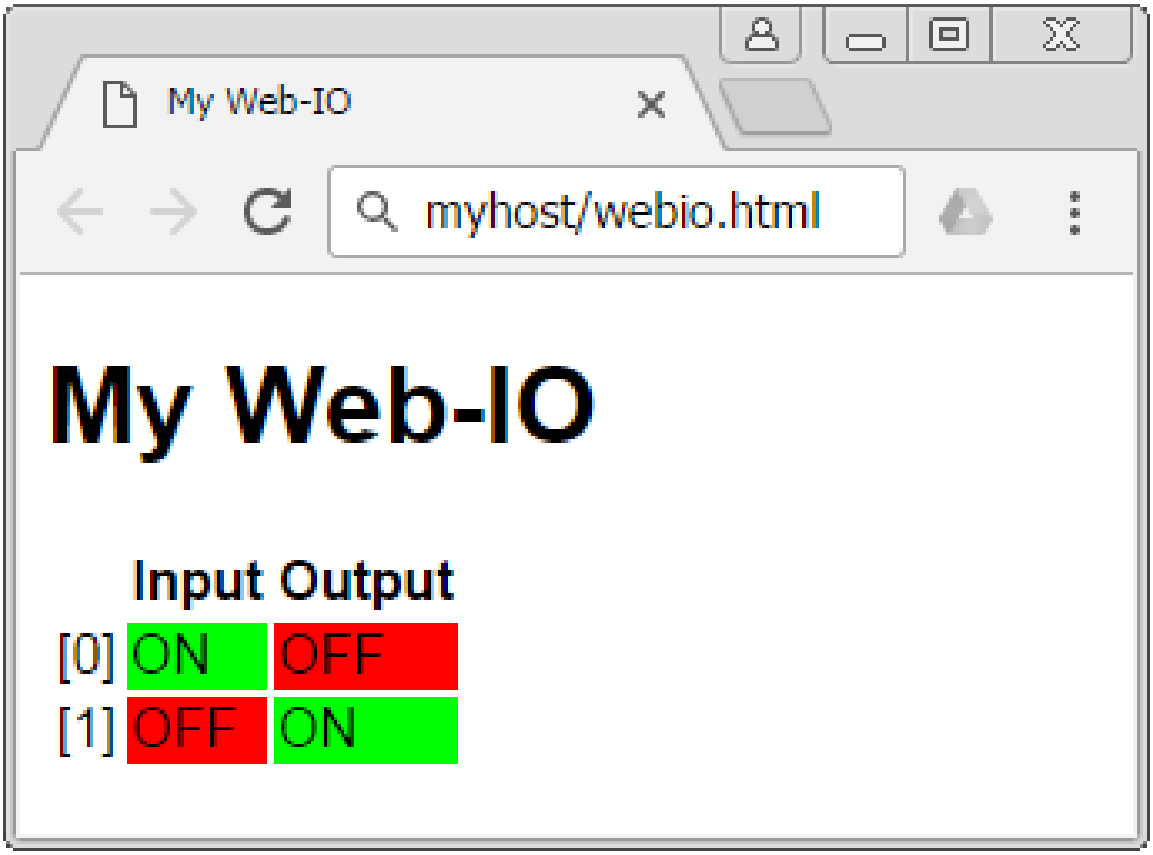
Applikation zum Web-IO

MQTT Web-Client mit JavaScript

als individuelle Oberfläche für Web-IO



In diesem Tutorial wird der JavaScript-Client aus dem Eclipse Paho-Projekt dazu genutzt, eine eigene webbasierte Oberfläche für das Web-IO 4.0 Digital zu erstellen.

Hierbei werden die Zustände des Web-IOs als Retained Message beim Broker zwischengespeichert und vom Webclient auf eine einfache HTML-Tabelle abgebildet.

Wird einer der Ausgänge angeklickt, sendet der Web-Client einen MQTT-Publish, der das Web-IO anweist, den entsprechenden Ausgang zu schalten.

