

Web-Programmierung

Dieses Buch ist für alle gemacht, die zwar ein Grundwissen in einer der gängigen Programmiersprachen haben, denen die Netzwerkprogrammierung aber bis jetzt fremd ist. Es ist in fünf Teile gegliedert:

- **Socket-Programmierung allgemein**
TCP-Client oder -Server, UDP-Peer oder Browser-basierende Anwendung - Hier finden Sie Entscheidungshilfe bei der Auswahl der richtigen Technik.
- **Netzwerkanalyse mit Wireshark**
Datensendung auf Netzwerkebene mitschneiden und bewerten.
- **Dynamische Web-Anwendungen**
Browser-basierende Visualisierungen und Steuerung von Anwendungen mit JavaScript, AJAX und PHP.

Die gezeigten Beispiele zeigen zunächst die Basistechniken der Netzwerk bzw. Web-Kommunikation und gehen dann Schritt für Schritt mehr in die Tiefe.

Autor:
Frank Thiel

© 05/2010 by Wiesemann & Theis GmbH
1. Vorabauflage Auflage

Nachdruck, auch auszugsweise, mit Quellenangabe einschließlich Internetadresse von W&T (<http://www.wut.de>) ausdrücklich erlaubt.

Microsoft, MS-DOS, Windows, Winsock und Visual Basic sind eingetragene Warenzeichen der Microsoft Corporation.

Irrtümer und Änderungen vorbehalten.

Da wir Fehler machen können, darf keine unserer Aussagen ungeprüft verwendet werden. Bitte melden Sie uns alle Ihnen bekannt gewordenen Irrtümer oder Missverständlichkeiten, damit wir diese so schnell wie möglich erkennen und beseitigen können.

Inhalt

Web-Programmierung	1
Socket-Programmierung allgemein	5
TCP – Verbindungsbezogene Datenkommunikation	6
Client und Server	6
Verbindung und Datenaustausch	6
Single-Client und Multi-Client Server-Anwendungen	7
UDP – Verbindungslose Datenkommunikation	9
UDP Broadcast	9
Besonderheiten und Tücken der UDP Kommunikation	11
Browser-gestützte Anwendungen	12
Statische Webseiten	12
Webseiten mit multimedialen Inhalten	13
Interaktion durch Formulare	14
AJAX - Echte Interaktion durch JavaScript und HTTP-Request-Technik	14
Java-Applets – Formatunabhängige Kommunikation	15
TCP-Sockets, UDP-Sockets oder Browseranwendung	17
TCP/UDP - Socketprogrammierung in Hochsprachen	17
TCP/UDP - Netzwerkkommunikation allgemein	18
TCP – Gesicherte Kommunikation	19
TCP-Client oder TCP-Server	19
UDP - Verbindungslos, flexibel und schnell	20
Browseranwendungen - Betriebssystemunabhängigkeit	21
Netzwerkanalyse	23
Netzwerkanalyse hilft bei der Socketprogrammierung	24
Wireshark - eine kurze Einführung	25
TCP-Datenverkehr mitschneiden und auswerten	27
UDP-Datenverkehr mitschneiden und auswerten	36
Mitschneiden auf einem unbeteiligten PC	39
Web-Programmierung in der Praxis	41
Web-Programmierung - Anwendungen im Browser	42
Aufbau von Web-Anwendungen	42
Werkzeuge für die Entwicklung von Web-Anwendungen	42

HTML – Hypertext Markup Language	44
Grundsätzlicher Aufbau einer HTML-Datei	46
Hyperlinks	47
Darstellung von multimedialen Inhalten	48
JavaScript	52
Eigenschaften eines HTML-Objektes nachträglich ändern	62
Mit zugeladenen Bildern arbeiten	64
Java-Applets für kontinuierlichen Datenaustausch	67
Digitale Signale mit einem Java-Applet überwachen	75
Digitale Outputs über ein Java-Applet steuern	82
AJAX - lebendige Webseiten ohne JavaApplets	87
Klimadaten mit AJAX visualisieren	94
Digitale Signale mit AJAX überwachen	98
Zustand eines Inputs anzeigen	104
Inputs und Counter mit AJAX anzeigen	108
Alle IOs eines Web-IO 12x Digital anzeigen	113
Schalten der Outputs mit AJAX	117
Mit AJAX alle IOs anzeigen und alle Outputs steuern	119
Webseiten mit einheitlichem Design	123
HTML-Objekte über Style-Attribute formatieren	125
Styles zentral verwalten - CSS-Stylesheets	127
Auch bei größeren Projekten Einheitlichkeit mit CSS	129
Optimierte Webseiten für mobile Anzeigegeräte	133
Klimadaten auf dem iPhone anzeigen	140
Analoge Signale (Strom/Spannung) im iPhone Anzeigen.	144
Outputspannung mit dem iPhone setzen	145
Graphische Elemente über Web-IO Status ändern	150
Einen Output grafisch visualisieren und schalten	154
Inputs und Outputs grafisch anzeigen	156
Mehre IOs grafisch visualisieren	158
HTML-Elemente automatisch mit JavaScript erzeugen	163
Analoge Signale grafisch darstellen	165
Eine Webseite für mehrere Web-IO	172
PHP-Script als IP-Kommando-Gateway	173
Zustände mehrerer Geräte grafisch darstellen	183
Gescannte Barcodes im Browser anzeigen	196
Datenaustausch mit seriellen Geräten über den Browser	201
Anhang zur Web-IO Programmierung	204
Character Sets	204
HTML-Tags - die Wichtigsten Tags im Überblick	205
Style-Informationen - Unterschiede CSS und JavaScript	211
Darstellung von Sonderzeichen innerhalb von HTML	214

Socket-Programmierung allgemein

TCP – Verbindungsbezogene Datenkommunikation

Bei der Programmierung von Netzwerkanwendungen unterscheidet man zwischen verbindungsbezogener Datenübertragung und verbindungsloser Datenübertragung.

TCP steht für *Transport Control Protocol*. Um Daten austauschen zu können, muss zunächst eine Verbindung zwischen zwei Kommunikationspartnern aufgebaut werden.

Client und Server

Den Kommunikationspartner, der die Verbindung aufbaut, bezeichnet man als Client. Der Partner der die Verbindung annimmt, wird als Server bezeichnet. Man spricht beim Programmieren auch von Client-Anwendung oder Server-Anwendung.

Verbindung und Datenaustausch

Der Ablauf einer TCP-Kommunikation ähnelt dem eines Telefongespräches. Der Client übernimmt die Rolle des Anrufers und der Server die Rolle des Angerufenen.

Ein Telefon muss eingeschaltet bzw. angesteckt sein, um Verbindungen annehmen zu können. Bei einer TCP-Verbindung gibt es analog dazu die Methode *Listen*, die serverseitig gestartet werden muss und mit deren Aufruf gleichzeitig festgelegt wird, auf welchem TCP-Port der Server lauscht.

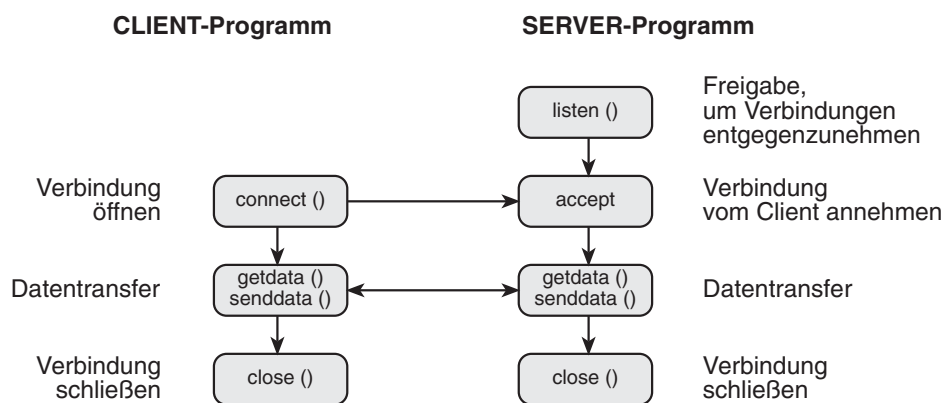
Anstelle von Rufnummern wird bei TCP mit IP Endpoints gearbeitet. Ein IP Endpoint setzt sich aus der IP-Adresse und dem TCP-Port zusammen. Dabei entspricht die IP-Adresse der grundsätzlichen Rufnummer und der TCP-Port der Nebenstelle im Gebäude.

Klingelt das Telefon, nimmt der angerufene Teilnehmer durch Abheben das Gespräch an.

Eine TCP-Verbindung wird vom Server durch Aufruf der Accept-Methode angenommen.

Über eine bestehende Telefonverbindung können beide Teilnehmer gleichberechtigt sprechen. Auch wenn eine TCP-Verbindung zustande gekommen ist, können Client und Server über die Methoden *Send* und *Receive* uneingeschränkt Daten über diese Verbindung austauschen.

Legt der Anrufer oder der Angerufene auf, ist das Gespräch beendet. Client oder Server können die TCP-Verbindung durch Aufruf der Disconnect Methode beenden.



Wie bereits angedeutet, stellen die gängigen Programmiersprachen Methoden zu Verfügung, die vom Programm aufgerufen werden können, um die nötigen Verbindungsschritte abzuwickeln.

Um den Transport der Daten kümmert sich im Hintergrund der TCP/IP-Stack. Bei Verwendung des TCP-Protokolls stellt der Stack sicher, dass die Daten ordnungsgemäß beim Kommunikationspartner ankommen. Kommt es zu Problemen löst der Stack Laufzeitfehler aus, auf die das Programm applikationsabhängig reagieren kann.

Single-Client und Multi-Client Server-Anwendungen

Eine Server-Anwendung stellt in aller Regel einen Dienst oder Service zur Verfügung, der vom Client genutzt wird.

Je nachdem, welcher Dienst angeboten wird, kann es sinnvoll sein, mehreren Clients oder nur einem die Verbindung auf den Server zu erlauben.

Analog zum Telefonieren kann man sich das so vorstellen, wie den Anschluss einer Servicehotline wo mehrere Sachbearbeiter unter der gleichen Rufnummer zur Verfügung stehen. Der Anrufer wird immer zum nächsten freien Sachbearbeiter durchgestellt.

Bei einem normalen Telefonanschluss hingegen, bekommt der Anrufer ein Besetztzeichen, wenn der Anschluss belegt ist.

Klassisches Beispiel für Multi-Client fähige Server-Anwendungen sind Webserver. Die Dienste von Google zum Beispiel werden mit Sicherheit von mehreren tausend Anwendern gleichzeitig benutzt.

Andere Server-Anwendungen lassen ganz bewusst nur eine Verbindung zu einer Zeit zu. Wird zum Beispiel die Programmierung einer CNC-Maschine über einen Com-Server abgewickelt, hätte es fatale Folgen, wenn gleichzeitig über zwei parallele Verbindungen versucht würde ein Programm einzuspielen.

Die Anforderungen an eine Server-Anwendung können also ganz unterschiedlich sein und sollten bei jedem Projekt ermittelt werden, bevor die eigentliche Programmierung beginnt.

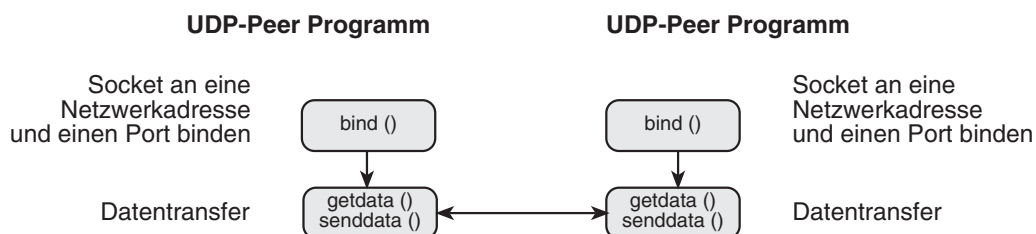
UDP – Verbindungslose Datenkommunikation

UDP steht für *User Datagram Protocol*. Damit zwei Kommunikationspartner Daten austauschen können, ist anders als bei TCP keine feste Verbindung nötig. Statt dessen werden einzelne, im Prinzip unabhängige Datensendungen ausgetauscht. Um solche Datagramme versenden zu können, muss programmtechnisch zunächst ein UDP-Socket geöffnet werden. Um zu bestimmen, wohin die Datagramme gesendet werden sollen, wird ein IP Endpoint, bestehend aus IP-Adresse und Portnummer des Empfängers, an den Socket gebunden. Zusätzlich wird bestimmt, auf welchem Port die eigene Anwendung Datagramme entgegennehmen soll.

Die Kommunikation über UDP ist am ehesten mit Sprechfunk zu vergleichen. Beim Polizeifunk zum Beispiel muss zunächst der Kanal festgelegt werden, auf dem gesendet wird – vergleichbar mit dem Port. Das Funkgespräch selbst beginnt mit der Ansage von Adressinformationen: „Möwe 3 für Möwe15“. Das Festlegen des Kanals und die Adressansage entspricht im Vergleich mit UDP dem Binden eines Sockets.

Im Anschluss erfolgt der Informationsaustausch durch Senden und Empfangen von Funksprüchen, bzw. bei UDP von Datagrammen.

Wann die Kommunikation beendet ist muss beim Funkverkehr abgesprochen werden. Bei UDP sollte ein Datenaustausch der über einzelne Datensendungen hinausgeht durch ein anwendungsabhängiges Protokoll geregelt werden.



UDP Broadcast

Ein Sonderfall der UDP-Kommunikation ist der Broadcast.

Betrachtet man wieder den Vergleich zum Polizeifunk, entspricht der Broadcast dem Rundruf an alle Einheiten.

Bei einem Broadcast-Socket wird als Zieladresse die Broadcast-Adresse des Netzwerkes angegeben. Die Broadcast-Adresse setzt sich aus der Net-ID und einer Host-ID zusammen, bei der alle Bits den Wert 1 haben (siehe auch TCP/IP-Ethernet bis Web-IO).

Bei Netzwerken mit glatten 8 Bit Grenzen zwischen Net-ID und Host-ID ist das ganz einfach.

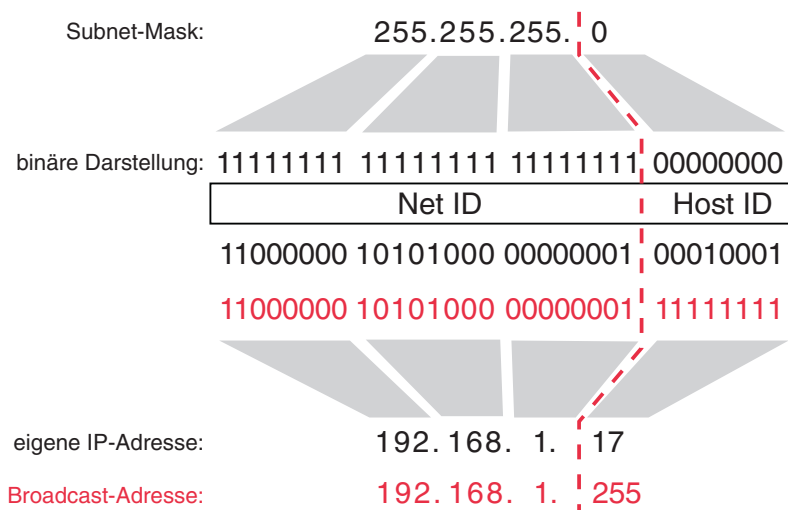
Beispiel: Die IP-Adresse des PC auf dem die Broadcast-Anwendung läuft ist 192.168.1.17, wobei mit der Subnet-Mask 255.255.255.0 gearbeitet wird. Die Grenze zwischen Net-ID und Host-ID liegt in diesem Fall zwischen den ersten 3 Byte und dem letztem Byte.

Net-ID = 192.168.1

Host-ID des PC = 17

Host-ID für Broadcast = 255

Broadcast-IP = 192.168.1.255



Ein UDP-Datagramm an einen IP-Endpoint, bestehend aus dieser Adresse und z.B. Port 8513, kann von allen Netzwerkteilnehmern empfangen werden, die einen Socket geöffnet haben, bei dem der Port 8513 als lokaler Port definiert wurde.

Besonderheiten und Tücken der UDP Kommunikation

Der Vergleich von UDP und Funkverkehr lässt bereits erahnen, dass beim Datenaustausch mit UDP je nach Anwendung einiges zu beachten ist.

Ein Vorteil von UDP ist, dass keine Zeit für einen Verbindungsaufbau benötigt wird. Außerdem sind UDP Sendungen auf dem Netzwerk kürzer, da keine Informationen zur Datensicherung mitübertragen werden. Dadurch ist UDP geringfügig schneller als TCP.

Da beim Funk keine Rufnummer gewählt werden muss, ist auch hier ein schnellerer Datenaustausch möglich.

Durch die fehlende Datensicherung gibt es allerdings keine Rückmeldung darüber, ob ein UDP-Datagramm beim Empfänger angekommen ist. Der Datenversender bemerkt nicht, wenn der Empfänger nicht bereit ist Daten entgegenzunehmen. Wird ein Funkspruch abgesetzt, ist nicht sichergestellt das ihn jemand gehört hat. Ist das Funkgerät des Empfängers abgeschaltet, merkt der Sender das gar nicht.

Ist ein UDP-Socket zum Empfang auf einem bestimmten Port geöffnet, kann jeder beliebige Netzwerkteilnehmer Daten an den offenen Port senden. Beim Funken kann jeder, der auf dem gleichen Kanal sendet einfach „dazwischen quatschen“.

Ein Anwendungsprogramm, das den Datenaustausch mittels UDP abwickelt, sollte Mechanismen beinhalten, welche mit den beschriebenen Besonderheiten umgehen können.

Browser-gestützte Anwendungen

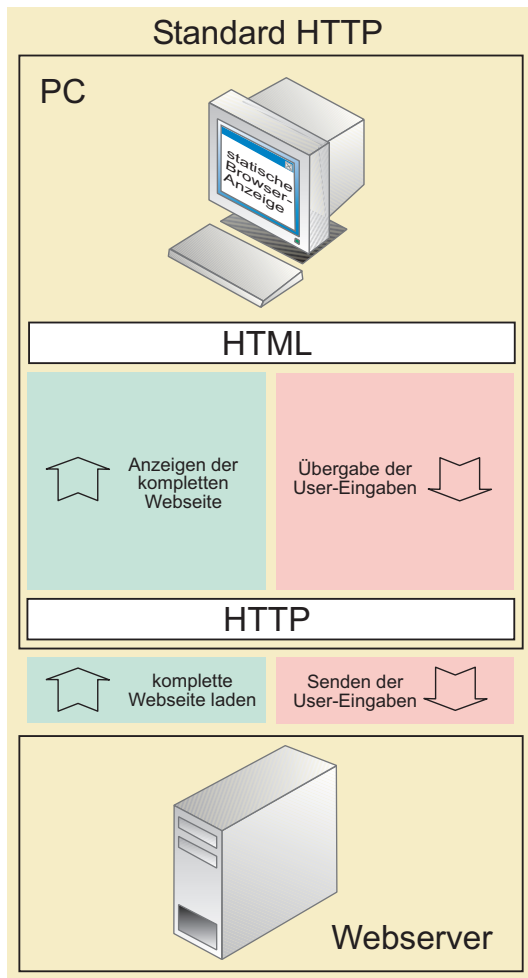
Seit Mitte der neunziger Jahre gehört der Browser bei allen gängigen Betriebssystemen zur Standardausstattung und ist längst nicht mehr nur Hilfsmittel zur Anzeige statischer Webseiten.

Moderne Browser, wie der Internet Explorer, Firefox, Netscape, Opera oder Safari sind viel mehr universelle Visualisierungstools für Inhalte aller Art. Die Unterstützung von Java-Applets, Multimedia-Plugins und verschiedenen Scriptsprachen erlaubt heute die Darstellung dynamischer, grafischer Anwendungen.

Statische Webseiten

Die ersten Versionen der Browser waren ausschließlich für die Anzeige von statischen Webseiten ausgelegt. Webseiten im ursprünglichen Sinn, sind in HTML (Hypertext Markup Language) verfasste Dateien. HTML ist keine Programmiersprache, sondern eine reine Formatbeschreibung. Im HTML-Code wird festgelegt, wie die einzelnen Elemente einer Webseite angezeigt werden sollen (Schriftgröße, Farbe, Ausrichtung,).

Den Aufruf einer rein in HTML aufgebauten Webseite kann man mit dem Kauf einer Zeitung vergleichen. Die Inhalte sind eine Momentaufnahme, vom Zeitpunkt des Aufrufs. Möchte man in der Zeitung neue Inhalte sehen, muss man eine neuere Ausgabe kaufen. So ist es auch bei HTML-Webseiten. Aktualisierte Inhalte bekommt man nur durch einen erneuten Aufruf der Webseite.



Webseiten mit multimedialen Inhalten

Wie bereits gesagt, ist HTML eine Beschreibungssprache, in der festgelegt wird, wie und wo auf der Webseite Elemente dargestellt werden. HTML erlaubt aber auch Fotos, Bilder im allgemeinen und andere Multimediale Inhalte wie Filme und animierte Grafiken darzustellen. Diese Elemente sind allerdings nicht Bestandteil des HTML-Codes sondern werden als separate Dateien dazugeladen.

Auch wenn eine Webseite auf diese Weise mit etwas Leben gefüllt wird, bleibt die Anzeige doch statisch, da bei jedem Aufruf das gleiche angezeigt wird.

Interaktion durch Formulare

Statische, in HTML verfasste Webseiten, können mit Eingabefeldern versehen werden. Der Anwender kann in den so aufgebauten Formularen Daten eingeben und mittels eines Submit-Buttons an den Webserver übergeben.

Damit das funktioniert, muss allerdings auf dem Server eine Anwendung laufen, welche die Daten entgegennimmt und irgendwie weiterverarbeitet.

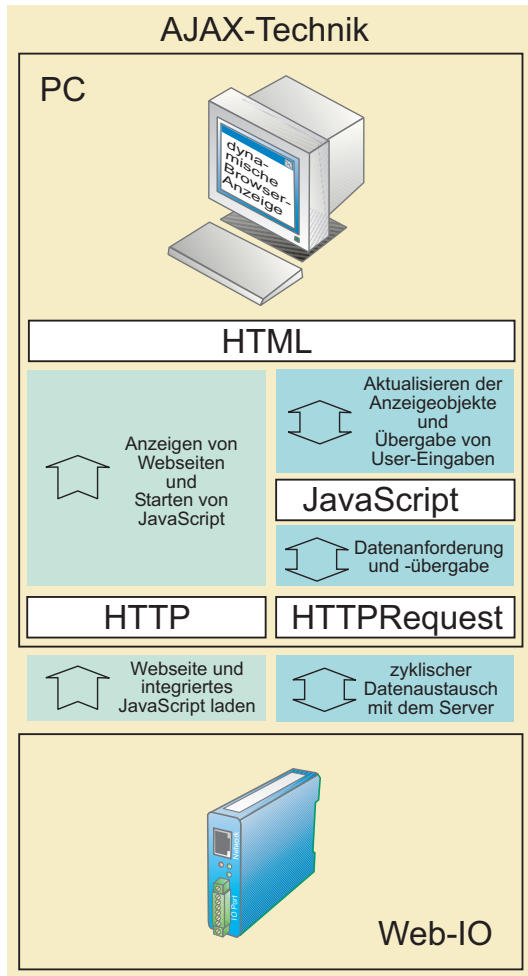
Ferner löst der Klick auf den Submit-Button bei Standard HTML-Seiten das Laden einer neuen Webseite aus, um z.B. den Empfang der übergebenen Daten zu bestätigen.

Der Nachteil dieser Technik liegt darin, dass bei jeder Aktion eine neue Webseite in den Browser geladen werden muss.

AJAX - Echte Interaktion durch JavaScript und HTTP-Request-Technik

Komplexere Anwendungen sollten mit dynamischen Anzeigelementen arbeiten. Gerade bei technischen Applikationen ist es sinnvoll, dass angezeigte Objekte sich selbstständig aktualisieren, ohne dass der Anwender eingreifen muss.

AJAX steht für *Asynchronous JavaScript and XML*. Auch wenn sich AJAX erst seit kurzem einer hohen Popularität erfreut, ist die verwendete Technik nicht neu. JavaScript kann auch nach Aufruf der Webseite über http-Requests eine erneute Verbindung zum Server aufbauen. Abhängig von den so übergebenen Daten können die Inhalte und Eigenschaften von HTML-Elementen nachträglich verändert werden.



Durch zyklische http-Requests kann so ein Polling realisiert werden, um z.B. in einem festen Intervall Messwerte abzufragen und anzuzeigen.

Die Kommunikation wird bei dieser Technik allerdings immer von Browserseite aus angestoßen und kann ausschließlich über das http-Format oder über den Austausch von XML Dateien erfolgen. Ereignisgesteuerte Anwendungen sind deshalb mit AJAX nicht bzw. nur in festen Intervallen möglich.

Java-Applets – Formatunabhängige Kommunikation

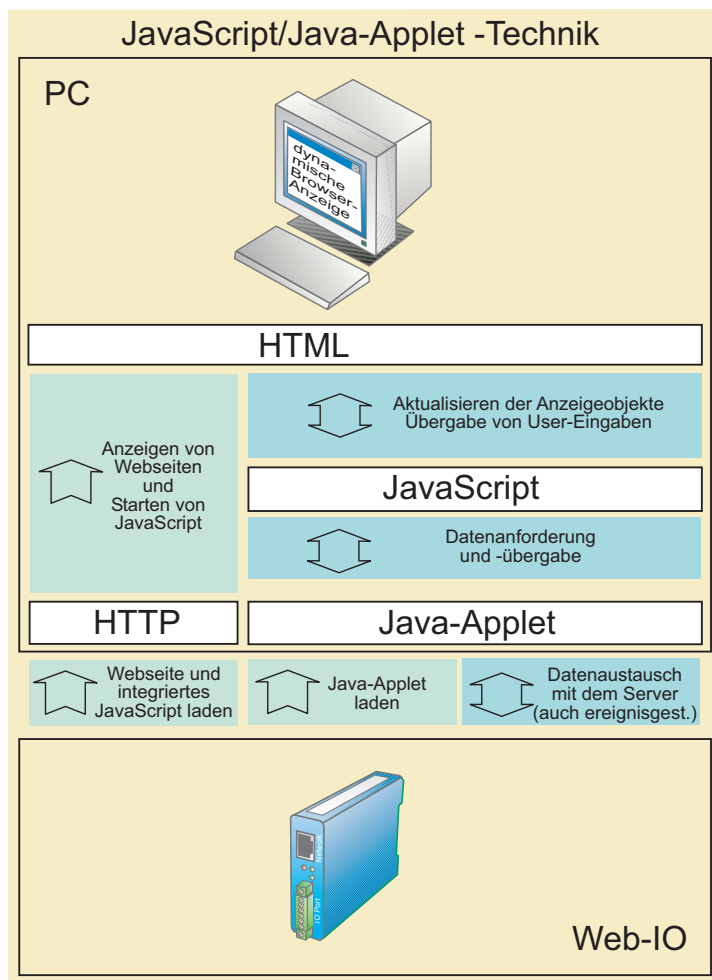
Java-Applets sind eigenständige Programme, die mit Laden der Webseite ebenfalls in den Browser geladen und dort ausgeführt werden. Java-Applet und JavaScript können über festgelegte Methoden Daten miteinander austauschen.

Um mit Java-Applets arbeiten zu können, muss im Browser die Java Unterstützung aktiviert sein. Bei den gängigen Browsern ist das von Hause aus der Fall.

Java-Applets können über normale TCP- oder UDP-Sockets kommunizieren. Damit ist der Datenaustausch, anders als bei AJAX, an kein bestimmtes Format gebunden. Es kann z.B. mit einer dauerhaften TCP-Verbindung gearbeitet werden, so dass von Serverseite Informationen an den Browser gesendet werden können, ohne dass diese explizit angefordert werden müssen.

Anwendungen die mit Java-Applets realisiert werden, können auch ereignisgesteuert arbeiten.

Wenn das Applet keine eigene Bedienoberfläche mitbringt, wird die Aktualisierung der Anzeige im Browser genauso über JavaScript abgewickelt wie bei AJAX.



TCP-Sockets, UDP-Sockets oder Browseranwendung

Alle drei Varianten haben eines gemeinsam – das Netzwerk wird als Kommunikationsweg benutzt. Dieses Buch soll dazu ermutigen, Netzerkanwendungen zu programmieren.

Welche Art der Kommunikation für die eigene Anwendung gewählt wird, hängt von vielen Faktoren ab und sollte zunächst gut geplant werden.

Wenn der Kommunikationspartner bereits vorhanden ist, wird einem die Entscheidung meist abgenommen. Bietet ein Endgerät als Netzwerkzugang z.B. einen TCP-Server an, ergibt sich zwangsläufig, dass die eigene Anwendung als TCP-Client arbeitet.

Genauer abwägen muss man dann, wenn das anzusprechende Endgerät mehrere Kommunikationsmöglichkeiten anbietet oder ohnehin beide Seiten der Kommunikation programmiert werden sollen.

TCP/UDP - Socketprogrammierung in Hochsprachen

Vor allem auf Windows Systemen bringen die diversen Programmiersprachen mächtige Entwicklungstools mit. Der Programmierer kann mittels *Drag and Drop* mit der Maus grafische Oberflächen zusammenschieben. Bestimmte Syntaxelemente werden farblich gekennzeichnet oder hervorgehoben. Zum Teil schlägt der Quelltexteditor dem Anwendungsentwickler schon beim Tippen in Frage kommende Befehle und Objekte vor.

Das Betriebssystem bietet dem Programmierer zudem den Zugriff auf API-Schnittstellen (Applikation Interface) bzw. ein Framework von benutzbaren Systemfunktionen. Solche Funktionen können mit Übergabe der nötigen Parameter vom Programm aufgerufen werden, und erledigen im Hintergrund die Arbeit, die früher in individuellen eigenen Funktionen verrichtet wurde.

Mit Entwicklungstools wie z.B. Delphi oder Visual Basic und sogar in C++ lassen sich so innerhalb weniger Minuten funktionsfähige Anwendungen programmieren. Im weiteren Verlauf werden Sie entsprechende Beispiele finden.

Microsoft und Borland bieten Express bzw. Turbovarianten ihrer Profientwicklungsumgebungen kostenlos zum Download an. Für kleinere Projekte reichen diese abgespeckten Varianten aber oft völlig aus.

TCP/UDP - Netzwerkkommunikation allgemein

Um die Abläufe einer Netzwerkkommunikation zu erklären, wird oft das OSI-Schichten Model benutzt. Die einzelnen Teilfunktionen einer Datenübertragung werden als Schichten dargestellt - angefangen bei der physikalischen Bitübertragung bis hin zur Anwendungsebene.

7	Anwendungsschicht Datenverarbeitung	Eigene Anwendung aber auch HTTP, FTP, SMTP,
6	Darstellungsschicht Festlegung des Datenformates	
5	Sitzungsschicht Verbindungskontrolle	
4	Transportschicht Ende zu Ende Kommunikation inkl. Fehlersicherung	TCP oder UDP
3	Vermittlungsschicht Switching und Routing	IP
2	Sicherungsschicht Checksummenbildung	Ethernet
1	Bitübertragungsschicht physikalische Übertragung	

Im Rahmen der Anwendungsprogrammierung basierend auf modernen Betriebssystemen wie Windows oder Linux reduzieren wir die Abwicklung auf zwei Ebenen:

- Dinge, um die man sich in der eigenen Anwendung kümmern muss
- Dinge, die einem von Betriebssystem, TCP/IP-Stack und Treibern abgenommen werden

TCP – Gesicherte Kommunikation

TCP bietet in der Beziehung optimale Voraussetzungen. Ist die Verbindung erst einmal aufgebaut, muss die Anwendung die zu transportierenden Daten nur noch an den TCP/IP-Stack übergeben und sich nicht weiter darum kümmern ob die Daten sicher ihren Weg gefunden haben. Im Fehlerfall bekommt die Anwendung vom Stack eine Meldung und muss erst dann reagieren.

Geht es also um eine sichere Datenübertragung, ist TCP das Protokoll der Wahl.

Wenn nur kleine Datenmengen mit wechselnden Kommunikationspartnern ausgetauscht werden sollen, muss berücksichtigt werden, dass der Verbindungsauf- und abbau einige Millisekunden Zeit in Anspruch nimmt. Das wirkt sich negativ auf die Übertragungsgeschwindigkeit aus. Bei der Übertragung größerer Datenmengen oder dann wenn TCP-Verbindungen länger andauern, fällt das allerdings nicht merklich ins Gewicht.

TCP-Client oder TCP-Server

TCP-Client und TCP-Server gehören natürlich zusammen, denn einer kann ohne den anderen nicht arbeiten.

Der Client ist immer der Part, der die Initiative ergreift und quasi die Fäden in der Hand hat. Der Server hingegen verharrt sozusagen in Ruhestellung und wartet darauf, dass ein Client Verbindung zu ihm aufnimmt.

Welchen Part die eigene Anwendung übernehmen soll, hängt also davon ab, wozu sie die Netzwerkverbindung nutzen soll.

Soll beispielsweise der Output eines Web-IO Digital von einem eigenen Programm geschaltet werden, sollte diese Anwendung als Client arbeiten, denn hier geht die Initiative eindeutig vom Programm aus.

Eine Anwendung, die hingegen Alarm-Meldungen von einem Web-Thermographen entgegen nehmen soll, muss sinnvollerweise als Server programmiert werden.



Zusammengefasst ist TCP das richtige Protokoll für alle Anwendungen, bei denen kontinuierlich Daten ausgetauscht werden oder größere Datenmengen übertragen werden oder die Datensicherheit eine große Rolle spielt.

UDP - Verbindungslos, flexibel und schnell

UDP arbeitet verbindungslos. Gerade dann, wenn es darum geht, kleine Datenmengen mit wechselnden Kommunikationspartnern auszutauschen, erhöht sich gegenüber TCP der Datendurchsatz, da die Zeiten für den jeweiligen Verbindungsaufbau gespart werden.

Wie bereits angedeutet, gibt es bei der Netzwerkkommunikation zwei Ebenen:

- Dinge um die man sich in der eigenen Anwendung kümmern muss
- Dinge die einem von Betriebssystem, TCP/IP-Stack und Treibern abgenommen werden

Anders als bei TCP findet bei UDP keinerlei Datensicherung durch das Betriebssystem bzw. den TCP/IP-Stack statt. Das bringt bei der Kommunikation einen weiteren Geschwindigkeitsvorteil. Allerdings muss bei Bedarf innerhalb der eigenen Anwendung Vorsorge getroffen werden, um Datenverlust zu bemerken und zu korrigieren.

Sollen Daten gleichzeitig an mehrere Netzteilnehmer gesendet werden, ist das ausschließlich mit UDP möglich. Es gilt aber zu beachten, dass solche Broadcasts nur innerhalb eines lokalen Subnetzes empfangbar sind.



Zusammengefasst ist UDP das richtige Protokoll für alle Anwendungen, bei denen kleinere Datenmengen mit wechselnden Kommunikationspartnern ausgetauscht werden. Auch der Datenversand an mehrere Empfänger ist nur mit UDP-Broadcasts möglich. Für den Austausch wichtiger Daten sollte die Anwendung, z.B. durch Quittierung sicherstellen, dass die Daten beim Gegenüber korrekt angekommen sind.

Browseranwendungen - Betriebssystemunabhängigkeit

Durch den ungeheuren Internetboom der letzten Jahre ist der Umgang mit dem Browser heute Standard. Onlinebanking, Reisebuchung, Bücherbestellung oder Routenplanung gehören heute zu den ganz selbstverständlichen Web-Anwendungen.

Wen wundert es da, dass der Browser auch bei technischen Anwendungen immer häufiger als Plattform dient. Die aktuellen Webtechniken bieten auch für technische Visualisierungs- und Steuerungsaufgaben tolle Möglichkeiten.

Allerdings müssen bei der Erstellung von professionellen Web-Anwendungen einige Dinge beachtet werden:

Die verschiedenen Browser wie Internet Explorer, Firefox, Opera oder Safari verhalten sich zwar im Großen und Ganzen gleich; in einigen Details gibt es aber Unterschiede, die beim Erstellen einer Anwendung berücksichtigt werden müssen.

Die Anwendungsgeschwindigkeit variiert je nach Betriebssystem und verwendetem Browser zum Teil erheblich.

Webanwendungen werden nicht in einer Programmiersprache erstellt, sondern bedienen sich einer Kombination verschiedener Programmier-techniken. Das fängt mit HTML für die Gestaltung der reinen Grunddarstellung an, geht mit JavaScript zur Handhabung dynamischer Prozesse weiter und endet ggf. mit Java-Applets oder PHP-Anteilen. Bedingt durch diese Kombination, gibt es auch nicht nur *ein* Entwicklungstool für Web-Anwendungen. Stattdessen wird mit verschiedenen Editoren und Tools gearbeitet, die sich in Ihrer Funktionalität ergänzen.

Dennoch wollen wir ausdrücklich dazu ermutigen, Anwendungen für den Browser zu erstellen, wenn Aufgabenstellung und Möglichkeiten zusammenpassen.

Wie bereits angedeutet: Anwendungen im Browser haben auch sehr viele Vorteile.

Wenn bei der Umsetzung der Web-Anwendung die richtigen Techniken eingesetzt werden, wird auf dem ausführenden PC

nur ein Browser benötigt. Alles andere wird bei Bedarf vom Server nachgeladen. Es ist also keine explizite Installation nötig.

Damit kann die Anwendung auf jedem beliebigen PC, der Zugang zum Server hat, sofort gestartet werden.

Auch das verwendete Betriebssystem ist unerheblich, solange ein Browser verfügbar ist. Selbst auf modernen Mobiltelefonen sind Web-Anwendungen ausführbar.



Zusammengefasst ist der Browser die richtige Plattform für Anwendungen die ohne großen Installations- und Einrichtungsaufwand an verschiedenen Orten und auf verschiedenen Systemen eingesetzt werden sollen und bei denen sich der Anspruch an Arbeits- und Reaktionsgeschwindigkeit in Grenzen hält. Für Echtzeitanwendungen, bei denen Reaktionszeiten von wenigen ms gefordert sind, ist der Browser nicht das richtige Medium.

Netzwerkanalyse

Unabhängig davon, für welche Programmiersprache man sich zur Erstellung der eigenen Socketanwendung entschließt - mit zunehmender Komplexität kommt man irgendwann an den Punkt, wo irgendetwas nicht so funktioniert wie geplant.

Dieses Kapitel beschreibt den Umgang mit dem kostenlosen Netzwerkanalyse-Tool Wireshark, mit dem der tatsächliche Datenaustausch auf dem Netzwerk aufgezeichnet und ausgewertet werden kann.

(Alle die lieber gleich zum Programmieren übergehen möchten, können dieses Kapitel zunächst überspringen.)

Netzwerkanalyse hilft bei der Socketprogrammierung

Aufbau und Syntax der eigenen Anwendung lassen sich innerhalb der Entwicklungsumgebung bei allen aktuellen Hochsprachen mittels integrierter Debugging-Tools überprüfen.

