Web-based Experiment Monitoring with HTML5

Stefan Ritt, Paul Scherrer Institute, Switzerland <stefan.ritt@psi.ch>

Abstract

This paper describes a set of techniques which allow the control and monitoring of experiments in real time with modern Web technologies using HTML5. A minimal web server based on the Mongoose Webserver is used to connect directly to the hardware for monitoring and control. JavaScript Object Notation (JSON) is used to exchange data between the server and the browser, and Typed Arrays are used for high-speed waveform transfer. The browser side uses Asynchronous JavaScript Extensions (AJAX) together with Remote Procedure Calls (JSON-RPC) to retrieve data from the web browser. The HTML "canvas" element is then used to render the graphics. Modern browsers are highly optimized and execute JavaScript as fast as native programs (such as ones programmed in Qt) a few years ago. This allows for effective data visualization and experiment control without having to install any program. **Try it yourself with your smartphone at http://elog.psi.ch/scope**

Web Server

The **Mongoose Webserver** [https://github.com/cesanta/mongoose] is a light-weight web server written in C. It supports features such as SSL, authentication and websockets. A plug-in architecture allows user code to access hardware devices through appropriate drivers, render measurement results as HTML text and pass it to any web browser.

```
#include <stdio.h>
#include "mongoose.h"

static struct mg_serve_http_opts s_http_server_opts;
static char *s_http_port = "8080";

static int rpc_setled(char *buf, int len, struct mg_rpc_request *req)
{
   digitalWrite(0, atoi(req->params[1].ptr));
   return mg_rpc_create_reply(buf, len, req, "status", 1);
}

static int rpc_readtemp(char *buf, int len, struct mg_rpc_request *req)
{
   return mg_rpc_create_reply(buf, len, req, "temp", read_temperature());
}

static void ev_handler(struct mg_connection *nc, int ev, void *ev_data)
{
   static const char *methods[] = { "setled", "readtemp", NULL };
   static mg_rpc_handler_t handlers[] = { rpc_setled, rpc_readtemp, NULL };

   struct http_message *hm = (struct http_message *) ev_data;
   char buf[1000];

   if (ev == MG_EV_HTTP_REQUEST) {
      if (mg_vcmp(&hm->uri, "/json-rpc") == 0) {
         // server JSON-RPC content
         mg_rpc_dispatch(hm->body.p, hm->body.len, buf, sizeof(buf),
                         methods, handlers);
         mg_printf(nc, "HTTP/1.0 200 OK\r\nContent-Length: %d\r\n"
                   "Content-Type: application/json\r\n\r\n%s",
                   (int) strlen(buf), buf);
         nc->flags |= MG_F_SEND_AND_CLOSE;
      } else {
         // serve static content
         mg_serve_http(nc, hm, s_http_server_opts);
      }
   }
}

int main(int argc, char *argv[])
{
   struct mg_mgr mgr;
   struct mg_connection *nc;

   mg_mgr_init(&mgr, NULL);
   nc = mg_bind(&mgr, s_http_port, ev_handler);
   mg_set_protocol_http_websocket(nc);
   s_http_server_opts.document_root = ".";
   s_http_server_opts.enable_directory_listing = "yes";

   while(1) {
      mg_mgr_poll(&mgr, 1000);
   }

   return 0;
}
```

A remote procedure call scheme (RPC) allows the definition of commands linked to call-back functions inside the server which allow the control of hardware. The example above shows a minimalistic program to read a temperature and control a LED on a Raspberry Pi computer.

Data Encoding

Data sent between the server and the browser can be encoded in different formats, each having certain advantages and disadvantages. While simple values can be passed as **plain text**, more complex data structures are best encoded in **Java Script Object Notation** (JSON). A simple JSON.parse() in the browser converts the string into a full Java Script object. For long arrays such as waveform values, **typed arrays** give an about five times speed improvement over JSON encoding. For effective JSON encoding on the server side, the "JSON for Modern C++" library can be used https://github.com/nlohmann/json

