ITDevCon
European Delphi Conference

14, 15 november 2013 - VERONA (Italy)

Realtime Websockets (and Socket.io) for Delphi

André Mussche, The Netherlands
andre.mussche@gmail.com

dts.nl
4dotnet.nl

Contents

- Websockets
  - What are Websockets?
  - Delphi implementation
  - Demonstration
- Socket.io (communication library)
  - Delphi implementation
  - Why or when to use
  - Demonstration
- The Smart Way (node.js in the cloud)

What are Websockets?

- Part of the HTML5 specification
- bi-directional direct communication over HTTP
  - bi-directional: server can push (!) data to one or more clients, no need for (long) polling!
  - direct: no http (header) overhead: data is send and received with minimal processing
- Looks similar to TCP
  - but “stream of messages” vs “stream of bytes”
- Full duplex: similar with Super TCP
  - but super tcp (in Delphi) heavy weight: uses a thread for each client connection (slow debugging in Delphi when many connection in services!)

What are Websockets? (2)

- So nothing really new…
- But a big step for web/html programming
  - uses same port 80 (firewall, proxies, etc)
  - fast web apps possible (realtime, low latency)
    - .Google Chrome demos with mobile as remote game controller
  - default full duplex: easy pushing from the server!
- http is only used for initial handshake
  - for the rest, it’s an independent protocol
- But, both server and client must support it...

What are Websockets? (3)

- **HTTP Handshake**

Client:

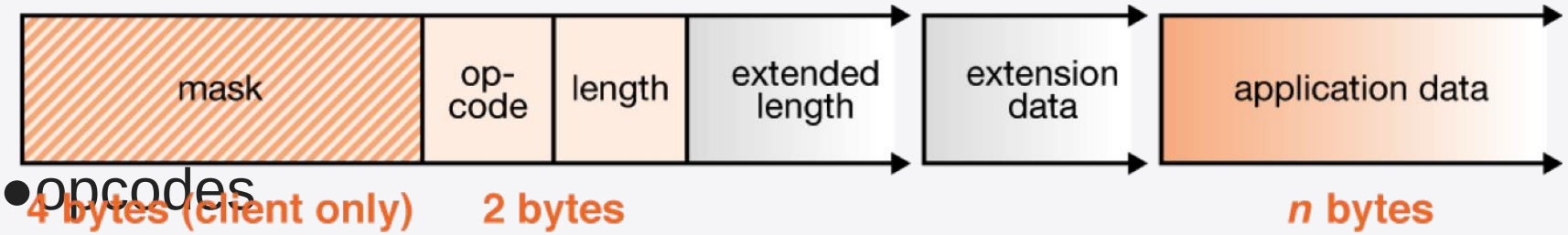
```
GET ws://server.example.com/mychat HTTP/1.1
Host: server.example.comUpgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Server response:

```
HTTP/1.1 101 Switching ProtocolsUpgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

What are Websockets? (4)

- Websocket frames
  - very small header:



- opcodes
  - Text and binary data
  - ping/pong frames (for heartbeat)

Delphi implementation

- Indy 10
  - websocket IOHandler
  - both server and client
  - basic support
    - no extensions like compresion, RFB, SRPC, etc
  - Mixed server: http and ws (and RemObjects SDK) connections
- Single wait thread per 64 client handles (using winsock2.select() API)

Demonstration

Broadcasting text messages…

That’s pretty it...

