Node Web Server WebSockets in UC-win/Road

A step by step tutorial on how to setup a Node.js Web Server with a WebSocket client that communicates with a UC-win/Road WebSocket Server

TUTORIAL

CONTENTS

WEB SERVER

INSTALL NODE.JS

Node is a javascript based server-side development ecosystem. Download and install it for your operating system (this tutorial assumes Windows) here:

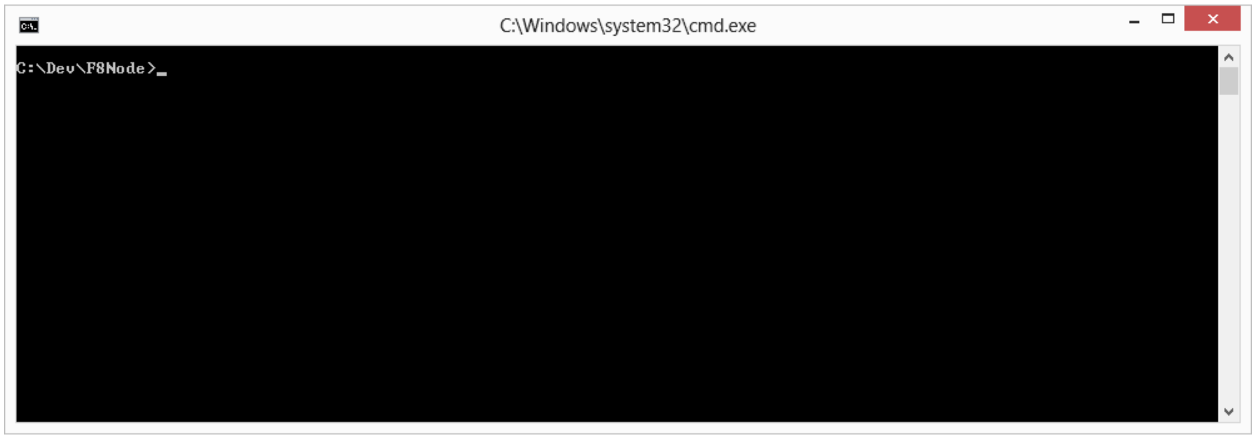
https://nodejs.org/en/download/

Node uses packages to allow users to develop and maintain modules of code. The Node Package Manager is npm, and a partial list of packages can be searched on the following site:

https://www.npmjs.com/

We start our first project by creating a new folder, for instance in C:\Dev\F8Node\.

Navigate to that empty folder. Hold the ‘Shift’ key and Right-Mouse-Click to open a context window and select ‘Open command window here’. This will open a command line interface.



```
C:\Dev\F8Node>_
```

Figure 1. Command Line Interface in our Node project folder

Initialize the project folder as Node project

```
npm init
```

Follow the instructions to fill out the information about the project. These details can be changes later in the package.json file that is produced at the end of this initialization.

INSTALL EXPRESS IN NODE

One of the primary ways Node can be used is as a Web Server. Express is one Node package that allows for easy creation and hosting of a Web Server. We will use npm to install the Express package.

```
npm install express --save
```

This installs the Express module into the node modules subdirectory. The –save makes a note in the package.json file that the module was installed. The package.json file can be used to store your Node projects without having to also store the modules, reducing the size of your repository storage needs.

CREATE BASIC WEB PAGE

To create a Web Server we need to create an initial javascript file to start the server, a subdirectory to store the client-side accessible content, and at least 1 webpage of content.

First create a file called app.js in the base directory and open it in a text editor. Notepad is fine, but other editors may be more efficient such as Notepad++, Brackets, or Atom.

```
notepad app.js
```

In the app.js file add the following and save it:

```
var express = require('express');
var app = express();
var server = require('http').Server(app);

app.get('/', function(req, res) {
	res.sendFile(__dirname + '/public/index.html');
});
```

```
app.use(express.static(__dirname + '/public'));

app.post('/', function(req, res) {
  console.log(req.body);
  res.send(200);
  doSomething();
});

var port = 8000;
server.listen(port);
console.log('Server listening on port '+port);
console.log('Point your browser to http://localhost:'+port);
```

Now in the base directory in the command line window create a 'public' subdirectory, enter that directory, and create an index.html file to edit in a text editor.

```
mkdir public
cd public
notepad index.html
```

In the index.html file add the following and save it:

```
<!DOCTYPE html>
<html>
<head>
		<meta charset="utf-8">
		<meta http-equiv="X-UA-Compatible" content="IE=edge">
		<title>UC-win/Road F8Node</title>
		<link rel="stylesheet" href="">
</head>
<body>
   X: <input type="number" id="xval" value="0"><br>
   Y: <input type="number" id="yval" value="0"><br>
   Z: <input type="number" id="zval" value="1000"></body>
</html>
```

RUN NODE SERVER WITH WEB SERVER

In the command line window, return to the base directory if you are not there already.

```
cd ..
```

Run the node instance.

```
node app.js
```

Open a web browser, such as Chrome, Firefox, or Internet Explorer. Navigate to the following address:

```
http://localhost:8000/
```

You should see the contents of the index.html webpage we just created. Now it is being hosted on our web server.