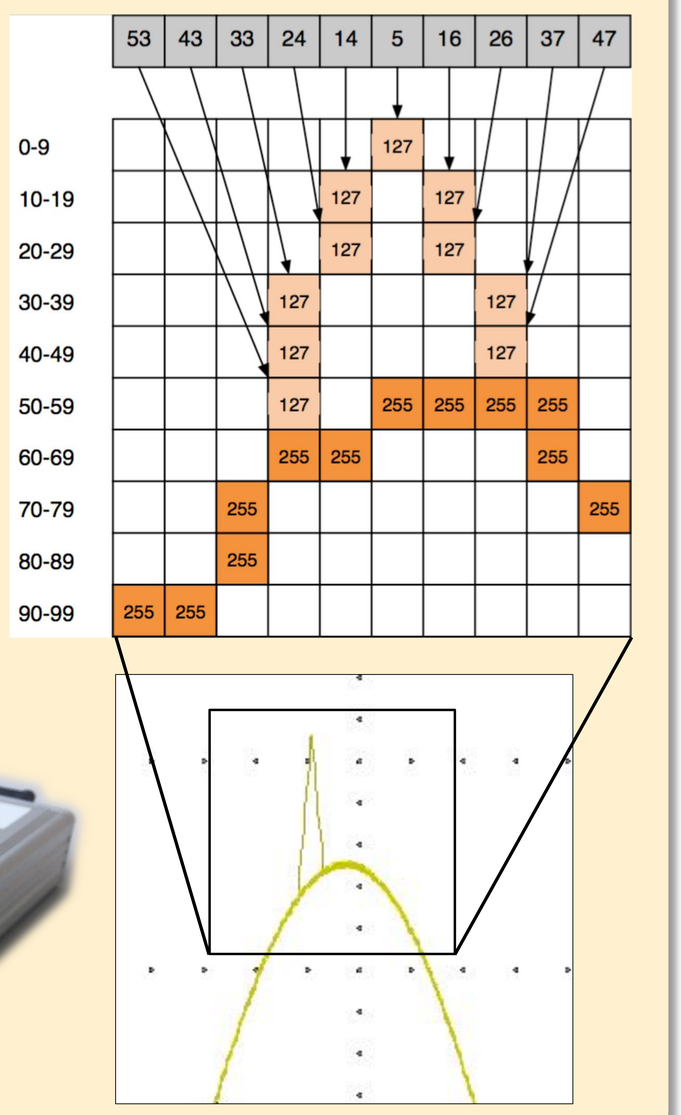
	Plain text	JSON	Typed arrays
Example	23.5 deg. C	{ "Temperature": { "value": 23.5, "unit": "deg. C" }}	0x41 0xBC 0x00 0x00

