APEX, Node.js and HTML5: Magic!

Alan Arentsen, Alex Nuijten
Ordina
The Netherlands

Keywords:

APEX, Node.js, HTML5, Application Express

Introduction

With Oracle Application Express (APEX) it is possible to create beautiful applications which run in every web browser. However, the need to communicate with other applications exists in every environment.
Most businesses have multiple applications which need to interact with each other. Getting relevant information from other systems can be quite challenging. Should you reach out to other applications and start looking around and hopefully find the information that you were looking for? Should you keep asking the other application for the information and only receive back the answer “I don’t have that information yet” (called polling). Wouldn’t it be nice if other applications will inform you when relevant information is available?
With HTML5, and especially the web sockets, you can enable other applications to inform you of events taking place. On the other side you will need a method to push the information to the web sockets, and that is where Node.js comes in.

This paper will introduce some concepts on how your APEX application can participate in inter-application communication.

HTML5 and Web Sockets

HTML5 is the latest standard to create webpages. HTML5 provides a better way to create webpages. The layout of the webpage is no longer determined by the use of TABLE elements, but are left to the use of Cascading Style Sheets (CSS; where CSS3 is the latest version and often used with HTML5).
HTML5 added new tags like HEADER, NAV, ARTICLE which clearly indicate the type of content that you are dealing with. It also added new functionality to represent graphic elements like SVG and CANVAS. Audio and video can also be used without the need for extra plugins in the browser.
These things are all very nice, but a real exciting feature are Web Sockets.

As an example, say you have an application which allows you to chat with a sales representative. How does the message that you type in get to the sales rep? The application that the sales rep uses could of course check every few seconds if there is a new message on the server. This would lead to lots of round trips to the server and most of the time without there actually being a relevant message. It would be a lot better if the server would tell the application of the sales rep that there is a new message.
This is similar to the way your smartphone knows there is a new email message that you haven’t read yet. This concept is called push notification.

Implementing Web Sockets in APEX is very straight forward. First of all the correct Doctype (indicating HTML5) should be used. In the past there were many DOCTYPE specification that you could choose from; indicating transitional, strictness, XHTML, version etc.
For HTML the declaration is:

```
<!DOCTYPE html>
```

When this declaration is at the top of the HTML document, you are using HTML5.
For this example we will use APEX 4.2.5, Google Chrome and the Sample Database Application. The Sample Database Application already has the correct HTML declaration if you use a modern browser.

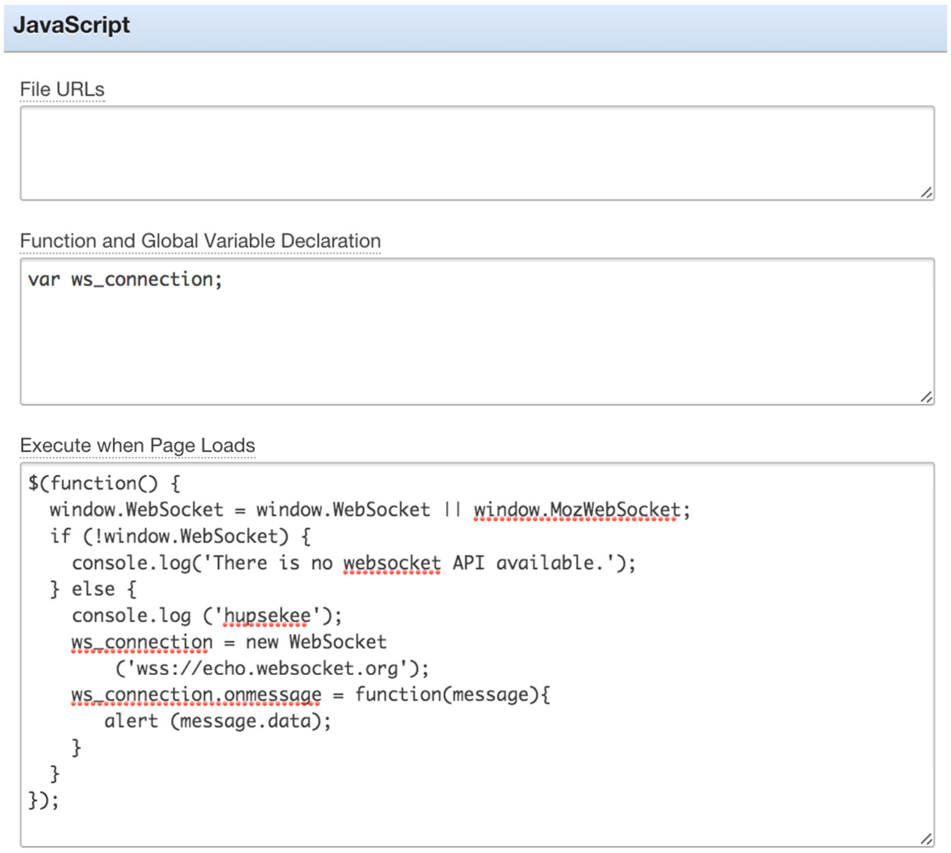
The Web Socket needs to establish a communication channel with the server (we will explain the server in the section about NodeJS). Establishing the connection can be done with the code below:

```
var ws_connection;

$(function() {
  window.WebSocket = window.WebSocket || window.MozWebSocket;
  if (!window.WebSocket) {
    console.log('There is no websocket API available.');
  } else {
    ws_connection = new WebSocket
         ('wss://echo.websocket.org');
  }
});
```

