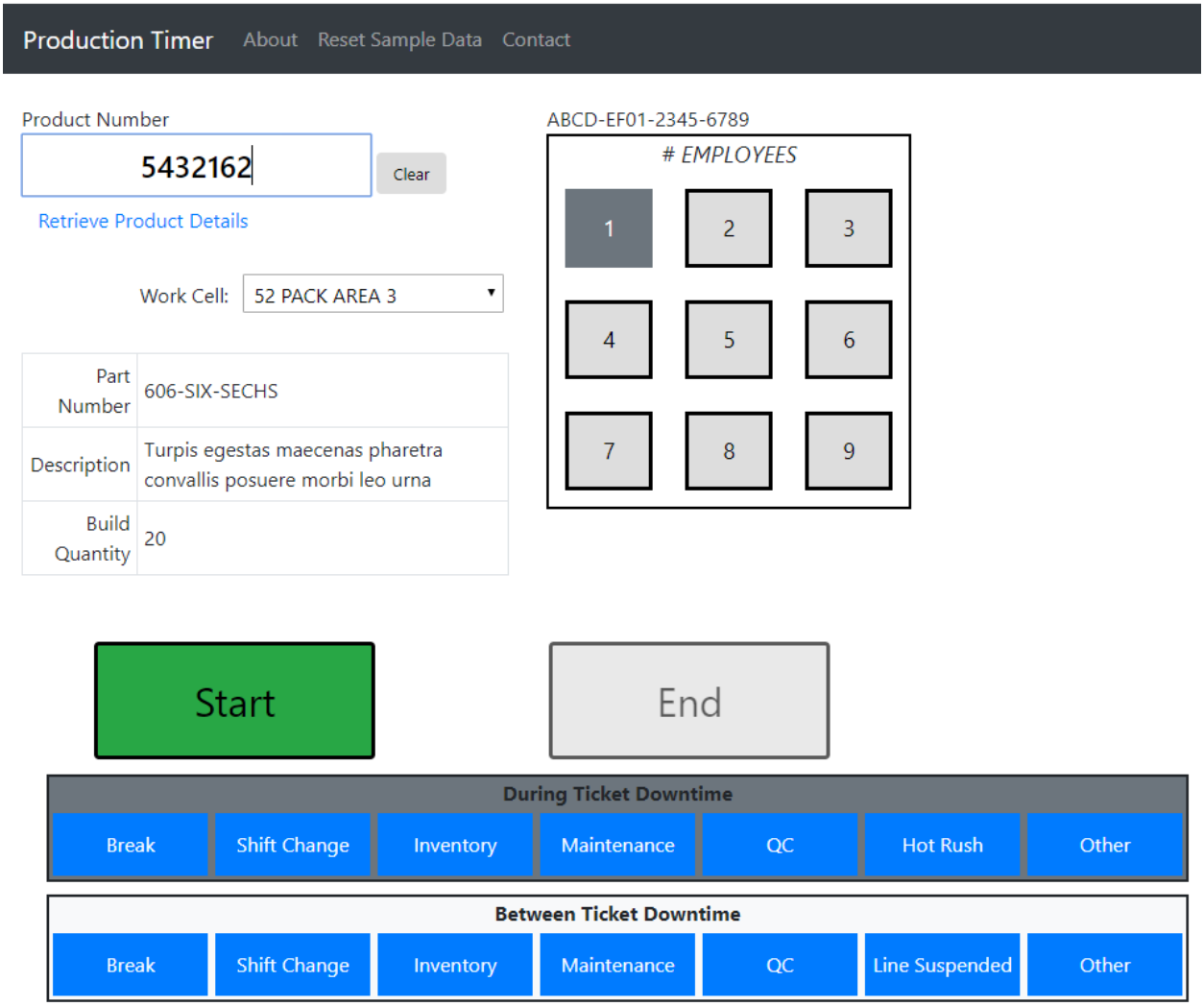
# Production Timer Client

By

David Scott Blackburn




The Production Timer Client is an HTML, CSS, JavaScript application that uses Bootstrap, and AJAX technologies. This document will cover how HTML elements are manipulated and how AJAX was implemented.

Basically, the JavaScript responds to events generated by the HTML code. The JavaScript manipulates other HTML elements, makes AJAX calls, and responds to AJAX returned values. CSS, of course, is used to format the HTML elements, where Bootstrap doesn't cover.