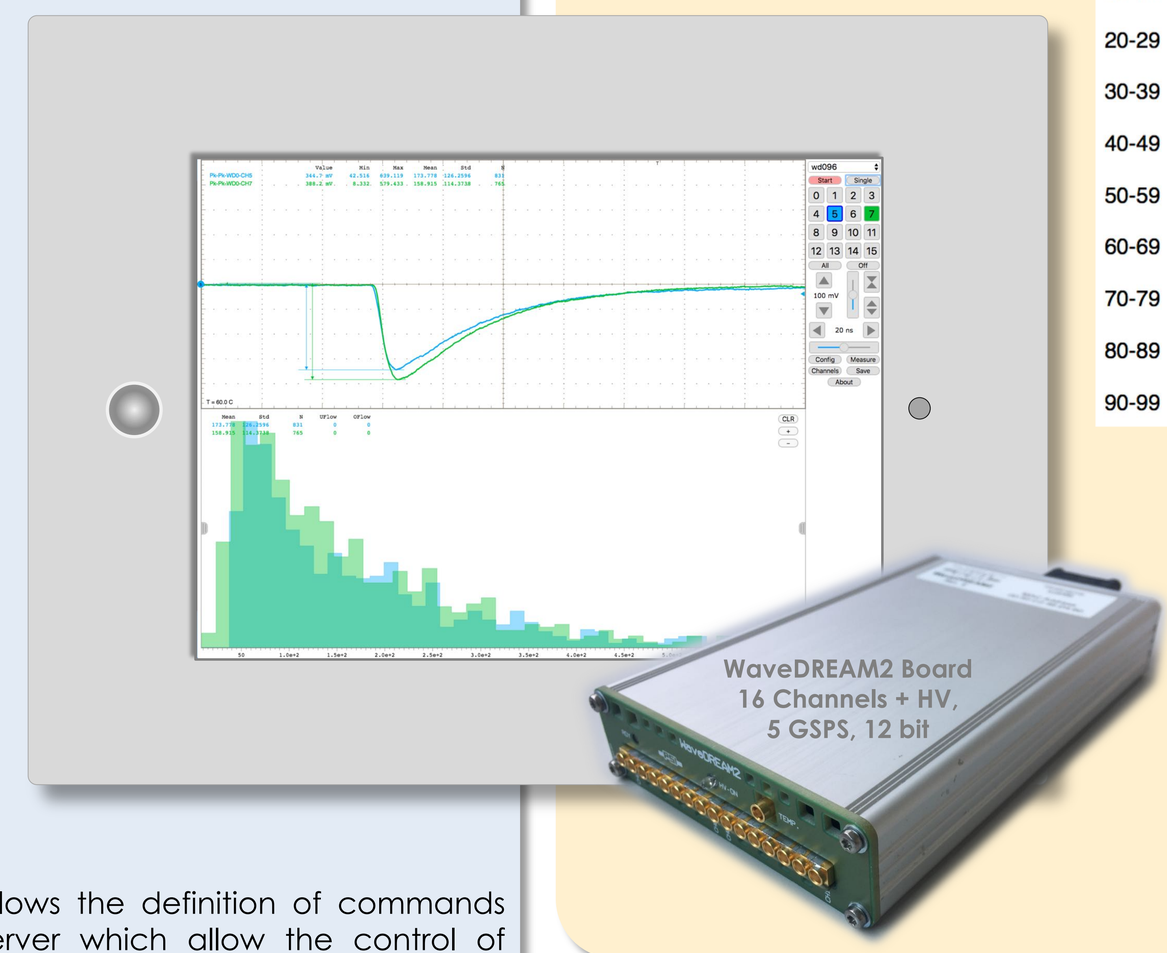
Server Side	mg_printf("%f deg. C", temp);	json j = {
		{ "Temperature": {
		"value", temp,
		"unit", "deg. C" }}
		};
		
		mg_printf(j.dump());
			mg_send(&temp, 4);

Browser Side	var t = parseFloat(response);	var obj = JSON.parse(response); var t = obj.temperatue.value;	var a = Float32Array(response); var t = a[0];
Pro	Simple	Simple to debug, allows complex objects	5x faster than JSON
Con	Complext objects are hard to parse	Slow for long arrays	Cannot be displayed directly in browser for debugging

Data Drawing

HTML5 contains the powerful "canvas" object, which lets you draw 2D and 3D graphics directly in the browser. The oscilloscope application shown in this paper allows rendering of 16 waveforms with 1024 points each at a frame rate of 60 Hz on most browsers with standard canvas functions. The only exception is the drawing of waveforms in "persistency mode", where the traces fades after a few seconds. Instead of using libraries which render "heat maps", a direct image manipulation technique was chosen for simplicity and efficiency. The **createImageData** method is used to obtain an array of all image pixels. After a trace has been drawn by setting pixel values to their maximum RGB value, a timer is started which periodically reduces the RGB value of a pixel. Large computer screens can contain more than one million pixels. Scanning each time all pixels reduces the update rate to below typically 10 Hz. To overcome this limitation, a one-dimensional array is kept which stores a reference to all non-zero pixels.
Scanning only those non-zero pixels reduces the computing needs by two orders of magnitude and brings the frame rate back to 60 Hz.





Custom Controls

Many controls such as drop-down lists and check boxes exist in the HTML standard. Typical control applications require however additional controls such as sliders, image buttons and floating dialog boxes. A light-weight library containing these controls has been developed for this application and can be downloaded at https://elog.psi.ch/scope/controls.js

Conclusions

Modern web technologies allow the development of efficient control and monitoring applications running in web browsers with certain advantages:

- Operating system independent
- Runs on PCs, tablets and smart phones
- Remote access
- No software installation necessary
- Automatic software updates

Graphical web applications will therefore replace traditional graphical programs in the near future. Being able to understand and to use these technologies for own developments will therefore be of great advantage in the field of controls and monitoring.