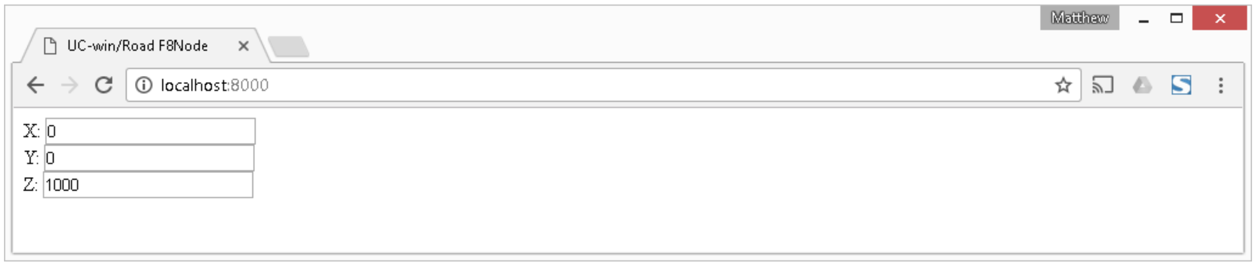


Figure 2. Screenshot of Web Browser of our Web Page

SUBMIT POST REQUEST ON CHANGE OF INPUT VALUES

We need to send information back to the Web Server. There are several ways to do this. The most common way is through GET and POST requests. Another way is through another pair of WebSockets or through Socket.IO. Where we will use POST in javascript. In the index.html we modify the input tag to include an onchange event hander.

```
   X: <input type="number" id="xval" value="0" onchange="MoveCamera()"><br>
   Y: <input type="number" id="yval" value="0" onchange="MoveCamera()"><br>
   Z: <input type="number" id="zval" value="1000" onchange="MoveCamera()">
```

Then we add the following script to send a post request back to the Web Server.

```
   <script>
         function MoveCamera(){
       var xhr = new XMLHttpRequest();
       xhr.open("POST", '/camera', true);
       xhr.setRequestHeader('Content-Type', 'application/json');
       xhr.send(JSON.stringify({
         'x': document.getElementById("xval").value,
         'y': document.getElementById("yval").value,
         'z': document.getElementById("zval").value
       }));
     }
   </script>
```

The current index.html file should look like the following:

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>UC-win/Road F8Node</title>
        <link rel="stylesheet" href="">
</head>
<body>
   X: <input type="number" id="xval" value="0" onchange="MoveCamera()"><br>
   Y: <input type="number" id="yval" value="0" onchange="MoveCamera()"><br>
   Z: <input type="number" id="zval" value="1000" onchange="MoveCamera()">
   <script>
         function MoveCamera(){
       var xhr = new XMLHttpRequest();
```

```
            xhr.open("POST", '/camera', true);
            xhr.setRequestHeader('Content-Type', 'application/json');
            xhr.send(JSON.stringify({
                'x': document.getElementById("xval").value,
                'y': document.getElementById("yval").value,
                'z': document.getElementById("zval").value
            }));
        }
    </script>
</body>
</html>
```

RECEIVE POST REQUEST ON WEB SERVER

On the server side we need to look for a specific incoming POST request. Our POST request is sent using JSON, so we need to also add a JSON parser to our WebServer in Node. In the command line add the body-parser package.

```
npm install body-parser --save
```

Then in the app.js file require the body-parser in the top of the file.

```
var bodyParser = require('body-parser');
```

Then after the app.use method add another app.use for the bodyParser for JSON objects.

```
app.use(bodyParser.json());
```

Finally, we can add our POST handling method to read in the JSON formatted POST data.

