Getting Started with SignalR

Slide 1: In the beginning…

From its inception, the WorldWideWeb was intentionally stateless and disconnected. It was, after all, designed basically as a means to transfer and display scientific papers and data between collaborators. Life was simple, a client made a request to a host, and the host returned a response. It worked.

Concurrent with this technology were synchronous channels, often wired directly to one another, allowing real time communication between machines when needed. Controller and monitoring systems were usually dedicated to their single purpose.

This was all fine—personal computers were a significant step up from typewriters and the postal service, so no one minded.

Slide 2: Times, they are a changin' (not really)

Once the Internet opened up, and people figured out you could make some actual money with this thing, its popularity exploded. We added a lot of new window dressing to the clients to improve the experience, but the basics were still the same request-response pattern. Real time communication evolved to have less dedicated wiring and more Ethernet connections, still usually with dedicated systems. PCs were faster, but still bound to desks.

Slide 3: Today's status

Then, technology went nuts. Laptops became more portable and less expensive, smartphones emerged and evolved into tabled, networks became faster, Starbucks expanded, and people strived to get away from their desk. One generation removed from being chained to a desk and typewriter, and people wanted to move around more, and expected their apps and data to be able to move with them, to be available when they wanted where they wanted. Although the use of the Internet had changed dramatically, the way it operated did not.

We have a number of technologies available to us to make our applications seem real time, but we're still just using workarounds.

In the simplest case, we just set a META REFRESH to have the page load and reload to reflect data changes. If we were really good, we'd use an Ajax request and partial page refresh in a single page application. Technologies such as jQuery make our lives easier, but we're still not as good as we could be.

Slide 4: Where are we headed?

Tin cans? Kind of!

Slide 5: We 3.0.0.0

Real time communications require something not in the traditional request-response paradigm—a persistent open channel. The desire to have communication in real time and independent of device, OS or location has led to adding sockets protocols to browsers, and sockets libraries are being created to harness the new functionality.

Slide 6: Approaches

Interval polling is using the META REFRESH or similar thing to check a source, usually on a schedule. These are typically short connections, open-check-close. This can be accomplished via HTTP, and fits the request-response pattern.

Long polling is an open-close connection, but the duration of the open is greater than interval polling. Often, as soon as the connection is closed, it is reopened.

Web Sockets is a new API, still being standardized by the W3C, which enabled persistent socket connections via HTTP. Here's the kicker—these are bidirectional, full duplex via TCP, but are asynchronous. Since these are handled via HTTP, they work with (or completely circumvent) some of the security measures in place for other protocols.

SignalR uses web sockets, and fails over to long polling. The default long poll is 110 seconds, but that can be configured in the global.asax.

Slide 7: Browser support

Web sockets are part of the HTML5 standard. Implementation-wise, we have some waiting to do before sockets are prevalent. This is why we need libraries like SignalR. Really, SignalR is so good, I don't care if Web Sockets are ever added to browsers.

Slide 8: Not just .NET!

Originated by David Fowler on the ASP.NET team, and originally had a dependency on ASP.NET MVC. However, has been split off from ASP.NET MVC and can stand alone. There are also client libraries for other technologies, including iOS.

Slide 9: Implementation

At its core, SignalR is a simple pub-sub message bus system.

If we're creating a web page as a client, we'll code in JavaScript. If we're using Silverlight, Windows Phone, etc., we'll use a .NET client library. The lowest level object on the server and client is a Persistent Connection. Persistent Connections are dynamic type in .NET, which is similar to a prototype in JavaScript. This means we can add functions and properties at run time. To JavaScript developers, this is nothing new, but this is new to many .NET developers. In addition to being able to add properties to

the object to be passed back and forth, we can also load additional function libraries to extend the objects.

Hubs add an MVC layer on top of a Persistent Connection. Hubs simplify working with SignalR, especially if we want to handle multiple types of messages being passed around. Not all features of a Persistent Connection are directly available in a Hub, but most of what we need is there. For example, a Hub does not have a Disconnected event, so we cannot tell if a client disconnects unless we roll our own.

Clients come in two varieties—Clients, and Callers. Clients are anything with a connection to a specific hub. Callers are the client which signaled the method.

When we write code on either the server or the client, we devs need to have knowledge of the other side. This is not a system where the client can infer the server (or vice versa) via a discovery file or other mechanism. Both client and server development will probably happen simultaneously.

To send a message to the server, the client is actually calling ("signaling") a method directly on the server. Not all messages need to be returned to a client, but if we do, we signal the client side method from the hub code. Signals can originate from either a client or a server, for true push functionality.

When run, SignalR autogenerates a script library at signalr/hubs. This script library implements the connection-specific details.

The danger here is that before, we had ports dedicated to protocols, and we could turn off protocols by blocking ports. Running everything via port 80 doesn't make life more secure. Smart firewalls capable of SPI and other security technologies will be even more important. And the onus will be on developers to be even more careful.

Demo

Since I'm not running IIS 8 locally, this demo will use long polling. There is no difference in code, the fallback from web sockets to long polling is automatic. If we were to implement web sockets, we would need to add a separate library. This library is separated out to reduce the size of SignalR core, and so it can be developed on a separate trajectory.

Chat App

We're going to build a simple chat application using a Hub on the server, and the clients for this sample will be written purely in JavaScript. Below is an overview of the steps this demo goes through (Steps 3-5

1. Start a new ASP.NET MVC 3 application
2. Go to Tools >> Library Package Manager, and install SignalR via NuGet ("Install-Package SignalR")
3. Open _Layout.cshtml, and add three references (in this order) if they don't exist.
4. Add a Hubs folder to the root, and define the different Hubs as classes which implement Hub. In this case, Chat.cs.

5. Add the client side code to the Home >> Index.cshtml file.

This demo shows the following:

1. Basic “call message on client side” functionality
2. Sending message to specific client
3. Adding additional data to the connection object on the client side
4. Grouping clients/callers, and sending group specific messages.

Voting Demo

This demo shows the following:

1. Tracking callers by ID in a dictionary, so caller specific data can be updated
2. Typed objects can be used as message data, and will be serialized into JSON
3. Loading existing vote data in the done promise after the hub starts on the client
4. A simple data injection to add a vote for “Bacon”.

Slide 10: What is it good for?

Real time communication means messages to clients need to originate at the source. With SignalR, we can push messages from server, which is a major change in communication techniques on the web.

These methods are not restricted to just passing messages, you can call databases and other services on the server side, and do all kinds of client manipulation. From monitoring database server load to tracking ambulances on a map, much can be done.

Slide 11: Other Libraries

SignalR is one of many good libraries for real time communications. As good as it is, it may not fit your needs, especially at this early stage. Here are a few other similar libraries.

Resources

http://en.wikipedia.org/wiki/Websockets

http://signalr.net/

http://www.hanselman.com/blog/AsynchronousScalableWebApplicationsWithRealtimePersistentLongrunningConnectionsWithSignalR.aspx

http://iwantmymvc.com/mvc-3-signalr-knockout-real-time-notifications

http://blog.maartenballiauw.be/post/2011/12/06/Using-SignalR-to-broadcast-a-slide-deck.aspx