Um auch solche Probleme schnell aufzuspüren, die die individuelle Netzwerkkommunikation betreffen, empfehlen wir das Netzwerkanalyse-Tool *Wireshark*.

Netzwerkanalyse - das klingt im ersten Moment kompliziert - ist mit Wireshark aber auch für den „Nichtprofi“ schnell zu erlernen.

Um Routine im Umgang mit Wireshark zu bekommen, werden wir bei allen Programmierbeispielen begleitend die entsprechenden Netzwerkmitschnitte betrachten und erklären.



Wireshark ist ein Opensource-Projekt und steht für Windows und viele andere Betriebssysteme unter www.wireshark.org kostenlos zum Download zur Verfügung.

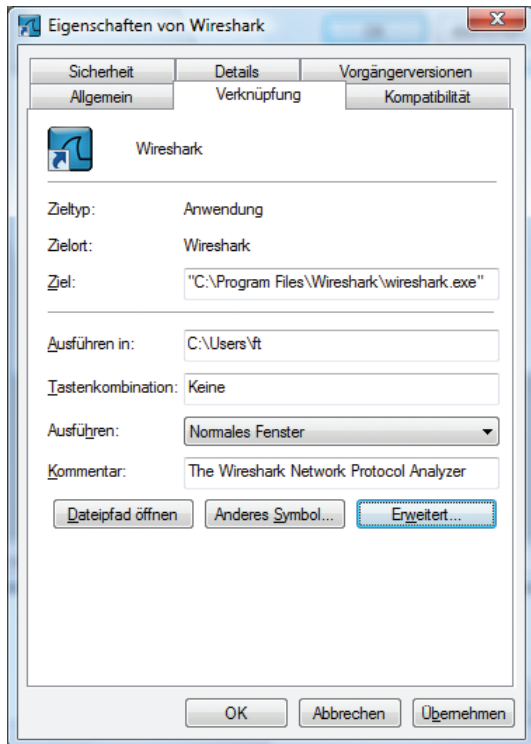
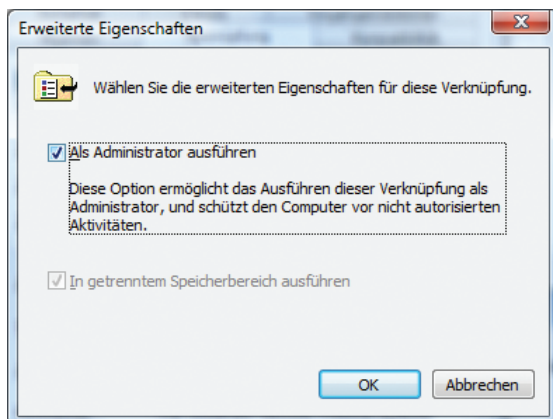
Die Installation von Wireshark ist einfach und unproblematisch. Folgen Sie einfach dem Installationsmenü. Im Zuge der Installation unter Windows wird auch der WinPcap-Treiber installiert.



Um Wireshark auf PCs mit Windows Vista in vollem Funktionsumfang nutzen zu können, muss das Programm mit Administratorrechten gestartet werden.

Um das bei jedem Start von Wireshark sicherzustellen, klicken Sie mit der rechten Maustaste auf das Icon, welches die Verknüpfung mit Wireshark repräsentiert.

Wählen Sie *Eigenschaften* und im Eigenschaftsfenster die Registerkarte *Verknüpfungen*. Dort wählen Sie den Button *Erweitert*.

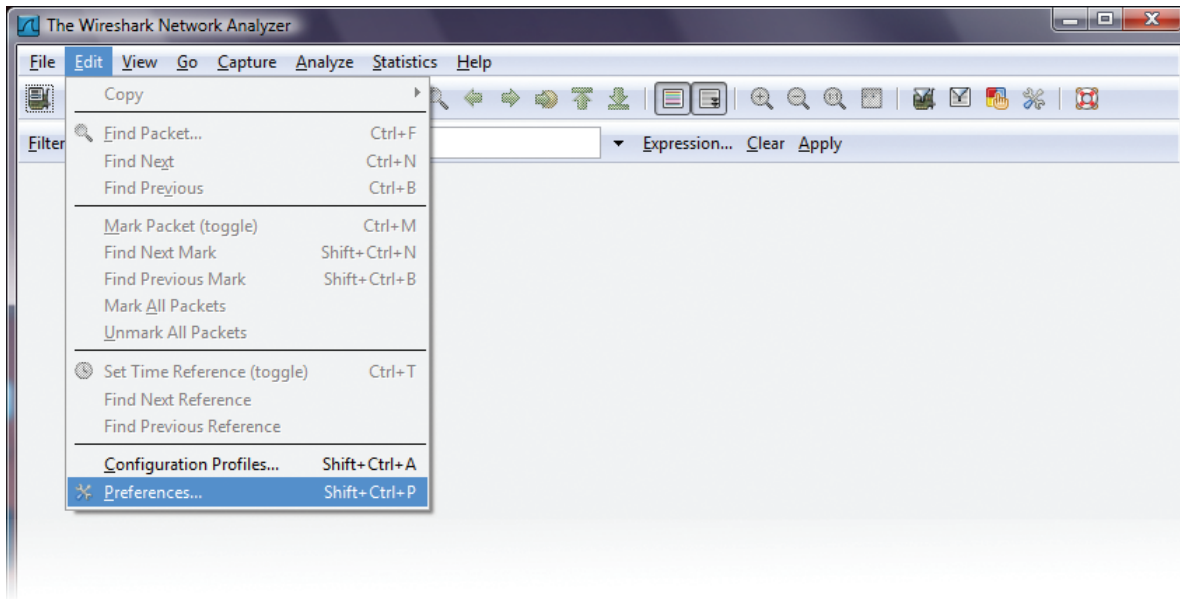


Setzen Sie das Häkchen bei *Als Administrator ausführen*.

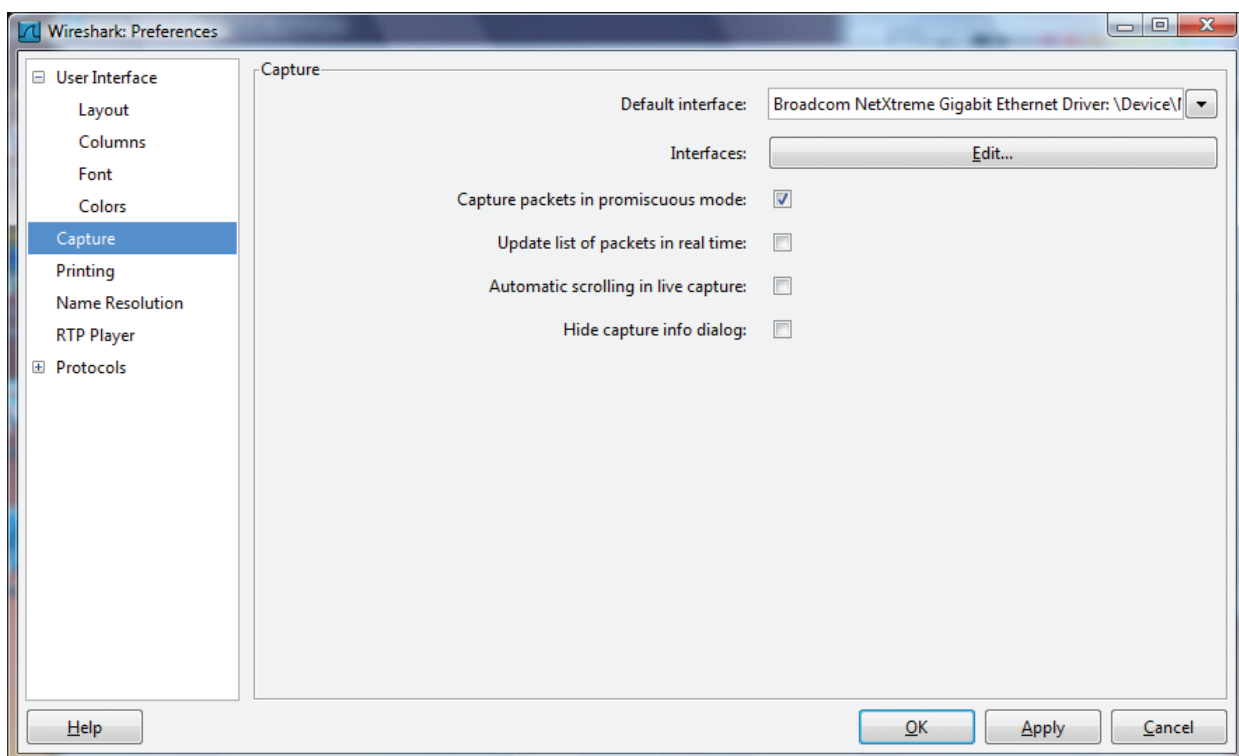
Wireshark - eine kurze Einführung

Nachdem die Installation von Wireshark erfolgreich abgeschlossen wurde, kann der erste Start erfolgen.

Wählen Sie im Hauptmenü von Wireshark unter *Edit* den Punkt *Preferences*.



Es öffnet sich ein Fenster in dem die Grundkonfiguration von Wireshark vorgenommen werden kann.

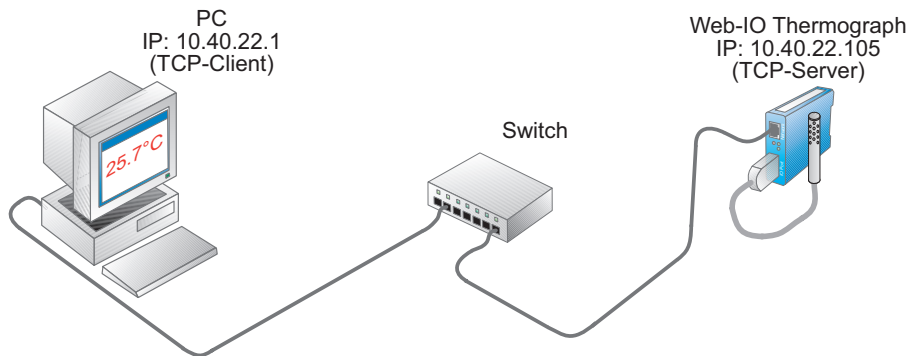


Wählen Sie auf der linken Seite *Capture* und überprüfen Sie, ob auf der rechten Seite unter *Default Interface* die Netzwerkkarte des PC eingetragen ist. Sollte das nicht der Fall sein, wählen Sie das Netzwerk Interface aus der Liste und bestätigen Sie mit OK.

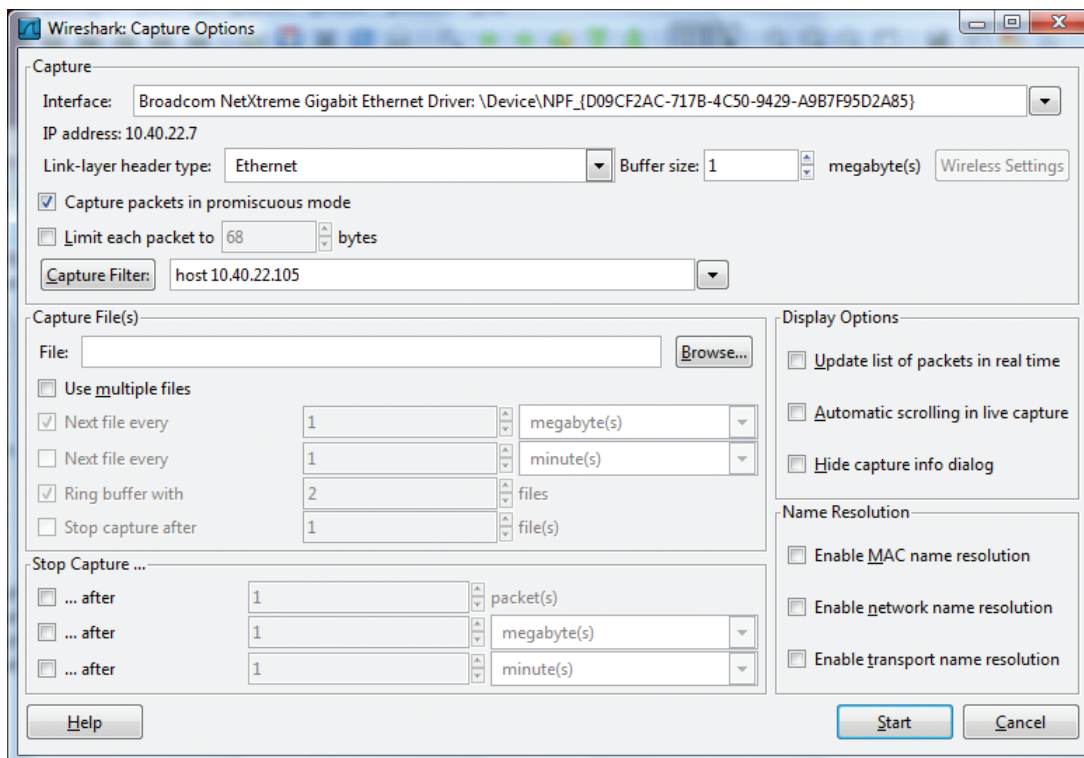
Wireshark ist nun bereit für den ersten Mitschnitt.

TCP-Datenverkehr mitschneiden und auswerten

Für dieses erste Beispiel soll per TCP-Client die aktuelle Temperatur von einem Web-Thermographen abgefragt werden.



Durch Anklicken von  öffnet sich ein Konfigurationsfenster.



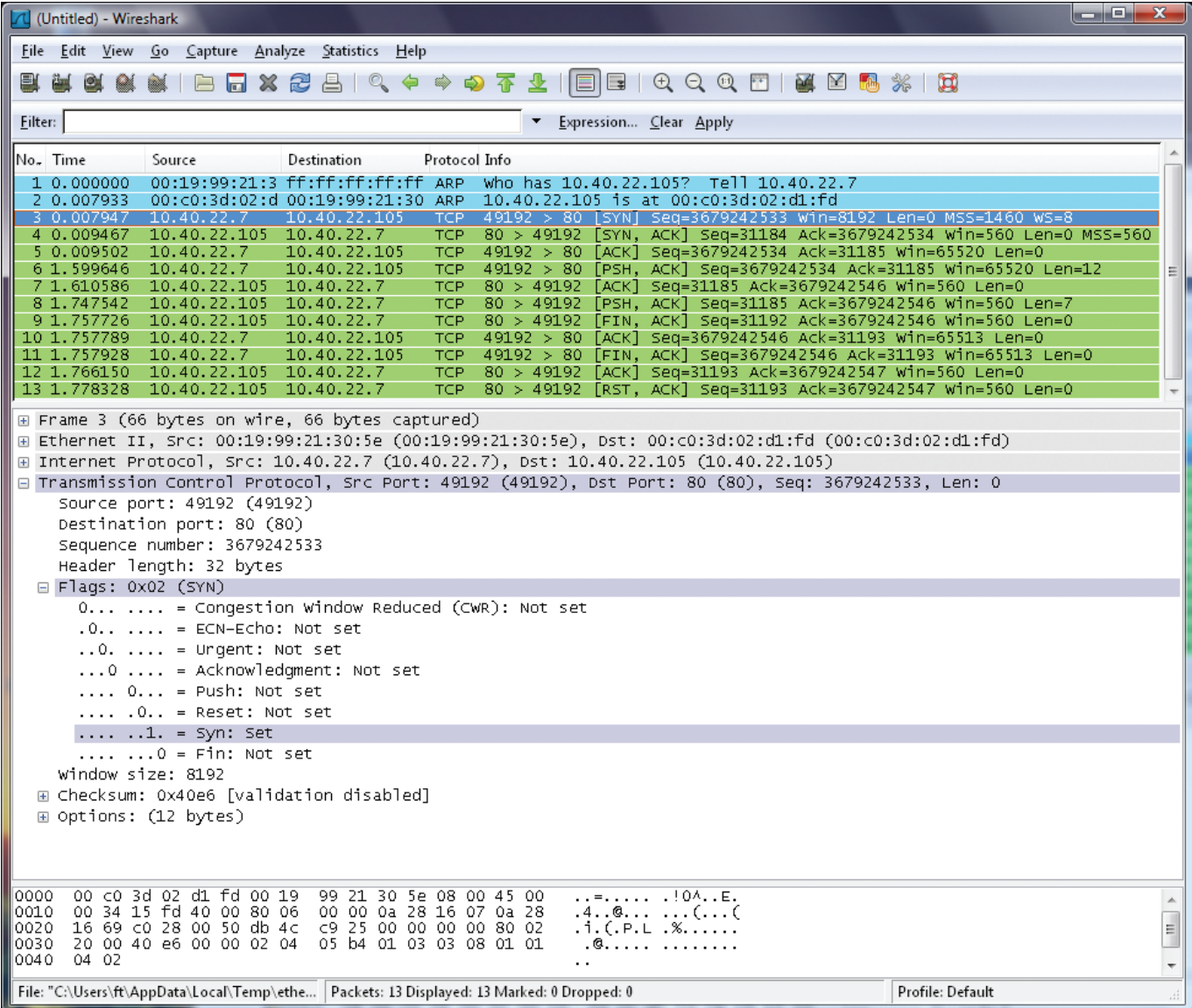
Um ausschließlich den Datenverkehr zwischen dem PC und dem Web-Thermographen mitzuschneiden, wird unter *Capture Filter* als Host die IP-Adresse des Web-Thermographen eingetragen. Durch das Deaktivieren von *Enable MAC name resolution* und *Enable transportname resolution* wird erreicht, dass Wireshark die mitgeschnittenen Daten nicht vor-

interpretiert (z.B. Portnummern durch etwaige Protokollnamen ersetzt). Der Mitschnitt wird dadurch besser lesbar.

Mit *Start* wird der Mitschnitt anschließend gestartet.

Nachdem das Anwendungsprogramm gestartet wurde, es die Temperatur ausgelesen hat und wieder beendet wurde, kann der Mitschnitt durch Klick auf  gestoppt werden.

Die mitgeschnittenen Daten können nun ausgewertet werden.



The screenshot displays the Wireshark interface with the following details:

- Filter:** Expression... Clear Apply
- Packet List:**

No.	Time	Source	Destination	Protocol	Info
1	0.000000	00:19:99:21:30:5e	ff:ff:ff:ff:ff:ff	ARP	who has 10.40.22.105? Tell 10.40.22.7
2	0.007933	00:c0:3d:02:d1:fd	00:19:99:21:30:5e	ARP	10.40.22.105 is at 00:c0:3d:02:d1:fd
3	0.007947	10.40.22.7	10.40.22.105	TCP	49192 > 80 [SYN] Seq=3679242533 win=8192 Len=0 MSS=1460 WS=8
4	0.009467	10.40.22.105	10.40.22.7	TCP	80 > 49192 [SYN, ACK] Seq=31184 Ack=3679242534 win=560 Len=0 MSS=560
5	0.009502	10.40.22.7	10.40.22.105	TCP	49192 > 80 [ACK] Seq=3679242534 Ack=31185 win=65520 Len=0
6	1.599646	10.40.22.7	10.40.22.105	TCP	49192 > 80 [PSH, ACK] Seq=3679242534 Ack=31185 win=65520 Len=12
7	1.610586	10.40.22.105	10.40.22.7	TCP	80 > 49192 [ACK] Seq=31185 Ack=3679242546 win=560 Len=0
8	1.747542	10.40.22.105	10.40.22.7	TCP	80 > 49192 [PSH, ACK] Seq=31185 Ack=3679242546 win=560 Len=7
9	1.757726	10.40.22.105	10.40.22.7	TCP	80 > 49192 [FIN, ACK] Seq=31192 Ack=3679242546 win=560 Len=0
10	1.757789	10.40.22.7	10.40.22.105	TCP	49192 > 80 [ACK] Seq=3679242546 Ack=31193 win=65513 Len=0
11	1.757928	10.40.22.7	10.40.22.105	TCP	49192 > 80 [FIN, ACK] Seq=3679242546 Ack=31193 win=65513 Len=0
12	1.766150	10.40.22.105	10.40.22.7	TCP	80 > 49192 [ACK] Seq=31193 Ack=3679242547 win=560 Len=0
13	1.778328	10.40.22.105	10.40.22.7	TCP	80 > 49192 [RST, ACK] Seq=31193 Ack=3679242547 win=560 Len=0
- Packet Details:**
 - Frame 3 (66 bytes on wire, 66 bytes captured)
 - Ethernet II, Src: 00:19:99:21:30:5e (00:19:99:21:30:5e), Dst: 00:c0:3d:02:d1:fd (00:c0:3d:02:d1:fd)
 - Internet Protocol, Src: 10.40.22.7 (10.40.22.7), Dst: 10.40.22.105 (10.40.22.105)
 - Transmission Control Protocol, Src Port: 49192 (49192), Dst Port: 80 (80), Seq: 3679242533, Len: 0
 - Source port: 49192 (49192)
 - Destination port: 80 (80)
 - Sequence number: 3679242533
 - Header length: 32 bytes
 - Flags: 0x02 (SYN)
 - 0... .. = Congestion window Reduced (CWR): Not set
 - .0.. .. = ECN-Echo: Not set
 - ..0. .. = Urgent: Not set
 - ...0 .. = Acknowledgment: Not set
 - 0... = Push: Not set
 -0.. = Reset: Not set
 -1. = Syn: Set
 -0 = Fin: Not set
 - window size: 8192
 - Checksum: 0x40e6 [validation disabled]
 - options: (12 bytes)
- Packet Bytes:**

Offset	Hex	ASCII
0000	00 c0 3d 02 d1 fd 00 19 99 21 30 5e 08 00 45 00	..=.....!0A..E.
0010	00 34 15 fd 40 00 80 06 00 00 0a 28 16 07 0a 28	.4..@... ..C...C
0020	16 69 c0 28 00 50 db 4c c9 25 00 00 00 00 80 02	.f..C.P.L .%.
0030	20 00 40 e6 00 00 02 04 05 b4 01 03 03 08 01 01	..@.....
0040	04 02	..

Im oberen Teil des Fensters findet man die Liste der mitgeschnittenen Netzwerkpakete.

Die einzelnen Spalten geben bereits einen groben Überblick über Art und Inhalt der Pakete.

Bei den beiden ersten Paketen handelt es sich um ARP-Request und ARP-Reply. Mittels ARP erfragt der TCP/IP-Stack, welche MAC-Adresse der Netzwerkteilnehmer hat, mit dem kommuniziert werden soll, damit die Datenpakete auf physikalischer Ebene adressiert werden können (mehr zu ARP finden Sie im Buch *TCP/IP-Ethernet bis Web-IO*).

Die eigentliche netzwerktechnische Abwicklung der Anwendung beginnt mit Paket Nr. 3.

Im mittleren Bereich des Fensters können die einzelnen Bereiche des Netzwerkpaketes detailliert betrachtet werden.

Die wichtigsten Punkte sind dabei:

Internet Protocol Src

Die IP-Adresse des Paketsenders

Hier 10.40.22.1, also die IP des PC, auf dem die Anwendung läuft.

Internet Protocol Dst

Die IP-Adresse des Empfängers

Hier 10.40.22.105, also die IP-Adresse des Web-Thermographen.

Transmission Control Protocol Src Port

Die lokale Portnummer, auf der die sendende Anwendung Daten empfangen kann, wenn nötig. Bei TCP-Client-Anwendungen wird die lokale Portnummer vom TCP/IP Stack dynamisch vergeben. Das Anwendungsprogramm hat darauf keinen Einfluss.

Transmission Control Protocol Dst Port

Die Portnummer, auf der der Empfänger Daten entgegen nehmen kann.

Beim Web-Thermografen ist das Port 80, also der HTTP-Port.

Sequence number

Eine willkürlich bei Verbindungsaufbau vom Versender generierte Nummer. Sowohl Client als auch Server generieren bei Verbindungsstart eine *Sequence number*.

Acknowledgement number

Mit der *Acknowledgement number* wird der Empfang von TCP-Paketen bzw. TCP-Daten quittiert. Der Sender nimmt dazu die letzte vom Gegenüber empfangene *Sequence number* und setzt deren um 1 erhöhten Wert als *Acknowledgement number* ein. Enthält ein TCP-Paket Nutzdaten wird der Wert nicht um 1 sondern um die Anzahl der empfangenen Nutzdaten erhöht. Eine *Acknowledgement number* gibt es erst ab dem zweiten TCP-Paket einer Verbindung.

Sequence und *Acknowledgement number* werden vom TCP/IP-Stack generiert und verwaltet, damit Datenverlust sofort erkannt wird.

Flags

Das TCP-Protokoll stellt 8 Flags zur Verfügung, mit denen die Funktion des Datenpaketes gekennzeichnet wird. Hier die wichtigsten:

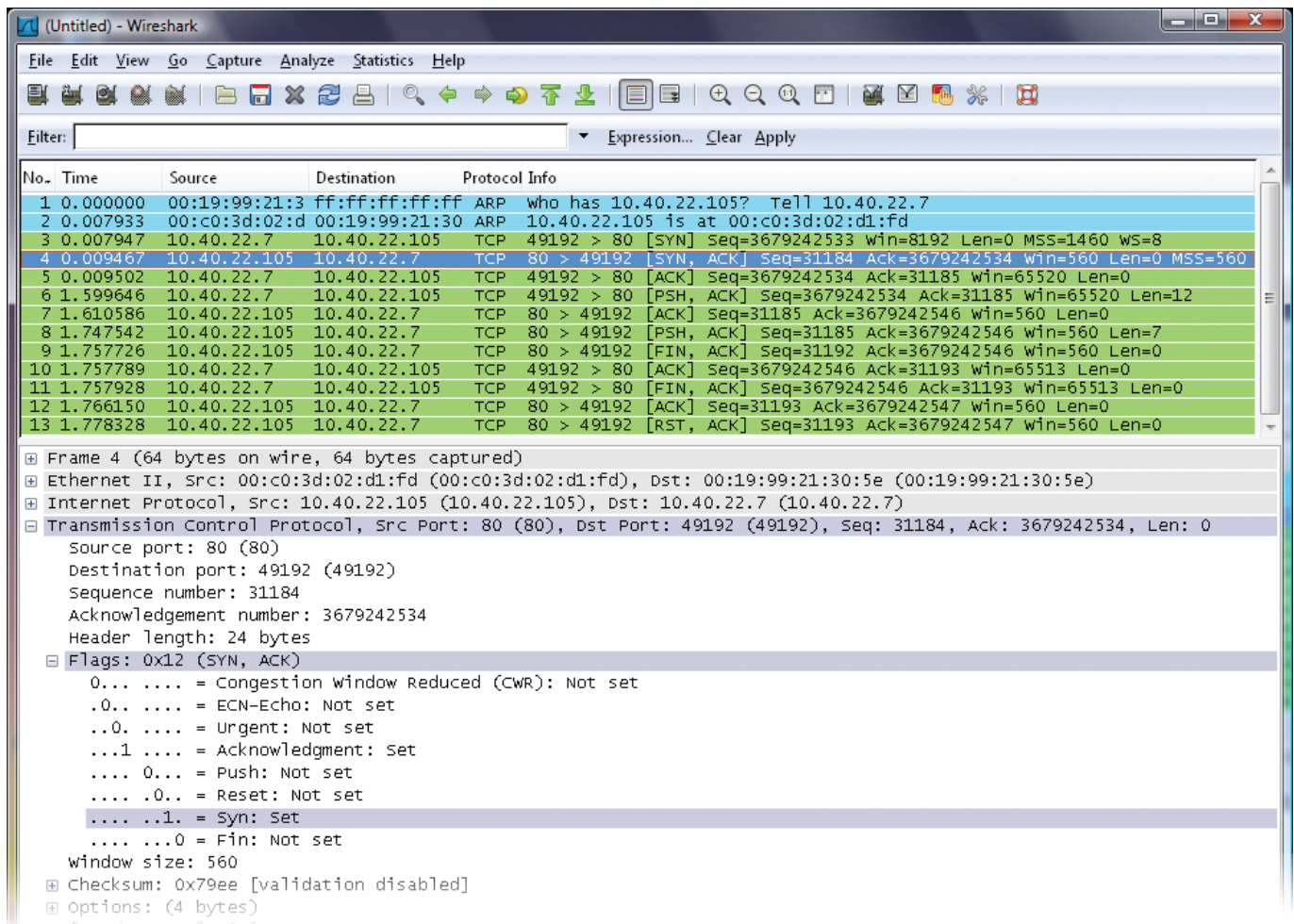
- Syn - Verbindungsaufbau
- Ack - Paket enthält eine Acknowledgement number
- Push - Paket enthält Nutzdaten
- Fin - Verbindung beenden
- Rst - Verbindungs-Reset

Es können mehrere Flags gesetzt werden.

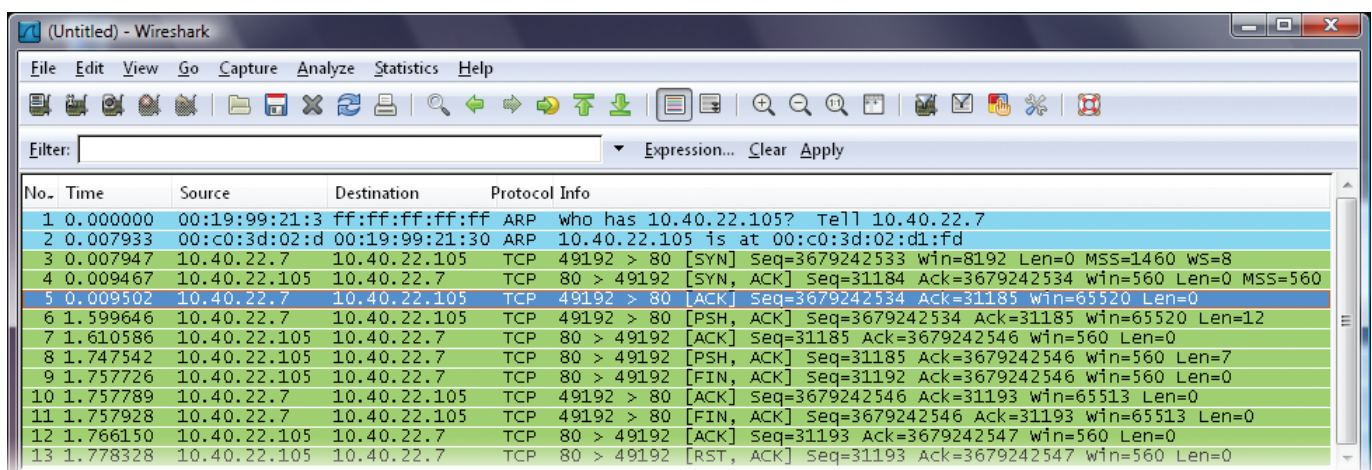
Das *Syn*-Flag zum Beispiel signalisiert, dass es sich um den ersten Teil eines Verbindungsaufbaus handelt.

Paket Nr. 4 zeigt das Akzeptieren der Verbindung durch den Server - hier den Web-Thermographen.

Neben dem *Syn*-Flag ist auch das *Ack*-Flag gesetzt. Das Paket enthält nun auch die *Acknowledgement number*, wobei diese den Wert der *Sequence number* aus Paket Nr. 3 plus 1 enthält.

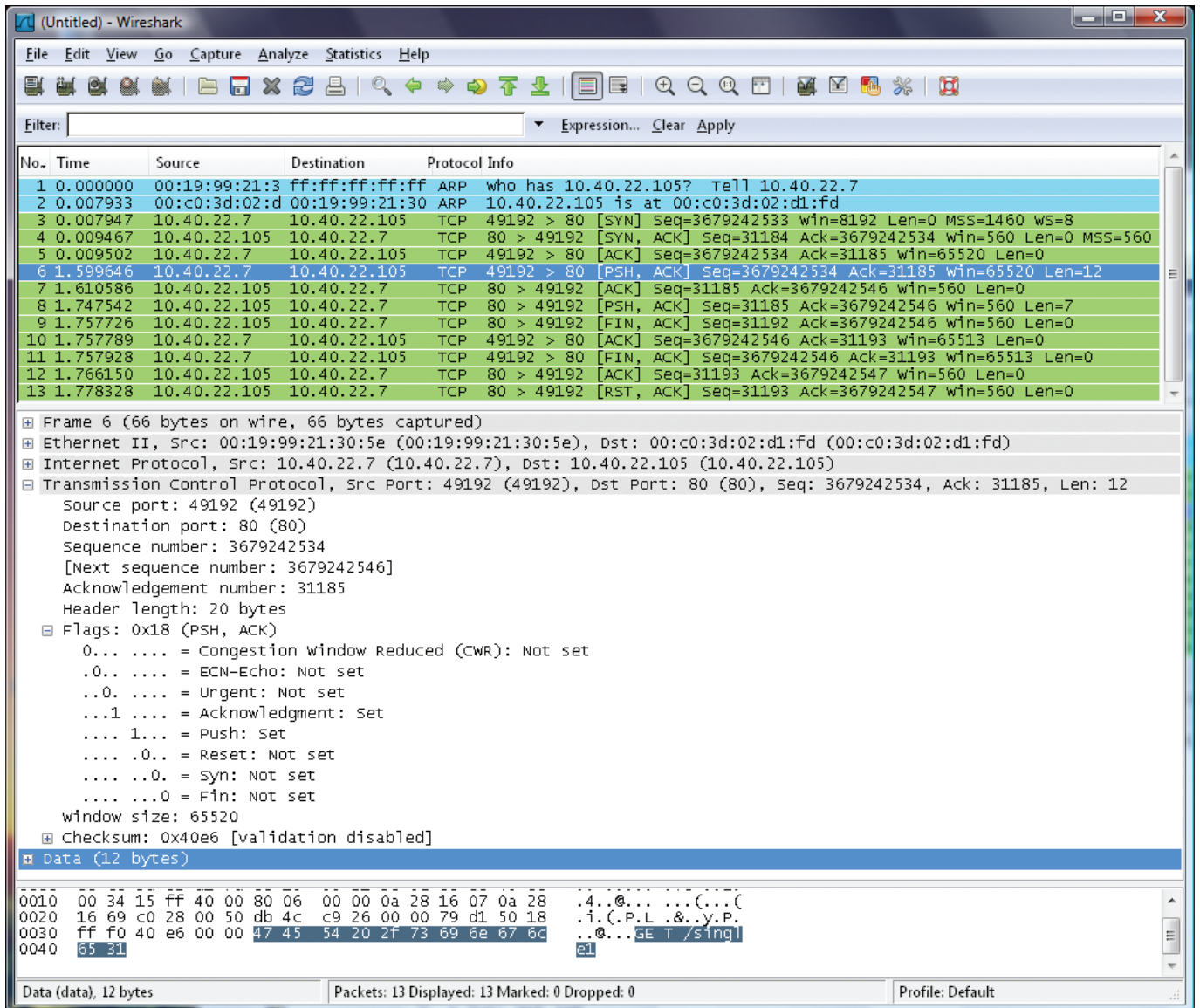


In Paket Nr.5 gibt der Client die Bestätigung, dass die Verbindung nun steht.



Das Ack-Flag ist gesetzt und auch in diese Richtung wurde die *Acknowledgement Number* um 1 hochgezählt.

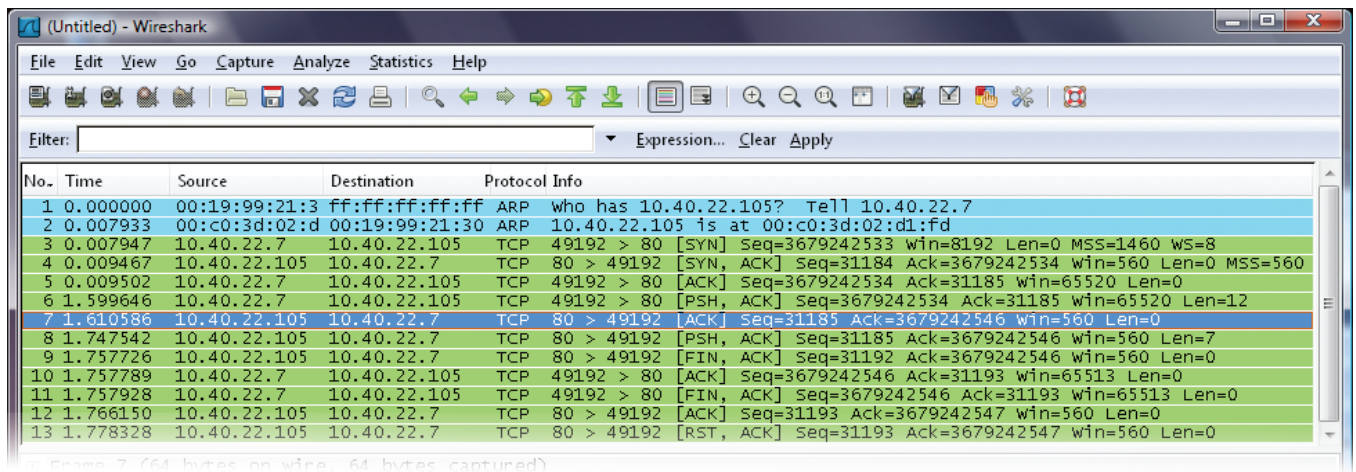
Der eigentliche Datenaustausch beginnt mit Paket Nr. 6.



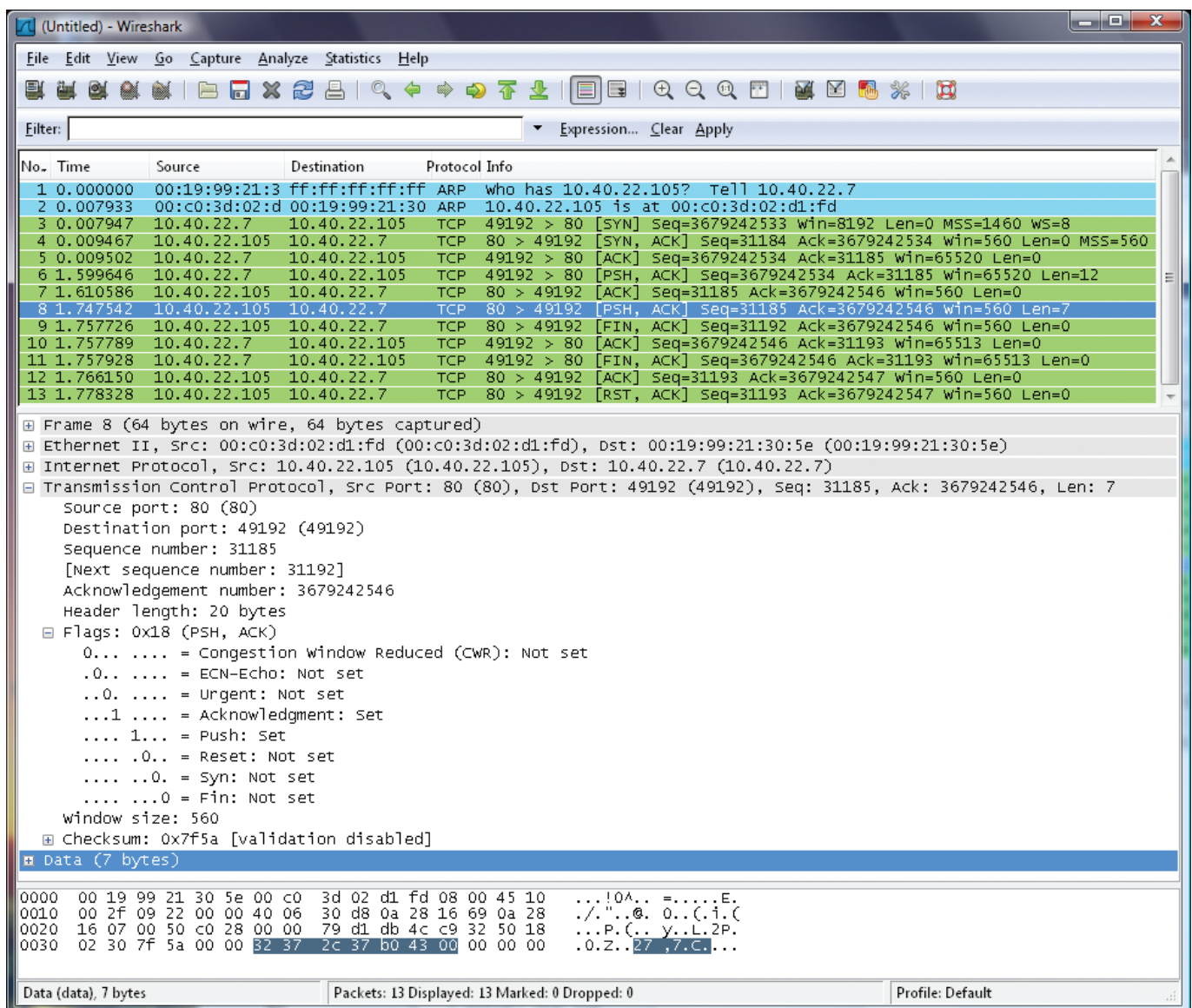
Push- und *Ack*-Flag sind gesetzt wobei das *Push*-Flag signalisiert, dass es sich um eine Nutzdatensendung handelt. Markiert man den Bereich *TCP segment data*, kann im unteren Teil des Fensters der Aufbau der Datensendung angesehen werden; links die einzelnen Bytes in hexadezimaler Schreibweise, rechts als ASCII-Zeichen.