Socket.io (communication library)

- Websocket itself very simple (only “messages”)
  - Need extra protocol/handling on top of it
    - RemObjects SDK, Socket.io
- Socket.io:
  - javascript communication library
    - node.js (server side javascript)
    - webbrowser (client side javascript)
  - Works in every browser (IE6, older Android devices)
    - works on top of websockets, http, Flash sockets
  - Event driven (async)

```
var socket = io.connect('http://localhost');
socket.on(‘eventfromserver’,
 function(data) {
                    console.log(data);
          });
         socket.emit(‘request2server’, {data: ‘text’}, function(ack){} );
```

Socket.io (protocol)

- Protocol
  - `1:::` = Connect
  - `2:::` = Heartbeat
  - `3:::  test` = Simple message

`3:1+:/chat:data` = Msg with msg number (1) and request for acknowledge (+) for only “/chat” room

  - `4:::{a:b}` = Json message
  - `5:2+::{"name":"myevent","args":["data"]}` Event with arguments and waiting for result (2+)
  - `6:::2[{a:b}]` = Acknowledge for msg nr 2 and with data

Delphi implementation

- Indy 10
  - uses websocket IOHandler (incl. http fallback)
  - seperate socket.io handling
  - both server and client
  - basic support (no rooms, subprotocols, authentication, etc)
  - Mixed server: http, ws, RO and socket.io connections
- Simple replacement for RemObjects and Datasnap?
  - events are very easy to use
  - default duplex communications
  - BUT: no type safety: hard coded strings and json...
    - .Using interfaces and TVirtualInterface for automatic client side wrapping?

Delphi implementation (2)

- Example:

```
FWSServer.SocketIO.OnEvent(
  'getProgress',
  procedure(const ASocket: ISocketIOContext; const
            aArgument: TSuperArray;
            const aCallbackObj: TSocketIOCallbackObj)
  begin
    aCallbackObj.SendResponse(SO('result'));
  end);
```

- It's open source!
  - https://github.com/andremussche/DelphiWebsockets
- So fork it and add missing features!

Why or when to use?

- Windows Service with html front-end for monitoring
  - direct pushing state changes, log messages, etc
  - Single port Service, one port for all communication:
    - normal http (static files, html front-end)
      - Can change web view without rebuilding service executable
    - normal ws (web + new Delphi clients?)
    - RemObjects over http (web clients, json msg)
    - RemObjects over ws (Delphi clients, bin msg)
      - Also between other services (SOA: Service Oriented Architecture)

Why or when to use? (2)

- HTML5 Web app
  - single page: static html+js files
    - direct change, no recompile needed
    - dynamic html/layout using javascript
  - only data is transferred (json/xml)
    - different apps/views possible
  - no app store needed, runs on any device (iOS, Android, Win8, Phone7, Blackberry, Firefox OS, Tizen, etc) within private/own network

Why or when to use? (3)

- Example: replace Adobe Flash visualization
  - Export to html5 canvas (createjs/easeljs library)
  - Websocket server:
    - serving static files
    - realtime pushing PLC changes
  - MongoDB: noSQL database using bson (binary json) for dynamic data

Demonstration

Event based visualization

The Smart Way (node.js in the cloud)

- Html5 client in Smart Mobile Studio
  - full client side socket.io implementation
  - single page app, can be used on any device
- Node.js in Smart Mobile Studio
  - full server side socket.io implementation
  - generated javascript can be directly used in cloud

Smart Mobile Studio

- Smart Mobile Studio (IDE)
  - pascal language
    - .with many cool improvements (type inference, inline variables, lambda expression, etc)
  - compiles to javascript
    - .smart linking (only “active” code)
    - .obfuscation possible
    - .“asm” sections for low level javascript code
    - .external classes for JS libraries (jQuery etc)
  - best of both worlds
    - .code completion, type checking, classes, etc
    - .runs on every device

Node.js

- Node.js: javascript on the server
  - Very different to “normal” browser
  - Google’s V8 javascript engine
    - almost native compilation (JIT), garbage collection
  - no webkit rendering (no “document” and “window” globals! no DOM, no jQuery)
  - no sandbox, but full access (files, processes, databases, etc)
    - own API (event based, async)
  - User code is single threaded, rest is async (background threads?)
    - so very fast for small requests or async stuff (waiting for files, database queries etc)
    - bad for complex calculations (compressing etc)

Demo

Smart node.js http server (with socket.io)
Smart html5 client