Why WebSockets?

In normal HTTP a client (i.e. the browser) sends a request to a server. After some processing time the server responses to the client. This is one way, the server can not serve files or data by itself without been requested before. This is a disadvantage i.e. if data must be asynchronously refreshed. In such cases the client has to poll the server which leads to a lot of traffic.

WebSockets are a possibilty for bidirectional communication between server and client. Client and server can send data on demand - Both are equal communication partners. While refreshing data it is enough to send data as soon they are available. It is not necessary anymore to poll the server.

Netzer supports Websockets since Version 1.5!

WebSocket URI

A WebSocket URI commonly starts with `ws://`, encrypted connections start with `wss://`. Netzer currently supports only unencrypted WebSocket connections.

The WebSocket URI of Netzer follows the schema `ws://NETZER/ws`. The placeholder NETZER stands either for the IP address or the Netzer mDNS name. The WebSocket of the Netzer with IP address `192.168.0.2` and the mDNS name `myNetzer.local` can be addressed with `ws://192.168.0.2/ws` or `ws://meinNetzer.local/ws`.

WebSockets in JavaScript

Open connection

A new WebSocket connecion can be opened with creating a new websocket object.

```
var myWebSocket = new WebSocket(myWebSocketURI);
```

In some Firefox versions (till version 11) the WebSocket object has the name “MozWebSocket”. All other names are equal (if implemented). An appropriate information which browser currently support WebSockets can be found here: WebSocket#Browser_support.

```
var myWebSocket = new MozWebSocket(myWebSocketURI);
```

Send data

WebSockets support two fundamental transmission types: Text (UTF-8) and binary. Data is send with `send(myData)`. The tranmission type is chosen through the data type of `myData`.

Netzer currently only supports text transmissions, thats why only this transmission type is shown here.

The text transmission type is only used, if the parameter for `send` is a string.

```
var myData = "My text is here.";
myWebSocket.send(myData);
```

Receive data

Receiving data is handled via an event handler `onmessage`. `event.data` contains the received data.

```
myWebSocket.onmessage = function(event) {alert("Received data:
"+event.data);};
```

Close connection

To close the WebSocket connection, the function `close()` must be called. Optional a closing code and a reason can be given. Both of them are currently not interpreted by Netzer.

```
myWebSocket.close();
```

Further event handler

onopen

onopen is fired, if a WebSocket connection is opened.

onclose

`onclose` is fired, if a WebSocket connection is closed. The close code and reason from server can be requested with `close` or `reason`.

```
myWebSocket.onclose = function(event) {alert("Connection closed. Code:
"+event.code+" Reason: "+event.reason);};
```

onerror

`onerror` is fired, if an error has occured.

A simple example

Showing the current value of the Netzer pin IO0.

websocket.htm

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <script type="text/javascript">
            var myWebSocket,
            // verbinden
            connect = function() {
                // Clean up old websocket connection.
                if(myWebSocket)
                {
                    myWebSocket.close();
                }
                // Extract URL
                url = document.getElementById("netzer_url").value;
                // Establish new connection.
                if("WebSocket" in window)
                {
                    myWebSocket = new WebSocket(url);
                }
                else if("MozWebSocket" in window)
                {
                    myWebSocket = new MozWebSocket(url);
                }
                else
                {
                    alert("The browser does not support WebSockets.");
                    return;
                }
                // Set handler for incoming messages.
                myWebSocket.onmessage = receiveMessage;
                // If the connection is established, send a command which
sets the trigger.
                myWebSocket.onopen = function()
{myWebSocket.send("tuv0=3");};
            },
```

```
            // Receive Message
            receiveMessage = function(event) {
                msgJSON = JSON.parse(event.data);
                // Is it an update of the value IO0?
                if(msgJSON && msgJSON.u && msgJSON.u.v && msgJSON.u.v["0"])
                {
                    document.getElementById("io_value").innerHTML =
msgJSON.u.v["0"];
                }
                // Ignore other messages.
            },
            // Function for initializing page.
            initPage = function() {
                // Delete javascript hint.
                document.getElementById("js_hint").innerHTML = '';
                // Set event handler
                document.getElementById("connect_button").onclick = connect;
            };
        </script>
    </head>
    <body onload=initPage()>
        <div id="js_hint">Please activate javascript!</div>
        <input value="ws://test.local/ws" id="netzer_url" type="url">
        <input id="connect_button" value="Connect" type="button">
        <p><div style="float:left; padding-right:5px">Value of
IO0:</div><div id="io_value">N/A</div></p>
        <div style="clear:both;">
    </body>
</html>
```