In diesem Fall das 12 Byte lange Kommando: „GET /single“

Mit Paket Nr. 7 wird der Empfang der Datensendung quittiert.



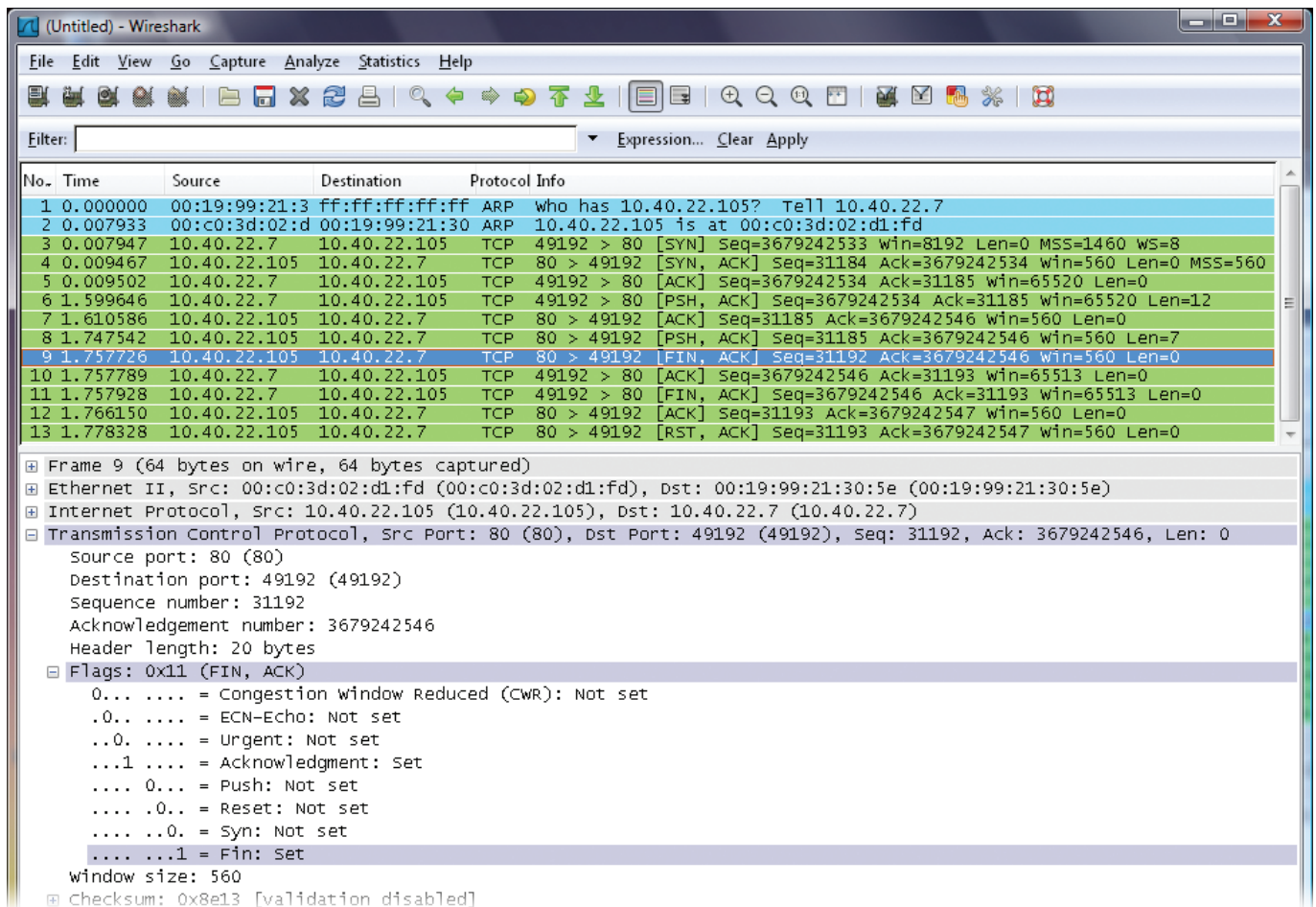
Das Ack-Flag ist gesetzt und die *Acknowledgement number* wurde um die empfangenen 12 Byte hochgezählt.



Die Antwort vom Web-Thermographen erfolgt mit Paket Nr. 8.

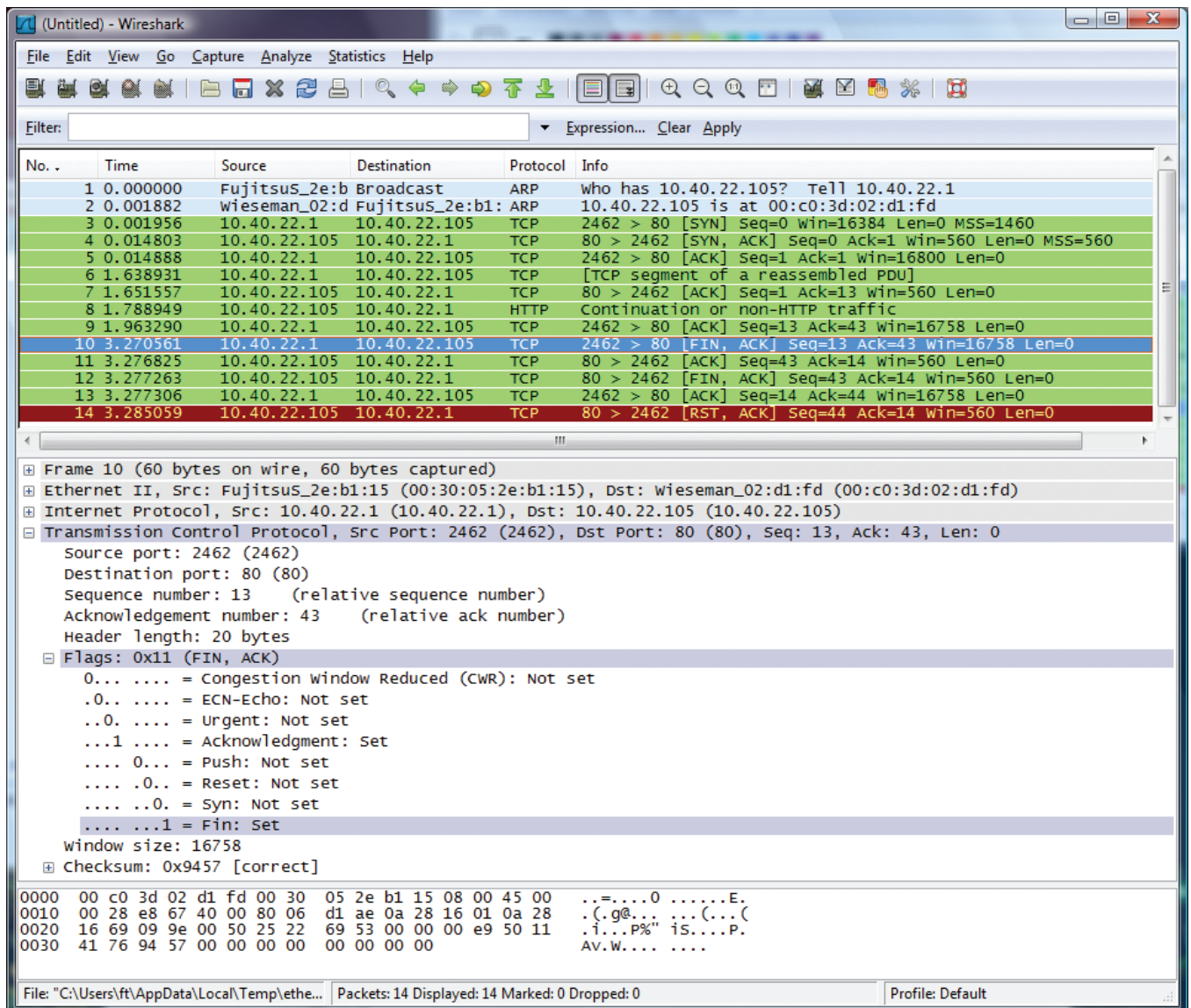
Push- und *Ack*-Flag sind gesetzt und im Datenbereich befindet sich die 7 Byte lange Antwort auf die Temperaturanfrage.

Mit Paket Nr. 9 wird die Verbindung vom Web-Thermographen beendet.



Das *Fin*- und *Ack*-Flag sind gesetzt und die *Sequence number* wurde um die gesendeten 7 Byte hochgezählt.

Anschließend beendet die Anwendung die TCP-Verbindung, was mit den Paketen 10-13 abgewickelt wird.



In Paket 10 signalisiert der Server - hier der Web-Thermograph - mit dem *Fin*- und dem *Ack*-Flag, dass die Verbindung beendet werden soll.

Der PC bestätigt das mit Paket Nr. 11 durch Setzen des *Ack*-Flags und Hochzählen der *Acknowledgement number*.

Anschließend beendet der Web-PC mit Paket 12 seinerseits die Verbindung. Auch hier sind das *Fin*- und das *Ack*-Flag gesetzt.

Der Web-Thermograph bestätigt das Verbindungsende mit Paket Nr.13 (*Ack*-Flag gesetzt und *Acknowledgement number* hochgezählt).

Abschließend sendet der Web-Thermograph Paket Nr. 14 mit gesetztem Rst- und Ack-Flag. Damit ist die Verbindung restlos gelöscht.

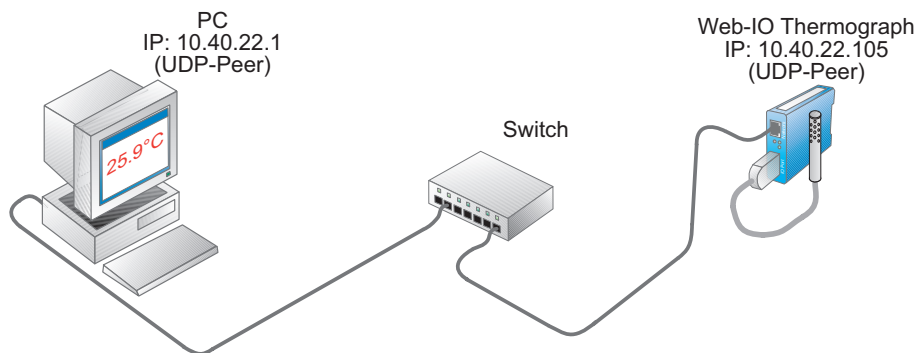


Im gezeigten Beispiel beendet der Server die Verbindung. Das hängt damit zusammen, dass es sich um eine Web-Anwendung handelt. Bei Web-Anwendungen nehmen im allgemeinen viele Nutzer den Dienst eines Servers in Anspruch. Um die Auslastung des Servers so gering wie möglich zu halten und damit möglichst vielen Usern zeitgleich den Zugriff zu erlauben, hält der Server die Verbindungen so kurz wie möglich. Nachdem die Anfrage abgewickelt wurde, beendet der Web-Thermograph deshalb sofort die Verbindung.

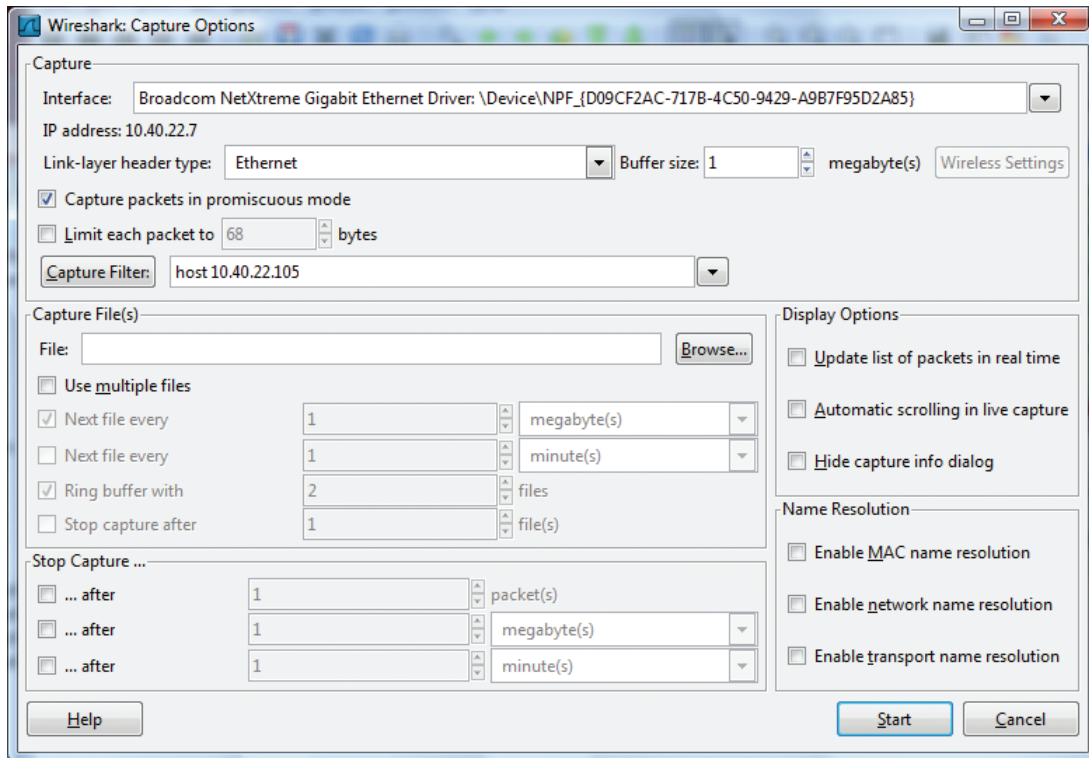
In Standard TCP-Anwendungen kann es auch der Client sein, der die Verbindung beendet.

UDP-Datenverkehr mitschneiden und auswerten

Für dieses Beispiel soll per UDP die aktuelle Temperatur von einem Web-Thermografen abgefragt werden.



Durch Anklicken von  öffnet sich ein Konfigurationsfenster.

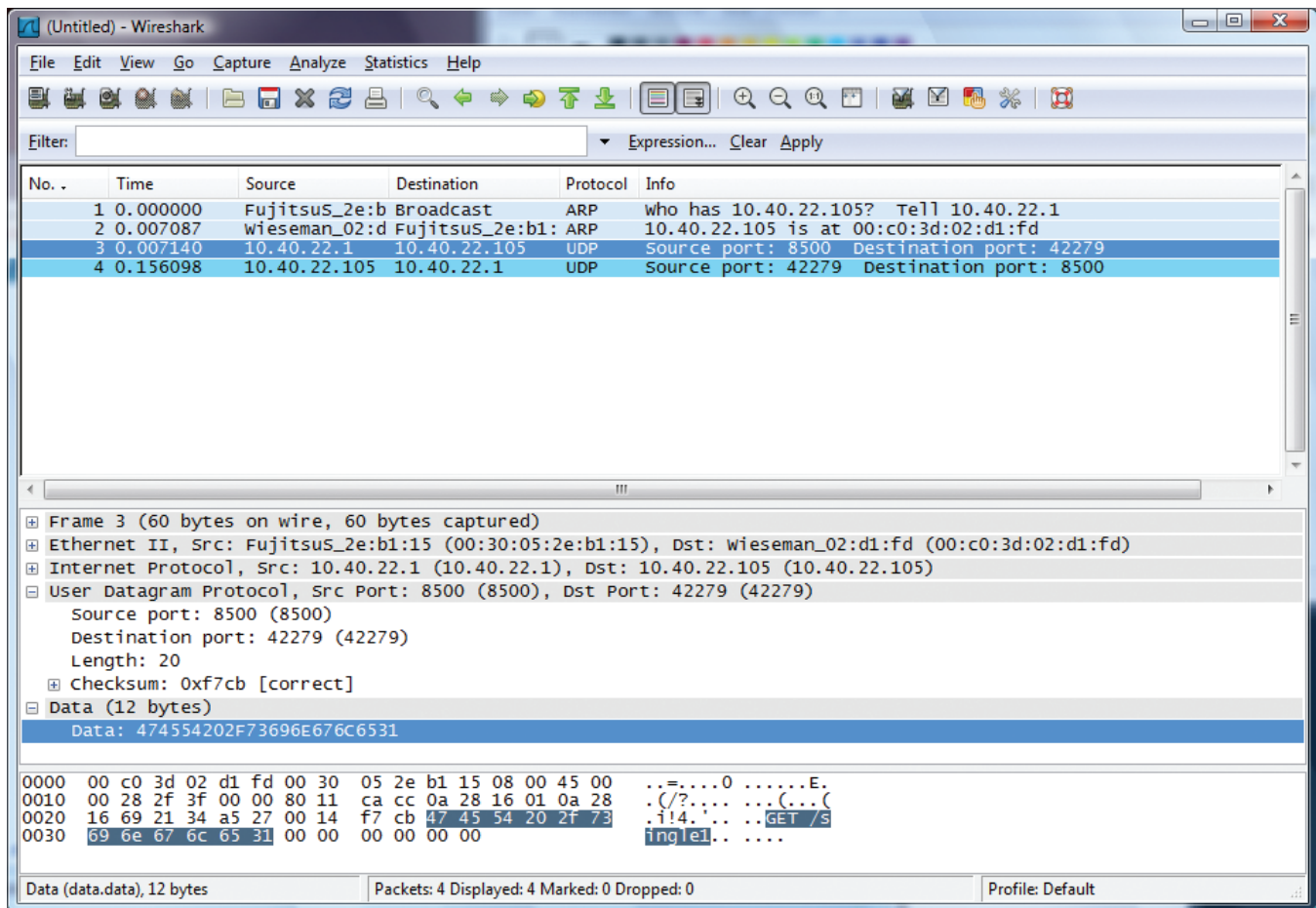


Um ausschließlich den Datenverkehr zwischen dem PC und dem Web-Thermographen mitzuschneiden, wird unter *Capture Filter* als host die IP-Adresse des Web-Thermographen eingetragen. Durch das Deaktivieren von *Enable MAC name resolution* und *Enable transportname resolution* wird erreicht, dass Wireshark die mitgeschnittenen Daten nicht vorinterpretiert (z.B. Portnummern durch etwaige Protokollnamen ersetzt). Der Mitschnitt wird dadurch besser lesbar.

Mit *Start* wird der Mitschnitt anschließend gestartet.

Nachdem das Anwendungsprogramm gestartet wurde, es die Temperatur ausgelesen hat und wieder beendet wurde, kann der Mitschnitt durch Klick auf  gestoppt werden.

Die Mitgeschnittenen Daten können nun ausgewertet werden.



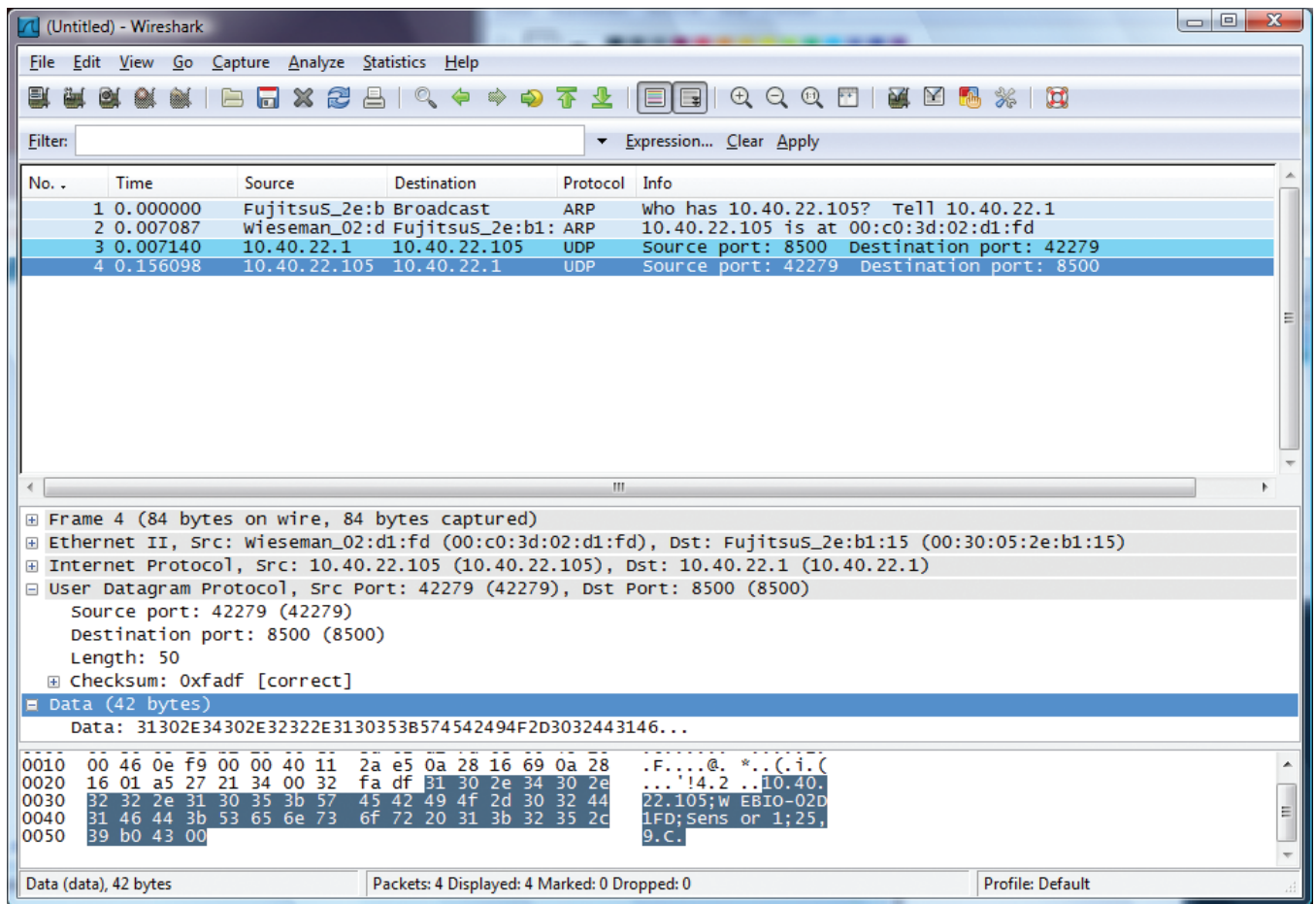
Im oberen Teil des Fensters findet man wieder die Liste der mitgeschnittenen Netzwerkpakete.

Bei den beiden ersten Paketen handelt es sich auch bei diesem Mitschnitt um ARP-Request und ARP-Reply (mehr zu ARP finden Sie im Buch *TCP/IP-Ethernet bis Web-I/O*).

Die eigentliche netzwerktechnische Abwicklung der Anwendung beginnt mit Paket Nr. 3.

Da UDP verbindungslos arbeitet, entfällt der Verbindungsaufbau. Stattdessen beginnt sofort die Kommunikation. Das Datagramm enthält 12 Bytes Nutzdaten - hier das „GET /single1“ Kommando - die an *Destination port* 42279 des Web-Thermographen gesendet werden. Der lokale Port der Anwendung erwartet die Antwort auf Port 8500. Im Gegensatz zu TCP, wo der lokale Port dynamisch generiert wird, bestimmt bei UDP die Anwendung den lokalen *Source port* - hier 8500.

Die Antwort vom Web-Thermographen zeigt Paket Nr. 4.



Der Web-Thermograph sendet 42 Byte Nutzdaten an die Anwendung, welche die Daten auf Port 8500 entgegen nimmt.

Mit diesen vier Paketen ist die Temperaturabfrage bereits abgeschlossen.

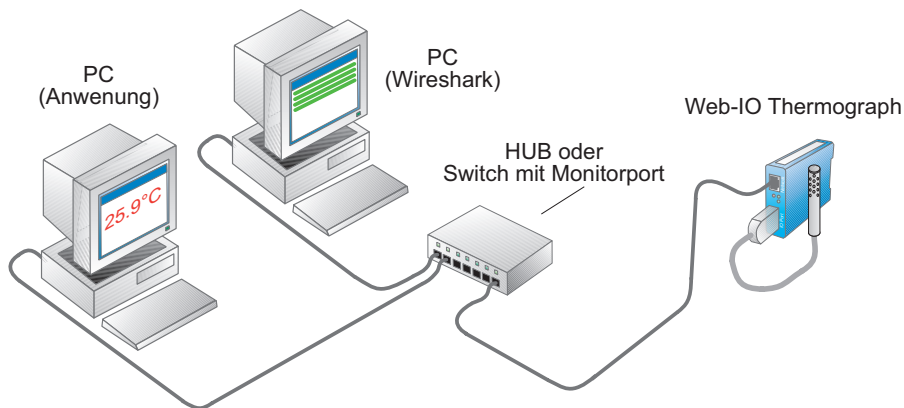
Da bei UDP keine Verbindung aufgebaut wurde, gibt es auch keinen Verbindungsabbau. Auch auf *Sequence* und *Acknowledgement number* wird bei UDP verzichtet, da es ja keine Datensicherung gibt.

Die beiden Mitschnitte zeigen sehr deutlich die Stärken und Schwächen von TCP und UDP.

Mitschneiden auf einem unbeteiligten PC

Um eigene Anwendungen und deren Netzwerkkommunikation zu prüfen und zu überwachen, wird Wireshark meist auf dem PC eingesetzt auf dem auch die Anwendung läuft.

Mit Wireshark kann der Netzwerkverkehr aber auch von einem dritten unbeteiligten PC aus mitgeschnitten werden.



Bei aktuellen Netzwerken werden fast ausschließlich Switches eingesetzt, um die Netzwerkteilnehmer miteinander zu verbinden. Nun liegt die Aufgabe eines Switches gerade darin, einem Netzwerkteilnehmer nur die Daten zu übermitteln, die auch an ihn adressiert sind. Das ist natürlich ein für Netzwerkmitschnitte ungeeignetes Verfahren, da der angeschlossene PC Datenverkehr, an dem er nicht selbst beteiligt ist, nicht „sieht“. In solchen Fällen sollte ein Hub oder ein Switch mit Monitorport verwendet werden.



Um den Netzwerkverkehr von anderen Netzteilnehmern mitschneiden zu können, muss der mitschneidende PC mit mindestens einem, an der Kommunikation beteiligten Netzteilnehmer am gleichen Hub oder einem Switch mit Monitorport angeschlossen sein.

Web-Programmierung in der Praxis

Im weiteren Verlauf dieses Buches werden Schritt für Schritt Techniken zum Aufbau dynamischer Webseiten vorgestellt.

Web-Programmierung - Anwendungen im Browser

Wie bereits in der Einführung angesprochen, sind Browser-gestützte Anwendungen vor allem für Visualisierungsaufgaben eine gute Alternative zu Standardanwendungen, die in einer der gängigen Programmiersprachen erstellt wurden. Aber auch für Steuerungs- und Bedienaufgaben bietet der Browser eine gute Plattform.

Der Vorteil Browser-basierender Anwendungen liegt darin, dass der Browser heute bei allen gängigen Betriebssystemen zur Grundausstattung gehört. Die Web-Anwendung selbst wird erst beim Start von einem Server in den Browser geladen und ausgeführt.

Aufwändige Installationsprozesse sind bei Web-Anwendungen nicht nötig.

Aufbau von Web-Anwendungen

Dynamische Web-Anwendungen sind immer eine Mischung aus verschiedenen Programmier-techniken:

- HTML
- JavaScript
- AJAX
- Java-Applets
- PHP
- Graphikverarbeitung

Werkzeuge für die Entwicklung von Web-Anwendungen

Wer bereits in Hochsprachen wie C, Delphi oder Visual Basic programmiert hat, ist es gewohnt, eine komfortable Entwicklungsumgebung vorzufinden.

Die Vielschichtigkeit der oben genannten Programmier-techniken bringt leider mit sich, dass für die Web-Programmierung mehr als ein Tool notwendig ist.

Es gibt zwar gute und ausgewachsene Profitools, die einen großen Teil der benötigten Techniken unterstützen, so z.B. den Dreamweaver von Adobe - wer aber die hohen Anschaffungskosten scheut, kann sich auch mit einer Auswahl von Freeware-Tools behelfen.

Hier einige Tipps zur benötigten Grundausstattung:

- **Texteditor**

Ein kostenloser Editor mit vielen Extras wie z.B. Syntax-Hervorhebung für die meisten Programmier- und Scriptsprachen (unter anderem HTML, JavaScript und PHP) ist PSPAD. Darüber hinaus bietet PSPAD eine Hex-Editor.

Kostenloser Download unter: www.pspad.com

- **WYSIWYG-HTML-Editor**

What you see is what you get Editoren erlauben es, Webseiten in einer graphischen Entwicklungsumgebung zusammenzubauen, wobei die Objekte der Webseite so dargestellt werden, wie später im Browser. Eine gute Umsetzung dieser Technik bietet NVU.

Kostenloser Download unter: www.nvu-composer.de

- **JavaScript Editor**

Für JavaScript gibt es leider keine richtige Entwicklungsumgebung. Ein Software-Paket mit Namen Aptana Studio bietet aber einen JavaScript Editor mit Befehlervervollständigung und Syntaxhervorhebung.

Kostenloser Download unter: www.aptana.com

- **CSS-Hilfstool**

Cascading Style Sheets erlauben feste Formatvorgaben für bestimmte HTML-Objekte. Das Programm TopStyle Lite gibt eine gute Hilfestellung bei der Erstellung von CSS-Formatvorgaben.

Kostenloser Download unter: topstyle.en.softonic.com

- **HTML und JavaScript-Debugging**

Da es für JavaScript keine echte Entwicklungsumgebung gibt, werden viele Fehler im Quelltext erst bei Aufruf der Webseite erkennbar. In vielen Fällen wird zwar angezeigt,

dass ein Fehler aufgetreten ist, aber nicht, in welchem Bereich des Quelltextes er zu suchen ist. Hier kann Firebug helfen. Firebug ist ein Add-on für den Browser FireFox.

Kostenloser Firefox-Download unter: www.firefox.de

Das *Firebug*-Add-on kann dann im Firefox unter *Extras* >> *Add-ons* heruntergeladen und hinzugefügt werden.

- **Zeichenprogramm**

Vor allem bei technischen Anwendungen ist es hilfreich, Anlagenteile mit einfachen, aber ausdrucksstarken Zeichnungen zu visualisieren. InScape ist eine Freeware, mit der sich Vectorgraphiken erstellen lassen.

Kostenloser Download unter: www.inkscape.org

- **Programm zur Graphikbearbeitung**

Für einige Anwendungen müssen Grafiken in ihrer Größe geändert oder in mehrere Teile zerschnitten werden. Einfache Operationen können mit dem Windows-eigenen Paint Programm durchgeführt werden. Für komplexere Aufgaben eignet sich Gimp.

Kostenloser Download unter: www.gimp.org

- **Web-Server mit PHP-Unterstützung**

Für komplexere Webseiten-Projekte, die mit der Programmiersprache PHP erstellt wurden, benötigt man zum Testen einen PHP-fähigen Web-Server. Mit Xamp kann man einen solchen Server auf dem eigenen PC starten und muss die Webseiten und Programme nicht bei jeder Änderung auf einen externen Web-Server hochladen.

Kostenloser Download unter:

<http://www.apachefriends.org/de/xampp-windows.html>

HTML – Hypertext Markup Language

Dass Grundgerüst für Web-Anwendungen ist HTML (Hypertext Markup Language). HTML ist keine Programmiersprache, sondern eine Auszeichnungssprache (Markup Language), die sich aus Schlüsselwörtern – auch Tags genannt – und den darzustellenden Inhalten zusammensetzt. Die Tags geben an, in welcher Art und Weise der nachfolgende Text darzustellen ist. So lassen sich z.B. Schriftgröße, -art und -ausrichtung vorgeben,

Inhalte können in Tabellen oder in Form einer numerischen Aufzählung dargestellt werden, die Farbe von Text und Hintergrund kann festgelegt werden usw. Der Browser interpretiert diese Angaben und stellt sie dar.

Neben Text können mit Hilfe von HTML auch Grafiken angezeigt werden, und sogar multimediale Inhalte wie Musik, Sprache oder Filmsequenzen lassen sich per HTML einbinden. Das HTML-Dokument selbst transportiert dabei ausschließlich Textinhalte. Für jedes andere darzustellende Element wird via HTML angegeben, von wo es geladen werden kann, wo es auf dem Bildschirm erscheinen soll und in welcher Größe es dargestellt werden soll.

Die wohl wichtigste Eigenschaft von HTML liegt darin, dass alle Elemente mit einem Verweis – auch *Hyperlink* oder kurz *Link* genannt – versehen werden können. Klickt der Anwender ein solches Element mit der Maus an, wird er automatisch auf eine weitere Website weitergeleitet, bekommt eine Grafik angezeigt oder startet einen Download.

Mit der Erklärung der von HTML bereitgestellten Tags lassen sich ganze Bücher füllen. Wer mit den im weiteren Verlauf vorgestellten Beispielen arbeiten möchte, sollte zumindest HTML-Grundkenntnisse besitzen. Wir beschränken uns hier also auf die Vorstellung der elementaren Tags und Eigenschaften von HTML.

Für HTML-Tags gilt ein festes Schema:

- Einzelne Tags sind in spitze Klammern „eingepackt“. *<HTML-Tag>*
- Das eigentliche Tag kann durch Angabe von Attributen erweitert werden. *<HTML-Tag Attribut="xy">*
- Überwiegend wird durch die paarweise Verwendung von Tags der Anfang und das Ende ihres Gültigkeitsbereichs festgelegt; die definierten Eigenschaften gelten dann für alles was zwischen den Tags steht. Das schließende Tag wiederholt das öffnende Tag mit einem vorangestellten Schrägstrich. Beispiel: *<title>Willkommen</title>*

- Bei HTML-Tags wird nicht zwischen Groß- und Kleinschreibung unterschieden. `<HTML>` ist gleichbedeutend mit `<html>`.

Grundsätzlicher Aufbau einer HTML-Datei

Jede HTML-Datei wird mit `<HTML>` eingeleitet und endet mit `</HTML>`. Man unterscheidet beim weiteren Aufbau einer Seite zwischen Kopf und Körper.

Alle Angaben im Kopf bleiben für den Betrachter unsichtbar und enthalten Eigenschaften der Seite, die nicht direkt die Darstellung betreffen. Einzige Ausnahmen sind der Titel, der in der Titelleiste des Browserfensters angezeigt wird und die Festlegung des Charsets. Mit der Charset-Angabe wird festgelegt, wie die Zeichen im Browser dargestellt werden. `charset=iso-8859-1` steht für europäische bzw. nordamerikanische Zeichen. Es gibt aber z.B. japanische Zeichen. Die verfügbaren Charsets sind im Anhang aufgelistet. Die Kopfinformationen stehen zwischen den Tags `<head>` und `</head>`.

Auf den Kopf folgt der Seitenkörper, der mit dem `<body>`-Tag eingeleitet wird. Im Körper der HTML-Seite sind alle Angaben zu finden, die den eigentlichen Inhalt der Seite und dessen Darstellung betreffen. Das Ende des Körpers wird mit dem `</body>`-Tag gekennzeichnet.



Einfache HTML-Dateien lassen sich sehr gut mit dem Programm Pspad erstellen und bearbeiten.