Aufbau des Clients

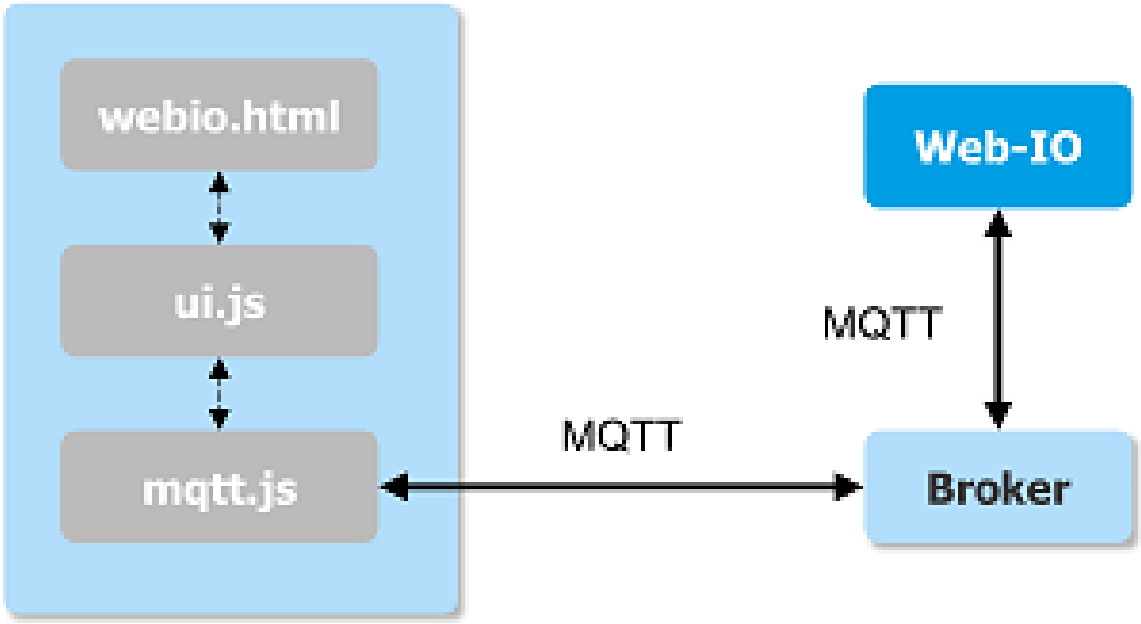
Der Webclient stellt eine Benutzeroberfläche für ein entferntes Web-IO bereit. Die Kommunikation zwischen Web-IO und Web-Client erfolgt über das IoT-Protokoll MQTT.

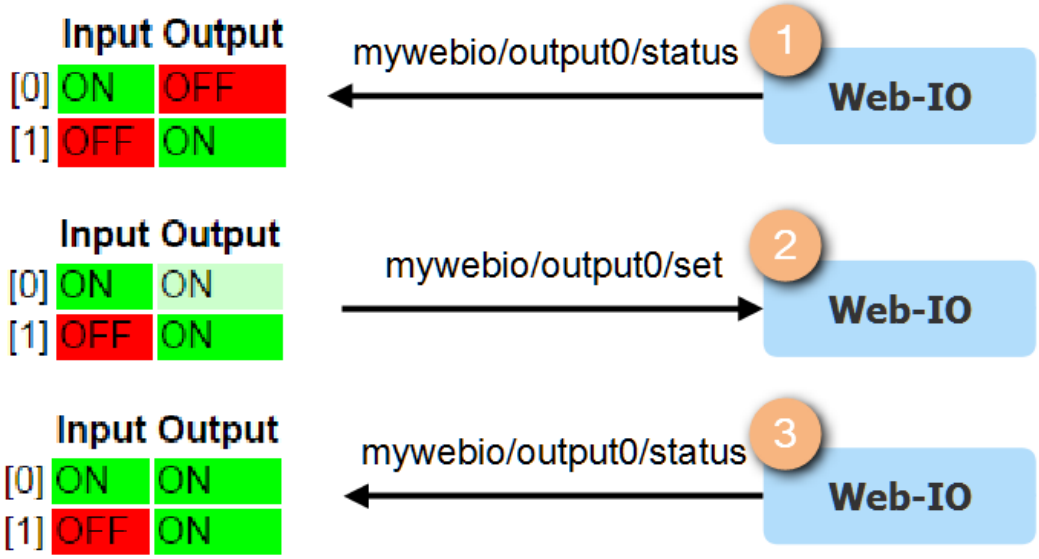
Der Client besteht aus den drei Dateien:

- **webio.html**
- **ui.js**
- **mqtt.js**

Die HTML-Datei stellt das Markup für die grafische Darstellung bereit. Die Javascript-Datei mqtt.js enthält die MQTT-spezifischen Funktionen wie Verbindungsmanagement, Publish und Subscribe. Die Datei ui.js ist die Schnittstelle zwischen MQTT-Client und Weboberfläche. Sie enthält die Funktionen, die bei Empfang einer Subscription die Oberfläche aktualisieren und die bei einer Benutzerinteraktion den Versand eines PUBLISH-Pakets auslösen.