## HTML Element Manipulation

The basic manipulation that takes place is as follows

| Manipulation Type | What is done | Example |
|---|---|---|
| Selecting an Element | document.getElementById(<Id>) | **Set elem to the element with the id of selWorkCell**<br>var elem = document.getElementById("selWorkCell") |

| | | |
|---|---|---|
| | This returns the element object which is used to refer to the object for further manipulation, | |
| Enable/Disable Button | .removeAttribute("disabled") – To Enable<br>.setAttribute("disabled","") – To Disable | **To enable the Start Button**<br>document.getElementById("btnStart").removeAttribute("disabled") |
| Hide/Show Element | .style.visibility = "hidden" – To Hide an element<br>style.visibility = "visible" – To Show a hidden element | **To show an Error message use:**<br>document.getElementById("lblProdNumberError"). style.visibility = "visible" |
| Add/Remove a Class Value | classList.remove(<classValue>)<br>classList.add(<classValue>) | **To set the Start Button to back ground of danger.**<br>document.getElementById("btnStart"). classList.add("bg-danger") |
| Retrieve/Change Value | .value | **Set the value of the element to 'Enter Production #'**<br>document.getElementById("txtProdNumber"). value = "Enter Production #" |
| Sub Elements | *The value and innerHTML properties are very similar, the innnerHTML; however, may include further HTML elements and the value statement is an base value type.*<br>.innerHTML = <some text value>\|<HTML snippet> | **Set the value of the element's innerHTML to a say Hello with bold emphasis.**<br>document.getElementById("sayHI"). value = "<Strong>Hello</Strong>" |
| Check Class for a value | .classList.contains(<class value>) | **Check for a background set to danger.**<br>.classList.contains("bg-danger") |

# AJAX

Asynchronous JavaScript and XML is the definition of AJAX. In general, what occurs is a request is made of a resource and the resource responds to the request and causes the OnReadyStateChange event to fire. The key to AJAX is the `XMLHttpRequest object. Despite its name, most AJAX calls now use JSON instead of XML.`

## Request

For this program, AJAX will be used to make GET and POST request to the Production Timer Web API. For information about the Production Timer Web API refer to the *Production Timer Web API Design Document*.

The **GET** is a simple request to return a List of Strings, the **POST**s make various requests to perform some type of update and perform specific queries regarding a specific Production Order. The data that is sent to and returned from the request are defined by the Web API. Each POST message has a different JSON structure that has to accompany the request.

Here is how the request is made…

```
        This sets up the request structure required by the Web API. In this case, a POST request
        to use a Production Order (call Set) is about to be made, for this request the Production
        Number (known as the id) and the Workstation Id is required.
var reqSet = {            id: document.getElementById("txtProductionNumber").value,
                    workstationId: document.getElementById("lblWorkStationId").innerHTML };

        This sets up the request structure required by the Web API.
var xhr = new XMLHttpRequest();

        This is the full URI where the request is to be sent.  In this case the POST will be made
        to the ProductionTimer Web API and is using the Set action.
var localURL = = 'http://www.dsbburn.com/api/ProductionTimer/Set';

        This covert the pure JSON structured data to a string format that is needed in order to
        transmit it to the Web API.
var payload = JSON.stringify(reqSet);

        This is the critical call that tell the XMLHttpRequest object that a POST is to made
        asynchronously to the URL specified by localURL. The 3rd parameter true is used to
        signifiy that the request is asynchronous.
xhr.open("POST", localURL, true);

        This tells the system that it is using JSON structured data in the request.
xhr.setRequestHeader("Content-type", "application/json");

        This sets the response handler.  A function named handleResponse which takes as a
        parameter the XMLHttpRequest object as a parameter.
```

```
        xhr.onreadystatechange = function () { handleResponse(xhr); };

              This makes the AJAX call, sending the data formatted earlier.
        xhr.send(payload);
```

This asynchrony POSTs a request to the localURL sending to the post the data stored in payload.

## Response

In order to respond to an AJAX request a function must be defined to handle the onreadystatechange event.

For this program handleResponse(xhr) is used. The actual handler takes a second parameter, responseOption, that tells the function how to process the returned data. The Web API has two standard POST responses: one for the **Set POST** call and a General response for the other POST calls.

The *General Post Response* contains a status code, where the value of 1 is a valid response and any other response signifies an invalid. The second field is an error text that contains a description of what is wrong. Normally the error text is null or blank.

The *Set Post Response* contains data that includes details about the Production Order, including the Product Number, Description, and Quantity to be Produced.

The primary logic for a response function is to check the XMLHTTPRequest object's reayState and status and values.

```
function handleResponse(xhr, responseOpt) {
    if (xhr.readyState === 4 && xhr.status === 200) {
        switch (responseOpt) {
            case "Set":
                recievedProductionNumber(JSON.parse(xhr.responseText));
                break;
       . . .
```

Refer to https://www.w3schools.com/js/js_ajax_http_response.asp for more information regarding the details about the response function and what it should do.

# Summary

Well that about it regarding this application. Cookies are used to persist data across multiple calls. Certain things were not implemented such as determining the Workstation Id and implementing the Retrieve Production Detail link. These may be implemented later.

This emulation does not perform exactly like the actual Production Timer.

If you have questions or comments please contact me via email.