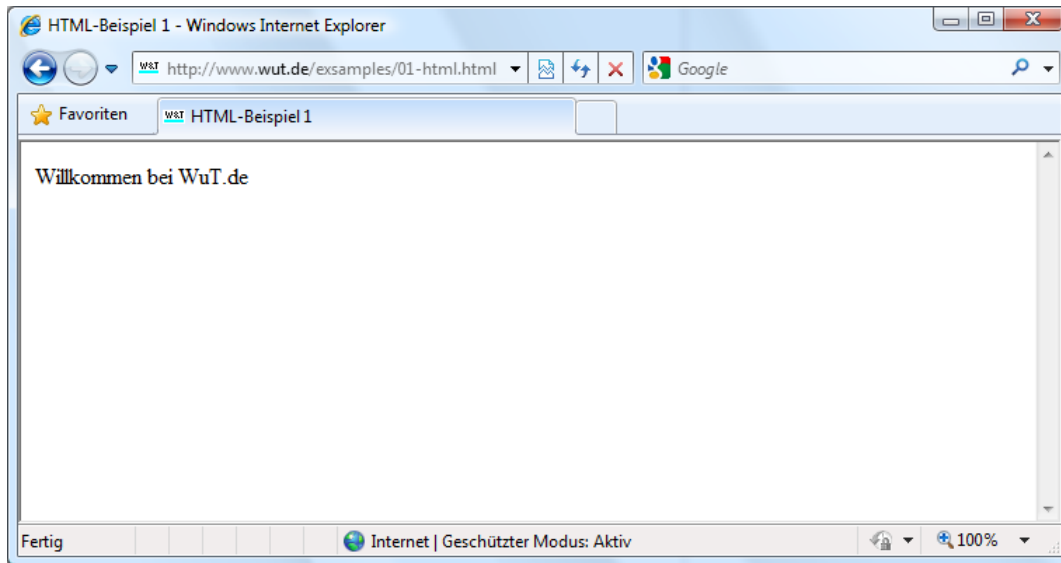
Hier ein einfaches Beispiel:

```
<html>
  <head>
    <title>HTML-Beispiel 1</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body bgcolor="#FFFFFF">
    Willkommen bei WuT.de
  </body>
</html>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/01-html.html

Im Browser sieht das dann so aus:



Bitte beachten Sie, dass beim `<body>`-Tag das Attribut `bgcolor="#FFFFFF"` für einen weißen Hintergrund angegeben wurde.

Hyperlinks

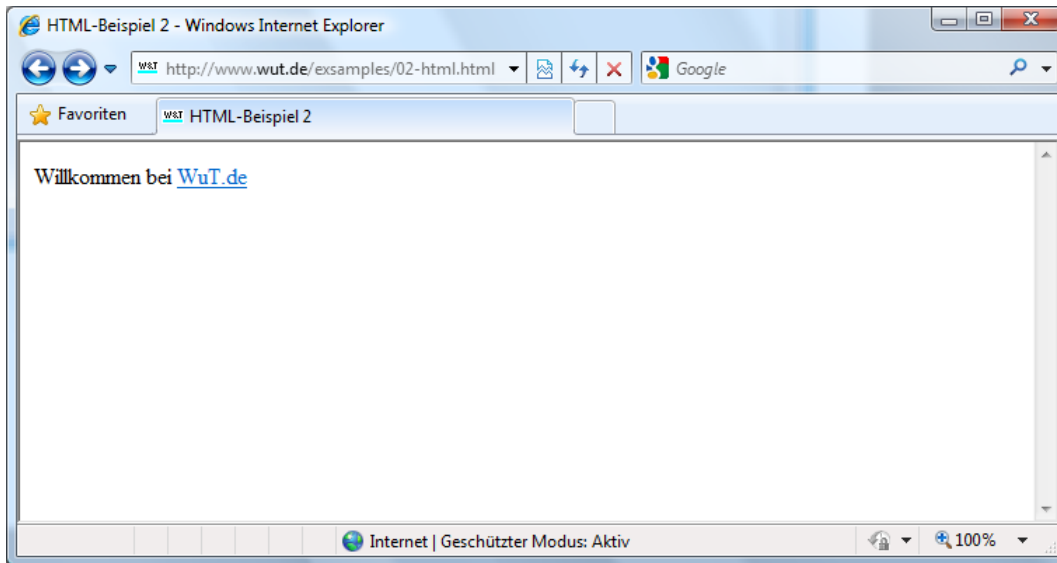
Einer der großen Vorteile von HTML liegt in der Möglichkeit, einzelne Inhaltselemente mit einem Hyperlink zu versehen. Klickt der Anwender auf ein solches verlinktes Element, wird er auf eine andere Webseite weitergeleitet.

Wir erweitern unseren HTML-Code um einen Hyperlink:

```
<body bgcolor="#FFFFFF">
  Willkommen bei <a href="http://www.wut.de">WuT.de</a>
</body>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/02-html.html



Bei einem Mausklick auf „WuT.de“ werden wir nun auf die Homepage von W&T gelenkt.

Das Pfadattribut des Tags `<a href="Pfadangabe">` kann die Pfadangabe entweder in absoluter oder in relativer Form enthalten.

- Absolut: es wird der komplette URL angegeben, auf den der Hyperlink verweisen soll.
- Relativ: es wird nur der Name der Datei angegeben auf die zugegriffen werden soll. Die Datei wird dann im gleichen Verzeichnis gesucht, in dem sich auch die aktuelle HTML-Datei befindet.



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch.

Download unter:

`www.wut.de/
examples/
03-html.html`

Darstellung von multimedialen Inhalten

Wie bereits angesprochen, erlaubt HTML die Darstellung von Inhalten, die nicht Bestandteil des HTML-Dokuments sind, sondern von anderer Stelle nachgeladen werden.

Bilder

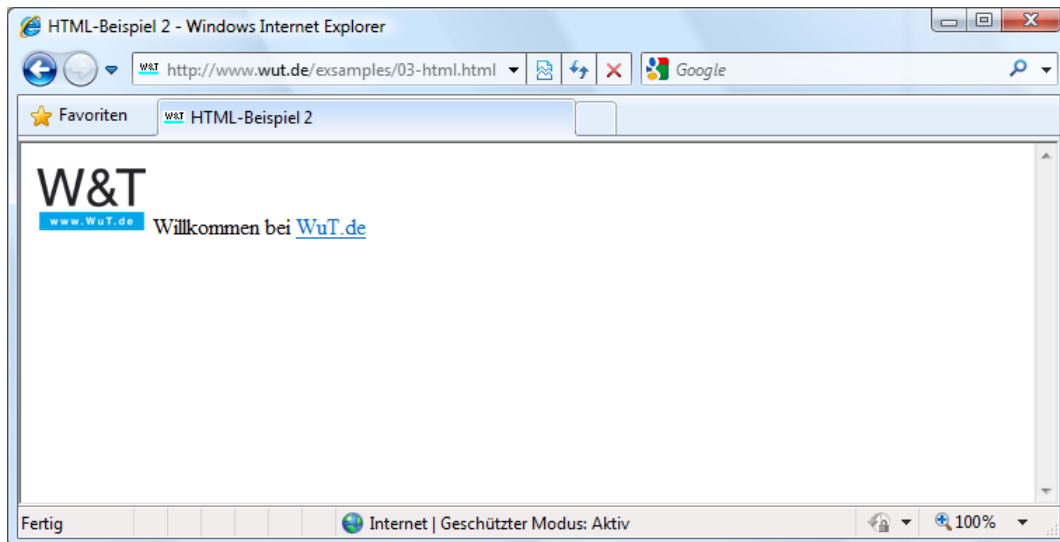
Zur Einbindung von Bilddateien stellt HTML z.B. das `<img>`-Tag (*img* für Image) zur Verfügung, wobei über das Attribut `src` Namen und Quelle der Bilddatei angegeben werden.

Wir erweitern unser HTML-Dokument um eine Grafik:

```
<body bgcolor="#FFFFFF">  
  
```

```
Willkommen bei <a href="http://www.wut.de">WuT.de</a>
</body>
```

Nun wird neben dem Text ein Logo im JPG-Format dargestellt, das aus dem Verzeichnis *examples* auf dem Webserver von W&T geladen wird. Der Pfad der Bilddatei kann wie auch beim Hyperlink absolut oder relativ angegeben werden.



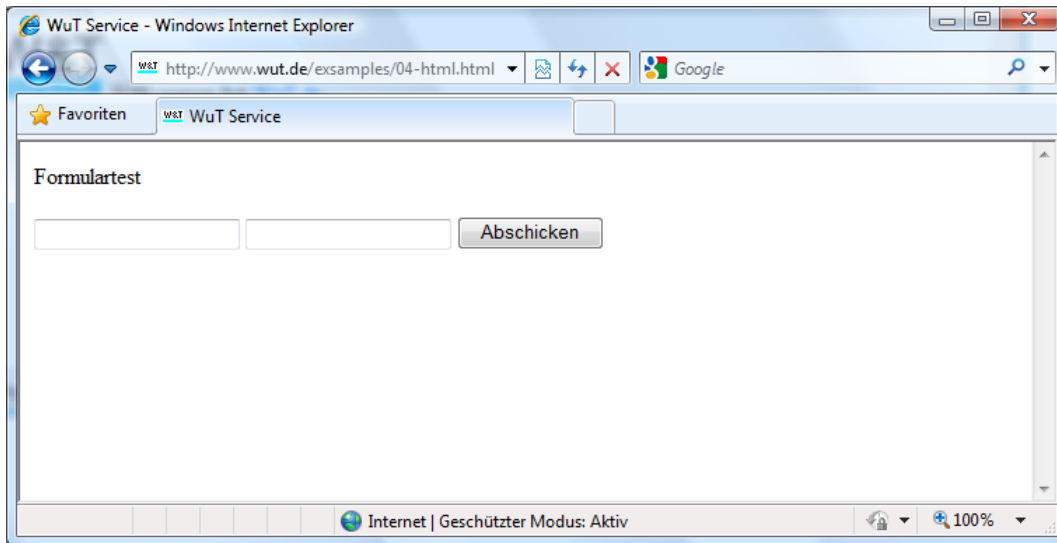
Über die Parameter *height* und *width* kann die Anzeigeröße einer Bilddatei vorgegeben werden. Es wird aber empfohlen, Bilddateien gleich in der gewünschten Skalierung abzuspeichern und nicht nachträglich im Browser anzupassen. Das Grafik-Tool Gimp bietet entsprechende Funktionen!

Formulare

HTML ist eine reine Darstellungssprache, die eine statische Anzeige im Browser generiert. Aber wie sieht es aus, wenn der Anwender Informationen an den WWW-Server zurückgeben möchte?

Als Lösung bietet HTML die Möglichkeit, Formulare anzuzeigen, die vom Anwender ausgefüllt werden können. Die so eingegebenen Informationen können durch Anklicken eines sogenannten „Submit-Buttons“ vom Browser zum WWW-Server gesendet werden.

Hier ein kurzes Beispiel:



Im HTML-Code sieht das so aus:

```
<html>
  <head>
    <title>WuT Service</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body bgcolor="#FFFFFF">
    Formulartest
    <form method="post" action="Formularauswertung.cgi" name="FORMULAR1">
      <input type="text" name="EINGABEFELD1">
      <input type="text" name="EINGABEFELD2">
      <input type="submit" name="submit" value="Abschicken">
    </form>
  </body>
</html>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/04-html.html

Sämtliche zum Formular gehörenden Elemente sind zwischen dem einleitenden `<form>`-Tag und dem abschließenden `</form>`-Tag zu finden.

Die Attribute des Form-Tags sind:

- | | |
|---------------|---|
| method | gibt an, wie HTTP die Eingaben an den WWW-Server übergibt. |
| action | legt fest, an welchen Prozess auf dem Server die Eingaben übergeben werden. |

name kann willkürlich vergeben werden und zeigt dem Prozess auf dem Server, von welchem Formular die Eingaben stammen (ein Prozess kann ggf. mehrere Formulare auswerten).

Die Eingabeelemente selbst werden über das `<input>`-Tag festgelegt, wobei das Attribut `type` angibt, um welche Art von Eingabeelement es sich jeweils handelt. Mögliche Attribute sind hier:

text	Texteingabefeld
checkbox	Ankreuzkästchen
radio	Optionsbutton
submit	Button zum Abschicken oder Zurücksetzen des Formulars

Über das Attribut `name` kann dem Element ein eindeutiger Name (vergleichbar mit einem Variablennamen) gegeben werden; mit dem Attribut `value` kann ihm ein Anfangswert zugeteilt werden.

Die Angabe `<input type="text" name="EINGABEFELD1" value="test1">` würde z.B. dazu führen, dass beim Öffnen des Formulars im Browser schon der Text `test1` im ersten Eingabefeld stehen würde.

Was mit den per Formular übergebenen Informationen geschieht – ob der Anwender eine Rückmeldung erhält und wie diese aussieht –, bestimmt allein der Prozess auf dem WWW-Server, der die Informationen entgegennimmt und auswertet.

Auch wenn durch die Einbindung von Bildern, Grafiken und Formularen vom Design her sehr hochwertige Webseiten erstellt werden können - nur mit HTML wird der Quelltext der Seite bei jedem Aufruf einfach von oben nach unten abgearbeitet und sieht immer gleich aus.

Wie bereits angesprochen, möchten wir hier nicht bis auf das letzte Detail von HTML eingehen. Wer Webseiten erstellen möchte, sollte sich auf jeden Fall eingehender mit diesem Thema beschäftigen. Eine hervorragende Quelle für weitere Informatio-

nen zu HTML findet man unter <http://www.selfhtml.org>; sehr nützlich ist selbstverständlich auch die Website des W3-Konsortiums (<http://www.w3.org>), das in Sachen HTML als normierende Körperschaft fungiert.

JavaScript

JavaScript (nicht zu verwechseln mit Java) ist eine Programmiersprache, deren Quelltext im Normalfall Bestandteil der HTML-Seite ist. In aller Regel werden die JavaScripte im `<head>`-Bereich der Web-Seite untergebracht.

Beispiel:

```
<html>
  <head>
    <title>Seitentitel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <script language="JavaScript" type="text/javascript">
      .....
      JavaScript-Anteil
      .....
    </script>
  </head>
  <body>
    HTML-Anteil der Seite
  </body>
</html>
```

Größere JavaScripte können aber auch in einer separaten JS-Datei untergebracht werden. Im Kopf der HTML-Seite wird dann entsprechend eine Verlinkung vorgenommen.

Beispiel:

```
<html>
  <head>
    <title>Seitentitel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <script type="text/javascript" src="http://www.meineseite.de/mein.js"></script>
```

```

</head>
<body>
  HTML-Anteil der Seite
</body>
</html>

```

Mit JavaScript werden Webseiten lebendig bzw. dynamisch. Hier einige Beispiele für das, was mit JavaScript in der Praxis umgesetzt werden kann:

- Das Aussehen einer Webseite kann bereits beim Laden abhängig von verschiedensten Bedingungen unterschiedlich gestaltet werden.
- Es können im Nachhinein Daten vom Server nachgeladen werden, und das Aussehen einer Webseite kann nachträglich verändert und angepasst werden.
- User Eingaben können überprüft bzw. bearbeitet werden, bevor sie zum Server gesendet werden.
- Mausbewegungen und Ereignisse lassen sich auswerten und z.B. für dynamische Menüs nutzen.
- Es kann ein Datenaustausch zwischen Java-Applets und JavaScript realisiert werden.



In Webseiten, die neben HTML auch JavaScript Anteile enthalten, schleichen sich sehr schnell Fehler ein. Eine gute Hilfe ist hier Aptana Studio. Aptana prüft sowohl den HTML-Teil als auch die JavaScripte auf syntaktische Fehler. Außerdem bietet Aptana eine halbautomatische Code-Vervollständigung.

Ein erstes Beispiel:

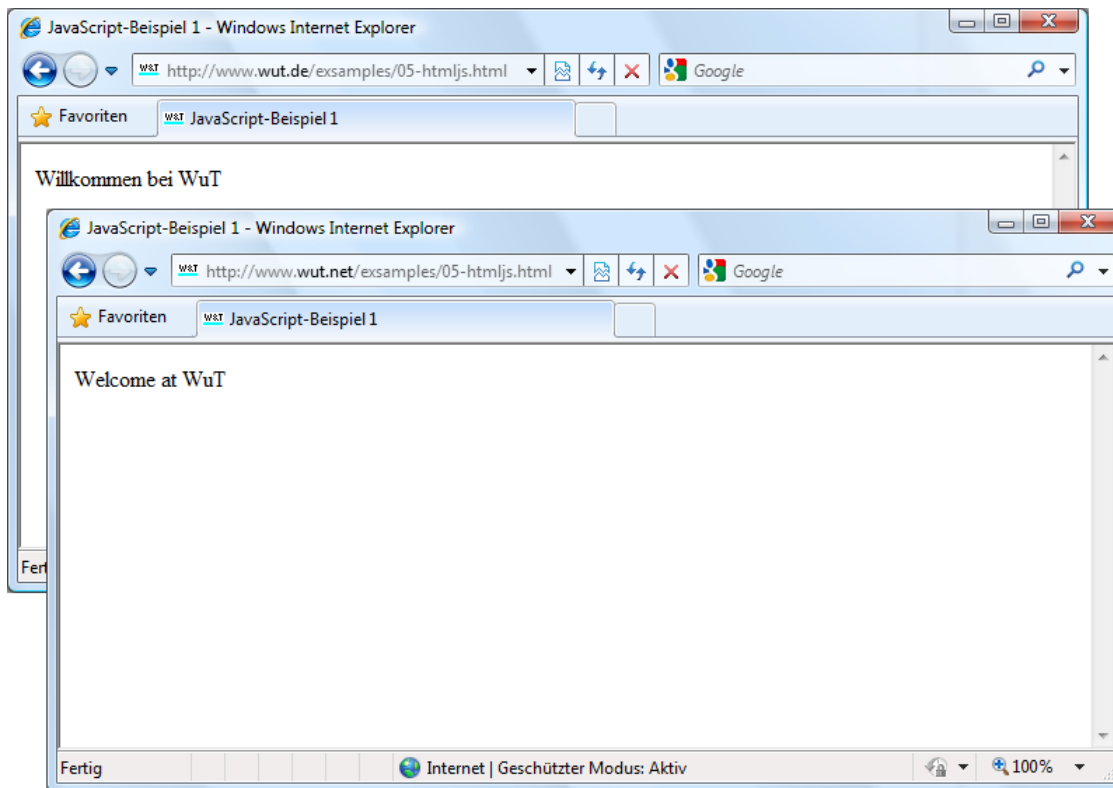
Begrüßungstext mit JavaScript dynamisch anzeigen

Für ein und den selben Webaufttritt können mehrere Domain-Namen vergeben werden.

Die Web-Präsenz von Wiesemann & Theis findet man gleichermaßen unter www.wut.de wie auch unter www.wut.net.

Mit JavaScript könnte man erkennen, ob ein Besucher über www.wut.de oder www.wut.net auf die Webseite gelangt ist. Ab-

hängig davon könnte der Besucher auf Deutsch oder in englischer Sprache begrüßt werden.



```
<html>
<head>
  <title>JavaScript-Beispiel 1</title>
  <script language="JavaScript">
    if (location.hostname=="www.wut.de")
    { document.write('Willkommen bei WuT');
    }
    else
    { document.write('Welcome at WuT');
    }
  </script>
</head>
<body bgcolor="#FFFFFF">
</body>
</html>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter:
[www.wut.de/
examples/
05-htmljs.html](http://www.wut.de/examples/05-htmljs.html)

Über den Parameter *location.hostname* wird über eine IF-Abfrage ermittelt, welcher Domainname eingegeben wurde. Abhängig davon wird mittels *document.write* der entsprechende Begrüßungstext in das Browserfenster geschrieben.

Text einer bereits geladenen Seite mit JavaScript ändern

Im nächsten Schritt soll ein angezeigter Text geändert werden, nachdem die Webseite bereits in den Browser geladen wurde.

Als Beispiel soll auf der Webseite die aktuelle Uhrzeit angezeigt werden und sich kontinuierlich aktualisieren.

Um Objekte nachträglich modifizieren zu können, müssen diese mit einer ID versehen werden. Um einen Text verändern zu können, muss dieser deshalb Teil eines Objektes sein.

Eine gute Möglichkeit Text gezielt zur Anzeige zu bringen bieten DIV-Elemente. DIV-Elemente sind rechteckige Flächen, denen Eigenschaften wie Höhe, Breite, XY-Position, Farbe und vieles mehr zugewiesen werden können.

Die Definition eines DIV-Objektes kann als HTML-Text im Body-Bereich erfolgen oder mittels des *document.write* Kommandos von JavaScript aus geschrieben werden.

Variante 1

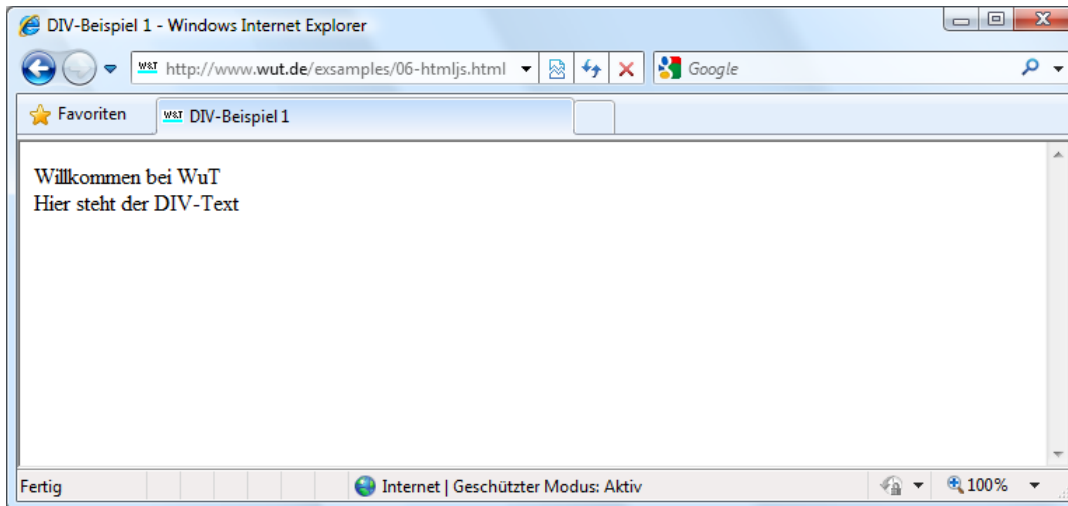
DIV-Element in JavaScript

```
<html>
  <head>
    <title>DIV-Beispiel 1</title>
  </head>
  <body bgcolor="#FFFFFF">
    Willkommen bei WuT
    <div id="timediv">Hier steht der DIV-Text</div>
  </body>
</html>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter:
www.wut.de/examples/06-htmljs.html

Im gezeigten Beispiel wurden dem DIV-Objekt keine weiteren Eigenschaften zugewiesen außer dem Text. Das DIV-Objekt wird somit linksbündig in den Text bzw. Objektfluss der Webseite eingegliedert.



Über die *id* innerhalb des DIV-Start-Tags kann das DIV-Objekt später angesprochen und somit dessen Eigenschaften geändert werden.

Zur Identifizierung des zu ändernden Objektes wird das Kommando `document.getElementById('meine_id')` genutzt. Mit einem Punkt getrennt wird dann der Name der zu ändernden Eigenschaft angehängt und mit = der gewünschte Wert bzw. Inhalt zugewiesen.

Beispiel:

```
document.getElementById('meine_id').innerHTML = "neuer Text";
```

In diesem Beispiel wird in das DIV-Objekt mit der *id* `meine_id` der Text *neuer Text* geschrieben

Nun zurück zu der ursprünglichen Aufgabe, in einem DIV-Objekt kontinuierlich die aktuelle Uhrzeit anzuzeigen.

```
<html>
<head>
  <title>JavaScript-Beispiel 1</title>
  <script language="JavaScript">
    setInterval("changeTime()", 1000);

    function changeTime()
    { var currentDate = new Date();
      var dd = currentDate.getDate();
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/08-htmljs.html

```

    var dm = currentDate.getMonth() + 1;
    var dy = currentDate.getFullYear();
    var th = currentDate.getHours();
    var tm = currentDate.getMinutes();
    var ts = currentDate.getSeconds();

    if (tm <10) { tm= "0" + String(tm); } // 0 vorweg falls einstellig
    if (ts <10) { ts= "0" + String(ts); }

    var timeString = dd+"."+dm+"."+dy+" "+th+": "+tm+": "+ts;
    document.getElementById('timediv').innerHTML = timeString;
}
</script>
</head>
<body bgcolor="#FFFFFF">
    Willkommen bei WuT
    <div id="timediv"></div>
</body>
</html>

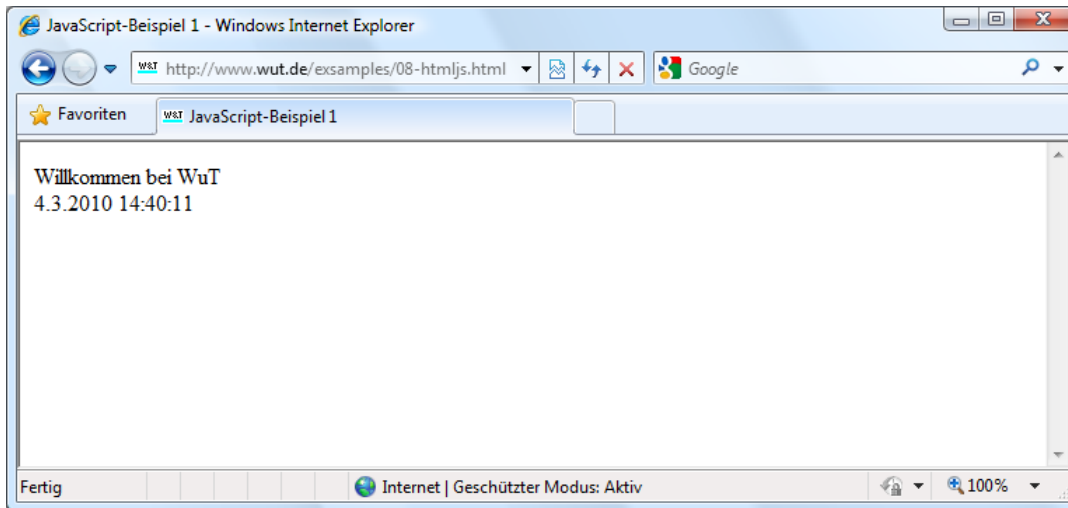
```

Wie im vorangegangenen Beispiel wurde im Body-Bereich ein DIV-Objekt mit der ID *timediv* angelegt. Hier sollen später die Uhrzeit und das Datum zur Anzeige gebracht werden.

Im JavaScript-Teil wurde die Funktion *changeTime* definiert. Bei Aufruf dieser Funktion wird zunächst der aktuelle *TimeDate* Wert in die Variable *currentDate* geschrieben. Aus dem *currentDate*-Wert werden über entsprechende Methodenaufrufe die Werte für Tag, Monat, Jahr, Stunde, Minute und Sekunde ermittelt und zu dem String *timeString* zusammengesetzt. Dabei wird für Minute und Sekunde sichergestellt, dass die Werte zweistellig eingetragen werden.

Über `document.getElementById('timediv').innerHTML = timeString` wird dann der String in das DIV-Objekt geschrieben.

Die Funktion *changeTime* wird über *setInterval* einmal pro Sekunde aufgerufen, so dass die Anzeige jede Sekunde auf die Aktuelle Uhrzeit wechselt.



Objekte mit HTML oder JavaScript anlegen?

Das gleiche Ergebnis kann man auch über eine reine JavaScript-Lösung erzielen. Die Anzeigeobjekte werden dann mittels `document.write` von JavaScript generiert. Der Body-Bereich bleibt dafür entsprechend leer.

Die reine JavaScript-Lösung zum vorangegangenen Beispiel würde so aussehen:

```
<html>
<head>
  <title>JavaScript-Beispiel 1</title>
  <script language="JavaScript">
    document.write("Willkommen bei WuT");
    document.write("<div id='timediv'></div>");

    function changeTime()
    { var currentDate = new Date();
      var dd = currentDate.getDate();
      var dm = currentDate.getMonth() + 1;
      var dy = currentDate.getFullYear();
      var th = currentDate.getHours();
      var tm = currentDate.getMinutes();
      var ts = currentDate.getSeconds();
      if (tm <10) { tm= "0" + String(tm); } // 0 vorweg falls einstellig
      if (ts <10) { ts= "0" + String(ts); }
      var timeString = dd+"."+dm+"."+dy+" "+th+": "+tm+": "+ts;
      document.getElementById('timediv').innerHTML = timeString;
    }
  </script>
</head>
<body>
</body>
</html>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/09-htmljs.html

```

        setInterval("changeTime()", 1000);
    </script>
</head>
<body bgcolor="#FFFFFF">
</body>
</html>

```

Wann macht es Sinn, die einzelnen Elemente einer Webseite per HTML zusammenzubauen und wann ist es besser, die Seitenelemente mit JavaScript zu generieren?

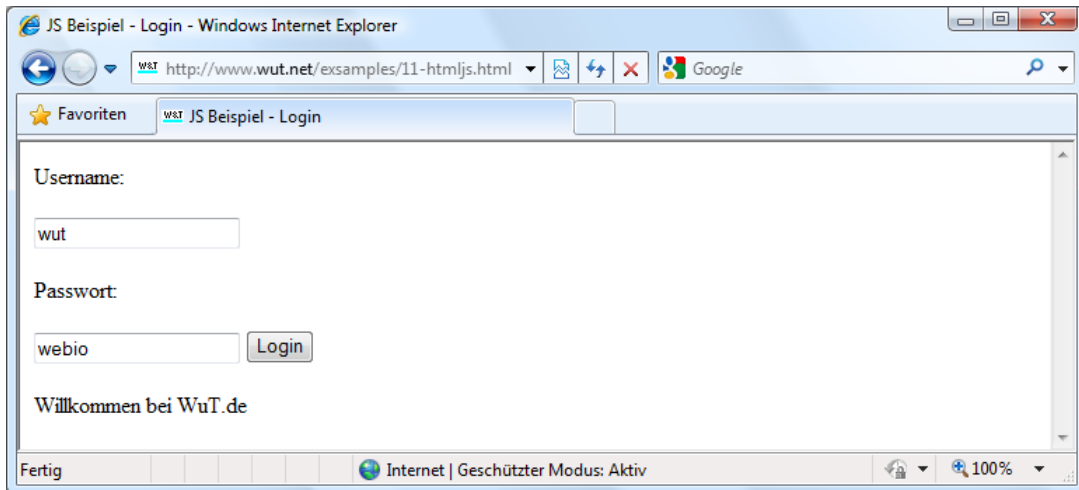
Als Faustregel kann man sagen:

- Webseiten mit zum größten Teil unterschiedlichen Elementen bzw. Objekten baut man sinnvoller Weise mit HTML zusammen.
- Enthält eine Webseite viele gleichgeartete Elemente, kann die JavaScript Methode den Aufwand deutlich verringern, weil gleichartige Objekte über Schleifenfunktionen generiert werden können.

Mit User-Eingaben arbeiten

Eine weitere Funktionalität von JavaScript ist die Möglichkeit, User-Eingaben einzulesen und zu verarbeiten bzw. auszuwerten.

Als einfaches Beispiel soll eine Webseite erstellt werden, auf der Username und Passwort eingegeben werden können. Klickt der Anwender auf den Login-Button, sollen in Abhängigkeit der Eingaben verschiedene Begrüßungstexte ausgegeben werden.



```
<html>
<head>
  <title>JS Beispiel - Login</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <script language="JavaScript" type="text/javascript">

    function checkLogin()
    {
      var username = document.forms[0].tb_username.value;
      var password = document.forms[0].tb_password.value;;
      if ((username == 'wut') && (password == 'webio'))
      {
        document.getElementById('text').innerHTML
          = 'Willkommen bei WuT.de';
      }
      else
      {
        document.getElementById('text').innerHTML
          = 'User nicht bekannt oder falsches Passwort!';
      }
    }

  </script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
  <form name="form1">
    <p>Username:</p>
    <p>
      <input type="text" name="tb_username">
    </p>
    <p>Passwort:</p>
    <p>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/11-htmljs.html

```

        <input type="text" name="tb_password" >
        <input type="button" value="Login" onclick="checkLogin();" >
    </p>
</form>
<div id="text"></div>
</body>
</html>

```

Die Felder zur Texteingabe sind in ein *Form* eingebettet. Der eingegebene Text kann über das JavaScript-Kommando *document.forms[0].tb_username.value* ausgelesen werden. Die 0 gibt dabei an, dass es sich um das erste Form der Webseite handelt. Auf die Textbox für den Usernamen ist z.B. über *tb_username* verwiesen.

Die Auswertung der Eingaben erfolgt in der Funktion *checkLogin*, die über den *Login*-Button aufgerufen wird. Dazu wurde im HTML-Teil der Webseite für den Button der Parameter *onclick* genutzt.

Die Funktion *checkLogin* prüft, ob der Username „wut“ und das Passwort „webio“ eingegeben wurden und gibt in dem DIV-Element mit der *id*=„text“ den Begrüßungstext aus.



Wenn der Text in dem Passworteingabefeld nicht lesbar sein soll, muss im HTML-Quelltext der Type des Eingabefeldes von „text“ auf „password“ geändert werden.

```
<input type="text" name="tb_password" >
```

wird ersetzt durch:

```
<input type="password" name="tb_password" >
```

Im Browser sieht es dann so aus:

Passwort:



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter:
[www.wut.de/
 examples/
 12-htmljs.html](http://www.wut.de/examples/12-htmljs.html)



Das Beispiel soll in erster Linie den Umgang mit Usereingaben zeigen. Einen wirksamen Passwortschutz kann man so natürlich nicht aufbauen.

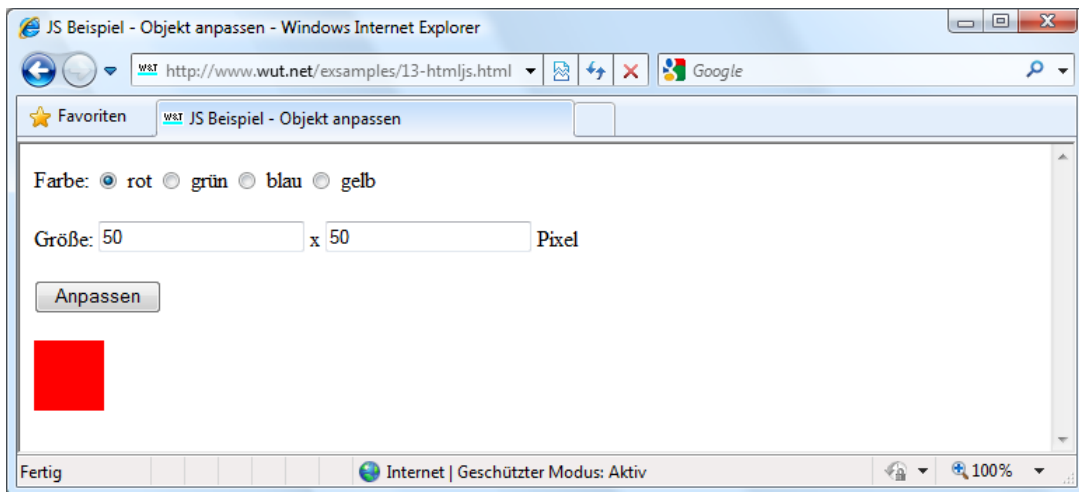
Eigenschaften eines HTML-Objektes nachträglich ändern

In den vorangegangenen Beispielen haben wir über die Vergabe einer ID die Möglichkeit geschaffen, den angezeigten Text eines Objektes auch nach dem Laden einer Webseite noch zu verändern.

An Hand des folgenden Beispiels soll gezeigt werden, dass sich auch fast alle anderen Eigenschaften von HTML-Objekten im Nachhinein ändern lassen.

Dazu erstellen wir eine Webseite mit einem Div-Objekt, dessen Farbe und Größe über Formulareingaben geändert werden soll.

Über Radiobuttons soll der Anwender die Hintergrundfarbe ändern können. Die Größe soll der Anwender in zwei Eingabefelder für Breite und Höhe schreiben können. Durch Klick auf den Anpassen-Button soll das Div-Objekt verändert werden.



```
<html>
<head>
  <title>JS Beispiel - Objekt anpassen</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <script language="JavaScript" type="text/javascript">

  function modify()
```

```

{  var divwidth = document.forms[0].tb_width.value;
    var divheight = document.forms[0].tb_height.value;
    var divcolor = [„red“, „lime“, „blue“, „yellow“];
    for(var i=0; i<4; i++)
    {  if (document.forms[0].rb_color[i].checked)
        {  document.getElementById('divlayer').style.backgroundColor = divcolor[i];
            }
        }
    document.getElementById('divlayer').style.height = divheight;
    document.getElementById('divlayer').style.width = divwidth;
}

</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
    <form name="form1">
        <p>Farbe:
        <input type="radio" name="rb_color" value="radiobutton" checked>
        rot
        <input type="radio" name="rb_color" value="radiobutton">
        gr&uuml;n
        <input type="radio" name="rb_color" value="radiobutton">
        blau
        <input type="radio" name="rb_color" value="radiobutton">
        gelb </p>
        <p>Gr&ouml;szlig;e:
        <input type="text" name="tb_height" maxlength="3" value="50">
        x
        <input type="text" name="tb_width" maxlength="3" value="50">
        Pixel</p>
        <p>
        <input type="button" value="Anpassen" onclick="modify();">
        </p>
    </form>
    <div id="divlayer" style="background-color:red;height:50px;width:50px"></div>
</body>
</html>

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/13-htmljs.html

Neu in diesem Beispiel ist der Parameter Style, der hier verwendet wird, um das Erscheinungsbild des Div-Objektes zu definieren. Nahezu alle Eigenschaften von Objekten können über Style

definiert werden (eine vollständige Liste findet sich im Anhang).

Dem Div-Objekt wurde in unserem Fall die Hintergrundfarbe rot und eine Größe von 50x50 Pixeln zugewiesen.

```
<div id="divlayer" style="background-color:red;height:50px;width:50px"></div>
```

Über die Style-Eigenschaften lässt sich das Erscheinungsbild von Objekten über JavaScript auch nachträglich verändern.

```
document.getElementById('divlayer').style.backgroundColor = divcolor[i];  
document.getElementById('divlayer').style.height = divheight;  
document.getElementById('divlayer').style.width = divwidth;
```

Schaut man genau hin, ist zu erkennen, dass sich die Schreibweise der Style-Parameter zwischen HTML und JavaScript zum Teil unterscheidet.

In HTML wird die Hintergrundfarbe mit *background-color* angegeben, in JavaScript heißt es *backgroundColor*, wobei die Groß-/Kleinschreibung wichtig ist. Die Unterschiede sind ebenfalls in der Liste im Anhang aufgezeigt.

In unserem Fall wird die Hintergrundfarbe abhängig vom aktivierten Radiobutton über ein Variablen-Array zugewiesen.

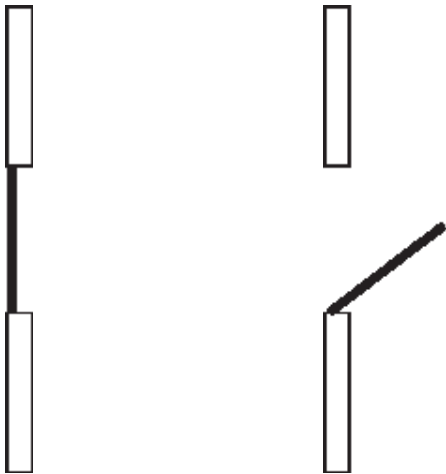
Mit zugeladenen Bildern arbeiten

Bilder sagen mehr als Worte. In der Praxis kann man z.B., angezeigt im Gebäudegrundriss, sofort den Überblick gewinnen, welche Tür gerade offen und welche geschlossen ist.

Diese Information einer Textliste zu entnehmen ist deutlich schwieriger.

Genau dieses Beispiel wollen wir aufgreifen.

Dazu werden zwei Bilder benötigt: eins der geschlossenen Tür und eins der offenen.

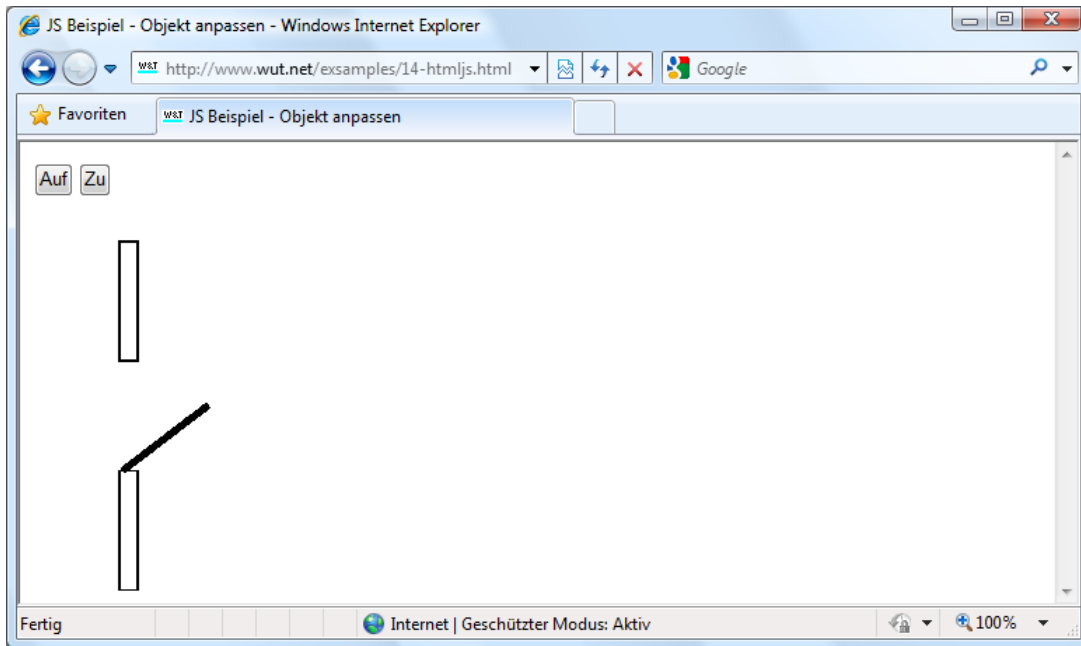


Um Bilder in einer Webseite anzuzeigen, wird das `<img>`-Tag genutzt. Der Trick um beide Zustände der Tür anzuzeigen zu können besteht darin, beide Bilder exakt übereinander zu legen.

Innerhalb des `<img>`-Tags wird dazu mit dem Parameter `Style` gearbeitet. Über den Style-Parameter `position: absolute` wird festgelegt, dass die Bilder an einer fest definierten Position angezeigt werden und nicht im Fluss der Seite mitlaufen (ähnlich wie im Fließtext z.B. bei Word).

Die Position selbst wird über die Style-Parameter `left` und `top` angegeben.

In unserem Beispiel soll der Anwender über zwei Buttons (Auf und Zu) festlegen, welcher Zustand angezeigt wird.