On the first line a global variable is declared called `ws_connection`. In the anonymous function that follows a check is done to verify that the browser support Web Sockets. If it doesn’t support Web Sockets a message is shown in the console. When the browser does support Web Sockets an attempt is made to create a connection. For this example a connection is made to a publicly available Web Socket Echo Server. This server will simply echo the text that you send to it.

This code can be placed on the page where you want to use Web Sockets or in the template.



```
var ws_connection;
```

```
$(function() {
  window.WebSocket = window.WebSocket || window.MozWebSocket;
  if (!window.WebSocket) {
    console.log('There is no websocket API available.');
  } else {
    console.log ('hupsekee');
    ws_connection = new WebSocket
        ('wss://echo.websocket.org');
    ws_connection.onmessage = function(message){
       alert (message.data);
    }
  }
});
```

When you want to use your own server fill out the correct IP address and Port Number for the location where the NodeJS server resides.

Now that we have established the connection with the Web Socket server, we can start using the Web Sockets. To receive messages that come through the Web Socket, we need to define a function that responds to the `onmessage` event.

The code to implement this function is trivial. This will simply display an alert box with the message.

```
ws_connection.onmessage = function(message){
         alert (message.data);
      }
```

This code needs to be placed right after you have established the Web Socket connection.

When you run the page, open the console (F12 in Google Chrome) and enter the following code there, you will see an alert with the text.

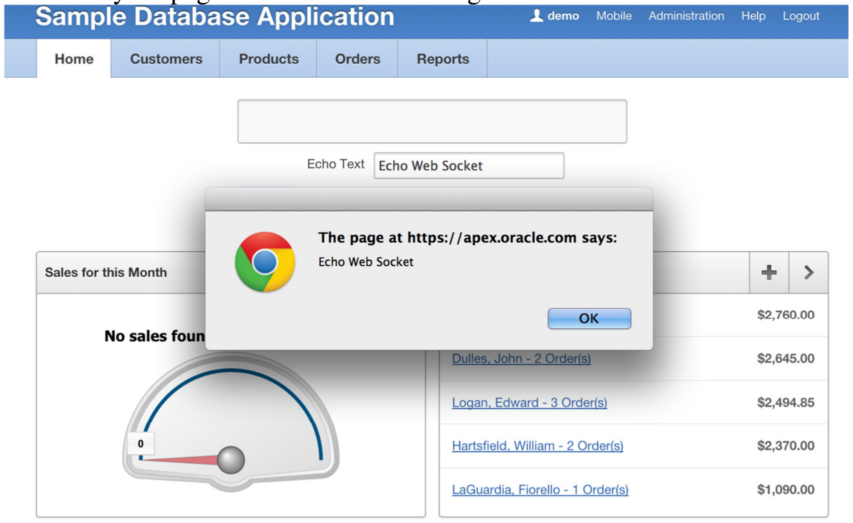
```
ws_connection.send ('My first Web Socket Message');
```

If you see the alert, you have successfully used Web Sockets!

