting client data

Just as you handle data on the JavaScript side, it is also good practice to do something similar on the kdb+ server side. With our current `.z.ws` setup it is easy to access functions, it is simply a string `"function[argument1;agrument2]"` with the `value` function applied to it. The current setup also means you will have no control over errors or how a certain data type is interpreted. The way to fix this is to use an evaluation function that is called every time the client sends a message to the server. The code below checks if the input value is a dictionary (JavaScript sent an object) and then checks if it is in the proper format which is `(`func`arg1`arg2)!("function";12;45)` and will return the function result. It doesn't accept strings. This is the beginning of properly handling different client data types and error trapping.

```
q)evaluate:{$[99h=abs[type x];$[not `func in key x;'"dictionary must contain `func in
     key";(value x`func) . value `func _ x];'"cannot handle this type"]}
q)evaluatetrapped:{@[evaluate;x;{'"failed to execute ",(-3!x)," : ",y}[x]]};
q).z.ws:{neg[.z.w] -8!evaluatetrapped[-9!x]}

JavaScript
var query = {func:"test",arg1:2,arg2:3};
```

On applications that are public facing it would be a good idea to only allow the client a certain amount of access, such as a whitelist of functions. It is good practice to restrict access, control what data the server will interpret and what data is returned.

6.9 Sending data when it's updated

As mentioned in the introduction, the major advantage to WebSockets is the ability for the server to send data when it is created or when previous data is updated. For a basic non bulletproof example, we will use a table that gains a new row each time it is updated. This will give you an analogous example that you could extend to many projects.

1. Create a function that sends data to a list of WebSocket handles
2. Create a function that checks the table count by comparing it to a variable that stores the previous count
3. Put that inside the `.z.ts` function and set the timer to `\t 1000`
4. Connect via WebSockets

```
q)sendData:{neg[x] -8!y}
q)prevc:0;
q)checkTable:{$[prevc<count x;[prevc::count x;sendData\:[key .z.W;x]];::]}
```



```
q).z.ts:{checkTable[table]}
q)\t 1000
```


Chapter 7

Examples

7.1 index.html file

7.1.1 HTML file

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>WebSocket and kdb+ Training</title>
<meta name="author" content="Glen Smith at AquaQ Analytics">
<!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
<link rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
<h3>Result:</h3>
<div id="result"></div>
<script src="http://kx.com/q/c/c.js"></script>
<script src="main.js"></script>
</body>
</html>
```

7.1.2 CSS - main.css

```
table{ border-spacing:0px; }
th{ border-bottom:1px solid #DDD; }
td{
  padding:5px;
  margin:0;
  border-width:0px 0px 1px 0px;
  border-color:#CCC;
  border-style:solid;
}
.keyed{ background:#CCC; }
```

7.1.3 JavaScript - main.js

```
var query = "x";        // Enter query here

var result = document.getElementById("result");
var websocket = new WebSocket("ws://HOST:PORT");
websocket.binaryType = 'arraybuffer';   // Required by c.js
websocket.onopen = function() { // Event handler configuration
  console.log("Websocket␣has␣opened");
  websocket.send(serialize(query));
}
websocket.onclose = function() {
  console.log("Websocket␣has␣closed");
}
websocket.onerror = function(err) { console.log(err); }
websocket.onmessage = function(e) {
  var data = deserialize(e.data);
  console.log(data);
}
```

7.2 Retrieve atom

7.2.1 HTML file

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>WebSocket and kdb+ Demonstration - 1</title>
  <meta name="author" content="Glen␣Smith␣at␣AquaQ␣Analytics">
  <!--[if lt IE 9]>
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
</head>
<body>
<h3>q code</h3>
<code>.z.ws:{neg[.z.w] -8!value -9!x}</code>
<h3>Result:</h3><div id="result"></div>
<script src="http://kx.com/q/c/c.js"></script>
<script src="example1.js"></script>
</body>
</html>
```

7.2.2 JavaScript - example1.js

```
var query = "1+1";      // Enter query here

var result = document.getElementById("result");
var websocket = new WebSocket("ws://HOST:PORT");
websocket.binaryType = 'arraybuffer';   // Required by c.js
websocket.onopen = function() { console.log("Websocket␣has␣opened"); websocket.send(
    serialize(query)); }        // Event handler configuration
websocket.onclose = function() { console.log("Websocket␣has␣closed"); }
websocket.onmessage = function(e) { var data = deserialize(e.data); console.log(data)
    ; result.innerHTML = data; }
websocket.onerror = function(err) { console.log(err); }
```


7.3 CSV table and format to HTML

Create a table `t` and copy the q code to your open q console.