```
<html>
<head>
  <title>JS Beispiel - Objekt anpassen</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <script language="JavaScript" type="text/javascript">

    function openDoor()
    { document.getElementById('door_open').style.visibility = „visible“;
    }

    function closeDoor()
    { document.getElementById('door_open').style.visibility = „hidden“;
    }

  </script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
  <form name="form1">
    <input type="button" value="Auf" onclick="openDoor();">
    <input type="button" value="Zu" onclick="closeDoor();">
  </form>
  
  
</body>
</html>
```



Dieses Beispiel finden
Sie in der Software-
Sammlung zum Buch.
Download unter:
[www.wut.de/
examples/
14-htmljs.html](http://www.wut.de/examples/14-htmljs.html)

Ein Klick auf einen der beiden Buttons ruft jeweils eine Funktion auf, bei der die Eigenschaft *visibility* des *door_open.gif* Bildes entweder auf *hidden* (unsichtbar), wenn die Tür geschlossen angezeigt werden soll oder auf *visible* (sichtbar), wenn die Tür geöffnet ist.



Über die Styleparameter *height* und *width* kann auch die Anzeigegröße einer Bilddatei vorgegeben werden. Es wird aber empfohlen, Bilddateien gleich in der gewünschten Skalierung abzuspeichern und nicht nachträglich im Browser anzupassen. Das Grafik-Tool Gimp bietet entsprechende Funktionen!

Java-Applets für kontinuierlichen Datenaustausch

Die vorangegangenen Scripte haben gezeigt, wie verschiedene Elemente einer Webseite nachträglich geändert werden können.

In folgenden Beispielen soll diese Technik genutzt werden, um eine Webseite abhängig von Information zu aktualisieren, die nachträglich vom Server abgefragt bzw. empfangen werden.

Eine Möglichkeit, auch nach dem Laden einer Webseite weiterhin Daten mit dem Server auszutauschen, ist das Einbinden eines entsprechenden Java-Applets.

Java-Applets sind speziell für den Betrieb im Browser kompilierte Java-Programme.

Im ersten Beispiel sollen über einen Web-Thermohygro-Barographen die aktuelle Temperatur, die Luftfeuchtigkeit und der Luftdruck gemessen und selbstaktualisierend angezeigt werden.

Der erste Schritt besteht darin, das Applet im Body-Bereich der Webseite einzubinden. Das geschieht mittels des `<applet>`-Tags und einiger weiterer Parameter.

```
<html>
<head>
  <title>JS Beispiel - Objekt anpassen</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```
</head>
<body bgcolor="#FFFFFF" text="#000000">
  <applet name="Web Thermo-Hygrobarograph" archive="A.jar"
    code="A.class" height="0" width="0" mayscript>
    <param name="device" value="0">
    <param name="showerrors" value="off">
    <param name="sensorpolling" value="on">
    <param name="pollingrate" value="4000">
  </applet>
  .....
  .....
</body>
</html>
```

Beim Aufruf des Applets werden neben Standardparametern einige Web-IO-spezifische Parameter angegeben.

Zunächst die Standardparameter:

name

gibt an, unter welchem Namen das Applet unter JavaScript ansprechbar ist.

archiv und code

geben an, welches Applet geladen wird (ein Web-Server kann mehr als 1 Applet anbieten).

codebase

Dieser Parameter gibt die IP-Adresse des Servers (hier des Web-IO) an und ist nur dann nötig, wenn die Webseite nicht vom gleichen Server geladen wird wie das Java-Applet

mayscript

Nur wenn dem Applet-Aufruf *mayscript* hinzugefügt wurde, können JavaScript und Java-Applet Daten austauschen.

height und width

Java-Applets können verschiedenste Funktionen bereitstellen. Es gibt visuelle Applets, die im Browser etwas anzeigen - es gibt aber auch Applets, die nicht sichtbar im Hintergrund arbeiten. In diesem Fall werden die Parameter *height* und *width* auf 0 gesetzt.

Die Web-Thermograph-spezifischen Parameter sind in dieser Form natürlich auf das spezielle Applet im Web-IO ausgerichtet. Für andere Server bzw. Anwendungen können an dieser Stelle andere Parameter zum Tragen kommen:

device

Soll mit mehr als einem Web-IO gearbeitet werden, ist durch diesen Parameter eine Unterscheidung möglich.

showerrors

legt fest, ob auftretende Fehler vom Browser angezeigt werden.

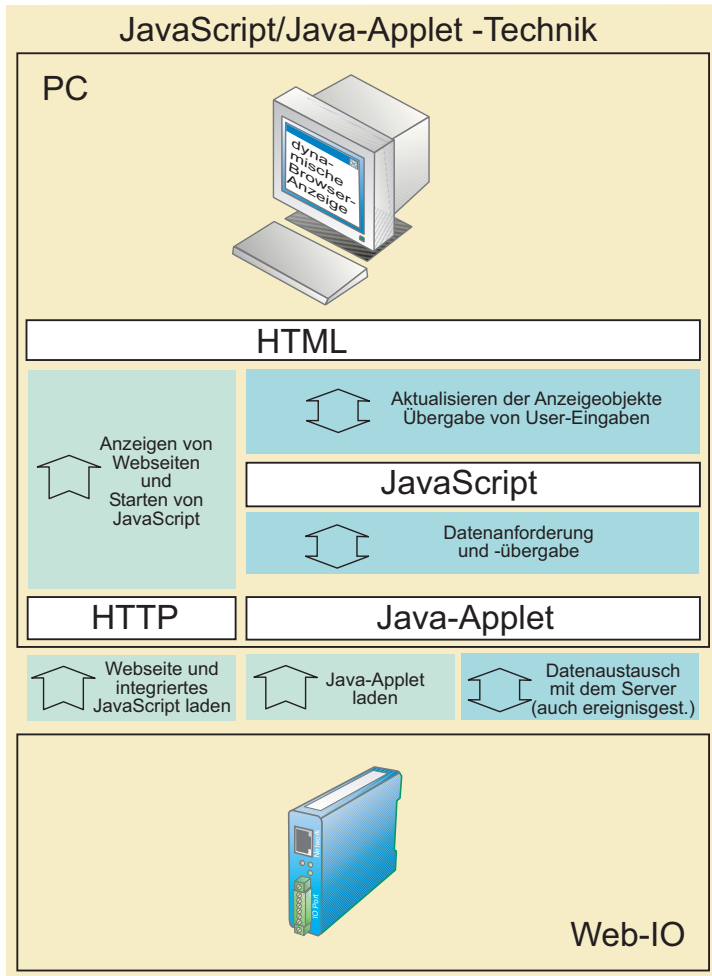
sensorpolling

Unter Polling versteht man das zyklische Abfragen von Daten. Mit dem o.g. Parameter wird festgelegt, ob die gemessenen Werte vom Applet automatisch abgefragt werden sollen.

pollingrate

legt den Abfragezyklus in ms fest .

Die folgende Grafik zeigt noch einmal den Ablauf bei der Arbeit mit dem Java-Applet des Web-Thermographen:



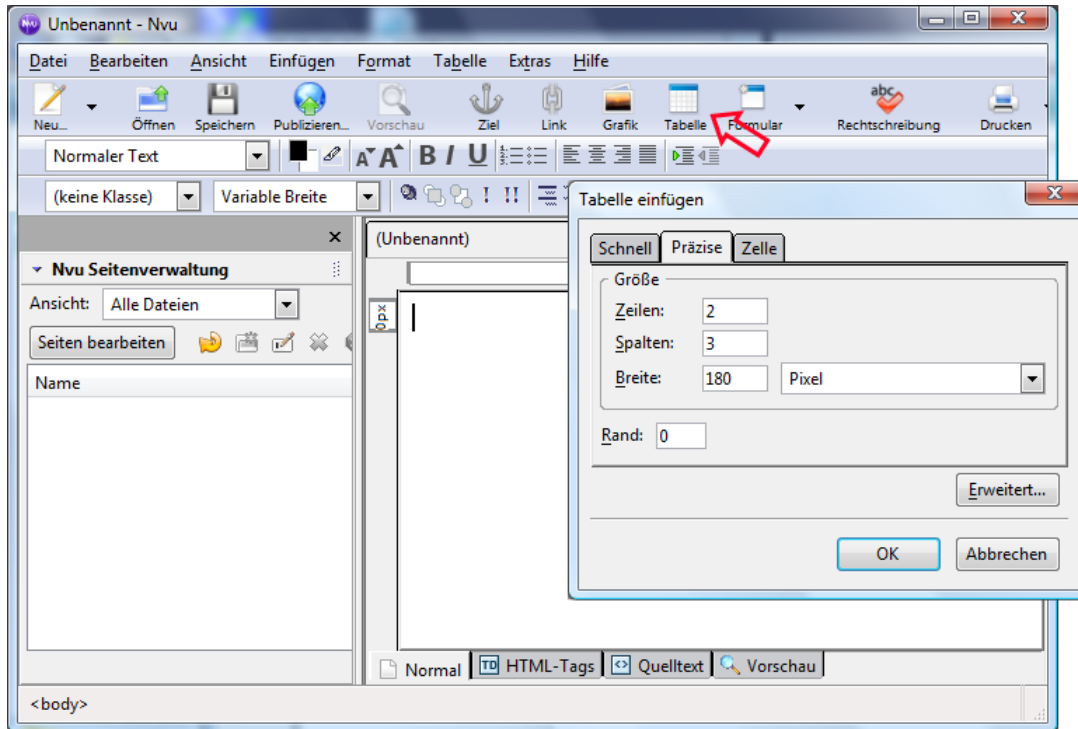
- Zunächst wird die Webseite vom Web-Thermographen in den Browser geladen.
- Der Browser arbeitet dabei Zeile für Zeile den Seiten Quelltext ab. Dabei stößt er auf das `<applet>`-Tag und lädt das Java-Applet aus dem Web-Thermographen.
- Anschließend ruft das Applet die Werte aus dem Web-Thermographen ab und übergibt sie über den Aufruf vorgegebener JavaScript-Funktionen weiter.

Für das aktuelle Beispiel soll eine Webseite mit einer Tabelle ausgestattet werden, die eine Zeile für die Beschriftung und eine weitere Zeile für die Werte hat. Dabei gibt es eine Spalte für die Temperatur, eine für die Luftfeuchtigkeit und eine für den Luftdruck.

Die Tabellenfelder für die Werte bekommen je eine eindeutige ID, damit mittels `document.getElementById` der Inhalt aktualisiert werden kann.



Das Erstellen von Tabellen mit HTML ist mit viel Schreibarbeit verbunden. Eine komfortable Alternative zum normalen Editor ist das WYSWYG-Tool NVU das bereits in der Einleitung vorgestellt wurde.



Sobald das HTML-Grundgerüst der Webseite mit NVU aufgebaut ist und es an den JavaScript-Teil der Seite geht, sollte wieder mit Aptana Studio weitergearbeitet werden, da hier JavaScript deutlich besser unterstützt wird.

Das mit NVU aufgebaute Grundgerüst sieht so aus.

```
<html>
  <head>
    <title>Java-Applet Beispiel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body bgcolor="#FFFFFF" text="#000000">
    <table style="text-align: center; width: 180px;" border="1">
      <tr>
        <td>Temperatur</td>
        <td>Luftfeuchte</td>
        <td>Luftdruck</td>
      </tr>
```

```
<tr>
  <td>--</td>
  <td>--</td>
  <td>--</td>
</tr>
</table>
</body>
</html>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/15-htmljs.html

Den Tabellenzellen, in denen die Werte ausgegeben werden sollen, müssen nun IDs hinzugefügt werden.

```
<td id="lt">--</td>
<td id="lf">--</td>
<td id="ld">--</td>
```

Außerdem muss das Java-Applet mit eingebunden werden. Wenn die Webseite nicht direkt aus dem Web-Thermographen geladen wird, muss der Parameter *codebase* gesetzt werden, damit das Applet weiss, von welcher IP-Adresse die Werte abgerufen werden sollen.

```
<applet name="Web Thermo-Hygrobarograph" archive="A.jar" code="A.class"
  codebase="http://10.40.22.18" height="0" width="0" mayscript>
  <param name="device" value="0">
  <param name="showerrors" value="off">
  <param name="sensorpolling" value="on">
  <param name="pollingrate" value="2000">
  Java ist nicht aktiviert oder wird nicht unterstützt
</applet>
```

Das Applet sendet im vorgegebenen Intervall die Werte für Temperatur, Luftfeuchte und Luftdruck. Dazu ruft das Java-Applet die JavaScript Funktion *sensorChanged* auf.



Die Funktion sensorChanged muss zwingend im JavaScript-Teil der Webseite vorhanden sein, wenn mit dem Applet des Web-Thermohygro-Barographen gearbeitet wird (Groß-/Kleinschreibung muss beachtet werden).

```
<script language="JavaScript" type="text/javascript">
```

```

function sensorChanged(iDevice, iSensor, iVal)
{
  if (iSensor == 0)
  {
    document.getElementById('lt').innerHTML = iVal + '°C';
  }
  if (iSensor == 1)
  {
    document.getElementById('lf').innerHTML = iVal + '%rel.';
  }
  if (iSensor == 2)
  {
    document.getElementById('ld').innerHTML = iVal + 'hPa';
  }
}
</script>

```

Mit dem Aufruf der sensorChange Funktion übergibt das Java-Applet 3 Parameter:

iDevice

Hier übergibt das Applet die Nummer, die als Applet-Parameter festgelegt wurde.

iSensor

0 = Temperatur
 1 = Luftfeuchte
 2 = Luftdruck

iVal

beinhaltet den aktuellen Wert des in iSensor angegebenen Sensors.

Hier noch einmal der komplette Seitenquelltext:

```

<html>
<head>
  <title>Java-Applet Beispiel</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

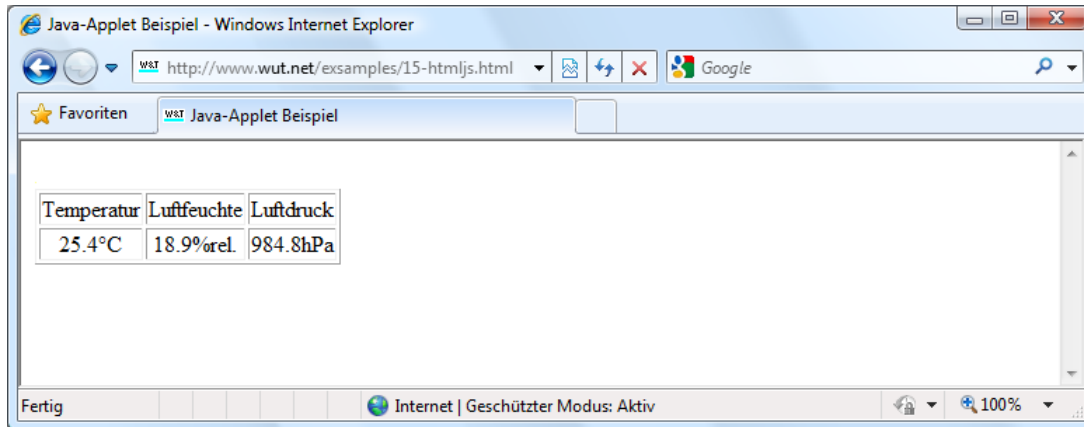
```

```
<script language="JavaScript" type="text/javascript">

function sensorChanged(iDevice, iSensor, iVal)
{
  if (iSensor == 0)
  {
    document.getElementById('lt').innerHTML = iVal + '°C';
  }
  if (iSensor == 1)
  {
    document.getElementById('lf').innerHTML = iVal + '%rel.';
  }
  if (iSensor == 2)
  {
    document.getElementById('ld').innerHTML = iVal + 'hPa';
  }
}

</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
  <applet name="Web Thermo-Hygrobarograph" archive="A.jar" code="A.class"
    codebase="http://10.40.22.18" height="0" width="0" mayscript>
    <param name="device" value="0">
    <param name="showerrors" value="off">
    <param name="sensorpolling" value="on">
    <param name="pollingrate" value="2000">
    Java ist nicht aktiviert oder wird nicht unterstützt
  </applet>
  <table style="text-align: center; width: 180px;" border="1">
    <tr>
      <td>Temperatur</td>
      <td>Luftfeuchte</td>
      <td>Luftdruck</td>
    </tr>
    <tr>
      <td id="lt">--</td>
      <td id="lf">--</td>
      <td id="ld">--</td>
    </tr>
  </table>
</body>
</html>
```

Mit einer *if*-Abfrage auf *iSensor* ermittelt die Funktion *sensorChanged* ob die Temperatur, die Luftfeuchte oder der Luftdruck übergeben werden. Über *getElementById* wird dann der Wert in das entsprechende Tabellenfeld geschrieben.



Digitale Signale mit einem Java-Applet überwachen

Dieses Beispiel soll die Status der 12 Inputs, 12 Counter und 12 Outputs eines Web-IO 12xDigital in einer Tabelle dynamisch anzeigen.

Dazu muss zunächst eine entsprechende Tabelle erstellt werden. Am schnellsten geht das, wie bereits im vorangegangenen Beispiel mit NVU.

21px 31px 31px 49px 42px 31px					
Nr.	State	Counter	State		
0	Input OFF	0	Output OFF		
1	Input OFF	0	Output OFF		
2	Input OFF	0	Output OFF		
3	Input OFF	0	Output OFF		
4	Input OFF	0	Output OFF		
5	Input OFF	0	Output OFF		
6	Input OFF	0	Output OFF		
7	Input OFF	0	Output OFF		
8	Input OFF	0	Output OFF		
9	Input OFF	0	Output OFF		
10	Input OFF	0	Output OFF		
11	Input OFF	0	Output OFF		

Der Quelltext wird dann wie gehabt in Aptana Studio kopiert.

```
<html>
  <head>
    <title>Java-Applet Beispiel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body bgcolor="#FFFFFF" text="#000000">
    <table style="text-align: left; width: 257px; height: 368px;" border="1">
      <tr>
        <td>Nr.</td>
        <td>&nbsp;</td>
        <td>State</td>
        <td>Counter</td>
        <td>&nbsp;</td>
        <td>State</td>
      </tr>
      .....
      .....
      <tr>
        <td>11</td>
        <td>Input</td>
        <td>OFF</td>
        <td>0</td>
        <td>Output</td>
        <td>OFF</td>
      </tr>
    </table>
  </body>
</html>
```



*Dieses Beispiel finden
Sie in der Software-
Sammlung zum Buch.
Download unter:
[www.wut.de/
examples/
16-htmljs.html](http://www.wut.de/examples/16-htmljs.html)*

Um die Werte nachträglich in die Tabelle schreiben zu können wäre es eine Möglichkeit, jeder Zelle eine individuelle ID zuzuweisen. Der Aufwand wäre allerdings erheblich (36 ID anlegen und später verwalten).

Deshalb weisen wir der gesamten Tabelle eine einzige ID zu.

```
<table ID="ioTable" style="text-align: left; width: 257px; height: 368px;" border="1">
```

Wie die einzelnen Zellen angesprochen werden, wird im Java-Script-Teil beschrieben.

Als nächstes muss das Java-Applet des Web-IO in die Webseite eingebunden werden.

```
<body bgcolor="#FFFFFF" text="#000000">
  <applet name="dio0" archive="dio.jar" code="dio.class" codebase="http://10.40.22.101"
    width="0" height="0" mayscript>
    <param name="device" value="0">
    <param name="showerrors" value="off">
    <param name="inputpolling" value="on">
    <param name="outputpolling" value="on">
    <param name="counterpolling" value="on">
    <param name="pollingrate" value="1000">
  </applet>
  <table style="text-align: left; width: 257px; height: 368px;" border="1">
```

Die grundsätzlichen Parameter des Applets entsprechen dem gleichen Schema wie beim Web-Thermographen.

Die gerätespezifischen Parameter sind den Bedürfnissen des Web-IO Digital angepasst.

device

Soll mit mehr als einem Web-IO gearbeitet werden, ist durch diesen Parameter eine Unterscheidung möglich.

showerrors

legt fest, ob auftretende Fehler vom Browser angezeigt werden.

inputpolling (outputpolling und counterpolling)

Unter Polling versteht man das zyklische Abfragen von Daten. Mit dem o.g. Parameter wird festgelegt, welche Werte (Inputs, Outputs oder Counter) vom Applet automatisch abgefragt werden sollen.

pollingrate

legt den Abfragezyklus in ms fest.

Nun müssen im head-Bereich der Seite die JavaScript- Funktionen eingefügt werden, die das Java-Applet bei Statusänderung aufruft.

Beim Aufruf der Funktion `inputChanged` (bitte Groß-/Kleinschreibung beachten) werden drei Werte übergeben:

iDevice

ist die Nummer, die beim Aufruf des Applets als Parameter *device* übergeben wurde. Immer dann, wenn mehr als ein Applet benutzt wird, kann anhand dieses Wertes identifiziert werden, von welchem Web-IO die Funktion aufgerufen wurde.

iNr

gibt an, bei welchem Input sich der Status geändert hat.

iVal

übergibt den Status des entsprechenden Inputs.

true = ON

false = OFF

Hier die `inputChanged` Funktion für unsere Webseite:

```
function inputChanged( iDevice, iNr, iVal )
{
  table = document.getElementById('ioTable');
  if(iVal == true)
  {
    table.rows[iNr+1].cells[2].innerHTML = 'ON';
    table.rows[iNr+1].cells[2].style.backgroundColor = '#00FF00';
  }
  else
  {
    table.rows[iNr+1].cells[2].innerHTML = 'OFF';
    table.rows[iNr+1].cells[2].style.backgroundColor = '#CCCCCC';
  }
}
```

Der Parameter `iDevice` wird in diesem Beispiel nicht ausgewertet, da die Webseite ohnehin nur die Status eines Web-IO visualisiert.

Über `iNr` wird übergeben, welcher Input seinen Status übergibt. Im Gegensatz zum vorangegangenen Beispiel, wo jede Tabellenzeile eine eigene ID hatte, gibt es hier für die gesamte Tabelle nur eine ID.

Über

```
table = document.getElementById('ioTable');
```

wird ein Delegat der Tabelle geschaffen.

Die Auswahl der Zellen, welche verändert werden sollen, erfolgt dann über die *rows* (Zeilen) und *cells* (Spalten) Eigenschaften des Tabellendelegates. Die Nummerierung beginnt mit 0 für die erste Zeile bzw. Spalte und wird in eckigen Klammern angefügt.

Über eine if-Abfrage wird der Status des Inputs ermittelt und entsprechend ON oder OFF in die Zelle eingetragen. Ferner wird der Hintergrund abhängig von *iVal* grün oder grau gefärbt.

Die Funktionen für die Outputs und Counter sind in der gleichen Weise aufgebaut, wobei übergebene Counter-Werte direkt in die Zellen eingetragen werden.

Der Seiten Quelltext sieht dann so aus:

```
<html>
<head>
  <title>Java-Applet Beispiel</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <script language="JavaScript" type="text/javascript">

  function inputChanged( iDevice, iNr, iVal )
  { table = document.getElementById('ioTable');
    if (iVal == true)
    { table.rows[iNr + 1].cells[2].innerHTML = 'ON';
      table.rows[iNr + 1].cells[2].style.backgroundColor = '#00FF00';
```

```
    }  
    else  
    { table.rows[iNr + 1].cells[2].innerHTML = 'OFF';  
      table.rows[iNr + 1].cells[2].style.backgroundColor = '#CCCCCC';  
    }  
  }  
}  
  
function counterChanged( iDevice, iNr, iVal )  
{ table = document.getElementById('ioTable');  
  table.rows[iNr+1].cells[3].innerHTML = iVal;  
}  
  
function outputChanged( iDevice, iNr, iVal )  
{ table = document.getElementById('ioTable');  
  if(iVal == true)  
  { table.rows[iNr+1].cells[5].innerHTML = 'ON';  
    table.rows[iNr+1].cells[5].style.backgroundColor = '#00FF00';  
  }  
  else  
  { table.rows[iNr+1].cells[5].innerHTML = 'OFF';  
    table.rows[iNr+1].cells[5].style.backgroundColor = '#CCCCCC';  
  }  
}  
}  
  
</script>  
</head>  
<body bgcolor="#FFFFFF" text="#000000">  
  <applet name="dio0" archive="dio.jar" code="dio.class" codebase="http://10.40.22.101"  
width="0" height="0" mayscript>  
    <param name="device" value="0">  
    <param name="inputpolling" value="on">  
    <param name="outputpolling" value="on">  
    <param name="counterpolling" value="on">  
    <param name="pollingrate" value="1000">  
  </applet>  
  <table ID="ioTable" style="text-align: left; width: 257px; height: 368px;" border="1">  
  <tr>  
    <td>Nr.</td>  
    <td>&nbsp;</td>  
    <td>State</td>  
    <td>Counter</td>
```

```

        <td>&nbsp;</td>
        <td>State</td>
    </tr>
    <tr>
        <td>0</td>
        <td>Input</td>
        <td>OFF</td>
        <td>0</td>
        <td>Output</td>
        <td>OFF</td>
    </tr>
    .....
    .....
    <tr>
        <td>11</td>
        <td>Input</td>
        <td>OFF</td>
        <td>0</td>
        <td>Output</td>
        <td>OFF</td>
    </tr>
</table>

</body>
</html>

```

Und hier das Ergebnis im Browser:

Nr.	Input	State	Counter	Output	State
0	Input	OFF	135	Output	OFF
1	Input	OFF	126	Output	ON
2	Input	OFF	126	Output	OFF
3	Input	OFF	126	Output	OFF
4	Input	OFF	126	Output	OFF
5	Input	OFF	126	Output	ON
6	Input	OFF	126	Output	OFF
7	Input	ON	126	Output	OFF
8	Input	OFF	126	Output	ON
9	Input	OFF	125	Output	OFF
10	Input	OFF	125	Output	OFF
11	Input	OFF	125	Output	OFF

Digitale Outputs über ein Java-Applet steuern

Bis jetzt haben wir das Java-Applet des Web-IO nur dazu genutzt, den Status der IOs anzufordern und zu visualisieren.

Das Java-Applet bietet aber auch die Möglichkeit, die Outputs ein- oder auszuschalten. Dazu muss JavaScript-Funktionen des Java-Applets aufrufen (bisher war es immer umgekehrt).

Der Aufruf einer Java-Appletfunktion erfolgt über das Kommando:

```
document.applets[Appletname].Applet-Funktion(Parameter);
```

In diesem Beispiel soll es für jeden der Outputs zwei Buttons (ON, OFF) geben. Dazu werden der Tabelle aus dem vorangegangenen Beispiel zwei Spalten angefügt.

```
<table ID="ioTable" style="text-align: left; width: 257px; height: 368px;" border="1">
  <tr>
    <td>Nr.</td>
    <td>&nbsp;</td>
    <td>State</td>
    <td>Counter</td>
```

```

        <td>&nbsp;</td>
        <td>State</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>
    <tr>
        <td>0</td>
        <td>Input</td>
        <td>OFF</td>
        <td>0</td>
        <td>Output</td>
        <td>OFF</td>
        <td><input onclick="setOutput(0,1);" type="button" value="ON"></td>
        <td><input onclick="setOutput(0,0);" type="button" value="OFF"></td>
    </tr>
    .....
    .....
    <tr>
        <td>11</td>
        <td>Input</td>
        <td>OFF</td>
        <td>0</td>
        <td>Output</td>
        <td>OFF</td>
        <td><input onclick="setOutput(11,1);" type="button" value="ON"></td>
        <td><input onclick="setOutput(11,0);" type="button" value="OFF"></td>
    </tr>
</table>

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/17-htmljs.html

Über das `<input>`-Tag wird dann in die neuen Zellen jeweils ein Button eingefügt. Als Einsprungfunktion wird für *onclick* die Funktion *setOutput* mit zwei Parametern eingetragen. Der erste Parameter steht für die Output-Nr, der zweite legt fest, in welchen Zustand der Output geschaltet werden soll (0=OFF, 1=ON).

Im JavaScript-Teil muss die Funktion *setOutput* angelegt werden.

```

function setOutput(iNr, iVal)
{
    var Mask = Math.pow( 2, iNr);

```

```
var State= 0x0fff * iVal;  
document.applets[„dio0“].outputAccess(Mask, State);  
}
```

Um das Java-Applet dazu zu veranlassen, einen Output zu schalten, wird folgendes Kommando aufgerufen:

```
document.applets[„dio0“].outputAccess(Mask, State);
```

Als Parameter werden die Variablen *Mask* und *State* übergeben.

Mask ist ein Zahlenwert, der die binäre Entsprechung des Output enthält. *Mask* legt fest, welcher bzw. welche Output(s) verändert werden sollen. Auch wenn in diesem Beispiel nur jeweils ein Output gesetzt wird, lässt das Applet auch das Setzen mehrerer Outputs gleichzeitig zu.

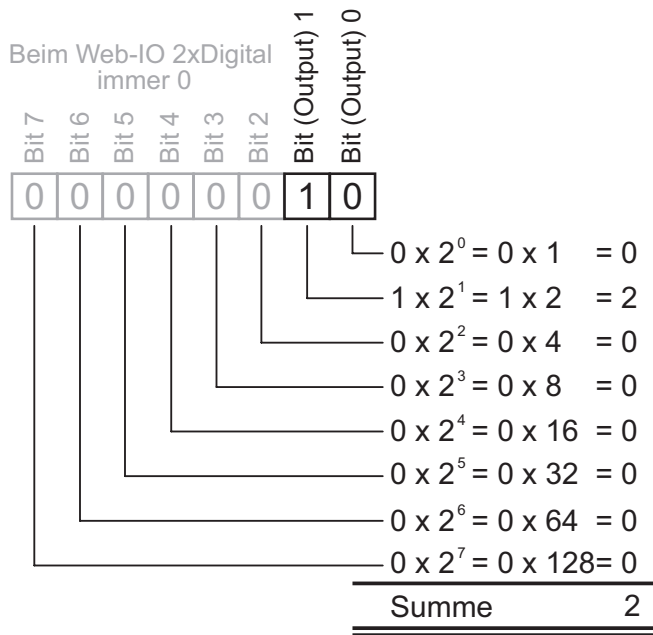
Das klingt zunächst kompliziert, ist aber eigentlich ganz einfach.

In der Computertechnik wird mit Bits und Bytes gearbeitet.

Ein Byte besteht aus 8 Bit. Jedes Bit kann zwei Zustände einnehmen: 1 oder 0 bzw. gesetzt oder nicht gesetzt bzw. ON oder OFF.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	0

Die einzelnen Bits stehen je für einen Output. Hinter jedem Bit verbirgt sich außerdem eine Stelle des dualen Zahlensystems.



Gerechnet wird also mit 2er-Potenzen, wobei die Nummer des Bits bzw. Outputs den Exponenten bildet. Addiert man die Potenzen der gesetzten Bits, erhält man den dezimalen Wert, der dem Bitmuster entspricht.

```
var Mask = Math.pow( 2, iNr);
```

Da in unserem Beispiel nur ein Output bzw. Bit gesetzt werden soll, entspricht der Wert genau dieser Zweierpotenz

In diesem Fall also *Output;2*

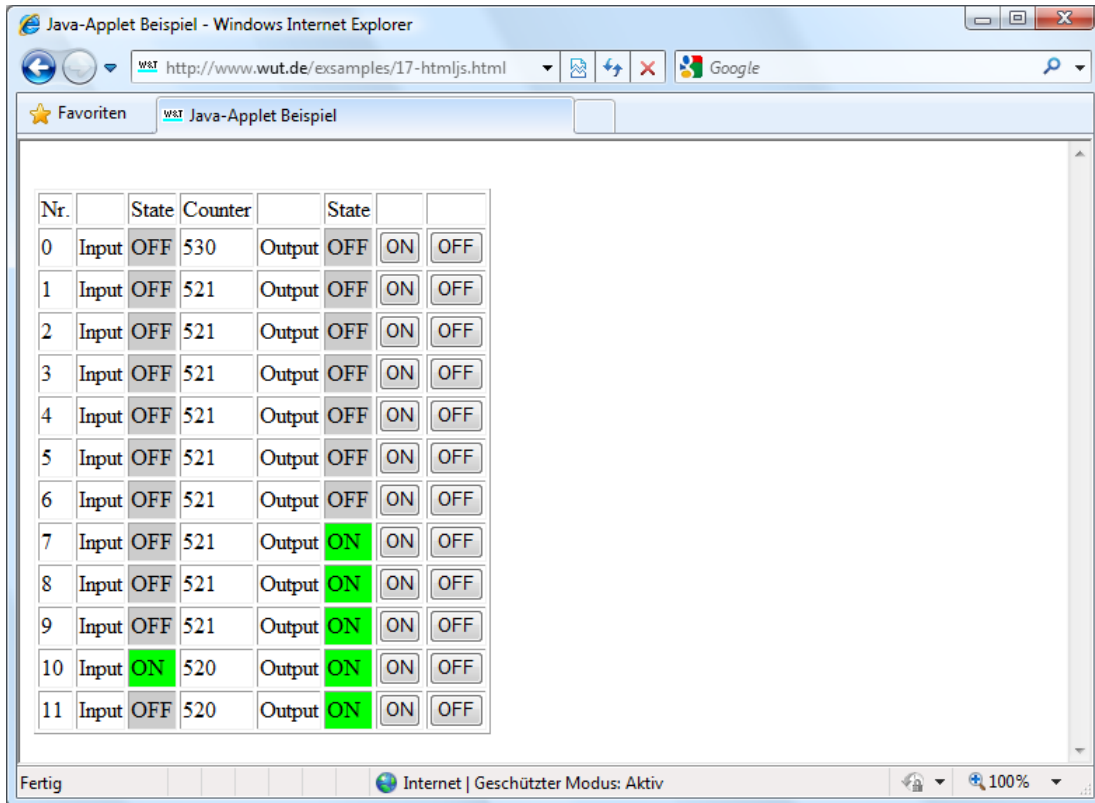
Für das 12fach Web-IO reicht ein Byte nicht aus. Es muss mit zwei Bytes gearbeitet werden, damit auch die Outputs 8-11 gesetzt werden können.

State gibt an, welchen Zustand der/die Output(s) einnehmen sollen. Dazu werden die entsprechenden Bits auf 0 für OFF bzw. 1 für ON gesetzt.

In unserem Beispiel werden alle Bits von *State* entweder auf 0 oder auf 1 gesetzt, da über Mask ohnehin festgelegt wird, welcher Output zu schalten ist.

```
var State= 0x0fff * iVal;
```

Wenn auf einen Schlag mehrere Outputs in verschiedene Zustände geschaltet werden sollen, muss der Wert für *State* individuell berechnet werden.



Neben den Funktionen für das Setzen der Outputs und das Lesen der IOs und Counter verfügt das Java-Applet des Web-IO Digital noch über eine Funktion, um die Counter auf 0 bzw. einen Startwert zu setzen

```
document.applets[Applettname].counterClear(Counterwert);
```

und über eine Funktion, um das Web-IO Systempasswort zu übergeben

```
document.applets[Appletname].setPassword(Passwort);
```

Die Übergabe des Passwortes ist immer dann notwendig, wenn dem Web-IO ein Administrator bzw. Operatorpasswort zugewiesen wurde.

Damit bieten die Java-Applets der Web-IO Geräte alle Voraussetzungen, dem Anwender lebendige Webseiten anzubieten.

Voraussetzung ist allerdings, dass der Browser die Java-Laufzeitkomponenten installiert und freigeschaltet hat.

Obwohl Java-Applets heute nicht mehr als Sicherheitsrisiko eingestuft werden, gibt es Firmen - und Behördennetzwerke, in denen die Nutzung und Installation der Java-Unterstützung nicht gestattet ist.

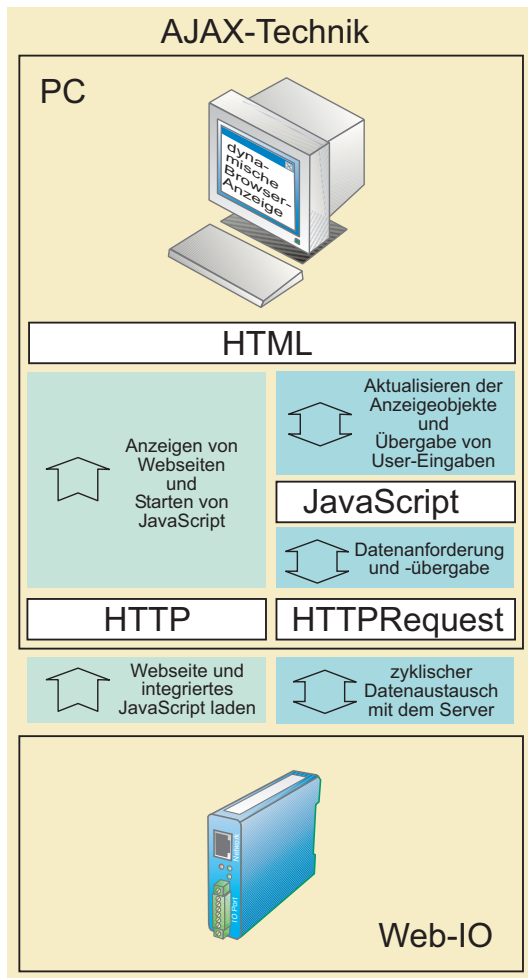
AJAX - lebendige Webseiten ohne JavaApplets

AJAX, das steht für Asynchronus JavaScript and XML und gewinnt bei der Gestaltung dynamischer Webseiten zunehmend mehr Popularität.

Das Kernstück von AJAX ist die in aktuellen Versionen von JavaScript angebotene HTTP-Request Methode. Hiermit kann JavaScript auch nach dem Laden und Anzeigen einer Webseite Daten vom Web-Server anfordern. Daten können, wenn es der Server unterstützt, im XML-Format angefordert werden. Alternativ wird aber auch die Übertragung von Textformaten unterstützt.

Im Gegensatz zur im vorangegangenen Abschnitt beschriebenen Java-Applet-Technik kommt AJAX also ohne das Laden zusätzlicher Applets aus. JavaScript übernimmt sowohl das nachträgliche Holen der Daten, als auch die Aktualisierung der Browser-Anzeige.

Allerdings kann JavaScript keine dauerhafte Verbindung zum Web-Server aufrechterhalten. Es kann aber durch zyklisches Nachladen von Prozessdaten eine selbstaktualisierende Prozessvisualisierung im Browser realisiert werden.



Als erstes Beispiel für die Verwendung von AJAX sollen wieder Temperatur, Luftfeuchte und Luftdruck, die mit einem Web-Thermohygro-Barographen ermittelt wurden, abgerufen werden.

Als Einstieg beschränken wir uns ausschließlich auf die Kommunikation mit dem Web-Thermographen. Die vom Web-Thermographen empfangenen Daten sollen einfach in einer Alert-box ausgegeben werden.

Dazu wird zunächst eine Webseite mit minimaler Ausstattung benötigt.