```
app.post('/camera', function(req, res) {
        var data = req.body;
        console.dir(data);
        res.sendStatus(200);
        //routeToUC();
});
```

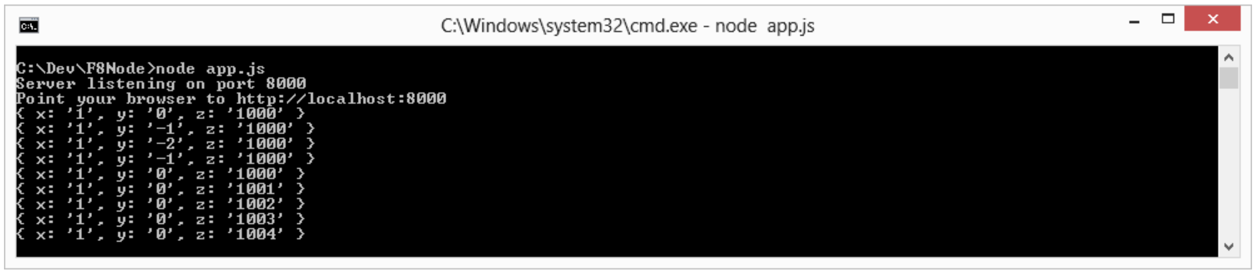
The current app.js file should look like the following:

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
var server = require('http').Server(app);

app.get('/', function(req, res) {
        res.sendFile(__dirname + '/public/index.html');
});

app.use(express.static(__dirname + '/public'));

app.use(bodyParser.json());

app.post('/camera', function(req, res) {
        var data = req.body;
        console.dir(data);
        res.sendStatus(200);
        //RouteToUC();
});

var port = 8000;
server.listen(port);
console.log('Server listening on port '+port);
console.log('Point your browser to http://localhost:'+port);
```

In the command line window running the server, stop the server by pressing `Ctrl + C` key. Then restart the server:

```
node app.js
```

After you reload the Web Page on the Web Browser, change one of the X, Y, or Z values. The comment in the command line window should appear. Now we can route that information through to the WebSocket.



```
C:\Dev\F8Node>node app.js
Server listening on port 8000
Point your browser to http://localhost:8000
{ x: '1', y: '0', z: '1000' }
{ x: '1', y: '-1', z: '1000' }
{ x: '1', y: '-2', z: '1000' }
{ x: '1', y: '-1', z: '1000' }
{ x: '1', y: '0', z: '1000' }
{ x: '1', y: '0', z: '1001' }
{ x: '1', y: '0', z: '1002' }
{ x: '1', y: '0', z: '1003' }
{ x: '1', y: '0', z: '1004' }
```

Figure 3. Console log of incoming POST data

WEB SOCKETS

INTRODUCTION TO WEBSOCKETS

WebSockets allow for a client and server to make an initial handshake through HTTP and then upgrade to a bi-directional TCP connection. This allows servers to send updates to clients when there is something that has changed. Traditionally this was performed through the client continuously polling the server for updates, which can clog up the network with requests to the server. Here we use WebSockets because it will allow us to have bi-directional communication with UC-win/Road. In this tutorial we only send data to UC-win/Road, but this can easily be expanded so that Web Clients can receive updates from UC-win/Road as updates occur to provide more responsive integration.

We will setup a WebSocket Server in UC-Win/Road and a WebSocket Client in our Node Server. We will then route our incoming POST data to the WebSocket Client, which then sends data to the WebSocket Server.

INSTALL WEBSOCKETS IN NODE

First we need to install and begin using the ws package for Node. In the command line type:

```
npm install ws --save
```

At the top of our app.js add a require method to use ws in our codebase. Then create a connection to a WebSocket Server on the localhost on port 9000, which we will setup later in UC-win/Road.

```
var WebSocket = require('ws');
var ws = new WebSocket('ws://localhost:9000');
```

CREATE WEBSOCKETS CLIENT IN NODE SERVER

In app.js replace the previously commented line //RouteToUC() with the following:

```
        ws.send(data.x+","+data.y+","+data.z);
```

Whenever POST data is received, it is now reformatted and routed to the WebSocket Server running on port 9000. Our Node Server will not function properly unless there is a WebSocket Server to connect to.

INSTALL WEBSOCKETS LIBRARY FOR DELPHI

First create a new UC-win/Road plugin in Delphi. A WebSockets implementation is available from André Mussche through a github repository:

https://github.com/andremussche/DelphiWebsockets

Download this repository and add the files into your Plugin project.