7.3.1 HTML file

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>WebSocket and kdb+ Demonstration - 2</title>
  <meta name="author" content="Glen␣Smith␣at␣AquaQ␣Analytics">
  <!--[if lt IE 9]>
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
</head>
<body>
<h3>q code</h3>
<code>tblCsv:{"\\n" sv .h.cd x};.z.ws:{x:value -9!x; neg[.z.w] -8!$[(type x) in 98 99
    h;(`table;tblCsv[x]);(`result;x)]}</code>
<h3>Result:</h3><div id="result"></div>
<script src="http://kx.com/q/c/c.js"></script>
<script src="example2.js"></script>
</body>
</html>
```

7.3.2 JavaScript file - example2.js

```
var query = "t";        // Enter query here

var result = document.getElementById("result");
var websocket = new WebSocket("ws://HOST:PORT");
websocket.binaryType = 'arraybuffer';   // Required by c.js
websocket.onopen = function() { console.log("Websocket␣has␣opened"); websocket.send(
    serialize(query)); }        // Event handler configuration
websocket.onclose = function() { console.log("Websocket␣has␣closed"); }
websocket.onmessage = function(e) { var data = deserialize(e.data); console.log(data)
    ; result.innerHTML = dataHandler(data); }
websocket.onerror = function(err) { console.log(err); }
function csvTbl(data){  // Change csv string into a HTML formatted table
  var i,table = "",rows = data.split("\\n"),header = rows[0].split(","),rowcount =
      rows.length;
  table += "<table><thead><th>" + header.join("</th><th>") + "</th></tr></thead><
      tbody>";
  for(i=1;i<rowcount;i++){
    table+="<tr><td>" + rows[i].split(",").join("</td><td>") + "</tr>";
  }
  table+="</tbody></table>";
  return table;
}
function dataHandler(data){     // What to do with the data
  switch(data[0]){
    case "result":
      return data[1];
      break;
    case "table":
      return csvTbl(data[1]);
    break;
```



```
  }
}
```

7.4 JSON table and format to HTML

Create an unkeyed table `t` and copy the q code to your open q console. You must load the json_training.q file in your kdb+ process.

7.4.1 HTML file

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>WebSocket and kdb+ Demonstration - 3</title>
<meta name="author" content="Glen␣Smith␣at␣AquaQ␣Analytics">
<!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
<style>table{border-spacing: 0px;}td,th{border:1px solid #DDD;}.keyed{background:#CCC
    ;}</style>
</head>
<body>
<h3>q code</h3>
<code>.z.ws:{x:value -9!x; neg[.z.w] -8!.utl.json[$[(type x) in 98 99h;(`type`data)!(
    type x;x);(`type`data)!(`result;x)]]}</code>
<h3>Result:</h3><div id="result"></div>
<script src="http://kx.com/q/c/c.js"></script>
<script src="example3.js"></script>
</body>
</html>
```

7.4.2 JavaScript file - example3.js

```
var query = "t";        // Enter query here
var result = document.getElementById("result");
var websocket = new WebSocket("ws://HOST:PORT");
websocket.binaryType = 'arraybuffer';   // Required by c.js
websocket.onopen = function() { console.log("Websocket␣has␣opened"); websocket.send(
    serialize(query)); }        // Event handler configuration
websocket.onclose = function() { console.log("Websocket␣has␣closed"); }
websocket.onmessage = function(e) { var data = JSON.parse(deserialize(e.data));
    result.innerHTML = dataHandler(data); }
websocket.onerror = function(err) { console.log(err); }
function jsonTbl(data){
  var table,colheaders,index,row,col;
  table = '<table><thead><tr>';
  // Set up column headers
  for(colheaders in data[0]){
    table+= '<th>' + colheaders + '</th>';
  }
  // Construct table body
  table+= '</tr></thead><tbody>';
  for(index in data){
    row = data[index];
```



```
    table+= '<tr>';
    for(col in row){
      table+= '<td>' + row[col] + '</td>';
    }
    table+= '</tr>';
  }
  table+= '</tbody></table>';

  return table;
}
function dataHandler(data){
  switch(data.type){
    case "result":
      return data.data;
    case 98:
      return jsonTbl(data.data,false);
    case 99:
      return jsonTbl(data.data,true);
  }
}
```