```
<user.htm>
<html>
  <head>
    <title>AJAX Beispiel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```

</head>
<body>

</body>
</html>

```



*AJAX benutzt das XMLHttpRequest Objekt von JavaScript.
Ein nachträglicher XMLHttpRequest kann nur an den Server gesendet werden, von dem die Webseite geladen wurde.*

Deshalb muss die erstellte Webseite in den Web-Thermographen geladen werden. Damit dieser die Seite akzeptiert, wird im Seitenquelltext das W&T spezifische <user.htm>-Tag vorangestellt. Wird die Seite vom Web-Thermographen in den Browser geladen, filtert der Web-Thermograph dieses Tag aus.

Im Kopf der Webseite wird ein JavaScriptteil mit zwei Funktionen eingefügt:

```

<script language="JavaScript" type="text/javascript">

var Command = 'single';
DataRequest(Command)

function DataRequest (SendString)
{
    var xmlHttp;
    if( window.ActiveXObject )           // Internet Explorer
    {
        xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
    }
    else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
    {
        xmlHttp = new XMLHttpRequest();
    }
    if (xmlHttp)
    {
        xmlHttp.onreadystatechange = function()
        {
            if (xmlHttp.readyState == 4)
            {
                if (xmlHttp.status == 200)
                {
                    updateDisplay(xmlHttp.responseText);
                    xmlHttp=null;
                }
            }
        }
    }
}

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter:
www.wut.de/examples/18-ajax.html

```
xmlHttp.open(„GET“, sendstring, true);  
xmlHttp.setRequestHeader(„Connection“, „close“);  
xmlHttp.setRequestHeader(„If-Modified-Since“, „Thu, 1 Jan 1970 00:00:00 GMT“);  
xmlHttp.send(null);  
}  
}  
  
function updateDisplay(Receivetext)  
{ alert(Receivetext)  
}  
  
</script>
```

Über die Funktion *DataRequest* kann ein Kommando zum Web-Thermographen gesendet werden.

Das Kommando zum Abrufen der aktuellen Werte lautet *single* und wird mit der Variablen *Command* übergeben.

Die Funktion *DataRequest* legt als erstes eine Instanz des *xmlHttp* Objektes an. Leider gibt es hier bei den verschiedenen Browsern ein unterschiedliches Vorgehen.



Um ein xmlHTTP Objekt anzulegen, nutzt der Internet Explorer das Kommando `new ActiveXObject("Microsoft.XMLHTTP")`. Firefox, Opera und Safari nutzen `new XMLHttpRequest()`.

Über eine If-Abfrage wird geprüft, ob ActiveX zur Verfügung steht. Wenn ja, handelt es sich um den Internetexplorer, und *xmlHTTP* wird als ActiveX Objekt angelegt.

Im nächsten Schritt wird mit

```
xmlHttp.onreadystatechange = function()
```

festgelegt, was geschehen soll, wenn sich der Status des *xmlHTTP*-Objektes ändert.

Mit

```
if(xmlHttp.readyState == 4)
```

wird geprüft, ob kein Fehler vorliegt und über

```
if(xmlHttp.status == 200)
```

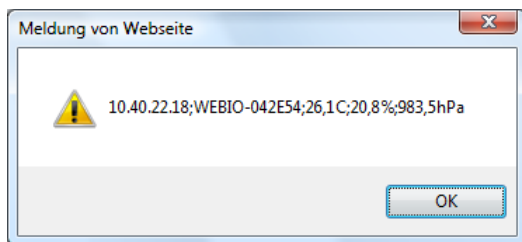
wird sichergestellt, dass die Status-Änderung durch Datenempfang ausgelöst wurde.

Die empfangenen Daten werden mit *xmlHttp.responseText* übergeben und an die Funktion *updateDisplay* weitergeleitet.

Abschließend wird die Instanz von *xmlHTTP* gelöscht, damit der Speicher nicht vollläuft

```
xmlHttp=null;
```

In der Funktion *updateDisplay* werden die Empfangsdaten dann einfach in einer Alert-Box zur Anzeige gebracht.

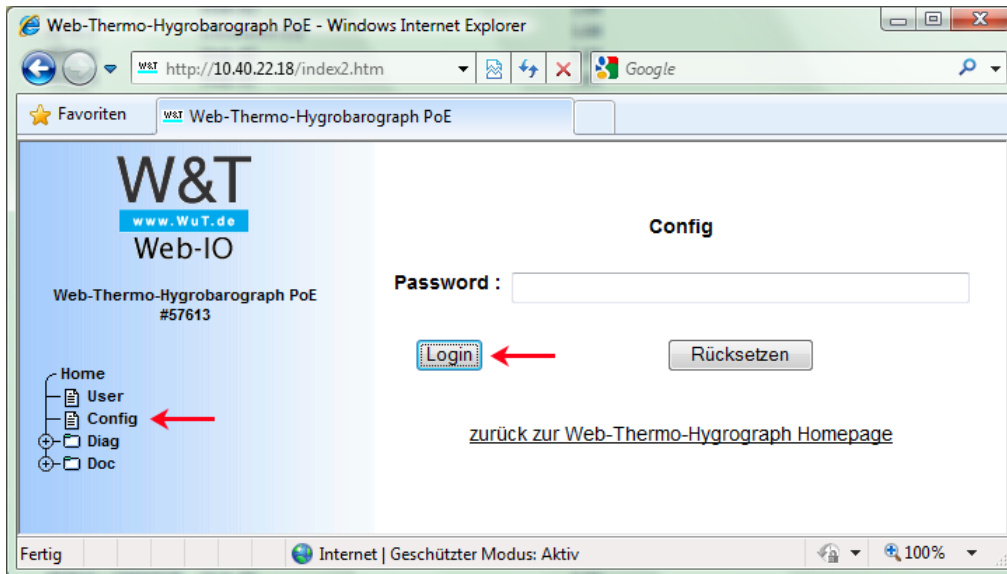


Als Antwort kommt vom Web-Thermohygro-Barographen ein String, der sich mit Semikolon getrennt aus IP-Adresse, Gerätenamen, Temperatur, Luftfeuchte und Luftdruck zusammensetzt.

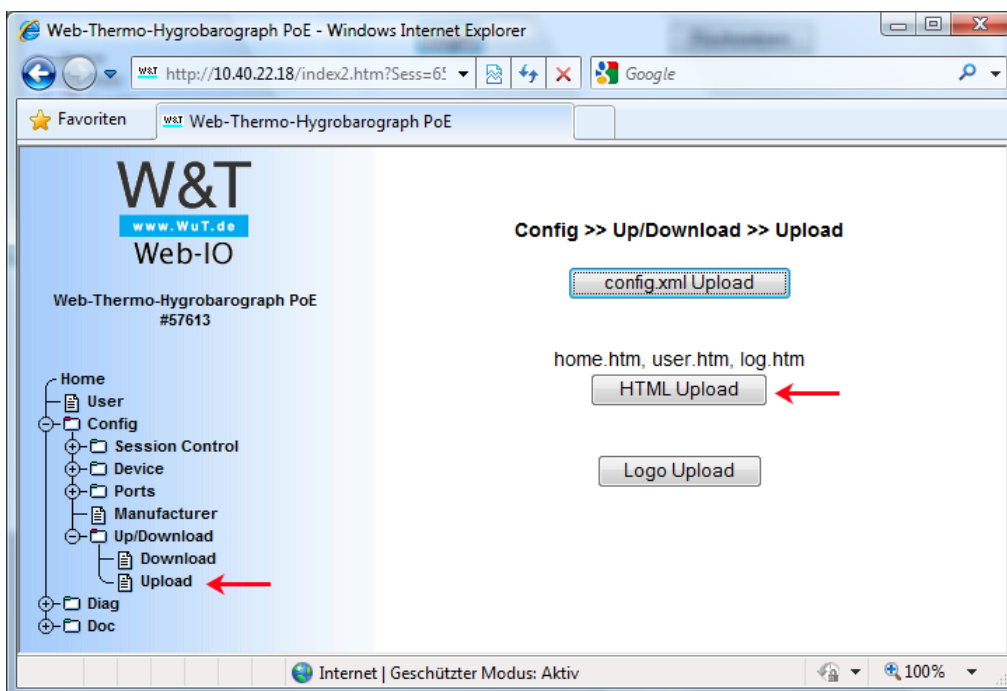
Damit haben wir eine universell einsetzbare Funktion, um Daten vom Server (hier der Web-Thermograph) nachzuladen.

Um die erstellte Webseite hochzuladen, öffnen Sie im Browser die Startseite des Web-Thermohygro-Barographen.

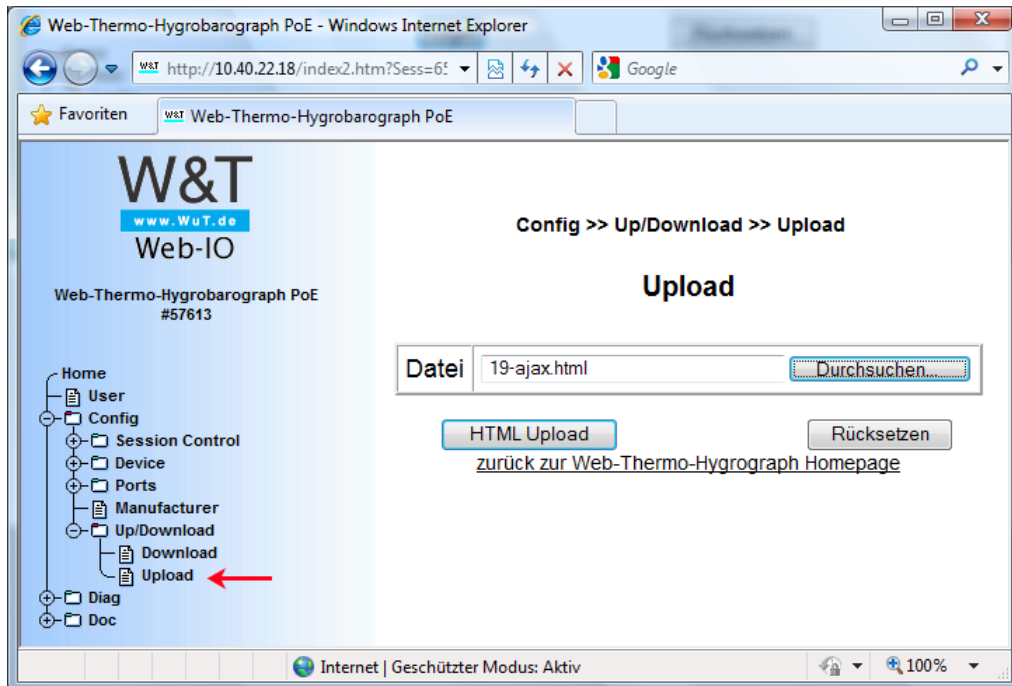
Wählen Sie im Navigationsbaum *Config* und loggen Sie sich als Administrator ein.



Im anschließend erscheinenden Fenster wählen Sie *Experten Modus*. Danach gehen Sie im Menübaum auf *Config >> Upload/Download >> Upload*.



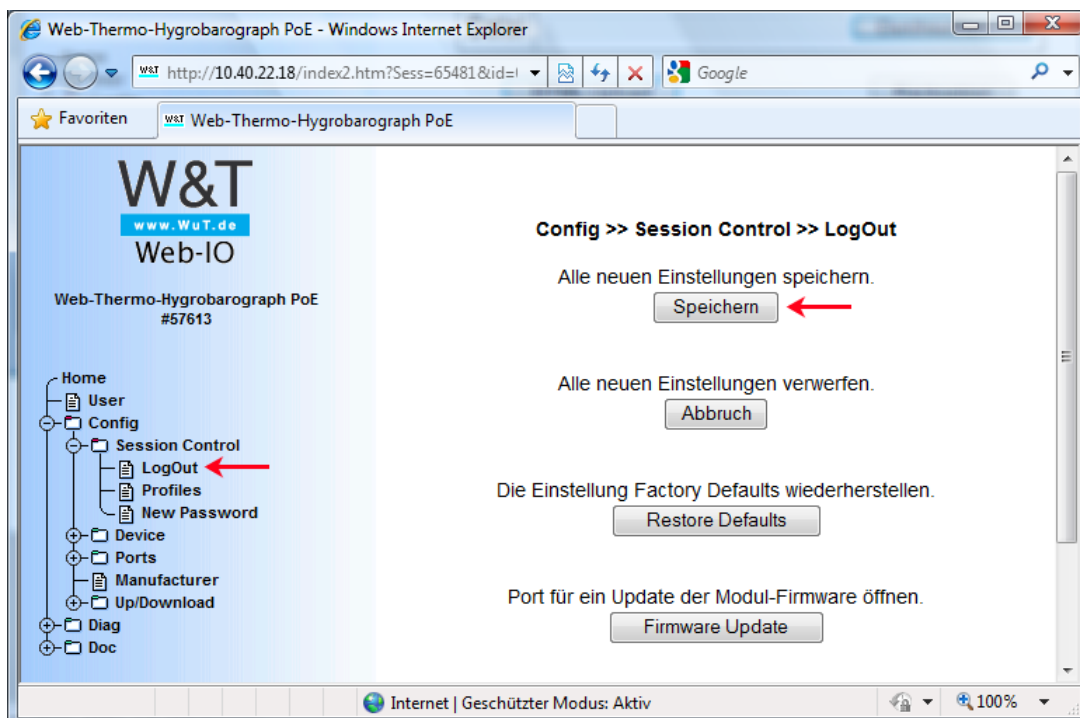
Klicken Sie auf den *HTML-Upload* Button.



Wählen Sie den Dateinamen der von Ihnen erstellten User-Web-seite aus und klicken Sie auf HTML-Upload.

Das Gerät antwortet mit *HTML Upload: Syntax Check Passed! HTML File geladen.*

Bestätigen Sie mit *OK und* wählen Sie im Menübaum *Config >> Session Control >> Logout* und klicken auf *Speichern*.

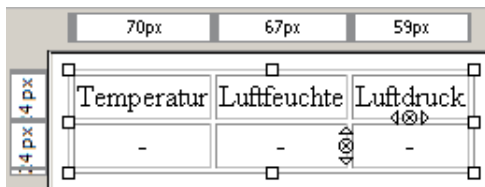


Die Webseite kann dann im Menübaum über User abgerufen werden.

Klimadaten mit AJAX visualisieren

Im folgenden Beispiel sollen die Klimadaten in einzelne Zellen einer Tabelle geschrieben werden.

Der Quelltext für die Tabelle kann wieder mit NVU erzeugt und in Aptana eingefügt werden.



70px	67px	59px
Temperatur	Luftfeuchte	Luftdruck
-	-	-

```
<table id="klima" style="text-align: center; width: 180px;" border="1">
  <tr>
    <td>Temperatur</td>
    <td>Luftfeuchte</td>
    <td>Luftdruck</td>
  </tr>
  <tr>
    <td>--</td>
    <td>--</td>
    <td>--</td>
  </tr>
</table>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/19-ajax.html

Der Tabelle wird im `<table>`-Tag eine ID vergeben.

Die JavaScript Funktion *updateDisplay* aus dem vorangegangenen Beispiel muss nun so geändert werden, dass die einzelnen Messwerte in die entsprechenden Zellen geschrieben werden.

```
function updateDisplay(Receivetext)
{
  var table = document.getElementById('klima');
  var Receivedata = Receivetext.split(';');
  alert(Receivedata[Receivedata.length - 3]);
  table.rows[1].cells[0].innerHTML = Receivedata[Receivedata.length - 3];
  table.rows[1].cells[1].innerHTML = Receivedata[Receivedata.length - 2];
  table.rows[1].cells[2].innerHTML = Receivedata[Receivedata.length - 1];
}
```

```
}
```

Dazu muss der Empfangsdatenstring in die einzelnen, mit Semikolon getrennten Blöcke getrennt werden.

Zur Erinnerung: Die Antwort des Web-Thermographen auf das Kommando *single* besteht aus IP-Adresse, Device-Namen, Temperatur, Luftfeuchte und Luftdruck.

```
10.40.22.18;WEBIO-042E54;24,7°C;22,2%;983,2hPa
```

Mit dem Kommando

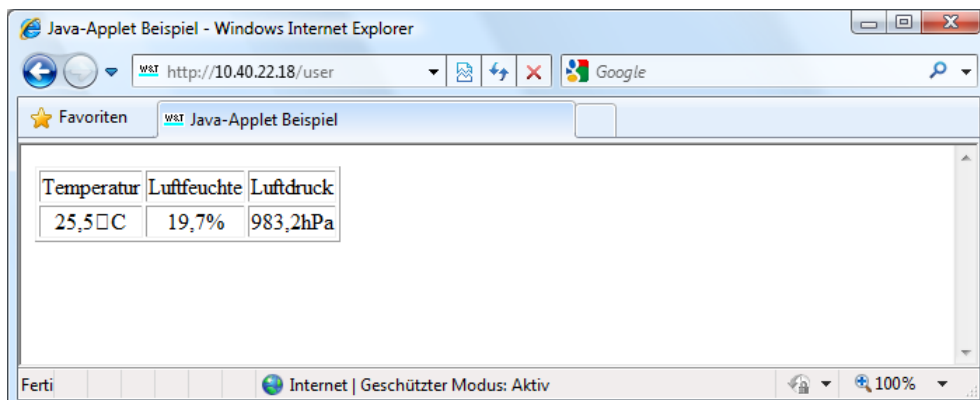
```
var Receivedata = Receivetext.split(';');
```

werden die einzelnen Blöcke von *Receivetext* als Array in *Receivedata* geschrieben.

Über die Variable *table* wird ein Delegat der Tabelle angelegt. Mittels *rows* und *cells* wird dann die benötigte Tabellenzelle ausgewählt.

```
table.rows[1].cells[0].innerHTML = Receivedata[Receivedata.length - 3];
table.rows[1].cells[1].innerHTML = Receivedata[Receivedata.length - 2];
table.rows[1].cells[2].innerHTML = Receivedata[Receivedata.length - 1];
```

Da der Device-Name frei wählbar ist, könnte er theoretisch ebenfalls Semikolon enthalten. Da die gewünschten Messwerte in den letzten drei Array-Feldern stehen, ermitteln wir über *length* die Gesamtzahl der Array-Felder und zählen einfach von hinten ab.

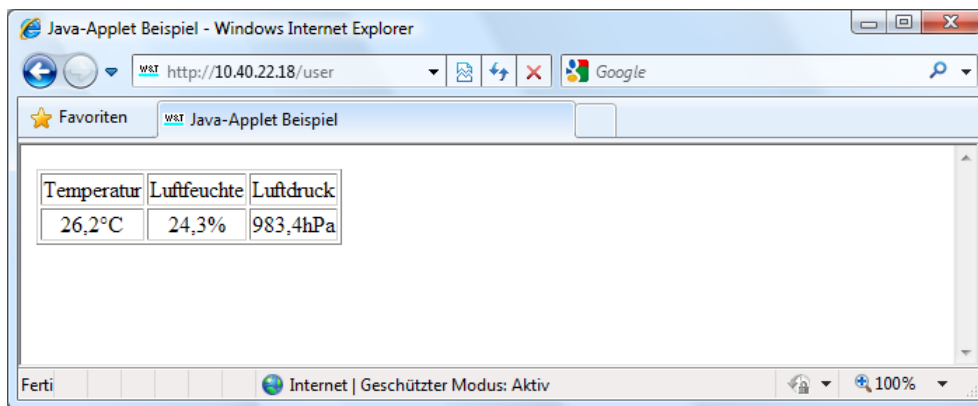


Bei der Anzeige im Browser gibt es noch ein Problem: Das Gradzeichen ° kann in dem übergebenen Format nicht dargestellt werden und muss gegen die HTML-Entsprechung ° ersetzt werden.

Dazu ist folgende Änderung notwendig:

```
var templength = Receivedata[Receivedata.length - 3].length - 2;
table.rows[1].cells[0].innerHTML = Receivedata[Receivedata.length - 3].substring(0,templength) + '&deg;C';
table.rows[1].cells[1].innerHTML = Receivedata[Receivedata.length - 2];
table.rows[1].cells[2].innerHTML = Receivedata[Receivedata.length - 1];
```

Die Browser-Ansicht ändert sich folgendermaßen:



Hier noch einmal der komplette Seiten Quelltext dazu:

```
<user.htm>
<html>
  <head>
    <title>Java-Applet Beispiel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <script language="JavaScript" type="text/javascript">

      var Command = 'single';
      var Interval = 1000;
      maintimer = setTimeout („DataRequest(Command)", Interval);

      function DataRequest(SendString)
      {
        var xmlhttp;
        if( window.ActiveXObject )           // Internet Explorer
```

```

    { xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
    }

    else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
    { xmlHttp = new XMLHttpRequest();
    }

    if (xmlHttp)
    { xmlHttp.onreadystatechange = function()
      { if (xmlHttp.readyState == 4)
        { if (xmlHttp.status == 200)
          { updateDisplay(xmlHttp.responseText);
            xmlHttp=null;
          }
        }
      }

      xmlHttp.open(„GET“, SendString, true);
      xmlHttp.setRequestHeader(„Connection“, „close“);
      xmlHttp.setRequestHeader(„If-Modified-Since“, „Thu, 1 Jan 1970 00:00:00 GMT“);
      xmlHttp.send(null);

      maintimer = setTimeout(„DataRequest(Command)“, Interval);
    }
  }

  function updateDisplay(Receivetext)
  { var table = document.getElementById('klima');
    var Receivedata = Receivetext.split(';');
    var templength = Receivedata[Receivedata.length - 3].length - 2;
    table.rows[1].cells[0].innerHTML = Receivedata[Receivedata.length -
3].substring(0,templength) + '&deg;C';
    table.rows[1].cells[1].innerHTML = Receivedata[Receivedata.length - 2];
    table.rows[1].cells[2].innerHTML = Receivedata[Receivedata.length - 1];
  }

</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<table id="klima" style="text-align: center; width: 180px;" border="1">
  <tr>
    <td>Temperatur</td>
    <td>Luftfeuchte</td>
    <td>Luftdruck</td>
  </tr>

```

```
<tr>
  <td>--</td>
  <td>--</td>
  <td>--</td>
</tr>
</table>
</body>
</html>
```

Digitale Signale mit AJAX überwachen

Stück für Stück sollen mit AJAX die Inputs, die Outputs und die Counter eines Web-IO Digital ausgelesen werden.

Um den entsprechenden JavaScript-Code entwickeln zu können, werfen wir zunächst einen Blick auf die Kommandos, die genutzt werden, um mit dem Web-IO Digital zu kommunizieren.

Hier die wichtigsten Kommandos:

Zustand der Inputs einlesen

`input?PW=password&`

Anstelle von *password* wird das für das Web-IO vergebene Passwort eingesetzt. Wurde kein Passwort vergeben, folgt hinter *PW=* bündig das *&*.

Das Web-IO antwortet mit:

`input; Hex-Inputstatus`

Hinter dem Semikolon wird der Status der Inputs in hexadezimaler Schreibweise übergeben. Beim 12fach Web-IO sind das 2 Byte - also 4 Stellen hexadezimal.

Zustand der Outputs einlesen

`output?PW=password&`

Anstelle von *password* wird das für das Web-IO vergebene Passwort eingesetzt. Wurde kein Passwort vergeben, folgt hinter *PW=* bündig das *&*.

Das Web-IO antwortet mit:

`output; Hex-Outputstatus`

Hinter dem Semikolon wird der Status der Outputs in hexadezimaler Schreibweise übergeben. Beim 12fach Web-IO sind das 2 Byte - also 4 Stellen hexadezimal.

Outputs ein- oder ausschalten

`outputaccessx?PW=password&State=Status&`

Das *x* steht für die Nr. des Output der geschaltet werden soll. Anstelle von *password* wird das für das Web-IO vergebene Passwort eingesetzt. Wurde kein Passwort vergeben, folgt hinter *PW=* bündig das *&*. Hinter *Status=* wird angegeben, in welchen Zustand der Output geschaltet werden soll.

Es gibt drei Möglichkeiten:

- *ON* - schaltet den Output ein
- *OFF* - schaltet den Output aus
- *TOGGLE* - wechselt den Schaltzustand

Das Web-IO antwortet mit:

`output; Hex-Outputstatus`

Hinter dem Semikolon wird der Status der Outputs in hexadezimaler Schreibweise übergeben. Beim 12fach Web-IO sind das 2 Byte - also 4 Stellen hexadezimal.

Zählerstände der Counter einlesen

`counter?PW=password&`

Anstelle von *password* wird das für das Web-IO vergebene Passwort eingesetzt. Wurde kein Passwort vergeben, folgt hinter *PW=* bündig das *&*.

Das Web-IO antwortet mit:

`Counter;n0;n1;n2;n3;n4;n5;n6;n7;n8;n9;n10;n11`

Hinter dem ersten Semikolon werden die Zählwerte der Counter 0 bis 11 mit weiteren Semikolon getrennt ausgegeben.

Counter löschen

`counterclear?PW=password&`

Anstelle von *password* wird das für das Web-IO vergebene Passwort eingesetzt. Wurde kein Passwort vergeben, folgt hinter *PW=* bündig das *&*.

Das Web-IO antwortet mit:

`Counter;0;0;0;0;0;0;0;0;0;0;0;0`

Hinter dem ersten Semikolon werden die Zählwerte (nach Löschen =0) der Counter 0 bis 11 mit weiteren Semikolon getrennt ausgegeben.

Eine komplette Übersicht aller Kommandos und aller Kommandooptionen finden Sie im Referenzhandbuch zum Web-IO Digital.



Den Status von Outputs und Inputs gibt das Web-IO in hexadezimaler Schreibweise zurück:

Hexadezimale Zahlen - Noch mal zur Erinnerung:

Wer im Umgang mit hexadezimalen Zahlen vertraut ist, kann diese kurze Abhandlung überspringen.

In der Computertechnik wird mit Bits und Bytes gearbeitet. Deshalb werfen wir zunächst einen Blick auf das duale bzw. binäre Zahlensystem.

Ein Byte besteht aus 8 Bit - also 8 Stellen binär. Jedes Bit kann zwei Zustände einnehmen: 1 oder 0 bzw. gesetzt oder nicht gesetzt bzw. ON oder OFF.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	1	0

Die einzelnen Bits stehen je für einen Input/Output. Hinter jedem Bit verbirgt sich außerdem eine Stelle des dualen Zahlensystems.

Gerechnet wird mit 2er-Potenzen, wobei die Nummer des Bits bzw. Outputs den Exponenten bildet. Addiert man die Potenzen der gesetzten Bits, erhält man den dezimalen Wert, der dem Bitmuster entspricht.

Beim Web-IO 2xDigital immer 0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit (Output) 1	Bit (Output) 0
0	0	0	0	0	0	1	0

$0 \times 2^0 = 0 \times 1 = 0$
$1 \times 2^1 = 1 \times 2 = 2$
$0 \times 2^2 = 0 \times 4 = 0$
$0 \times 2^3 = 0 \times 8 = 0$
$0 \times 2^4 = 0 \times 16 = 0$
$0 \times 2^5 = 0 \times 32 = 0$
$0 \times 2^6 = 0 \times 64 = 0$
$0 \times 2^7 = 0 \times 128 = 0$
Summe
2

Zum Vergleich:

Beim dezimalen Zahlensystem kann jede Stelle 10 verschiedene Ziffern von 0 - 9 beinhalten. Ist die 9 erreicht, fängt es wieder bei 0 an und die nächsthöhere Stelle wird um eins hochgezählt. Gerechnet wird mit Zehnerpotenzen.

Im hexadezimalen Zahlensystem wird mit 16 verschiedenen Ziffern bzw. Zeichen gearbeitet. Das sind die Ziffern 0 - 9, gefolgt von den Buchstaben A - F. Ist also das F erreicht, fängt es wieder bei 0 an und die nächsthöhere Stelle wird um eins hochgezählt.

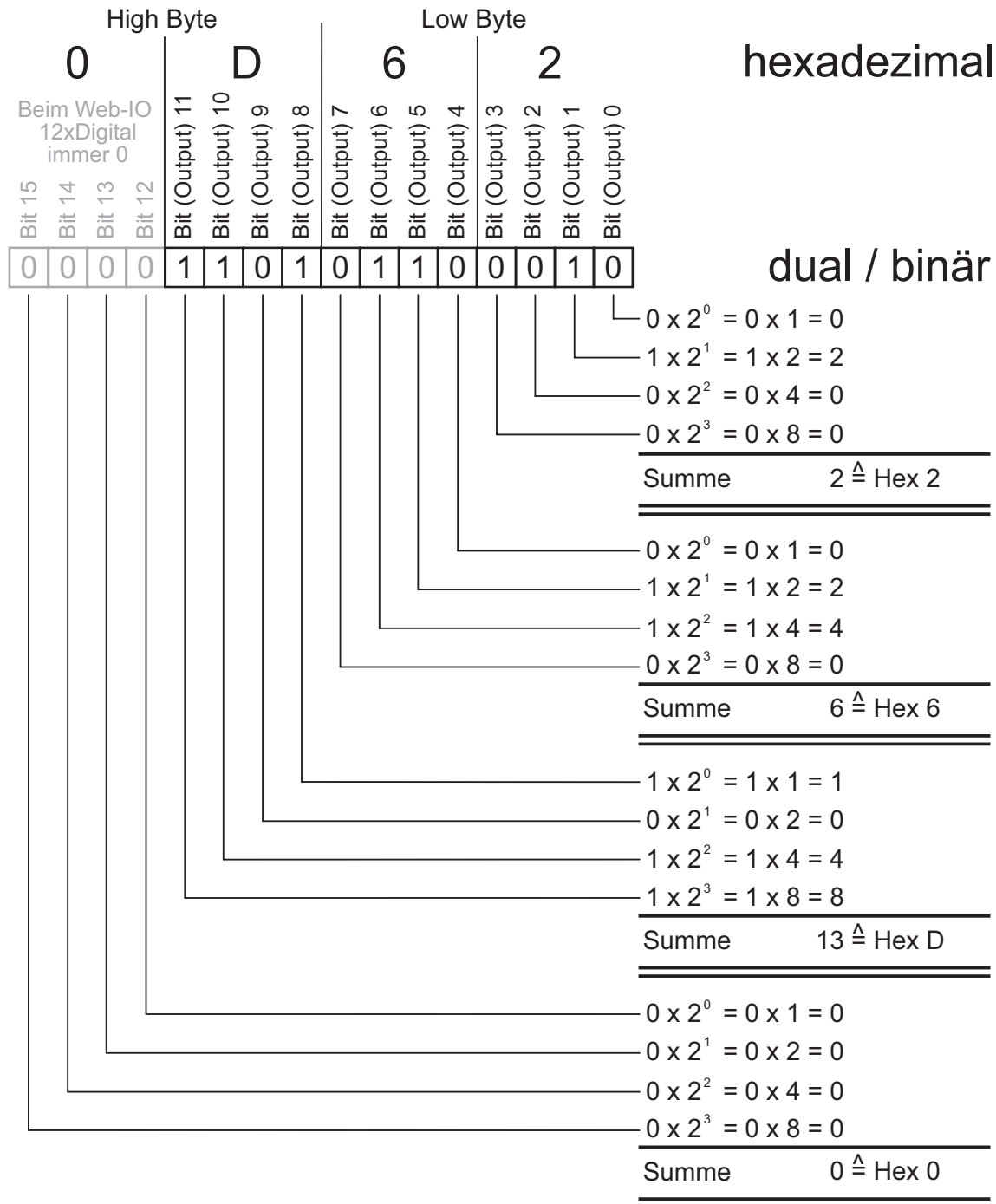
Auf den ersten Blick könnte man denken, dass der Einsatz des hexadezimalen Zahlensystems die Sache komplizierter macht, als sie ohnehin schon ist.

Schaut man allerdings genauer hin, wird es mit den Hexzahlen deutlich einfacher und übersichtlicher.

Ein Byte, bzw. 8 Bit entsprechen einer 8 stelligen Dualzahl. Hexadezimal entsprechen 8 Bit einer 2 stelligen Zahl.

Es lassen sich also je 4 Stellen dual mit einer Stelle hexadezimal darstellen.

Das Umrechnen auch mehrerer Bytes in eine hexadezimale Zahl schafft man, wenn immer nur einzelne 4 Bit-Bereiche umgerechnet werden, sogar im Kopf.



Umgekehrt kann man anhand einer hexadezimalen Zahl relativ leicht das dazugehörige Bitmuster überblicken.

Deshalb gibt das Web-IO den Status von Outputs und Inputs in hexadezimaler Schreibweise zurück.

Um Zahlen, die in hexadezimaler Schreibweise übergeben werden, in eine echte Zahlenvariable umzuwandeln, kann folgende Funktion verwendet werden:

```
function HexToInt(HexStr)
{
    var TempVal;
    var HexVal=0;
    for( var i=0; i<HexStr.length;i++)
    {
        if (HexStr.charCodeAt(i) > 57)
        {
            TempVal = HexStr.charCodeAt(i) - 55;
        }
        else
        {
            TempVal = HexStr.charCodeAt(i) - 48;
        }
        HexVal=HexVal+TempVal*Math.pow(16, HexStr.length-i-1);
    }
    return HexVal;
}
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/20a-ajax.html

Die Funktion prüft für die übergebene Zeichenkette für jedes Zeichen, ob der ASCII-Zeichencode zwischen 48 und 57 liegt (48 = "0", 57 = "9"). Zieht man in diesem Fall 48 vom Zeichencode ab, erhält man die zugehörige Zahl. Ist der Zeichencode größer als 57, handelt es sich um eines der Zeichen „A“ bis „F“. Der Zeichencode für „A“ ist 65; der Wert von „A“ ist dezimal 10. In diesem Fall müssen also 55 abgezogen werden.

Der zur hexadezimalen Stelle gehörende dezimale Wert wird dann mit der der Stelle entsprechenden 16er Potenz multipliziert. Bei der niedrigsten Stelle z.B. mal 16^0 . Die Werte werden dann addiert.

Zustand eines Inputs anzeigen

Um die grundsätzliche Vorgehensweise bei der Kommunikation mit dem Web-IO über AJAX zu zeigen, soll im ersten Beispiel der Zustand eines einzigen Inputs angezeigt werden.

Dazu wird eine Tabelle mit zwei Zeilen und drei Spalten angelegt. Die Zelle, in der der Status angezeigt werden soll, bekommt die ID *input0*.

```
<user.htm>
<html>
<head>
<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">
```

```

<title>IOs mit AJAX</title>
</head>
<body>
  <table style="text-align: left;" border="1">
    <tr>
      <td>Nr</td>
      <td>&nbsp;</td>
      <td>State</td>
    </tr>
    <tr>
      <td>0</td>
      <td>Input</td>
      <td id="input0">OFF</td>
    </tr>
  </table>
</body>
</html>

```

Im Browser sieht das ganze dann so aus:

Nr		State
0	Input	OFF

Im JavaScript-Teil der Seite muss die Funktion *HexToInt* eingefügt werden.

Um den Status der Inputs abzufragen, wird über die Funktion *DataRequest* das Kommando *input?PW=&* an das Web-IO gesendet.

```
<script language="JavaScript" type="text/javascript">
```

```

var CommandString = 'input?PW=&';
var Interval = 500;

```

```
DataRequest(CommandString);
```

```

function HexToInt(HexStr)
{
  var TempVal;
  var HexVal=0;

```

```
for( var i=0; i<HexStr.length;i++)
{ if (HexStr.charCodeAt(i) > 57)
  { TempVal = HexStr.charCodeAt(i) - 55;
  }
  else
  { TempVal = HexStr.charCodeAt(i) - 48;
  }
  HexVal=HexVal+TempVal*Math.pow(16, HexStr.length-i-1);
}
return HexVal;
}

function DataRequest(SendString)
{ var xmlHttp;
  if( window.ActiveXObject )      // Internet Explorer
  { xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
  }
  else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
  { xmlHttp = new XMLHttpRequest();
  }
  if (xmlHttp)
  { xmlHttp.onreadystatechange = function()
    { if (xmlHttp.readyState == 4)
      { if (xmlHttp.status == 200)
        { updateDisplay(xmlHttp.responseText);
          xmlHttp=null;
        }
      }
    }
  }
  xmlHttp.open(„GET“, SendString, true);
  xmlHttp.setRequestHeader(„Connection“, „close“);
  xmlHttp.setRequestHeader(„If-Modified-Since“, „Thu, 1 Jan 1970 00:00:00 GMT“);
  xmlHttp.send(null);
  maintimer = setTimeout(„DataRequest(CommandString)“, Interval);
}
}
```

Über `setTimeout` ruft sich die Funktion `DataRequest` nach einer Intervallzeit von 500ms immer wieder erneut auf.

Nachdem das Web-IO eine Antwort gesendet hat, wird diese an die Funktion `updateDisplay` übergeben.

```
function updateDisplay(ReceiveStr)
{
    var HexStr = ReceiveStr.substring(6, 10);
    var HexVal= HexToInt(HexStr);
    if ((HexVal & 1) == 1)
    {
        document.getElementById('input0').innerHTML = 'ON';
        document.getElementById('input0').style.backgroundColor = '#00FF00';
    }
    else
    {
        document.getElementById('input0').innerHTML = 'OFF';
        document.getElementById('input0').style.backgroundColor = '#CCCCCC';
    }
}
```

Das Web-IO sendet als Antwort einen String bestehend aus 'input;' und einer vierstelligen hexadezimalen Zahl.

Beispiel: input;0031

Über die *substring* Funktion werden die vier Hex-Ziffern aus dem String ausgeschnitten. Die Funktion *HexToInt* macht aus dem hexadezimalen Zahlenstring einen Zahlenwert.