Kommunikation zwischen Web-Client und Web-IO





In der nebenstehenden Darstellung der Kommunikation zwischen Web-IO und Web-Client wird der Einfachheit halber auf den Broker verzichtet.

Die Kommunikation erfolgt über die Topics **mywebio/<io>/status**, über welches das Web-IO den aktuellen Schaltzustand veröffentlicht, und **mywebio/<io>/set**, über welches das Web-Interface eine Schaltanforderung an das Web-IO sendet.

1 Nachdem sich der Web-Client verbunden hat, empfängt er die Zustände der einzelnen Ein- und Ausgänge als **Retained Messages** und stellt sie in einer HTML-Tabelle dar.

2 Sobald der Benutzer auf einen Ausgang klickt (hier Output 0), setzt der Web-Client einen Publish mit der Payload "ON" und dem Topic "mywebio/output0/set" ab. Die unbestätigte Anforderung wird im Web-Client durch eine blassere Farbe angezeigt.

3 Das Web-IO reagiert auf den Empfang der Subscription, indem es den entsprechenden Ausgang einschaltet. Dann sendet es einen Publish mit dem neuen Zustand als Payload und dem Topic "mywebio/output0/status". Sobald der Web-Client die Nachricht empfangen hat, stellt es den Ausgang wieder vollfarbig dar.

Hinweis: *Bei mehreren aufeinanderfolgenden Änderungsanforderungen ohne eine Bestätigung des Web-IOs wird der entsprechende Ausgang ausgegraut und erst dann wieder aktiviert, wenn das Web-IO seinen gegenwärtigen Status veröffentlicht hat.*

Vorbereitung: Konfiguration des Brokers und des Web-IOs

Benutzer und Zugriffsrechte festlegen

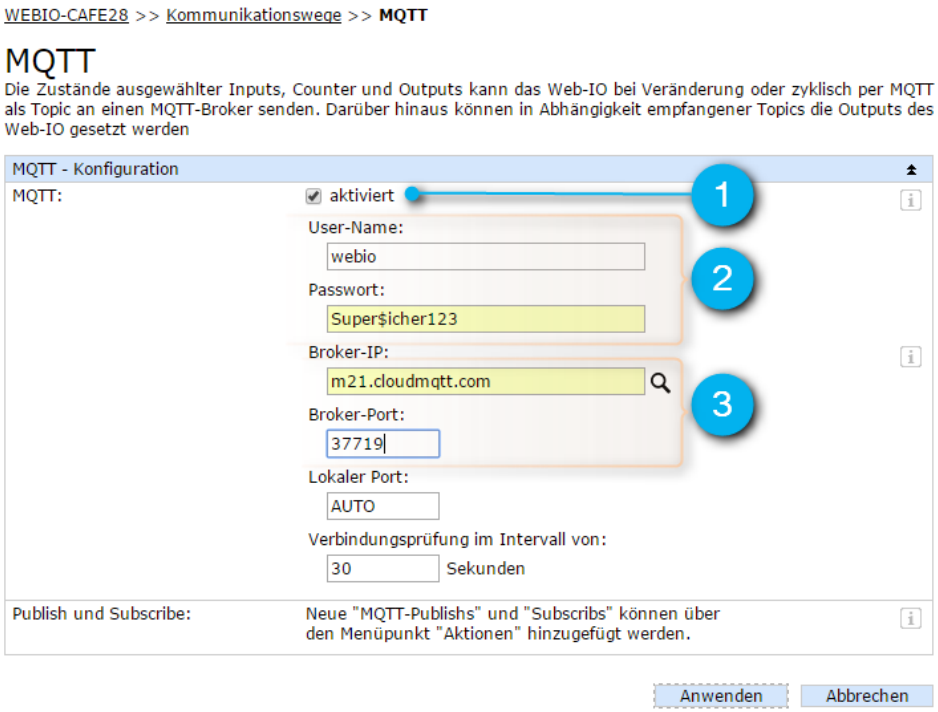
Für dieses Turorial wird, wie schon im Tutorial Box-2-Box via MQTT, der für kleine Anwendungen kostenlose Broker cloudmqtt.com eingesetzt.