7.5 JSON table function for formatting keyed and unkeyed tables

If the table is keyed, the second argument should be true and vice versa. You must modify `.z.ws` and the `dataHandler` function. A suitable `dataHandler` function is shown in 7.4. This function should be added to the web page's JavaScript file.

```
function jsonTbl(data,keyed){
  var table,keyheaderdata,headerdata,keyedbodydata,colheaders,i,j,col;

  table = '<table><thead><tr>';
  headerdata = data[0];  // Default headerdata for unkeyed cols
  bodydata = data;

  if(keyed && data.key){
    keyheaderdata = data.key[0]; // First row of keyed data
    headerdata = data.value[0];  // First row of unkeyed data
    bodydata = data.value; // Change unkeyed body data

    // Set up keyed column headers
    for(colheaders in keyheaderdata){
      table+= '<th class="keyed">' + colheaders + '</th>';
    }
  }
  // Set up column headers
  for(colheaders in headerdata){
    table+= '<th>' + colheaders + '</th>';
  }
  // Finish headers and construct table body
  table+= '</tr></thead><tbody>';
  for(i in bodydata){
    table+= '<tr>';
    row = bodydata[i];
    // Keyed body data
    if(keyed){
      keyedbodydata = data.key[i];
      for(j in keyedbodydata){
```



```
        table+= '<td class="keyed">' + keyedbodydata[j] + '</td>';
      }
    }
    // unkeyed body data
    for(col in row){
      table+= '<td>' + row[col] + '</td>';
    }
    table+= '</tr>';
  }
  table+= '</tbody></table>';
  return table;
}
```


Chapter 8

Exercises

8.1 HTML and JavaScript

1. Complete all lessons `http://www.codecademy.com/tracks/web`
2. Complete lessons 1 to 4 `http://www.codecademy.com/tracks/javascript`
3. Copy the HTML5 Template found in section 3.4 to a file called `template.html`. Change the `<h1>` tag's content to say your name.
4. Change the `<h1>` content using JavaScript.
5. Use jQuery and repeat above but only change its value when it is clicked.

8.2 kdb+ and JavaScript

1. Copy the code found in section 7.1 to a file called index.html. Modify the .z.ws function to receive deserialized incoming messages and return the serialized result back down the handle. Create a WebSocket connection and get the value of "1+1" from the kdb+ server using your index.html file. You will have to open the developer console in Chrome to see the result.
   Hint: You can only send a message once the WebSocket has opened.
2. Let's retrieve more complex objects. Create an arbitrary unkeyed table in your q session and assign it to a variable. Retrieve it using JavaScript.
3. So far we have been using the console to see the data. Now display this table data in a HTML formatted table.
   Hint: In order to display your table use
   `result.innerHTML = jsonTbl(deserialize(data))`
4. Build a data handler function to handle different data types
   Hint: `result.innerHTML = dataHandler(deserialize(data))`

Chapter 9

Answers

9.1 HTML and JavaScript

1. The answers are provided on the site
2. The answers are provided on the site
3. `<h1 id="header">Change this</h1>`
4.
```
var header = document.getElementById("header");
header.innerHTML = "Changed this using JavaScript";
```

```
$('#header').click(function(){
   $(this).html("Changed using jQuery");
 });
```

9.2 kdb+ and JavaScript

5. Copy the code from section 7.1 and change `var query = "1+1"`. Check the JavaScript console for the result. The `.z.ws` function is shown below.

```
.z.ws:{neg[.z.w] -8!value -9!x}
```

2. After you have created the unkeyed table in your q session and assigned it to a variable, change `var query = "tablename"`.
3. Copy the `jsonTbl` function from section 6.6 into your JavaScript code. Also write `result.innerHTML=jsonTbl(data)` inside the `websocket.onmessage` handler function.
4. Copy the `dataHandler` function from section 6.7 and include it in your JavaScript code. Also write `result.innerHTML=dataHandler(data)` inside the `websocket.onmessage` handler function. The `.z.ws` function is shown below.

```
.z.ws:{x:value -9!x; neg[.z.w] -8!(`type`data)!(type x;x)}
```