Über eine UND-Verknüpfung mit dem Wert, der dem zum Output passend gesetzten Bit entspricht, wird ermittelt (ausmaskiert), ob der Output gesetzt ist oder nicht.

Im konkreten Fall soll der Zustand von Output0 ermittelt werden.

Der Bitwert ist in diesem Fall 1. Die Ausmaskierung findet also durch UND-Verknüpfung und Vergleich mit dem Wert 1 statt.

Für den Output1 wäre es 2, für Output2 entsprechend 4 usw.

Mittels *ID* und *document.getElementById* wird dann je nach Zustand *ON* oder *OFF* angezeigt und der Hintergrund der Tabellenzelle grün oder grau eingefärbt.

Nr		State
0	Input	OFF

oder

Nr		State
0	Input	ON

Inputs und Counter mit AJAX anzeigen

Im nächsten Schritt soll die Anzeige um den Zählerwert des Counters 0 erweitert werden.

Dazu ist es nötig, die Tabelle um eine Spalte zu erweitern.

```
<table style="text-align: left;" border="1">
  <tr>
    <td>Nr</td>
    <td>&nbsp;</td>
    <td>State</td>
    <td>Counter</td>
  </tr>
  <tr>
    <td>0</td>
    <td>Input</td>
    <td id="input0">OFF</td>
    <td id="counter0">0</td>
  </tr>
</table>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/20b-ajax.html

Die `DataRequest` Funktion kann mit Ausnahme der `setTimeout` Anweisung unverändert übernommen werden.

Zusätzlich zum Input-Abfragekommando muss der Zählerstand der Counter abgefragt werden. Um nicht zuviel Datenverkehr zu erzeugen, sollte die Abfrage von Inputs und Countern im Wechsel erfolgen.

Dazu wird eine Funktion *Commandloop* angelegt, die abwechselnd die Funktion `DataRequest` aufruft. Einmal mit Übergabe der Kommandos *input?PW=&* und *counter?PW=&*.

Durch eine Variable *Applicationstep* und eine *If*-Abfrage wird der Wechsel sichergestellt.

```
var Applicationstep = 0;
```

```

function Commandloop()
{ Applicationstep++;
  if (Applicationstep == 1)
  { CommandString = 'input?PW=&';
  }
  else
  { CommandString = 'counter?PW=&';
    Applicationstep = 0;
  }
  DataRequest(CommandString);
  maintimer = setTimeout(„Commandloop()“, Interval);
}

```

Die Funktion *updateDisplay* muss nun unterscheiden, ob bei der Antwort des Web-IO der Zustand der Inputs oder die Counter-Werte gesendet wurden.

Dazu kann einfach der erste Buchstabe des Antwortstring überprüft werden. Ist der erste Buchstabe ein 'i', enthält die Antwort die Inputzustände. Ist es ein 'c' sind es die Counter-Werte.

```

function updateDisplay(ReceiveStr)
{ var HexVal
  if (ReceiveStr.substring(0,1)=='i')
  { HexVal= HexToInt(ReceiveStr.substring(6, 10))
    if ((HexVal & 1) == 1)
    { document.getElementById('input0').innerHTML = 'ON';
      document.getElementById('input0').style.backgroundColor = '#00FF00';
    }
    else
    { document.getElementById('input0').innerHTML = 'OFF';
      document.getElementById('input0').style.backgroundColor = '#CCCCC';
    }
  }
  else
  { var Countervalue = ReceiveStr.split(';');
    document.getElementById('counter0').innerHTML = Countervalue[1];
  }
}

```

Um die Counter auszuwerten, wird die Antwort des Web-IO mit der *split*-Anweisung in ein Array zerlegt.

Zur Erinnerung:

Das Web-IO sendet mit Semikolon getrennt die Werte aller Counter.

```
counter;n0;n1;n2;n3;n4;n5;n6;n7;n8;n9;n10;n11
```

```
Countervalue = ReceiveStr.split(';');
```

Im Array *Countervalue* sind nach dem Aufruf von `ReceiveStr.split(';')` 13 Einzelvariablen beginnend mit „counter“ gefolgt von den 12 Zählerwerten.

In unserem Beispiel soll Counter0 angezeigt werden. Der Wert ist über *Countervalue[1]* abrufbar.

Hier noch einmal der komplette Quelltext:

```
<user.htm>
<html>
<head>
  <meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">
  <title>IOs mit AJAX</title>
  <script language="JavaScript" type="text/javascript">

var CommandString;
var Interval = 500;
var Applicationstep = 0;
Commandloop();

function HexToInt (HexStr)
{ var TempVal;
  var HexVal=0;
  for( var i=0; i<HexStr.length;i++)
  { if (HexStr.charCodeAt(i) > 57)
    { TempVal = HexStr.charCodeAt(i) - 55;
    }
    else
    { TempVal = HexStr.charCodeAt(i) - 48;
```

```

    }

    HexVal=HexVal+TempVal*Math.pow(16, HexStr.length-i-1);
    }

    return HexVal;
}

function Commandloop()
{ Applicationstep++;
  if (Applicationstep == 1)
  { CommandString = 'input?PW=&';
  }

  else
  { CommandString = 'counter?PW=&';
    Applicationstep = 0;
  }

  DataRequest(CommandString);
  maintimer = setTimeout(„Commandloop()“, Interval);
}

function DataRequest(SendString)
{ var xmlHttp;
  if( window.ActiveXObject )      // Internet Explorer
  { xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
  }

  else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
  { xmlHttp = new XMLHttpRequest();
  }

  if (xmlHttp)
  { xmlHttp.onreadystatechange = function()
    { if (xmlHttp.readyState == 4)
      { if (xmlHttp.status == 200)
        { updateDisplay(xmlHttp.responseText);
          xmlHttp=null;
        }
      }
    }
  }

  xmlHttp.open(„GET“, SendString, true);
  xmlHttp.setRequestHeader(„Connection“, „close“);
  xmlHttp.setRequestHeader(„If-Modified-Since“, „Thu, 1 Jan 1970 00:00:00 GMT“);
  xmlHttp.send(null);
}

```

```

}

function updateDisplay(ReceiveStr)
{
  var HexVal
  if (ReceiveStr.substring(0,1)=='i')
  {
    HexVal= HexToInt(ReceiveStr.substring(6, 10))
    if ((HexVal & 1) == 1)
    {
      document.getElementById('input0').innerHTML = 'ON';
      document.getElementById('input0').style.backgroundColor = '#00FF00';
    }
    else
    {
      document.getElementById('input0').innerHTML = 'OFF';
      document.getElementById('input0').style.backgroundColor = '#CCCCC0';
    }
  }
  else
  {
    var Countervalue = ReceiveStr.split(';');
    document.getElementById('counter0').innerHTML = Countervalue[1];
  }
}

</script>
</head>
<body>
<table style="text-align: left;" border="1">
  <tr>
    <td>Nr</td>
    <td>&nbsp;</td>
    <td>State</td>
    <td>Counter</td>
  </tr>
  <tr>
    <td>0</td>
    <td>Input</td>
    <td id="input0">OFF</td>
    <td id="counter0">0</td>
  </tr>
</table>
</body>
</html>

```

Um auch noch den Zustand von Output0 mitanzuzeigen, könnte die Webseite nach dem gleichen Schema erweitert werden. Wir verzichten an dieser Stelle darauf, diese Erweiterung auch noch vorzustellen.

Alle IOs eines Web-IO 12x Digital anzeigen

Für das folgende Beispiel kann der HTML-Teil des Beispiels „digitale Signale mit einem Java-Applet überwachen“ genutzt werden. Der JavaScript-Teil und die Einbindung des Applets können gelöscht werden.

```
<html>
  <head>
    <title>Java-Applet Beispiel</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <script language="JavaScript" type="text/javascript">
      JavaScript-Teil
    </script>
  </head>
  <body bgcolor="#FFFFFF" text="#000000">
    <applet name="dio0" archive="dio.jar" code="dio.class" codebase="http://10.40.22.101"
width="0" height="0" mayscript>
      <param name="device" value="0">
      <param name="inputpolling" value="on">
      <param name="outputpolling" value="on">
      <param name="counterpolling" value="on">
      <param name="pollingrate" value="1000">
    </applet>
    <table ID="ioTable" style="text-align: left; width: 257px; height: 368px;" border="1">
    <tr>
      <td>Nr.</td>
      <td>&nbsp;</td>
      <td>State</td>
      <td>Counter</td>
      <td>&nbsp;</td>
      <td>State</td>
    </tr>
    <tr>
      <td>0</td>
      <td>Input</td>
      <td>OFF</td>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/20-ajax.html

```
<td>0</td>
<td>Output</td>
<td>OFF</td>
</tr>
.....
.....
<tr>
<td>11</td>
<td>Input</td>
<td>OFF</td>
<td>0</td>
<td>Output</td>
<td>OFF</td>
</tr>
</table>
</body>
</html>
```

Die Anzeige aller IOs bzw. Counter ist mit AJAX kein Problem. Damit auch die Outputs Berücksichtigung finden, muss die *Commandloop*-Funktion aus dem vorangegangenen Beispiel entsprechend erweitert werden.

```
function Commandloop()
{ Applicationstep++;
  switch (Applicationstep)
  { case 1:
      CommandString = 'input?PW=&';
      break;
    case 2:
      CommandString = 'output?PW=&';
      break;
    case 3:
      CommandString = 'counter?PW=&';
      Applicationstep = 0;
      break;
  }
  DataRequest(CommandString);
  maintimer = setTimeout(„Commandloop()“, Interval);
}
```

Anstelle der *If*-Abfrage wird *ApplicationStep* nun mit einer *Switch/Case*-Abfrage überprüft. Das erhöht die Übersichtlichkeit.

Die Funktion *DataRequest* bleibt unverändert.

Allerdings muss die Funktion *updateDisplay* ebenfalls verändert werden. Zum einen ist es nötig, eine Erweiterung für die Output-Anzeige einzupflegen, zum anderen sollen ja alle 12 IOs angezeigt werden.

```
function updateDisplay(ReceiveStr)
{
  var HexVal;
  var state = false;
  table = document.getElementById('ioTable');
  if (ReceiveStr.substring(0, 1) == 'i')
  {
    HexVal = HexToInt(ReceiveStr.substring(6, 10));
    for (i = 0; i < 12; i++)
    {
      if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
      {
        table.rows[i + 1].cells[2].innerHTML = 'ON';
        table.rows[i + 1].cells[2].style.backgroundColor = '#00FF00';
      }
      else
      {
        table.rows[i + 1].cells[2].innerHTML = 'OFF';
        table.rows[i + 1].cells[2].style.backgroundColor = '#CCCCCC';
      }
    }
  }
  if (ReceiveStr.substring(0, 1) == 'o')
  {
    HexVal = HexToInt(ReceiveStr.substring(7, 11));
    for (i = 0; i < 12; i++)
    {
      if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
      {
        table.rows[i + 1].cells[5].innerHTML = 'ON';
        table.rows[i + 1].cells[5].style.backgroundColor = '#00FF00';
      }
      else
      {
        table.rows[i + 1].cells[5].innerHTML = 'OFF';
        table.rows[i + 1].cells[5].style.backgroundColor = '#CCCCCC';
      }
    }
  }
}
```

```
if (ReceiveStr.substring(0, 1) == 'c')
{
    var Countervalue = ReceiveStr.split(';');
    for (i = 0; i < 12; i++)
    {
        table.rows[i + 1].cells[3].innerHTML = Countervalue[i + 1];
    }
}
}
```

Über

```
table = document.getElementById('ioTable');
```

wird ein Delegat der Tabelle geschaffen.

Die Auswahl der Zellen, welche verändert werden sollen, erfolgt dann über die *rows* (Zeilen) und *cells* (Spalten) Eigenschaften des Tabellendelegates. Die Nummerierung beginnt mit 0 für die erste Zeile bzw. Spalte und wird in eckigen Klammern angefügt.

Wie gehabt wird über den ersten Buchstaben ermittelt, ob es um Inputs, Outputs oder Counter geht. Damit alle 12 IOs berücksichtigt werden können, wird mit einer For/Next-Schleife gearbeitet.

Die Ausmaskierung der einzelnen Inputs bzw. Outputs erfolgt über Zweierpotenzen deren Wert mit der JavaScript-Funktion *Math.pow(2, i)* generiert werden. Die Variable *i* wird bei jedem Schleifendurchlauf um 1 hochgezählt.

Dadurch ergeben sich die aufsteigenden Zweierpotenzen:

1, 2, 4, 8, 16, 32,

In jedem Durchlauf wird dann geprüft, ob das entsprechende Input-/Output-Bit gesetzt ist.

```
if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
```

Die Anzeige der zugehörigen Tabellenfelder wird dann entsprechend angepasst.

Schalten der Outputs mit AJAX

In diesem Beispiel soll Output 0 über zwei Buttons ein- bzw. ausgeschaltet werden. Der aktuelle Output-Zustand soll in einem vorangestellten Tabellenfeld angezeigt werden.

Das passende HTML-Grundgerüst sieht so aus:

```
<user.htm>
<html>
<head>
  <meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">
  <title>IOs mit AJAX</title>
  <script language="JavaScript" type="text/javascript">

    </script>
</head>
<body>
<table style="text-align: left;" border="1">
  <tr>
    <td>Nr</td>
    <td>&nbsp;</td>
    <td>State</td>
    <td>Switch</td>
    <td>Swicht</td>
  </tr>
  <tr>
    <td>0</td>
    <td>Output</td>
    <td id="output0">OFF</td>
    <td><input type="button" value="ON" onclick="setOutput(1);"></td>
    <td><input type="button" value="OFF" onclick="setOutput(0);"></td>
  </tr>
</table>
</body>
</html>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/21b-ajax.html

Im Browser wird das ganze so angezeigt:

Nr		State	Switch	Swicht
0	Output	OFF	ON	OFF

Wie gehabt wird unverändert die Funktion *DataRequest* genutzt, um Kommandos an das Web-IO zu senden.

Unter *onclick* wurde im HTML-Teil für die Buttons festgelegt, dass die Funktion *setOutput* aufgerufen wird und ob als Parameter 1 oder 0 mitgegeben wird. Abhängig davon wird der passende Befehl zum Ein- oder Ausschalten von Output0 gesendet.

```
function setOutput(State)
{ if (State == 1)
  { DataRequest('outputaccess0?PW=&State=ON&');
  }
  else
  { DataRequest('outputaccess0?PW=&State=OFF&');
  }
}
```

Das Web-IO antwortet daraufhin in gleicher Weise wie bei dem Kommando *output?PW=&* mit einem vierstelligen Hex-Code. Die Antwort wird wieder in der Funktion *updateDisplay* ausgewertet.

```
function updateDisplay(ReceiveStr)
{ var HexVal
  if (ReceiveStr.substring(0,1)=='o')
  { HexVal= HexToInt(ReceiveStr.substring(7, 11))
    if ((HexVal & 1) == 1)
    { document.getElementById('output0').innerHTML = 'ON';
      document.getElementById('output0').style.backgroundColor = '#00FF00';
    }
    else
    { document.getElementById('output0').innerHTML = 'OFF';
      document.getElementById('output0').style.backgroundColor = '#CCCCCC';
    }
  }
}
```

Nr		State	Switch	Swicht
0	Output	ON	ON	OFF

Mit AJAX alle IOs anzeigen und alle Outputs steuern

Abschließend sollen innerhalb einer Tabelle alle 12 Inputs, Outputs und Counter angezeigt werden und darüberhinaus eine Steuermöglichkeit für alle Outputs gegeben werden.

Es soll, wie vorher für jeden der Outputs zwei Buttons (ON, OFF) geben. Dazu werden der Tabelle aus dem Beispiel in dem alle IOs angezeigt wurden, zwei Spalten angefügt.

```
<table ID="ioTable" style="text-align: left; width: 257px; height: 368px;" border="1">
  <tr>
    <td>Nr.</td>
    <td>&nbsp;</td>
    <td>State</td>
    <td>Counter</td>
    <td>&nbsp;</td>
    <td>State</td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
  <tr>
    <td>0</td>
    <td>Input</td>
    <td>OFF</td>
    <td>0</td>
    <td>Output</td>
    <td>OFF</td>
    <td><input onclick="setOutput(0,1);" type="button" value="ON"></td>
    <td><input onclick="setOutput(0,0);" type="button" value="OFF"></td>
  </tr>
  .....
  .....
  <tr>
    <td>11</td>
    <td>Input</td>
    <td>OFF</td>
    <td>0</td>
    <td>Output</td>
    <td>OFF</td>
    <td><input onclick="setOutput(11,1);" type="button" value="ON"></td>
    <td><input onclick="setOutput(11,0);" type="button" value="OFF"></td>
  </tr>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/22-ajax.html

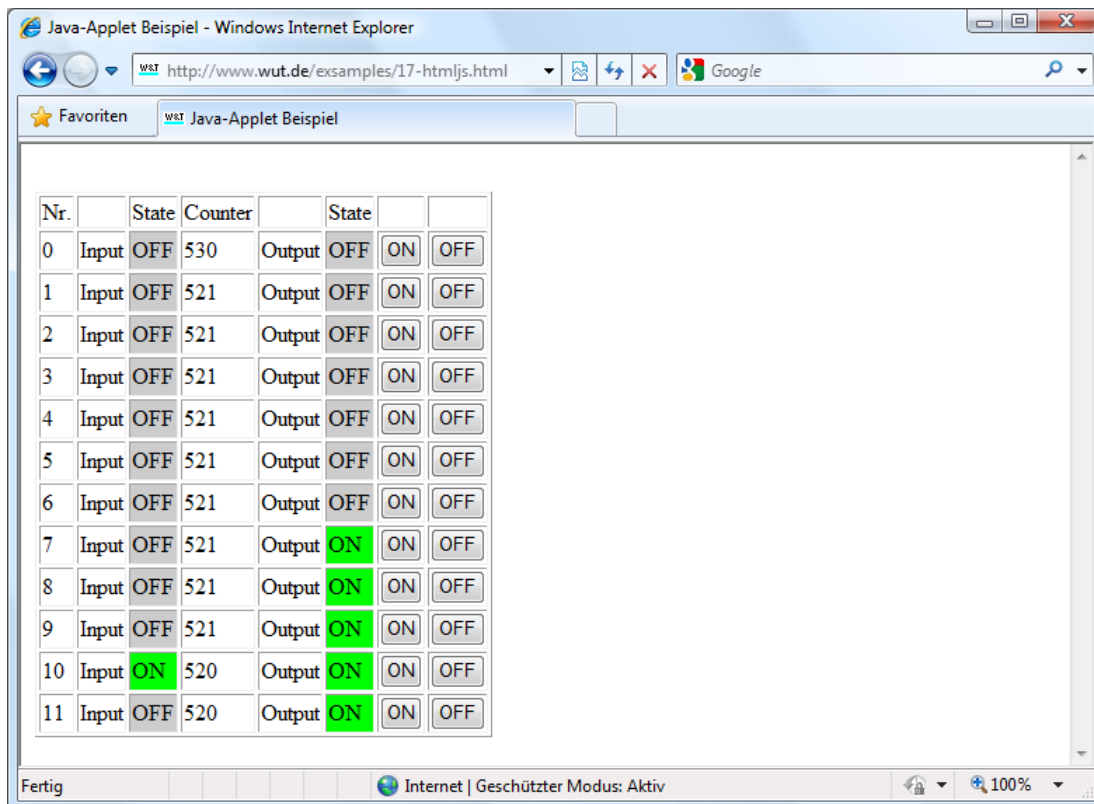
```
</tr>
</table>
```

Über das `<input>`-Tag wird dann in die neuen Zellen jeweils ein Button eingefügt. Als Einsprungfunktion wird für *onclick* die Funktion *setOutput* mit zwei Parametern eingetragen. Der erste Parameter steht für die Output-Nr, der zweite legt fest, in welchen Zustand der Output geschaltet werden soll (0=OFF, 1=ON).

Im JavaScript-Teil muss die Funktion *setOutput* angelegt werden.

```
function setOutput(Nr, State)
{
  if (State == 1)
  {
    DataRequest('outputaccess'+Nr+'?PW=&State=ON&');
  }
  else
  {
    DataRequest('outputaccess'+Nr+'?PW=&State=OFF&');
  }
}
```

Die Funktion *updateDisplay* kann unverändert übernommen werden.



Bei allen gezeigten AJAX-Beispielen wurde auf die Übergabe eines Passwortes verzichtet. Bei realen Anwendungen ist es absolut zu empfehlen, den Zugriff auf die IOs mit einem Passwort zu schützen.

Dazu kann man der Tabelle ein Formular mit Eingabefeld für das Passwort und einem Login-Button voranstellen.

Im HTML-Teil wird dazu hinter dem `<body>`-Tag der Formular-
teil eingefügt:

```
<form name="form1">
  <p>Passwort:</p>
  <p>
    <input type="password" name="tb_password" >
    <input type="button" value="Login" onclick="Login();">
  </p>
</form>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter:
www.wut.de/
examples/
23-ajax.html

Im JavaScript-Bereich wird die Variable *Password* und die Funktion *Login* hinzugefügt:

```
var Password = '';  
  
function Login()  
{ Password = document.forms[0].tb_password.value;  
  document.forms[0].tb_password.value = '';  
}
```

Überall da, wo im JavaScript-Teil Kommandos für das Web-IO zusammengestellt werden, muss das Passwort mit eingefügt werden. Konkret sind das die Funktionen *Commandloop* und *setOutput*.

```
function Commandloop()  
{ Applicationstep++;  
  switch (Applicationstep)  
  { case 1:  
    CommandString = 'input?PW='+Password+'&';  
    break;  
    case 2:  
    CommandString = 'output?PW='+Password+'&';  
    break;  
    case 3:  
    CommandString = 'counter?PW='+Password+'&';  
    Applicationstep = 0;  
    break;  
  }  
  DataRequest(CommandString);  
  maintimer = setTimeout("Commandloop()", Interval);  
}  
  
function setOutput(Nr, State)  
{ if (State == 1)  
  { DataRequest('outputaccess'+Nr+'?PW='+Password+'&State=ON&');  
  }  
  else  
  { DataRequest('outputaccess'+Nr+'?PW='+Password+'&State=OFF&');  
  }  
}
```

Damit ist der Passwortschutz fertig.

Webseiten mit einheitlichem Design

In den vorangegangenen Beispielen haben wir rein funktionale Webseiten aufgebaut, die keinen speziellen Anspruch an das Design hatten.

Mit wenig Aufwand können Webseiten deutlich ansprechender gestaltet werden. Bei den bisherigen Beispielen wurde keinerlei Wert auf Schriftart, Schriftgröße oder die Breite/Höhe von Tabellenzeilen gelegt.

Es gibt verschiedene Möglichkeiten, das Erscheinungsbild von HTML-Objekten zu bestimmen.

Am Beispiel der Outputvisualisierung wollen wir die Möglichkeiten vergleichen.

```
<table style="text-align: left;" border="1">
  <tr>
    <td>Nr</td>
    <td>&nbsp;</td>
    <td>State</td>
    <td>Switch</td>
    <td>Swicht</td>
  </tr>
  <tr>
    <td>0</td>
    <td>Output</td>
    <td id="output0">OFF</td>
    <td><input type="button" value="ON" onclick="setOutput(1);"></td>
    <td><input type="button" value="OFF" onclick="setOutput(0);"></td>
  </tr>
</table>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/24-style.html

Hier wird der Standardzeichensatz verwendet und die Spaltenbreite der Tabellenfelder ergibt sich aus der Laufweite des Textinhaltes. Das gilt auch für die Breite der Buttons.

Nr		State	Switch	Swicht
0	Output	OFF	ON	OFF

Eigenschaften von HTML-Objekten mit HTML anpassen

Im ersten Beispiel soll mit HTML-Tags das Erscheinungsbild der Webseite angepasst werden.

```
<table style="text-align: left;" border="1">
  <tr>
    <td height="19">
      <b><font face="Arial, Helvetica, sans-serif" size="2">Nr</font></b>
    </td>
    <td height="19">
      <font face="Arial, Helvetica, sans-serif" size="2">&nbsp;</font>
    </td>
    <td height="19">
      <b><font face="Arial, Helvetica, sans-serif" size="2">State</font></b>
    </td>
    <td height="19">
      <b><font face="Arial, Helvetica, sans-serif" size="2">Switch</font></b>
    </td>
    <td height="19">
      <b><font face="Arial, Helvetica, sans-serif" size="2">Swicht</font></b>
    </td>
  </tr>
  <tr valign="middle">
    <td height="25" align="center">
      <font face="Arial, Helvetica, sans-serif" size="2">0</font>
    </td>
    <td height="25">
      <font face="Arial, Helvetica, sans-serif" size="2">Output</font>
    </td>
    <td id="output0" height="25" align="center">
      <font face="Arial, Helvetica, sans-serif" size="2">OFF</font>
    </td>
    <td height="25">
      <font face="Arial, Helvetica, sans-serif" size="2">
        <input type="button" value="ON" onclick="setOutput(1);">
      </font>
    </td>
    <td height="25">
      <font face="Arial, Helvetica, sans-serif" size="2">
        <input type="button" value="OFF" onclick="setOutput(0);">
      </font>
    </td>
  </tr>
</table>
```



Dieses Beispiel finden
Sie in der Software-
Sammlung zum Buch.
Download unter:
[www.wut.de/
examples/
24a-style.html](http://www.wut.de/examples/24a-style.html)

```

    </td>
  </tr>
</table>

```

Nr		State	Switch	Swicht
0	Output	OFF	ON	OFF

`<font>` und die damit übergebenen Attribute bestimmen wie ein Text dargestellt wird.

`<b>` steht für bold und bestimmt, dass der Text zwischen den `<b>`-Tags fettgedruckt angezeigt wird.

Innehalb der `<tr>` und `<td>`-Tags können Attribute zur Ausrichtung und Größe gesetzt werden.

Eine Liste möglicher HTML-Tags und Attribute findet sich im Anhang.

Schon dieses kleine Beispiel zeigt, dass der Aufwand relativ groß ist, da für jedes Objekt die entsprechenden Eigenschaften per HTML zugewiesen werden. Für große Projekte ist diese Lösung eher ungeeignet.

HTML-Objekte über Style-Attribute formatieren

Eine weitere Methode HTML-Objekte anzupassen, ist das Setzen von Style-Attributen. Für die Veränderung von Objekten haben wir das bereits in den vorangegangenen Beispielen getan.

```

<table style="text-align: left;" border="1">
  <tr>
    <td style="text-align:center;
      font-family:Arial,Verdana, Helvetica;
      font-size:14px;
      font-weight:bold;
      height:18px;
      width:25px;">Nr</td>
    <td style="text-align:center;
      height:18px;
      width:60px;">&nbsp;</td>
    <td style="text-align:center;

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/24b-style.html

```

        font-family:Arial,Verdana, Helvetica;
        font-size:14px;
        font-weight:bold;
        height:18px;
        width:40px;">State</td>
<td style="text-align:center;
        font-family:Arial,Verdana, Helvetica;
        font-size:14px;
        font-weight:bold;
        height:18px;
        width:50px;">Switch</td>
<td style="text-align:center;
        font-family:Arial,Verdana, Helvetica;
        font-size:14px;
        font-weight:bold;
        height:18px;
        width:50px;">Switch</td>
</tr>
<tr>
<td style="text-align:center;
        font-family:Arial,Verdana, Helvetica;
        font-size:14px;
        height:24px;
        width:25px;">0</td>
<td style="text-align:center;
        font-family:Arial,Verdana, Helvetica;
        font-size:14px;
        height:24px;
        width:60px;"> Output</td>
<td id="output0" style="text-align:center;
        font-family:Arial,Verdana, Helvetica;
        font-size:14px;
        height:24px;
        width:40px;">OFF</td>
<td style="text-align:center;
        height:24px;
        width:50px;"><input type="button" style="text-align:center;
        font-family:Arial,Verdana, Helvetica;
        font-size:14px;
        height:22px;
        width:48px;" value="ON" onclick="setOutput(1);"></td>

```

```
  |
```

Über den Parameter *style* können den einzelnen Elementen Attribute für verschiedenste Eigenschaften übergeben werden.

Die Attributbezeichnungen sind durch ihren Wortlaut weitestgehend selbsterklärend.

Der Aufwand ist aber ebenso hoch, wie beim Umgang mit HTML-Tags.

Allerdings ist der Quelltextaufbau übersichtlicher und im Ergebnis können die Elemente feiner angepasst werden. Auch die Buttons lassen sich so in ihrem Erscheinungsbild anpassen.

Nr		State	Switch	Swicht
0	Output	OFF	ON	OFF

Styles zentral verwalten - CSS-Stylesheets

Betrachtet man das vorangegangene Beispiel, stellt man fest, dass diverse Style-Eigenschaften bei allen Tabellenelementen gleich sind.

Über einen Style-Bereich im Kopf der HTML-Seite lassen sich sowohl globale als auch spezielle Styles für bestimmte HTML-Elemente und Objekte festlegen.

```

<user.htm>
<html>
<head>
  <meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">
  <title>IOs mit AJAX</title>
  <style type="text/css">

```

```

td
{ text-align:center;
  font-family:Arial,Verdana, Helvetica;
  font-size:14px;}
td.header
{ font-weight:bold;}
td.nr
{ width:25px;}
td.caption
{ width:60px;}
td.state
{ width:50px;}
input
{ text-align:center;
  font-family:Arial,Verdana, Helvetica;
  font-size:14px;
  height:22px;
  width:48px;}
</style>
<script language="JavaScript" type="text/javascript">
  JavaScript-Teil
</script>
</head>

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/25-style.html

Die Style-Vorgaben im Bereich *td* werden auf alle Tabellenzellen angewendet. Die Vorgaben in *td.nr*, *td.header* und *td.caption* zeigen nur da Wirkung, wo in der *td*-Deklaration unter *class* die Bezeichnung hinter dem Punkt angegeben ist.

```

<table border="1">
  <tr>
    <td class="header nr">Nr</td>
    <td class="header caption">&nbsp;</td>
    <td class="header state">State</td>
    <td class="header">Switch</td>
    <td class="header">Switch</td>
  </tr>
  <tr>
    <td>0</td>
    <td>Output</td>
    <td id="output0">OFF</td>

```

```

<td><input type="button" value="ON" onclick="setOutput(1);"></td>
<td><input type="button" value="OFF" onclick="setOutput(0);"></td>
</tr>
</table>

```

Es können auch mehrere Style-Klassen mit Leerzeichen getrennt angegeben werden.

Die Style-Vorgaben für die Buttons wurden im Style-Bereich über *input* festgelegt.

Nr		State	Switch	Switch
0	Output	OFF	ON	OFF

Mit nur wenig Quelltext kommt man über Style-Vorgaben zum gleichen Ergebnis wie im vorangegangenen Beispiel.

Auch bei größeren Projekten Einheitlichkeit mit CSS

Die gleichen CSS-Vorgaben lassen sich genauso auf unser Beispiel für alle 12 IOs anwenden.

Dazu muss natürlich im Head-Bereich die Style-Vorgabe eingefügt werden. Ferner ist es nötig im HTML-Teil nur die *class* Deklaration einzufügen.

```

<table ID="ioTable" border="1">
  <tr>
    <td class="header nr">Nr.</td>
    <td class="header caption">&nbsp;</td>
    <td class="header state">State</td>
    <td class="header state">Counter</td>
    <td class="header caption">&nbsp;</td>
    <td class="header state">State</td>
    <td class="header">Switch</td>
    <td class="header">Switch</td>
  </tr>
  <tr>
    <td>0</td>
    <td>Input</td>
    <td>OFF</td>
    <td>0</td>

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/26-style.html

```

        <td>Output</td>
        <td>OFF</td>
        <td><input onclick="setOutput(0,1);" type="button" value="ON"></td>
        <td><input onclick="setOutput(0,0);" type="button" value="OFF"></td>
    </tr>
    .....
</table>

```

Die Tabelle sieht dann so aus:

Nr.		State	Counter		State	Switch	Switch
0	Input	ON	16	Output	OFF	ON	OFF
1	Input	OFF	1	Output	OFF	ON	OFF
2	Input	OFF	2	Output	ON	ON	OFF
3	Input	ON	1	Output	OFF	ON	OFF
4	Input	ON	2	Output	OFF	ON	OFF
5	Input	ON	1	Output	OFF	ON	OFF
6	Input	ON	2	Output	OFF	ON	OFF
7	Input	ON	1	Output	ON	ON	OFF
8	Input	ON	2	Output	OFF	ON	OFF
9	Input	ON	1	Output	OFF	ON	OFF
10	Input	ON	2	Output	ON	ON	OFF
11	Input	ON	1	Output	OFF	ON	OFF

Ein weiterer Vorteil von CSS-Styles liegt darin, dass wenn das HTML-Grundgerüst einmal steht, das komplette Design der Webseite durch bloßes Ändern der Style-Vorgaben verändern werden kann.

Ein Beispiel:

Das wird um einen Punkt erweitert

```

tr.gray
{ background-color:#CCCCCC; }

```

Im HTML-Teil wird jeder zweiten Tabellenzeile im <tr>-Tag die Klasse *gray* eingefügt. Innerhalb des <table>-Tags wird die Eigenschaft border auf 0 gesetzt und es wird cellspacing gleich 0 gesetzt. Dadurch entfallen die Tabellentrennlinien und die einzelnen Zellen werden bündig aneinander angezeigt.

```

<body bgcolor="#FFFFFF" text="#000000">
  <table ID="ioTable" border="0" cellspacing="0">
    <tr>
      <td class="header nr">Nr.</td>
      <td class="header caption">&nbsp;</td>
      <td class="header state">State</td>
      <td class="header state">Counter</td>
      <td class="header caption">&nbsp;</td>
      <td class="header state">State</td>
      <td class="header">Switch</td>
      <td class="header">Switch</td>
    </tr>
    <tr class="gray">
      <td>0</td>
      <td>Input</td>
      <td>OFF</td>
      <td>0</td>
      <td>Output</td>
      <td>OFF</td>
      <td><input onclick="setOutput(0,1);" type="button" value="ON"></td>
      <td><input onclick="setOutput(0,0);" type="button" value="OFF"></td>
    </tr>
    <tr>
      <td>1</td>
      <td>Input</td>
      <td>OFF</td>
      <td>0</td>
      <td>Output</td>
      <td>OFF</td>
      <td><input onclick="setOutput(1,1);" type="button" value="ON"></td>
      <td><input onclick="setOutput(1,0);" type="button" value="OFF"></td>
    </tr>
    <tr class="gray">
      <td>2</td>
      <td>Input</td>
      <td>OFF</td>
      <td>0</td>
      <td>Output</td>
      <td>OFF</td>
      <td><input onclick="setOutput(2,1);" type="button" value="ON"></td>
      <td><input onclick="setOutput(2,0);" type="button" value="OFF"></td>

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/27-style.html

```
</tr>
.....
```

Eine kleine Anpassung nehmen wir auch noch in der JavaScript-Funktion *updateDisplay* vor.

```
function updateDisplay(ReceiveStr)
{
    var HexVal;
    var state = false;
    table = document.getElementById('ioTable');
    if (ReceiveStr.substring(0, 1) == 'i')
    {
        HexVal = HexToInt(ReceiveStr.substring(6, 10));
        for (i = 0; i < 12; i++)
        {
            if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
            {
                table.rows[i + 1].cells[2].innerHTML = 'ON';
                table.rows[i + 1].cells[2].style.color = '#006600';
                table.rows[i + 1].cells[2].style.fontWeight = 'bold';
            }
            else
            {
                table.rows[i + 1].cells[2].innerHTML = 'OFF';
                table.rows[i + 1].cells[2].style.color = '#000000';
                table.rows[i + 1].cells[2].style.fontWeight = 'normal';
            }
        }
    }
    if (ReceiveStr.substring(0, 1) == 'o')
    {
        HexVal = HexToInt(ReceiveStr.substring(7, 11));
        for (i = 0; i < 12; i++)
        {
            if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
            {
                table.rows[i + 1].cells[5].innerHTML = 'ON';
                table.rows[i + 1].cells[5].style.color = '#006600';
                table.rows[i + 1].cells[5].style.fontWeight = 'bold';
            }
            else
            {
                table.rows[i + 1].cells[5].innerHTML = 'OFF';
                table.rows[i + 1].cells[5].style.color = '#000000';
                table.rows[i + 1].cells[5].style.fontWeight = 'normal';
            }
        }
    }
    if (ReceiveStr.substring(0, 1) == 'c')
```

```

{ var Countervalue = ReceiveStr.split(';');
  for (i = 0; i < 12; i++)
  { table.rows[i + 1].cells[3].innerHTML = Countervalue[i + 1];
  }
}
}

```

Die Änderungen bewirken, dass für die Inputs und Outputs im ON-Fall die Schriftfarbe auf grün und die Schrift selbst auf fett gedruckt gesetzt wird.

Nr.		State	Counter		State	Switch	Switch
0	Input	ON	16	Output	OFF	ON	OFF
1	Input	OFF	1	Output	ON	ON	OFF
2	Input	OFF	2	Output	OFF	ON	OFF
3	Input	ON	1	Output	ON	ON	OFF
4	Input	ON	2	Output	ON	ON	OFF
5	Input	ON	1	Output	ON	ON	OFF
6	Input	ON	2	Output	ON	ON	OFF
7	Input	ON	1	Output	ON	ON	OFF
8	Input	ON	2	Output	ON	ON	OFF
9	Input	ON	1	Output	ON	ON	OFF
10	Input	ON	2	Output	ON	ON	OFF
11	Input	ON	1	Output	ON	ON	OFF

Mit minimalen Änderungen hat die Webseite nun ein völlig anderes Design.

Optimierte Webseiten für mobile Anzeigegeräte

Der große Vorteil beim Einsatz von Webtechniken ist, dass diese heute fast überall zur Verfügung stehen.

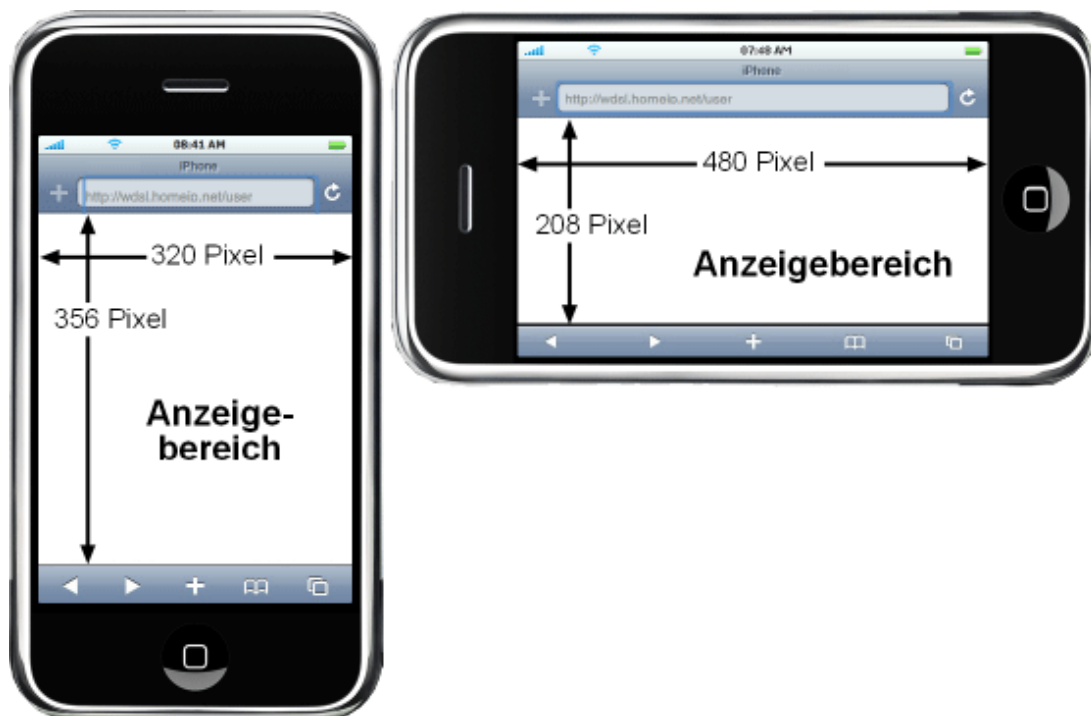
Visualisiert man die Zustände einer technischen Anlage auf einer Webseite und macht diese über das Internet zugänglich, kann die Anlage von überall überwacht werden. Dank moderner Smartphones mit UMTS Internet-Zugang sogar von unterwegs.