Der Einfachheit halber wird nur ein einziger Benutzer mit weitreichenden Rechten angelegt:

Benutzername	Passwort	Schreibrechte	Leserechte
webio	Super$icher123	#	#

Grundeinstellungen für MQTT

Das Web-IO wird auf der Seite "Kommunikationswege >> MQTT" für das Zusammenspiel mit dem Broker konfiguriert.



1 Aktivieren Sie zunächst MQTT.

2 Geben Sie den Benutzernamen und das Passwort an.

3 Geben Sie die Verbindungsinformationen für den Broker an.

Aktionen für das Schalten der Ausgänge anlegen

Öffnen Sie jetzt die Seite "Aktionen" und klicken Sie auf "Hinzufügen", um die Aktionen für den Empfang einer Subscription anzulegen.

1 Aktivieren Sie die Aktion und bennennen Sie sie sinnvoll.

2 Auslöser ist der Empfang einer **MQTT-Subscription** mit dem **Topic-Pfad** mywebio/output0/set und dem **Topic-Text** ON.

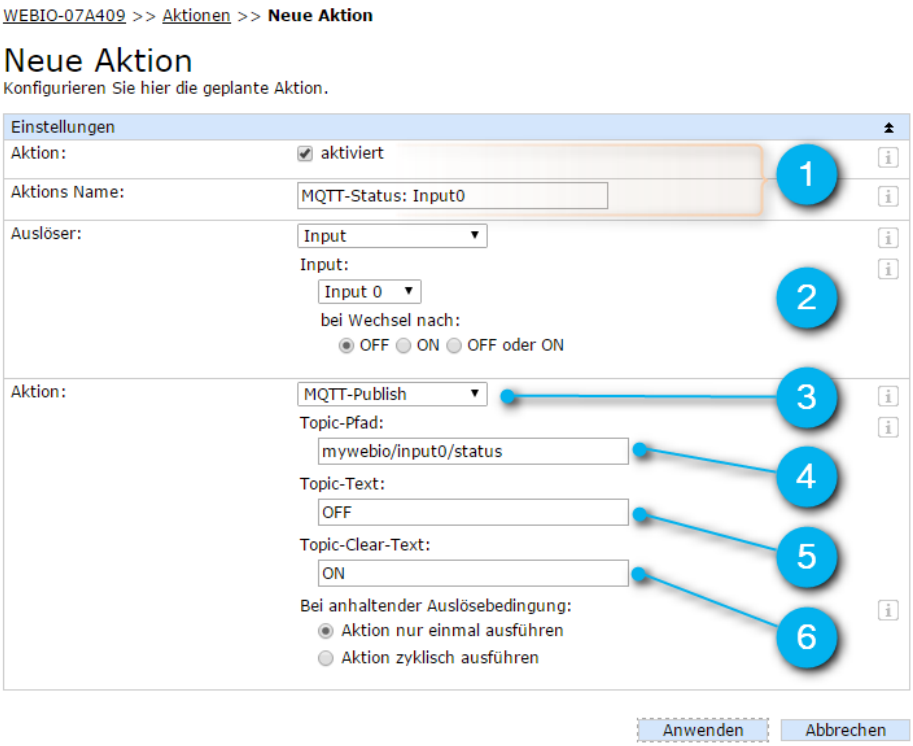
3 Tritt der Auslöser auf, soll das Web-IO einen **Output schalten**, in diesem Fall **Output 0** auf "ON".

Wiederholen Sie den letzten Schritt, bis Sie insgesamt vier Aktionen definiert haben: Output 0 auf ON, Output 0 auf OFF,

Output 1 auf ON und Output 1 auf OFF.

Aktionen für den Publish von IO-Zuständen anlegen

Im letzten Konfigurationsschritt werden die Aktionen erstellt, die bei einem Zustandswechsel an einem beliebigen Ein- oder Ausgang einen MQTT-Publish absetzen.



