SignalR and WebSockets Demo Notes

# Web Sockets Client (All Windows Versions)

* Demo the ‘To Implement’ Application, show the drawing on the canvas
* Now going to add Websocket connection to do reflection
* Add Code in chunks before event handler attachment, first create the socket

 var socket = new WebSocket("ws://localhost:12345");

* Now deal with connection events

 socket.onopen = function () {

 $('#connectionState').toggleClass('connected', true).html('Connection Online');

 };

 socket.onclose = function () {

 $('#connectionState').toggleClass('connected', false).html('Connection Offline');

 };

* And finally the message handler

 socket.onmessage = function (dataPacket) {

 var data = dataPacket.data;

 var pair = data.split(',');

 drawToCanvas(pair[0], pair[1], reflectContext, startedDrawingReflect);

 startedDrawingReflect = true;

 };

* Finally we add the call to send some data to the server in the mouse move handler

socket.send('Reflect (' + x + ',' + y + ')');

# Web Sockets On Windows 8

* New Project – ASP.NET Web Application(web Forms) (in c:\projects)
* Install Package Microsoft.WebSockets
* Add New Generic Handler to project (ashx) called SocketEndPoint
* Replace default Process with:

if (context.IsWebSocketRequest)

{

 context.AcceptWebSocketRequest(

* Show the overloads available – now add the using for Microsoft.WebSockets

using Microsoft.Web.WebSockets;

* Go back to the Accept line and show better overloads

context.AcceptWebSocketRequest(new MyWebSocketHandler());

* Now add a new class (can be below this one)

public class MyWebSocketHandler : WebSocketHandler

{}

* Add override for OnOpen()
* Remove the call to base
* We want to track connections, so use an additional class to hold the connections

private static WebSocketCollection \_connections = new WebSocketCollection();

* In OnOpen, add:

\_connections.Broadcast(“Another One Joined!”);

\_connections.Add(this);

Send(“Hello From the Server”);

* Override onMessage(string)
* Remove base call
* Add:

Send(“Your message backwards is:” + new string (message.Reverse().ToArray()));

* Open About.aspx
* Clear content area.
* Add Script block

var ws = new WebSocket(‘ws://localhost:80/SocketEndPoint.ashx’);

ws.onmessage = function(message) {

alert(message.data);

};

ws.onopen = function() {

ws.send(‘Hello World’);

}

function sendMessage() {

ws.send(‘Message from the Button’);

return false;

}

* Add a button

<button onclick=”return SendMessage();”>ClickMe</button>

* Build
* All Done, Now F5 and show it does not work!
* Need to use full IIS – also need to have installed Web Sockets Functionality
* Point IIS at folder
* Hit it and it will work
* Demo second browser window and the ‘Another Joined

# SignalR Hubs

* Demo the To Implement Application again
* Look at hub Implementation in hubs\DrawHub.cs
* Highlight the base class (Hub)
* Highlight the HubName attribute
* Highlight the **Caller**
* Now Adding SignalR to the app, open SignalRDrawImplement view:
* Need to Reference the SignalR library (Note JQuery already referenced)

<script src="/Scripts/jquery.signalR.js" type="text/javascript"></script>

* Next need to reference the Hubs – this is the clever JavaScript proxy

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

Now configure the connection:

var hub = $.connection.draw;

 hub.draw = function (x, y) {

 drawToCanvas(x, y, reflectContext, startedDrawingReflect);

 startedDrawingReflect = true;

 };

 $.connection.hub.start();

And add call to hub:

hub.reflect(x, y);

Explain that this is calling the proxy method on the hub resulting in the data being sent to the server

Mention Performance – Long poll and re-establish connection

# SignalR and Web Applications

Example is a multiuser web application to demonstrate some useful features of SignalR, and how you can use it in more real world situations.

Demo the application with two browser windows (making sure you use different usernames)

Look at the code:

* Highlight that it is a client side Knockout application
* Show view model implementation in KnockoutCode.js

Look at connection code

* Connection made as previously
* Wire up client side methods to hub
* Start makes use of .done to wait until connected before running next bit
* Username set to the server – note casing of method – camel case

Switch to hub code to look at the implementation of this method

* Show the method name in the .NET is Pascal case
* Common pattern, stash connection information in dictionary – use cookie or in our case connection ID as key (note connection ID bad for apps which reconnect!)
* Concurrent dictionary here
* Discuss use of static dictionary as every call is a new hub
* Highlight the Caller.userName value setting – any property on set on hub is passed back to client (much like method calls)

Switch back to View

* After username set up event handlers for client side events
* Explore focus handler first

Switch to Hub implementation

* Highlight Clients to call method on all clients
* Mention groups, which allows you to group connection and send data to just connections in a group – might be useful scoping to records?
* No way of excluding a connection - we filter on the client side using username
* Note parameters – first is a simple string – second is a .NET object – this gets serialised using JSON and magically appears at the client

Switch back to View

* Note the use of the data from the .Net object
* Note the use of the variable set on the hub server side
* Look at the keypress handler
* Makes use of JavaScript object sent to server
* Timeout used to improve performance – making connection for each key is rather inefficient, might use Reactive Extensions for JavaScript here to do rate limiting.
* Look at hub implementation, JavaScript object deserialise to .NET object

# Windows Phone & SignalR

**Don’t Forget to Close Virtual Box Machine**

* Create a new Windows Phone Silverlight application
* Add a new MVC Application (Internet Application Template)
* Add Hubs Folder
* Add New Class to Hubs Folder – MySampleHub.cs
* Install Package SignalR (Make sure you select the MVC Application Project
* In MySampleHub.cs

[HubName("MySample")]

 public class MySampleHub : Hub

 {

 public void ReverseString(string stringToReverse)

 {

 var reversed = new string(stringToReverse.Reverse().ToArray());

 Caller.SetReversedText(reversed);

 }

 }

* Set the MVC app as startup and F5 – make a note of the address
* That’s our SignalR Hub done, over to WP7
* In MainPage1.xaml, Drag a textbox and button onto the page
* Set the text for the Textbox to ‘Hello World’
* Set the button Content to ‘Reverse Me’
* Double Click Button to create event handler
* Install Package SignalR.Client.WP7 (make sure that you set the project to the WP7 App)
* Add page scoped variables:

 private HubConnection \_connection;

 private IHubProxy \_sampleProxy;

* Create Set Reversed Method t update display

 void SetReversedText\_Called(object[] obj)

 {

 Dispatcher.BeginInvoke(() => textBox1.Text = obj[0].ToString());

 }

* In Page Startup create the connection
* \_connection = new HubConnection("http://localhost:YOURPORTHERE");

 \_connection = new HubConnection("http://localhost:28696");

 \_sampleProxy = \_connection.CreateProxy("MySample");

 var sub = \_sampleProxy.Subscribe("SetReversedText");

 sub.Data += SetReversedText\_Called;

 \_connection.Start();

* Note: Need to get correct address for the site
* Finally, in button click

\_sampleProxy.Invoke("ReverseString", textBox1.Text);

* Properties on Solution, setup multiple startup projects, and then F5