CREATE WEBSOCKETS SERVER IN UC-WIN/ROAD PLUGIN

Based on the examples from the SDK, plugins usually being with a PluginMain.pas. In this file we add a few uses:

```
Windows,Messages,SysUtils,Classes,VCL.Dialogs,VCL.Menus,VCL.Forms,upfIntf,upfVisualClasses,PluginCore,F8OpenGL,Data.DBXJSon,superobject,IdWebsocketServer,IdSocketIOHandling,IdServerWebsocketContext,IdHTTPWebsocketClient,IdContext,IdCustomHTTPServer,IdServerSocketIOHandling;
```

Some are for the WebSockets library and some are for accessing the camera in UC-win/Road.

Before adding more functionality, we first need to create a menu for starting and stopping the WebServer within UC-win/Road. In Delphi, create a new VLC form and call it mainform. Add a TButton labelled ‘Start’ and a TButton labelled’ Stop. Add a TMemo to the bottom, which we will use to log the incoming data. Also we need to add a TActionList to route start and stop button commands, and a TIdTCPServer on which the WebSockets connection sits. The mainform.pas should look like the following:

```
unit mainform;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs,Vcl.ActnList, Vcl.StdCtrls, IdContext,
  IdBaseComponent, IdComponent, IdCustomTCPServer, IdTCPServer;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Memo1: TMemo;
    ActionList1: TActionList;
    acStart: TAction;
    acStop: TAction;
```

```
   IdTCPServer1: TIdTCPServer;
   procedure StartButtonClick(Sender: TObject);
   procedure StopButtonClick(Sender: TObject);

  strict private
   { Private declarations }

  public
   { Public declarations }
      OnStart : TNotifyEvent;
      OnStop : TNotifyEvent;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
procedure TForm1.StopButtonClick(Sender: TObject);
begin
     OnStop(self);
end;

procedure TForm1.StartButtonClick(Sender: TObject);
begin
      OnStart(self);
end;

end.
```

In the PluginMain.pas add mainform to the uses. Under the TAPlugin type declaration add:

```
TAPlugin = class(TupfVisualPlugin, IF8UserPlugin)
   procedure upfVisualPluginCreate(Sender: TObject);
   procedure upfVisualPluginDestroy(Sender: TObject);

   private
   { Private declarations }
     aMenu : TMenuItem;
     aForm : TForm1;

     aJSON : TJSONObject;

   var
     aServer: TIdWebsocketServer;
     aClient: TIdHTTPWebsocketClient;
   const
     C_CLIENT_EVENT = 'CLIENT_TO_SERVER_EVENT_TEST';
     C_SERVER_EVENT = 'SERVER_TO_CLIENT_EVENT_TEST';
   function GetVersion : String;
     procedure AbleMenus(enable : boolean);
     procedure AMenuClick(Sender: TObject);
     procedure OnFormStart(Sender: TObject);
     procedure OnFormStop(Sender: TObject);
     procedure WebsocketMessageText(const AContext: TIdServerWSContext; const aText: string);
     procedure HTTPCommandEvent(AContext: TIdContext; ARequestInfo: TIdHTTPRequestInfo; AResponseInfo:
TIdHTTPResponseInfo);
     procedure ClientBinDataReceived(const aData: TStream);
   public
   { Public declarations }
     winRoadApplication : IF8ApplicationServices;
     procedure AfterConstruction; override;
     procedure BeforeDestruction; override;
     procedure ServerMessageTextReceived(const AContext: TIdServerWSContext; const aText: string);
   end;
```

Add the AfterConstruction procedure for the TAPlugin to initialize the webserver and the WebSocket server.

```
procedure TAPlugin.AfterConstruction;
   aMenu := TMenuItem.Create(nil);
   aMenu.Name := 'CameraSocket';
   aMenu.Caption := 'Websocket Camera';
   aMenu.OnClick := AMenuClick;
   winRoadApplication.mainForm.optionsMenu.Add( aMenu );

   aServer := TIdWebsocketServer.Create(Self);
   aServer.DefaultPort := 9000;
   aServer.KeepAlive := True;
   aServer.Active := False;

   aServer.OnCommandGet  := HTTPCommandEvent;
   aServer.OnMessageText := ServerMessageTextReceived;

   {aServer.SocketIO.OnEvent(C_CLIENT_EVENT,
   procedure(const ASocket: ISocketIOContext; const aArgument: TSuperArray; const aCallback:
ISocketIOCallback)
   begin
     if aCallback <> nil then
       aCallback.SendResponse( SO(['succes', True]).AsJSon );
   end);

   aServer.SocketIO.OnEvent('connect',
   procedure(const ASocket: ISocketIOContext; const aArgument: TSuperArray; const aCallback:
ISocketIOCallback)
   begin
     aForm.Memo1.Lines.Add('connected');
   end);

   aServer.SocketIO.OnEvent('connecting',
   procedure(const ASocket: ISocketIOContext; const aArgument: TSuperArray; const aCallback:
ISocketIOCallback)
   begin
     aForm.Memo1.Lines.Add('connecting');
   end);
   }
```

```
end;
```