1. Setzen Sie die Aktion auf **aktiv** und geben Sie ihr einen sinnvollen **Namen.**
2. Wählen Sie als Auslöser **Input** und wählen Sie dann einen Wechsel auf **OFF** für **Input0**.
3. Als Aktion soll ein **MQTT-Publish** ausgeführt werden.
4. Das **Topic** für Veröffentlichung eines neuen Zustands ist "mywebio/input0/status".
5. Die Payload **(Topic-Text)** für einen Wechsel nach OFF ist "OFF".
6. Der **Topic-Clear-Text** ist die Nachricht, die versendet wird, wenn der Trigger nicht mehr vorliegt, also "ON".

Wiederholen Sie die Einrichtung der Aktion für den anderen Eingang und für die beiden Ausgänge.

Implementierung des Web-Clients

Nachfolgend werden die drei Dateien **webio.html**, **mqtt.js** und **ui.js** im Detail besprochen. Die HTML-Datei enthält lediglich grundlegende Gestaltungselemente, daher wird nur kurz darauf eingegangen. Es folgt eine intensivere Diskussion der beiden JavaScript-Dateien.

Alle Dateien herunterladen

webio.html: Die grafische Darstellung

```
<!doctype html>
<html lang="de">
 <head>
  <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.js"></script>
  <script src="webio_mqtt.js" type="text/javascript"></script>
  <script src="ui.js" type="text/javascript" defer></script>
  <meta charset="utf-8">
  <title>My Web-IO</title>
  <style type="text/css">
   body {font-family: sans-serif;}
   table {empty-cells: show;}
   .on {background-color: #00FF00;}
   .off {background-color: #FF0000;}
   .set_on {background-color: #CCFFCC;}
   .set_off {background-color: #FFCCCC;}
   .unknown {background-color: #DDDDDD;}
   .on:before, .set_on:before {content: "ON";}
   .off:before, .set_off:before {content: "OFF";}
   .unknown:before {content: "?";}
   #output0,#output1{cursor: pointer;}
  </style>
 </head>

 <body>
  <h1>My Web-IO</h1>
  <table>
  <tr>
   <th></th>
   <th>Input</th>
   <th>Output</th>
  </tr>
  <tr>
   <td >[0]</td>
   <td class="unknown" id="input0"></td>
   <td class="unknown" id="output0"></td>
  </tr>
  <tr>
   <td >[1]</td>
   <td class="unknown" id="input1"></td>
   <td class="unknown" id="output1"></td>
  </tr>
  </table>
 </body>
</html>
```

Zeilen 4-6:
Einbinden des PAHO-JavaScript-Clients. Alternativer URL: https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.js
Zeile 7 u. 8:
Einbinden der lokalen JS-Dateien. Das Attribut **defer** sorgt dafür, dass die Funktionen, die auf HTML-Elemente zugreifen, erst dann ausgeführt werden, wenn die gesamte Seite geladen wurde.
Zeile 12-24:
Style-Informationen für die verschiedenen Zustände. Neben der Zuweisung einer Farbe wird dem Elements entweder der Text **ON**, **OFF** oder **?** vorangestellt.
Zeile 29-45:
Die Tabelle, die Ein- und Ausgänge anzeigt. Die einzelnen Ein- und Ausgänge werden hier durch Tabellenzellen repräsentiert, denen eine CSS-Klasse für die Darstellung zugewiesen wird. Identifiziert werden die Elemente über IDs, die genau so benannt sind, wie die Ein- und Ausgänge des Web-IOs.