Of course you can also place a text item and a button on the page, and attach a Dynamic Action to the button, then you don't need the browser console.
Simply add this code to the Dynamic Action (Execute Javascript):

```
ws_connection.send (apex.item("P1_ECHO_TEXT").getValue());
```

In APEX your page should look something like this:



The page at https://apex.oracle.com says:
Echo Web Socket

	$2,760.00
Dulles, John - 2 Order(s)	$2,645.00
Logan, Edward - 3 Order(s)	$2,494.85
Hartsfield, William - 2 Order(s)	$2,370.00
LaGuardia, Fiorello - 1 Order(s)	$1,090.00

Node.js
In the previous example we used the Web Socket Echo Server to echo back the message we have sent to it. In a real world example this is quite useless, but it conveys the steps needed to implement Web Sockets in your application.
Of course you can resort to public Web Sockets Servers like Beaconpush or Socket.io, but we wanted to have our own.

What is Node.js? Basically it is a method of running JavaScript on a server.
The definition (from the nodejs.org website) states the following:
"Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices."

The fact that it is event-driven is ideal to use Node.js as our Web socket Server.
After installation of Node.js (out of scope for this document) you will also need to install two modules: http and websocket. You can install them using the Node Package Manager (npm).

The following code must be placed on your server and executed using node.

As mentioned before, the two modules must be present. This is checked in the code below.

```
"use strict";
process.title = 'Basic NodeJS Echo server';

/*
 *   load requirements
 */
var http = require('http');
var webSocketServer = require('websocket').server;
```

An array of clients is used to track who needs a message send to:

```
var clients = [ ];
```

Next, the HTTP server is defined and listening on port 4780. When ready this is shown in the console (on the server).

```
/*
 *    HTTP server
 */
var server = http.createServer(function(req, resp){});
server.listen(4780, function() {
  console.log((new  Date())  +  "  HTTP  server  is  listening  on  port 
4780");
});
```

Now the Web Socket Server is defined and bound to the HTTP server.

```
/*
 *    WebSocket server
 */
var wsServer = new webSocketServer({
    httpServer: server
});
```

When a new request is done to establish a Web Socket connection, it is first verified that the same protocol is used before it is accepted and placed on the array of clients. The array will help to keep track of known connections.

```
/*
 *    Events on WebSocket server
 */
wsServer.on('request', function(request) {
    console.log((new Date()) + ' Websocketconnection from origin ' + 
request.origin + '.');
   var connection = request.accept('echo-protocol', request.origin);
   var index = clients.push(connection) - 1;
```

```
    console.log((new Date()) + ' Connection accepted [' + index +
']');
```

When a message is received, it is converted to UTF8 and forwarded to all the clients which are registered with our Node.js server.

```
    /* Events on websocket connection */
    connection.on('message', function(message) {
        var msgString = message.utf8Data;

        for(var i in clients){
            clients[i].sendUTF(msgString);
        };
    });
```

When a connection is closed, the client is removed from the array.

```
    connection.on('close', function(connection) {
        clients.splice(index, 1);
      console.log((new Date()) + ' Peer ' + connection.remoteAddress
+ ' disconnected.');
    });
});
```

It may seem like a lot of code, but as you can see it is pretty straightforward to implement. With all the components in place, adjust the connection in the APEX page to:

```
ws_connection = new WebSocket
        ('ws://ip-address:4780', 'echo-protocol')
```

and you are using your own Web Socket Server.

Of course there are many more modules available for Node.js, like oracle (to establish a connection directly with the database) or the rest-package (to implement REST web services), and many many more. You can even write a module of your own if you want.

This is in a nutshell the magic that you can do with HTML5, Node.js with APEX. Using Web Sockets and Node.js opens up new possibilities to interact with different applications and devices.

The real magic comes when you create your own program that interacts with different devices such as, for example, a LEGO Mindstorm Robot.

Thank you for reading.
If you want to discuss the content further, don’t hesitate to contact us.

Contact address:

Alan Arentsen
Ordina
Ringwade 1
3438 MN, Nieuwegein

Phone:	+31 (0) 6 13 01 88 32
Email	alan.arentsen@ordina.nl
Internet:	alanarentsen.blogspot.nl

Alex Nuijten

Phone:	+31 (0) 6 10 39 56 54
Email	alex.nuijten@ordina.nl
Internet:	nuijten.blogspot.nl