Then create a procedure for the HTTPCommandEvent in the TAPlugin to create a basic http server.

```
procedure TAPlugin.HTTPCommandEvent(AContext: TIdContext; ARequestInfo: TIdHTTPRequestInfo;
AResponseInfo: TIdHTTPResponseInfo);
    begin
    AResponseInfo.ContentText := 'Hello World';
    aForm.Memo1.Lines.AddObject('context',AContext.Connection);
    aForm.Memo1.Lines.Add(ARequestInfo.Command);
    aForm.Memo1.Lines.AddObject('requestinfo',ARequestInfo);
    aForm.Memo1.Lines.AddObject('responseinfo',AResponseInfo);
    end;
```

Then create a procedure for the OnFormStart in the TAPlugin to activate the server and add a line to the memo.

```
procedure TAPlugin.OnFormStart(Sender: TObject);
begin
    aServer.Active := True;
    aForm.Memo1.Lines.Add('started server');
end;
```

Then create a procedure for the ServerMessageTextReceived in the TAPlugin. This is where the incoming WebSocket data is parsed into float values and assigned to the camera’s view target in OpenGL.

```
procedure TAPlugin.ServerMessageTextReceived(const AContext: TIdServerWSContext; const aText: string);
var
  List: TStrings;
begin
  List := TStringList.Create;
  ExtractStrings([','], [], PChar(aText), List);
  aForm.Memo1.Lines.Add(aText);
  winRoadApplication.mainForm.openGL.Camera.Eye.X:=strtofloat(List[0]);
  winRoadApplication.mainForm.openGL.Camera.Eye.Y:=strtofloat(List[1]);
  winRoadApplication.mainForm.openGL.Camera.Eye.Z:=strtofloat(List[2]);
  winRoadApplication.mainForm.openGL.Refresh();
  List.Free;
end;
```

Then create a procedure for the OnFormStop in the TAPlugin to deactivate the server and add a line to the memo.

```
procedure TAPlugin.OnFormStop(Sender: TObject);
begin
   aServer.Active := False;
   aForm.Memo1.Lines.Add('stopped server');
end;
```

Then create a procedure for the BeforeDestruction in the TAPlugin to make sure all the form and server objects are released and freed from memory cleanly.

```
procedure TAPlugin.BeforeDestruction;
    begin
    inherited;
```

```
   winRoadApplication.mainForm.optionsMenu.Remove( aMenu );
   aMenu.Free;
   if Assigned(aForm) then
     aForm.Free;
   if Assigned(aServer) then
     aServer.Free;
   if Assigned(aClient) then
     aClient.Free;
   end;
```

Then create a procedure for the WebsocketMessageText in the TAPlugin to add lines to the memo in the form.

```
procedure TAPlugin.WebsocketMessageText(const AContext: TIdServerWSContext;
  const aText: string);
begin
  aForm.Memo1.Lines.Add(aText);
end;
```

Compile and load the plugin into UC-win/Road.

INTERACT WITH UC-WIN-ROAD THROUGH WEB CLIENT BROWSER

Run the plugin in UC-win/Road and press the 'Start' button to start up the http server and WebSocket Server. Then start up the Node Server with WebSocket Client and WebServer. Direct a web browser to localhost:8000. When you change values in inputs of the web page, the camera view in UC-win/Road is adjusted.

EXTENTIONS- ASSIGNMENTS

1. Extend the functionality of the Web Client Web Page to pull mouse location information on mouse move events. Transfer those events to the Web Server through POST.
2. Make a different modification to UC-win/Road than moving the camera view target.
3. Upgrade the Web Client Web Page to use WebSockets or Socket.io (npm install socket.io) to allow the Node Web Server to send information and updates back to the Web Client. Then connect multiple clients and synchronize their values.
4. Pull data from UC-win/Road and send it back to the Node Web Server via the WebSockets connection. Then reroute that information to all of the Web Clients connected to the Node Web Server.
5. Use HTML5 to access the sensor hardware on mobile devices. Use the accelerometer, gyro, or other sensor input to control UC-win/Road.

This tutorial was created by Matthew Swarts

Permission is granted to Forum8 for distribution.

matthew.swarts@design.gatech.edu

http://www.matt-swarts.com