mqtt.js: Der eigentliche MQTT-Client

Die Datei **mqtt.js** enthält die Funktionen für den Verbindungsaufbau zum MQTT-Broker und für das Empfangen und Versenden von Nachrichten.

```
/*
 * MQTT-WebClient example for Web-IO 4.0
*/
var hostname = "m21.cloudmqtt.com";
var port = 37719;
var clientId = "webio4mqttexample";
clientId += new Date().getUTCMilliseconds();;
var username = "webclient";
var password = "Super$icher123";
var subscription = "mywebio/+/status";

mqttClient = new Paho.MQTT.Client(hostname, port, clientId);
mqttClient.onMessageArrived = MessageArrived;
mqttClient.onConnectionLost = ConnectionLost;
Connect();

/*Initiates a connection to the MQTT broker*/
function Connect(){
 mqttClient.connect({
 onSuccess: Connected,
 onFailure: ConnectionFailed,
 keepAliveInterval: 10,
 userName: username,
 useSSL: true,
 password: password});
}

/*Callback for successful MQTT connection */
function Connected() {
 console.log("Connected");
 mqttClient.subscribe(subscription);
}

/*Callback for failed connection*/
function ConnectionFailed(res) {
 console.log("Connect failed:" + res.errorMessage);
}

/*Callback for lost connection*/
function ConnectionLost(res) {
 if (res.errorCode !== 0) {
  console.log("Connection lost:" + res.errorMessage);
  Connect();
 }
}

/*Callback for incoming message processing */
function MessageArrived(message) {
 console.log(message.destinationName +" : " + message.payloadString);
 switch(message.payloadString){
  case "ON":
   displayClass = "on";
   break;
  case "OFF":
   displayClass = "off";
   break;
  default:
   displayClass = "unknown";
 }
 var topic = message.destinationName.split("/");
 if (topic.length == 3){
  var ioname = topic[1];
  UpdateElement(ioname, displayClass);
 }
}
```

Konfigurationswerte und Verbindungsaufbau

Zunächst werden ein paar Konfigurationswerte gesetzt:

```
var hostname  = "m21.cloudmqtt.com";
var port   = 37719;
var clientId = "webio4mqttexample";
clientID += new Date().getUTCMilliseconds();
var username = "webclient";
var password = "Super$icher123";
var subscription = "mywebio/+/status";
```

Zeilen 4 u. 5:
Die Verbindungsinformationen für den MQTT-Broker
Zeile 6:
Eine einzigartiger String, der den zugreifenden Client identifiziert.
Zeile 7:
Die Anweisung `new Date().getUTCMilliseconds()` sorgt dabei dafür, dass mehrere Instanzen des Clients gleichzeitig geöffnet sein dürfen, indem es einen aktuellen Zeitstempel an die Client-ID anfügt. Mehrere Verbindungen mit der selben Client-ID würde der Broker dem Standard entsprechend unterbrechen.
Zeilen 8 u. 9:
Der Login-Daten des Benutzers
Zeile 10:
Das Topic, über das das Web-IO seine Zustände veröffentlicht.
Es folgt die Initialisierung des Clients:

```
mqttClient = new Paho.MQTT.Client(hostname, port, clientId);
mqttClient.onMessageArrived = MessageArrived;
mqttClient.onConnectionLost = ConnectionLost;
Connect();
```

Zeile 12:
Initialisierung des MQTT-Clients mit dem Hostnamen des Brokers, dem Port und der einzigartigen Client-ID.
Zeile 13:
`MessageArrived()` ist die Callback-Funktion, die bei Empfang einer Subscription aufgerufen wird.
Zeile 14:
`ConnectionLost()` wird im Falle eines Verbindungsabbruchs aufgerufen.
Zeile 15:
Mit Connect wird die Verbindung aufgebaut.
Die Funktion `Connect()` ist verantwortlich für den Verbindungsaufbau zum MQTT-Broker. Das Konfigurationsobjekt enthält neben den bereits angegebenen Parametern die Callback-Funktionen für einen erfolgreichen bzw. fehlgeschlagenen Verbindungsaufbau.

```
/*Initiates a connection to the MQTT broker*/
function Connect(){
 mqttclient.connect({
  onSuccess: Connected,
  onFailure: ConnectionFailed,
  keepAliveInterval: 10,
  userName: username,
  useSSL:  true,
  password: password
 });
}
```

Zeile 19:
Die Verbindung ruft die Methode `connect()` des MQTT-Clients auf.
Zeile 20:
Nach erfolgreichem Verbindungsaufbau wird die Callback-Funktion `Connected()` ausgeführt.
Zeile 21:
Nach einem gescheiterten Verbindungsversuch wird `ConnectionFailed()` aufgerufen.
Zeile 24:
Die Kommunikation erfolgt verschlüsselt über TLS.

Die Callback-Funktionen

Im ersten Abschnitt wurden vier Callback-Funktionen erwähnt: `MessageArrived()` und `ConnectionLost()` wurden als Attribute des MQTT-Clients gesetzt, `Connected()` und `ConnectionFailed()` wurden der Funktion `Connect()` im Konfigurationsobjekt übergeben.

Connected() wird ausgeführt, wenn der Verbindungsaufbau erfolgreich war.

```
/*Callback for successful MQTT connection */
function Connected() {
 console.log("Connected");
 mqttclient.subscribe(subscription);
}
```

Zeile 31:
Der Verbindungsstatus wird über die JavaScript-Konsole ausgegeben.

Zeile 32:
Da es sich nicht um eine persistente Verbindung handelt, muss das Topic für Nachrichten vom Web-IO bei jedem Verbindungsaufbau neu abonniert werden.
Bei einer fehlgeschlagenen Verbindung gibt **ConnectionFailed()** eine Information auf der JavaScript-Konsole aus.

```
/*Callback for failed connection*/
function ConnectionFailed(res) {
  console.log("Connect failed:" + res.errorMessage);
}
```

Im Falle eines Verbindungsabbruchs wird **ConnectionLost()** aufgerufen.

```
/*Callback for lost connection*/
function ConnectionLost(res) {
 if (res.errorCode !== 0)
 {
 console.log("Connection lost:" + res.errorMessage);
 Connect();
 }
}
```

Zeile 43:
Über die JavaScript-Konsole werden Informationen zum Verbindungsabbruch ausgegeben.
Zeile 44:
Dann wird eine neue Verbindung aufgebaut.
Die umfangreichste Funktion ist **MessageArrived()**. Der obere Teil der Funktion wertet den PayLoad-String aus. Über diesen wird die CSS-Klasse bestimmt, die dem entsprechenden Element zugewiesen wird.

```
/*Callback for incoming message processing */
function MessageArrived(message) {
 switch(message.payloadString){
  case "ON":
   displayClass = "on";
   break;
  case "OFF":
   displayClass = "off";
   break;
  default:
   displayClass = "unknown";
 }
```

Zeile 50:
Das Message-Objekt besitzt ein Attribut payloadString, das den Inhalt der übermittelten Nachricht enthält.
Zeilen 51-56:
Wenn der Nachrichteninhalt "ON" ist, wird die Variable displayClass auf "on" gesetzt. Bei der PayLoad "OFF" auf "off".
Zeilen 57-58:
Bei einer fehlerhaften Payload wird die Display-Klasse auf "unknown" gesetzt.
Der untere Teil der Funktion bestimmt, welcher In- oder Output aktualisiert werden soll:

```
var topic = message.destinationName.split("/");
 if (topic.length == 3){
  var ioname = topic[1];
  UpdateElement(ioname, displayClass);
 }
}
```

Zeile 61:
Zunächst wird der message.destinationName, also das Topic der Nachricht, mit der String-Methode split() am Zeichen "/" in seine einzelnen Glieder zerlegt und diese im Array topic gespeichert.
Zeile 63:
Bei mywebio/**ioname**/status ist das mittlere Element der Name des entsprechenden Ein- oder Ausgangs.
Zeile 64:
Das DOM-Element ioname bekommt in der Funktion UpdateElement() die CSS-Klasse displayClass zugewiesen.

ui.js: Die Verbindung zwischen HTML-Seite und MQTT-Client

Die **ui.js** enthält die Funktionen, die die Darstellung der HTML-Seite aktualisieren und die nach einer Benutzerinterkation die MQTT-Client-Funktionen aufrufen.

```
/*
 * Web-IO 4.0: MQTT WebSocket example
 */

/* Updates the CSS class of an DOM element */
function UpdateElement(ioname, displayClass){
 var cell = document.getElementById(ioname);
 if (cell){
  cell.className = displayClass;
 }
 }

 /* Toggles an input in the web interfaces and
 * initiates an MQTT publish */
function ToggleOutput(ioname){
 var cell = document.getElementById(ioname);
 switch (cell.className){
 case "on":
  var message = new Paho.MQTT.Message("OFF");
  message.destinationName = "webio/" + ioname + "/set";
  mqttClient.send(message);
  cell.className = "set_off";
  break;
 case "off":
  var message = new Paho.MQTT.Message("ON");
  message.destinationName = "webio/" + ioname + "/set";
  mqttClient.send(message);
  cell.className = "set_on";
  break;
 default:
  cell.className = "unknown";
  break;
 }

};

/* Adds an Click-Event-Listener to a table cell, so that after
 * a click the element can is toggeled */
function EnableToggle(ioname){
  var cell = document.getElementById(ioname)
  if (cell){
   cell.addEventListener("click",
    function(){
    ToggleOutput(ioname)
   },
    true);
  }
 }

/*Initialize elements that can be toggled my by click*/
EnableToggle("output0");
EnableToggle("output1");
```

Aktualisieren der Anzeige durch den MQTT-Client

Mit der Funktion `UpdateElement()` setzt der MQTT-Client bei Empfang einer Subscription die CSS-Klasse der entsprechenden Tabellenzelle auf den gegenwärtigen Zustand.

```
/* Updates the CSS class of an DOM element */
function UpdateElement(ioname, displayClass){
 var cell = document.getElementById(ioname);
 if (cell){
  cell.className = displayClass;
 }
 }
```

Zeile 6:
Die Funktion bekommt die ID des DOM-Elements und die CSS-Klasse übergeben.
Zeile 9:
Diese Klasse wird dem Element zugewiesen.

Benutzerinteraktionen

Die Funktion **`ToggleOutput()`** passt bei einem Klick auf einen Output dessen Darstellung an und versendet einen

entsprechenden MQTT-Publish. Ist die gegenwärtige Klasse `on`, wird ihr die Klasse `set_off` zugewiesen, ist die Klasse `off`, wird ihr die neue Klasse `set_on` zugewiesen. In allen anderen Fällen ist der aktuelle Status des Ausgangs unbestätigt und der Anzeige wird die Klasse `unknown` zugewiesen.

```
/* Toggles an input in the web interfaces and
 * initiates an MQTT publish
 */
function ToggleOutput(ioname){
 var cell = document.getElementById(ioname);
 switch (cell.className){
 case "on":
  var message = new Paho.MQTT.Message("OFF");
  message.destinationName = "webio/" + ioname + "/set";
  mqttClient.send(message);
  cell.className = "set_off";
  break;
 case "off":
  var message = new Paho.MQTT.Message("ON");
  message.destinationName = "webio/" + ioname + "/set";
  mqttClient.send(message);
  cell.className = "set_on";
  break;
 default:
  cell.className = "unknown";
  break;
 }
};
```

Zeile 16:
Die Funktion bekommt die ID des DOM-Elements übergeben, das den Output darstellt.
Zeile 18:
Abhängig von der gegenwärtigen Klasse des Elements wird
Zeilen 19-24, 25-30 und 21-33:
die neue Klasse auf `set_on`, `set_off` oder `unknown` gesetzt.
Damit die Funktion `ToggleOutput()` bei einem Klick auf eine Tabellenzelle aufgerufen werden kann, fügt `EnableToggle()` den übergebenen Elementen einen Event-Handler für einen Mausklick hinzu.

```
/* Adds an Click-Event-Listener to a table cell, so that after
 * a click the element can is toggeled
 */
function EnableToggle(ioname){
  var cell = document.getElementById(ioname)
  if (cell){
   cell.addEventListener("click",
    function(){
    ToggleOutput(ioname)
   },
    true);
  }
 }
```

Zeile 40:
Die Methode bekommt den Namen eines IOs übergeben.
Zeile 41:
Dieser Name wird genutzt, um das entsprechende Tabellenzelle zu referenzieren.
Zeile 43-47:
Über die Methode `addEventListener()` wird dem Element `cell` ein Event-Handler für einen `click` zugewiesen. Wird auf das entsprechende Element geklickt, soll die Funktion `ToggleOutput()` mit dem Namen des IOs als Parameter aufgerufen werden.
Abschließend wird `EnableToggle()` für die beiden Elemente `"output1"` und `"output2"` aufgerufen:

```
/*Initialize elements that can be toggled my by click*/
EnableToggle("output0");
EnableToggle("output1");
```

Dieses Tutorial mit Hardware durcharbeiten

Sie möchten dieses Tutorial durcharbeiten, Ihnen fehlt aber die notwendige Hardware? Gerne stellen wir Ihnen Web-IO Digital 4.0 als Muster zur Verfügung.

Offene Fragen zu Web-IO Digital und MQTT?

Herr Thiel hilft Ihnen gerne weiter.
Telefon: 0202/2680-110 (Mo-Fr. 8-17 Uhr)
E-Mail: f.thiel@wut.de

Wir sind gerne persönlich für Sie da:

Wiesemann & Theis GmbH
Porschestr. 12
42279 Wuppertal
Tel.: 0202/2680-110 (Mo-Fr. 8-17 Uhr)
Fax: 0202/2680-265
info@wut.de