Allerdings ist die zur Verfügung stehende Anzeigegröße bei Smartphones gegenüber der eines PC eingeschränkt und variiert von Modell zu Modell.

Vorreiter der Smartphone Internet-Technik ist das iPhone, weshalb die folgenden Beispiele für das iPhone optimiert sind. Die gezeigte Technik lässt sich aber problemlos für alle anderen Modelle anpassen.

Beim iPhone gibt es zwei verschiedene Anzeigeformate:

- Hochformat 320 x 356 Pixel,
- Querformat 480 x 208 Pixel



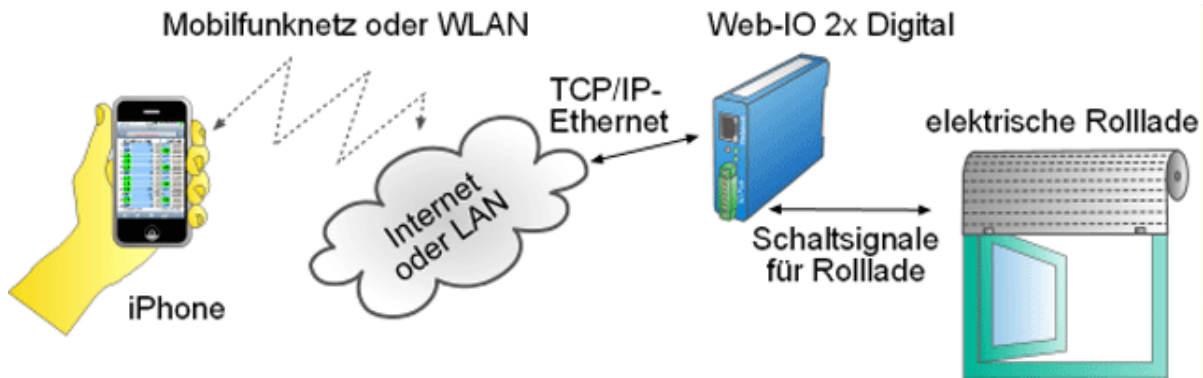
Selbst wenn man die Elemente einer Webseite Pixel-genau auf diese angepasst aufbaut, scaliert das iPhone die Seite auf einen größeren Anzeigebereich.

Um das zu verhindern, fügt man im Head-Bereich der Webseite folgendes Meta-Tag ein:

```
<meta name="viewport" content="width=320" />
```

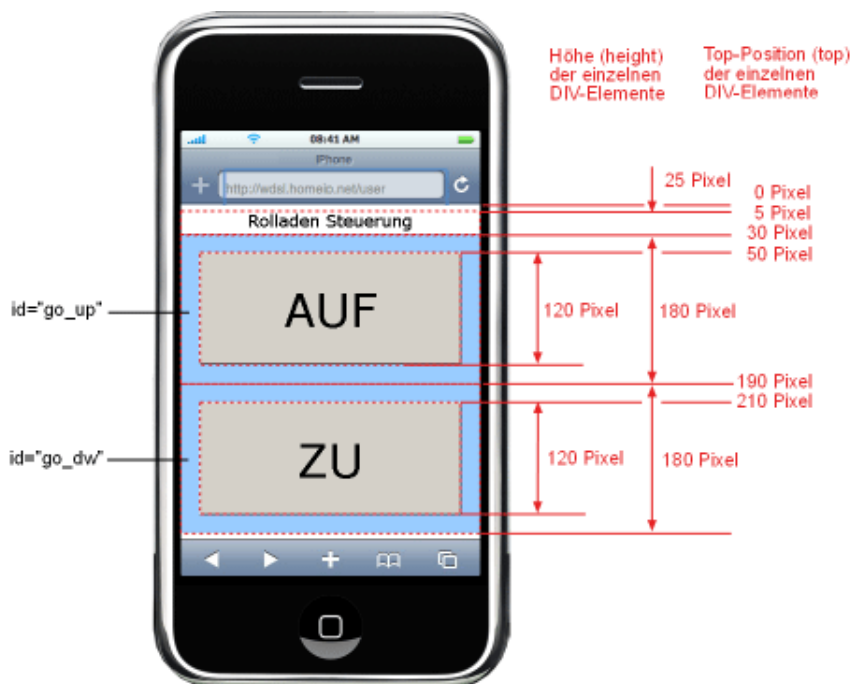
Mit *width* wird die gewünschte Angebreite vorgegeben.

Das erste Beispiel soll eine Rolladen Steuerung mit dem iPhone als Fernbedienung zeigen.



Auf dem Bildschirm sollen zwei Buttons zum Öffnen und Schließen der Rolllade zur Verfügung stehen. Die Buttons sollen auf einem blauen Hintergrund platziert sein. Sobald der Anwender einen der Buttons anklickt, soll der entsprechende Schaltbefehl zum Web-IO gesendet werden und sich als Quit- tierung der Hintergrund grau einfärben. Sobald die Antwort vom Web-IO empfangen wurde, dass erfolgreich geschaltet wurde, soll der Hintergrund wieder Blau gefärbt werden.

Die Bildschirmaufteilung gestaltet sich so:



Für die Buttons und die Hintergrundfelder, sowie für den beschreibenden Text werden im Kopf der Seite entsprechende Style-Vorgaben zu Größe, Farbe und Schriftgröße festgelegt.

<user.htm>

```
<html>
<head>
  <title>iPhone steuert Rollade</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="viewport" content="width=320" />
  <style type="text/css">
    * { font-family:Verdana, Helvetica; }
    .description { position:absolute;
                  text-align:center;
                  font-size:18px;
                  left:0px;
                  height:25px;
                  width:320px; }
    .confirm      { position:absolute;
                  background-color:#99CCFF;
                  left:0px;
                  height:160px;
                  width:320px; }
    .buttons      { position:absolute;
                  font-size:50px;
                  width:280px;
                  height:120px;
                  left:20px; }
  </style>
  <script language="JavaScript" type="text/javascript">
    JavaScript-Teil
  </script>
</head>
```



Dieses Beispiel finden
Sie in der Software-
Sammlung zum Buch.
Download unter:
[www.wut.de/
examples/
30-iphone.html](http://www.wut.de/examples/30-iphone.html)

Neu ist das Attribut *position:absolut*, welches eine pixelgenaue Positionierung erlaubt (Elemente ohne dieses Attribut schwimmen sonst im Fließtext der Webseite).

Im HTML-Teil werden den Objekten mittels der *class*-Deklarationen die Stylevorgaben zugewiesen.

```
<body>
  <div class="description" style="top:5px;">Rolladen Steuerung</div>
  <div id="go_up" class="confirm" style="top:30px;"></div>
  <input type="button" class="buttons" onclick="goup();" style="top:50px;" value="AUF">
  <div id="go_dw" class="confirm" style="top:190px;"></div>
```

```



```

Im JavaScript-Teil kommt wie gehabt die *HexToInt* Funktion zum Umrechnen von hexadezimalen Zahlen zum Einsatz. Für die Übergabe der Kommandos an das Web-IO wird wie gehabt die Funktion *DataRequest* verwendet.

Die Funktionen *godown* und *goup* werden über die beiden Buttons aufgerufen und übergeben der Funktion *DataRequest* das Kommando zum Setzen der Outputs 0 bzw. 1. Außerdem werden die Hintergrundfelder eingegraut.

```

function godown()
{
  DataRequest('outputaccess1?PW=&State=ON&')
  document.getElementById('go_dw').style.backgroundColor= '#CCCCCC';
}

function goup()
{
  DataRequest('outputaccess0?PW=&State=ON&')
  document.getElementById('go_up').style.backgroundColor= '#CCCCCC';
}

```

Die vom Web-IO empfangenen Daten werden von der Funktion *updateDisplay* ausgewertet. Sobald das Web-IO das Schalten eines Outputs bestätigt, wird der Hintergrund wieder blau gefärbt.

```

function updateDisplay(ReceiveStr)
{
  var HexVal
  var ReceiveData = ReceiveStr.split(';')
  if (ReceiveData[0].substring(0,1)=='o')
  {
    HexVal= HexToInt(ReceiveData[1])
    if ((HexVal & 1) == 1)
    {
      document.getElementById('go_up').style.backgroundColor= '#99CCFF';
    }
    if ((HexVal & 2) == 2)
    {
      document.getElementById('go_dw').style.backgroundColor= '#99CCFF';
    }
  }
}

```

}

Beim Anklicken der Buttons wird Output0 oder Output1 gesetzt. Bei vielen Rolladensteuerungen ist es so, dass die Rollade anfängt in die gewünschte Richtung zu laufen, wenn auf einem entsprechenden Eingang ein kurzer Schaltimpuls empfangen wurde.

Das hier gezeigte Beispiel schaltet allerdings die Outputs nur ein, aber nicht wieder aus. Um einen Puls zu generieren, gibt es zwei Möglichkeiten:

- Der gewünschte Output kann im Web-IO auf den Pulsmodus konfiguriert werden.
- Das JavaScript kann so erweitert werden, dass der Output nach der gewünschten Pulsdauer wieder zurückgesetzt wird.

Outputs für Pulsmodus konfigurieren

Um einen Output des Web-IO in den Pulsmodus zu konfigurieren wählen Sie im Navigationsbaum *Config >> Ports >> Outputs >> Output 0 >> Puls*.

The screenshot shows the W&T Web-IO configuration interface. On the left is a navigation tree with the following structure:

- Home
- User
- Config
 - Session Control
 - LogOut
 - Profiles
 - New Password
 - Device
 - Ports
 - Inputs
 - Outputs
 - Output 0
 - Logic
 - Puls (highlighted with a red arrow)
 - Output 1

The main content area displays the configuration for **Config >> Ports >> Outputs >> Output 0 >> Puls**. It includes the following fields and controls:

- Duration :** Dauer des Pulses in 1/1000 sek. The input field contains the value **1000**, with a red arrow pointing to it.
- Puls Polarity :** Polarität des Startpulses
 - ☐ negative
 - ☒ positive (with a red arrow pointing to it)
- Freier Speicher:** 35788 Bytes
- At the bottom, there are three buttons: **Zwischenspeichern** (with a red arrow pointing to it), **Rücksetzen**, and **Logout**.

Unter *Duration* wird die Pulslänge in ms eingetragen und *Puls polarity* sollte auf positiv gestetzt sein. Mit *Zwischenspeichern* werden die Änderungen zunächst temporär gespeichert.

Wenn die Konfiguration aller betroffenen Outputs abgeschlossen ist, werden alle Einstellungen über *Logout* > *Speichern* final ins Web-IO übernommen.

Outputs die über die Webseite eingeschaltet wurden, schalten sich nun nach der gesetzten Pulslänge automatisch wieder ab.

Outputs über JavaScript automatisch wieder abschalten

Um die Outputs über JavaScript nach einer definierten Zeit automatisch zurückzusetzen, müssen die Funktionen *godown* und *goup* erweitert werden.

```
function godown()
{
  DataRequest('outputaccess1?PW=&State=ON&')
  document.getElementById('go_dw').style.backgroundColor= '#CCCCCC';
  maintimer = setTimeout(„DataRequest('outputaccess1?PW=&State=OFF&')“, 1000);
}

function goup()
{
  DataRequest('outputaccess0?PW=&State=ON&')
  document.getElementById('go_up').style.backgroundColor= '#CCCCCC';
  maintimer = setTimeout(„DataRequest('outputaccess0?PW=&State=OFF&')“, 1000);
}
```

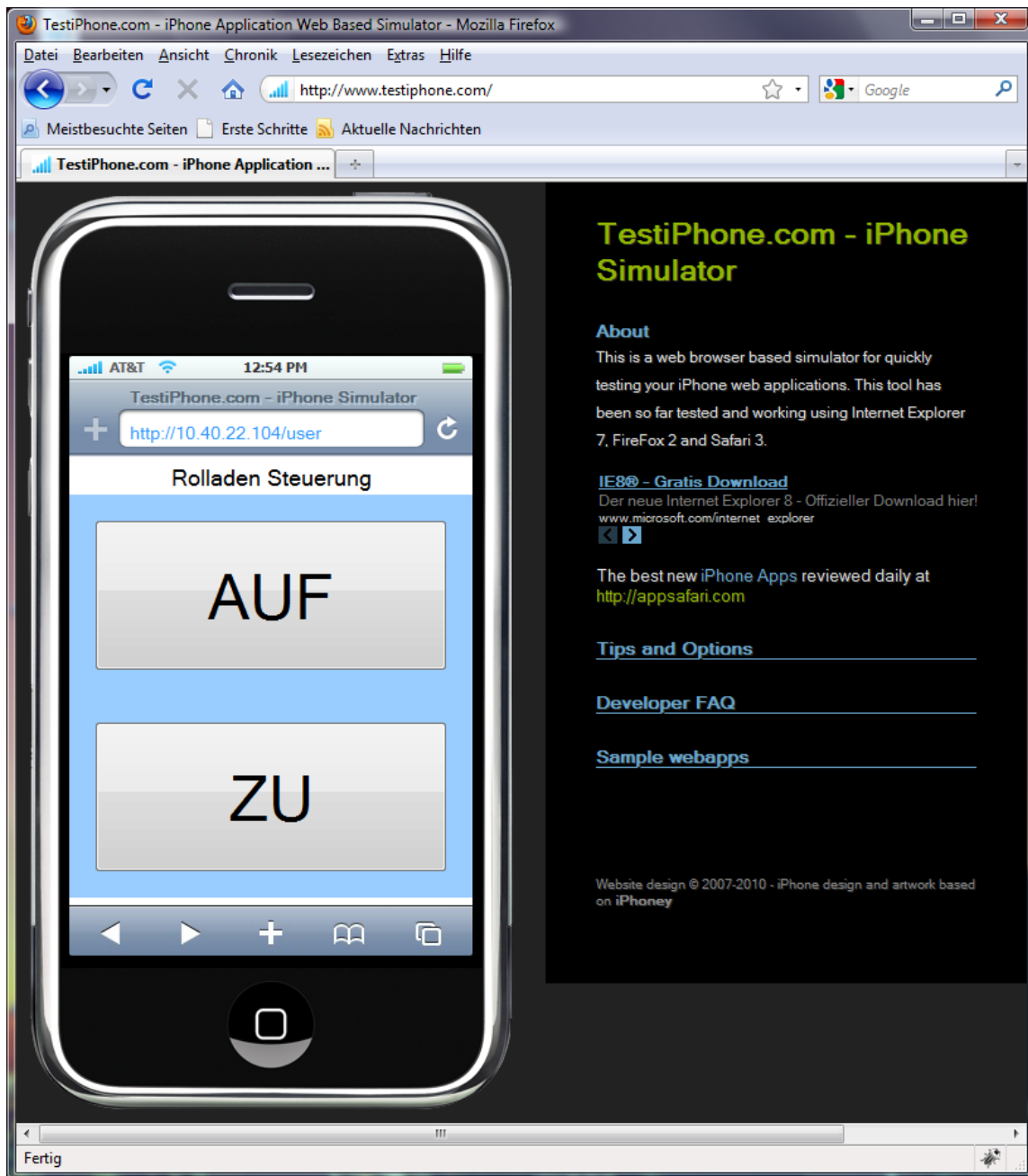
Über die *setTimeout* Funktion von JavaScript wird mit einer Verzögerung von 1000ms das Kommando zum Rücksetzen des Outputs an die Funktion *DataRequest* und damit an das Web-IO übergeben.



Um speziell für das iPhone generierte Webseiten auf korrekte Anzeige zu prüfen, benötigt man nicht zwingend ein iPhone. Unter www.testiphone.com können eigene Webseiten in einer virtuellen iPhone angezeigt werden.



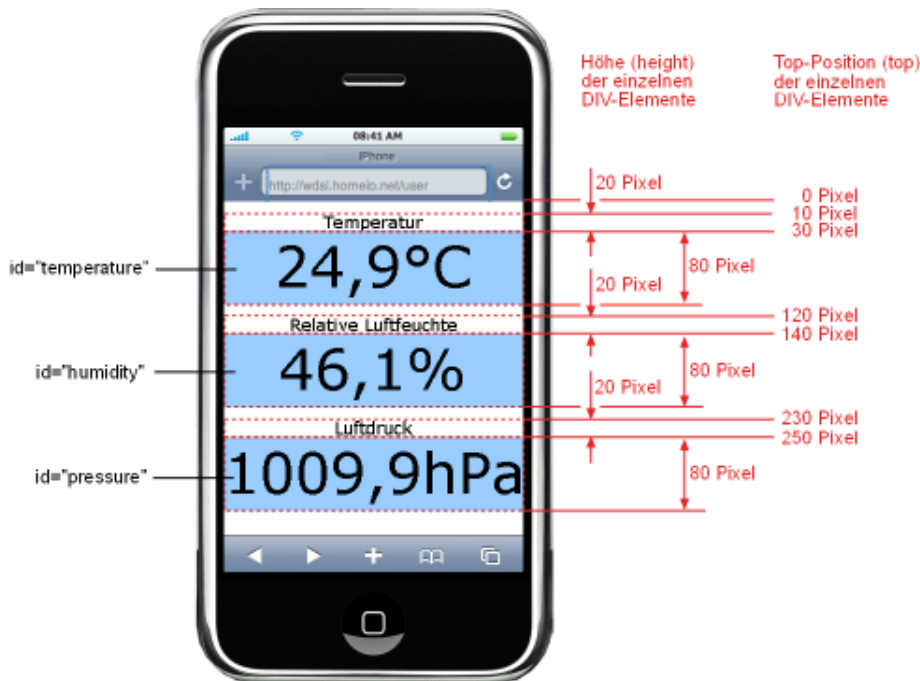
Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/31-iphone.html



Klimadaten auf dem iPhone anzeigen

In diesem Beispiel sollen mit einem Web-Thermohygro-Barograph Temperatur, Luftfeuchte und Luftdruck gemessen und auf dem iPhone angezeigt werden.

Die Anzeige soll folgendermaßen aufgebaut sein:



Wie im letzten Beispiel werden für die Anzeigeelemente im Kopf der Webseite Stylevorgaben eingefügt:

```
<user.htm>
<html>
  <head>
    <title>Aktuelles Klima</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <meta name="viewport" content="width=320" />
    <style type="text/css">
      * { font-family:Verdana, Helvetica; }
      .description { position:absolute;
                     text-align:center;
                     font-size:18px;
                     left:0px;
                     height:20px;
                     width:320px; }
      .value       { position:absolute;
                     background-color:#99CCFF;
                     text-align:center;
                     font-size:60px;
                     left:0px;
                     height:80px;
                     width:320px; }
    </style>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/32-iphone.html

```
<script language="JavaScript" type="text/javascript">  
    JavaScript-Teil  
</script>  
</head>
```

Die class-Zuweisungen werden wie gehabt im HTML-Teil eingepflegt:

```
<body>  
    <div class="description" style="top:10px; „>  
        Temperatur  
    </div>  
    <div id="temperature" class="value" style="top:30px;">  
        <w&t_tags=t1>°C  
    </div>  
    <div class="description" style="top:120px; „>  
        Relative Luftfeuchte  
    </div>  
    <div id="humidity" class="value" style="top:140px; „>  
        <w&t_tags=h1> %  
    </div>  
    <div class="description" style="top:230px; „>  
        Luftdruck  
    </div>  
    <div id="pressure" class="value" style="top:250px;">  
        <w&t_tags=pa>hPa  
    </div>  
</body>  
</html>
```

Die `<w&t_tags>` sind Platzhalter, die beim Laden der Webseite durch die aktuellen Werte für Temperatur, Luftfeuchte und Luftdruck ausgetauscht werden. Allerdings findet von Hause aus keine nachträgliche Aktualisierung dieser Werte statt. Diese Aufgabe wird wieder von AJAX übernommen.

Dazu werden wieder die Funktionen *DataRequest* und *updateDisplay* verwendet. Die Funktion *DataRequest* schließt mit einem *setTimeout* Aufruf mit dem sie sich zeitversetzt selber aufruft. Dadurch wird ein zyklischer Datenabgleich erreicht.

```

function DataRequest(SendString)
{
    var xmlHttp;
    if( window.ActiveXObject )           // Internet Explorer
    {
        xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
    }
    else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
    {
        xmlHttp = new XMLHttpRequest();
    }
    if (xmlHttp)
    {
        xmlHttp.onreadystatechange = function()
        {
            if (xmlHttp.readyState == 4)
            {
                if (xmlHttp.status == 200)
                {
                    updateDisplay(xmlHttp.responseText);
                    xmlHttp=null;
                }
            }
        }
        xmlHttp.open(„GET“, SendString, true);
        xmlHttp.setRequestHeader(„Connection“, „close“);
        xmlHttp.setRequestHeader(„If-Modified-Since“, „Thu, 1 Jan 1970 00:00:00 GMT“);
        xmlHttp.send(null);
        maintimer = setTimeout(„DataRequest('single')“, 5000);
    }
}

function updateDisplay(ReceiveStr)
{
    var ReceiveData = ReceiveStr.split(„;“);
    document.getElementById('temperature').innerHTML =
        ReceiveData[ReceiveData.length-3].
        substring(0,ReceiveData[ReceiveData.length-3].length-2) + '°C';
    //im original kein Zeilenumbruch!
    document.getElementById('humidity').innerHTML = ReceiveData[ReceiveData.length-2]
    document.getElementById('pressure').innerHTML = ReceiveData[ReceiveData.length-1]
}

DataRequest('single');

```

Die Funktion updateDisplay trägt die vom Web-Thermogarphen empfangenen Werte in die entsprechenden Div-Objekte ein.

Analoge Signale (Strom/Spannung) im iPhone Anzeigen.

Um die Spannungs- bzw. Strommesswerte eines Web-IO Analog im iPhone anzeigen zu lassen, muss die Webseite aus dem vorangegangenen Beispiel nur geringfügig geändert werden.

Die Style-Vorgaben können 1:1 übernommen werden, genau wie die JavaScript-Funktion *DataRequest*.

Angepasst wird nur die Funktion *updateDisplay* und der HTML-Teil, da ja nur zwei Werte angezeigt werden müssen.

```
function updateDisplay(ReceiveStr)
{
    var ReceiveData = ReceiveStr.split(",");
    document.getElementById('m1').innerHTML = ReceiveData[ReceiveData.length-2];
    document.getElementById('m2').innerHTML = ReceiveData[ReceiveData.length-1];
}

DataRequest('single');

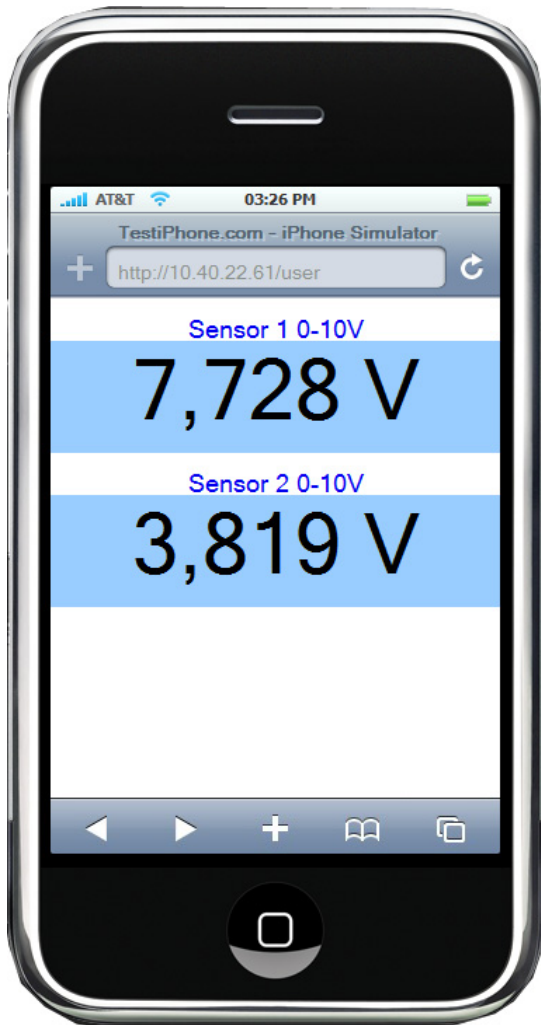
</script>
</head>
<body>
    <div class="description" style="top:10px; ">
        <w&t_tags=sensor1>
    </div>
    <div id="m1" class="value" style="top:30px; ">
        <w&t_tags=m1>
    </div>
    <div class="description" style="top:120px; ">
        <w&t_tags=sensor2>
    </div>
    <div id="m2" class="value" style="top:140px; ">
        <w&t_tags=m2>
    </div>
</body>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/33-iphone.html

Auch in diesem Fall werden einige Anzeigeelemente wie Sensorbezeichnung und erster gezeigter Wert durch *<w&t_tags>* vorgegeben.

Auf dem iPhone sieht das so aus:



Outputspannung mit dem iPhone setzen

Bei den Web-IO Analog Boxen können die zwei Kanäle so konfiguriert werden, dass entweder eine Spannung / ein Strom gemessen oder ausgegeben wird.

In diesem Beispiel soll Kanal 1 einen Spannungswert messen, Kanal 2 einen regelbaren Spannungswert ausgeben.

Dazu zunächst eine Übersicht über die benötigten Kommandos, die zum Web-IO Analog gesendet werden müssen um Werte zu lesen oder einen Output auf einen bestimmten Wert zu setzen:

single

liest die Werte aller vorhandenen Sensoren bzw. Inputs. Dieses Kommando kennen wir bereits aus den vorangegangenen Beispielen mit dem Web-Thermographen und den analogen Inputs.

Zur Erinnerung: Die Antwort des Web-IO auf das Kommando *single* besteht aus IP-Adresse, Device-Namen, Wert1, Wert2 getrennt durch Semikolon.

Beispiel:

```
10.40.22.61;WEBIO-042E54;3,231 V;7,937 V
```

Durch entsprechende Auswertung des Antwortstrings können also problemlos die aktuellen Werte ermittelt werden.

outputaccessx

Mit *outputaccess* kann für den Output *x* innerhalb der zulässigen Grenzen (0-10V) willkürlich ein Ausgangswert gesetzt werden. Anders als bei den digitalen Web-IO beginnt die Zählweise für *x* bei 1.

```
outputaccessx?PW=password&State=value&
```

Hinter *PW=* kann wenn nötig das Passwort eingesetzt werden.

Über den Parameter *State=* kann der einzustellende Wert festgelegt werden.

Beispiel:

```
outputaccess2?PW=blau&State=5.000&
```

setzt Output2 auf 5V wobei das Passwort *blau* übergeben wurde.

Die Antwort des Web-IO sieht so aus:

```
10.40.22.61;WEBIO-04D9D2;Sensor 2 0-10V;output2;5,000 V
```

Die Antwort des Web-IO auf das Kommando *Outputaccess* besteht aus IP-Adresse, Device-Namen, Sensor Nr. und Type, Output-Nr., Wert getrennt durch Semikolon.



Das Einregeln der gewünschten Ausgangsspannung kann bis zu einigen hundert Millisekunden Zeit in An-

spruch nehmen. Der Rückgabewert auf das outputaccess Kommando wird aber bereits ca. 20ms nach Empfang gesendet und gibt deshalb noch nicht den eingestellten Wunschwert zurück. Der eingeregelter Sollwert sollte also zeitversetzt mit dem single Kommando nachgefordert werden.

Der unterschiedliche Aufbau der Antworten des Web-IO erfordert eine entsprechende Differenzierung innerhalb der *updateDisplay* Funktion.

In den vorangegangenen Beispielen haben wir bereits mit der JavaScript-eigenen Funktion *split* gearbeitet, um einen String in einzelne Blöcke zu zerlegen.

Um für beide Antwortvarianten die gleiche *updateDisplay* Funktion nutzen zu können nehmen wir also das Semikolon als Trennzeichen und zerlegen die Empfangsdaten in einzelne Blöcke die dann als Teile eines Arrays zur Verfügung stehen.

Bei einer Antwort auf das *outputaccess* Kommando steht im vorletzten Block *outputx*

Diese Information kann über eine If-Abfrage ausgewertet werden.

```
function updateDisplay(ReceiveStr)
{ var ReceiveData = ReceiveStr.split(",");
  if (ReceiveData[ReceiveData.length - 2].substring(0, 1) == 'o')
  { maintimer = setTimeout("DataRequest('single')", 750);
  }
  else
  { document.getElementById('m1').innerHTML = ReceiveData[ReceiveData.length-2];
    document.getElementById('m2').innerHTML = ReceiveData[ReceiveData.length-1];
  }
}
```

Beginnt der vorletzte Block mit *o* wird mit 750ms Zeitversatz das *single* Kommando gesendet. Mit den Werten aus der Antwort auf *single* wird dann die Anzeige aktualisiert.

Das aktuelle Beispiel soll so aufgebaut sein, dass die Anzeige der analogen Werte wie schon zuvor in zwei Div-Elementen angezeigt werden. Darunter sollen 11 Buttons positioniert werden, über die glatte Spannungswerte von 0 bis 10 geschaltet werden können.

```
<user.htm>
<html>
  <head>
    <title>Aktuelles Klima</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <meta name="viewport" content="width=320" />
    <style type="text/css">
      * { font-family:Verdana, Helvetica; }
      .description { position:absolute;
                     text-align:center;
                     font-size:18px;
                     left:0px;
                     height:20px;
                     width:320px; }
      .value       { position:absolute;
                     background-color:#99CCFF;
                     text-align:center;
                     font-size:60px;
                     left:0px;
                     height:80px;
                     width:320px; }
      .bt          { position:absolute;
                     font-size:16px;
                     top:250px;
                     width:29px;
                     height:29px; }
    </style>

    <script language="JavaScript" type="text/javascript">
      Javascript-Teil
    </script>
  </head>
  <body>
    <div class="description" style="top:10px; ">
      <w&t_tags=sensor1>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/34-iphone.html

```

</div>
<div id="m1" class="value" style="top:30px;">
  <w&t_tags=m1>
</div>
<div class="description" style="top:120px; „>
  <w&t_tags=sensor2>
</div>
<div id="m2" class="value" style="top:140px; „>
  <w&t_tags=m2>
</div>
<input type="button" class="bt" onclick="setOutput(0);" style="left:6;" value="0">
<input type="button" class="bt" onclick="setOutput(1);" style="left:34;" value="1">
<input type="button" class="bt" onclick="setOutput(2);" style="left:62;" value="2">
<input type="button" class="bt" onclick="setOutput(3);" style="left:90;" value="3">
<input type="button" class="bt" onclick="setOutput(4);" style="left:118;" value="4">
<input type="button" class="bt" onclick="setOutput(5);" style="left:146;" value="5">
<input type="button" class="bt" onclick="setOutput(6);" style="left:174;" value="6">
<input type="button" class="bt" onclick="setOutput(7);" style="left:202;" value="7">
<input type="button" class="bt" onclick="setOutput(8);" style="left:230;" value="8">
<input type="button" class="bt" onclick="setOutput(9);" style="left:258;" value="9">
<input type="button" class="bt" onclick="setOutput(10);" style="left:286;" value="10">
</body>
</html>

```

Durch Klick auf einen der Buttons wird die `setOutput` Funktion aufgerufen, wobei der einzustellende Spannungswert als Parameter mitgegeben wird.

```

function setOutput(Value)
{
  DataRequest('outputaccess2?PW=&State='+ Value + '&');
}

```

Die Funktion *Commandloop* sorgt durch zyklisches Senden von *single* für eine kontinuierliche Aktualisierung der Werte

```

function Commandloop()
{
  DataRequest('single');
  maintimer = setTimeout(„DataRequest('single')“, 2000);
}

```

```

function DataRequest(SendString)

```

```
{ var xmlHttp;
  if( window.ActiveXObject )      // Internet Explorer
  { xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
  }
  else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
  { xmlHttp = new XMLHttpRequest();
  }
  if (xmlHttp)
  { xmlHttp.onreadystatechange = function()
    { if (xmlHttp.readyState == 4)
      { if (xmlHttp.status == 200)
        { updateDisplay(xmlHttp.responseText);
          xmlHttp=null;
        }
      }
    }
  }
  xmlHttp.open(„GET“, SendString, true);
  xmlHttp.setRequestHeader(„Connection“, „close“);
  xmlHttp.setRequestHeader(„If-Modified-Since“, „Thu, 1 Jan 1970 00:00:00 GMT“);
  xmlHttp.send(null);
}

function updateDisplay(ReceiveStr)
{ var ReceiveData = ReceiveStr.split(„;“);
  if (ReceiveData[ReceiveData.length - 2].substring(0, 1) == 'o')
  { maintimer = setTimeout(„DataRequest('single')“, 750);
  }
  else
  { document.getElementById('m1').innerHTML = ReceiveData[ReceiveData.length-2];
    document.getElementById('m2').innerHTML = ReceiveData[ReceiveData.length-1];
  }
}

Commandloop();
```

Graphische Elemente über Web-IO Status ändern

Bis hierher haben wir Messergebnisse und Status der Web-IO Geräte mit reinen Text- bzw. Farbanzeigen visualisiert.

In kleineren Applikationen reicht das meist auch völlig aus.

Bei komplexeren Projekten erhöhen grafische Elemente aber erheblich die Übersicht.

Im folgenden Beispiel soll im Browser eine Signallampe angezeigt werden, die bei Statuswechsel an Input0 die Farbe wechselt. Bei ON-Status soll die Lampe rot angezeigt werden, bei OFF-Status grün.

Dazu werden zwei Bilddateien (rote Lampe, grüne Lampe) benötigt.



Beide Bilder werden Pixel-genau übereinander gelegt.

Das geschieht über entsprechende Stylevorgaben und die HTML-Einbindung:

```
<user.htm>
<html>
  <head>
    <title>Aktuelles Klima</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <meta name="viewport" content="width=320" />
    <style type="text/css">
      * {background-color:#808080 }
      .lamp { position:absolute;
              left:10px;
              top:10px;
              border:none; }
    </style>

    <script language="JavaScript" type="text/javascript">

    </script>
  </head>
  <body>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/40-grafic.html

```


</body>
</html>
```

Das Bild der grünen Lampe liegt unter dem Bild der roten Lampe. Über *style="visibility:hidden"* ist das Bild der roten Lampe auf unsichtbar gesetzt.

Im JavaScript-Teil wird wie gehabt mit den Funktionen *HexToInt* und *DataRequest* gearbeitet.

```
function HexToInt(HexStr)
{
    var TempVal;
    var HexVal=0;
    for( var i=0; i<HexStr.length;i++)
    {
        if (HexStr.charCodeAt(i) > 57)
        {
            TempVal = HexStr.charCodeAt(i) - 55;
        }
        else
        {
            TempVal = HexStr.charCodeAt(i) - 48;
        }
        HexVal=HexVal+TempVal*Math.pow(16, HexStr.length-i-1);
    }
    return HexVal;
}

function DataRequest(SendString)
{
    var xmlHttp;
    if( window.ActiveXObject )           // Internet Explorer
    {
        xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
    }
    else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
    {
        xmlHttp = new XMLHttpRequest();
    }
    if (xmlHttp)
    {
        xmlHttp.onreadystatechange = function()
```

```

    { if (xmlHttp.readyState == 4)
      { if (xmlHttp.status == 200)
        { updateDisplay(xmlHttp.responseText);
          xmlHttp=null;
        }
      }
    }

    xmlHttp.open("GET", SendString, true);
    xmlHttp.setRequestHeader("Connection", "close");
    xmlHttp.setRequestHeader("If-Modified-Since", "Thu, 1 Jan 1970 00:00:00 GMT");
    xmlHttp.send(null);
    maintimer = setTimeout("DataRequest('input?PW=&')", 500);
  }
}

```

In der Funktion *updateDisplay* wird der Status von Input0 durch Ausmaskieren ermittelt und das Bild der roten Lampe bei ON-Status auf *visible* gesetzt.

```

function updateDisplay(ReceiveStr)
{ var HexVal
  var ReceiveData = ReceiveStr.split(';');
  if (ReceiveData[0].substring(0,1)=='i')
  { HexVal= HexToInt(ReceiveData[1])
    if ((HexVal & 1) == 1)
    { document.getElementById('lamp').style.visibility = 'visible';
    }
    else
    { document.getElementById('lamp').style.visibility = 'hidden';
    }
  }
}

```

Für die erste Statusabfrage wird DataRequest zeitversetzt aufgerufen.

```

maintimer = setTimeout("DataRequest('input?PW=&')", 500);

```

Einen Output grafisch visualisieren und schalten

Auch den Zustand eines Outputs kann man auf die gleiche Weise grafisch anzeigen. Als Bild soll in diesem Fall ein Knebel-schalter angezeigt werden:



Ein Klick mit der Maus auf den Schalter soll einen Zustandswechsel des Outputs auslösen.

Auch die Bilder für die Output-Schalter werden über Stylevorgaben und HTML-Einbindung in der Webseite positioniert.

```
<style type="text/css">
* {background-color:#808080 }
.switch { position:absolute;
          cursor:pointer;
          left:10px;
          top:10px;
          border:none; }
</style>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/41-grafic.html

Das Attribut *cursor:pointer* sorgt dafür, dass sich der Maus-Cursor bei Überfahren des Bildes ändert so verändert wie bei einem Hyperlink. Der Anwender wird so darauf hingewiesen, dass die Schalter eine Funktion haben, die durch Mausklick ausgelöst werden kann.

```
<body>
  <a href="javascript:setOutput(0)">
    
  </a>
  <a href="javascript:setOutput(0)">
    
  </a>
```

```
</body>
```

Über das `<a>`-Tag wird den Bildern ein Hyperlink hinzugefügt. In diesem Fall wird auf die JavaScript-Funktion `setOutput` verlinkt, wobei als Parameter `0` für `Output0` mitgegeben wird.

```
<a href="javascript:setOutput(0)">
```

Die Funktionen `HexToInt` und `DataRequest` werden wie immer übernommen, wobei der zeitversetzte Selbstaufruf von `DataRequest` entfällt.

```
function setOutput(Nr)
{
  DataRequest('outputaccess' + Nr + '?PW=&State=TOGGLE&');
}
```

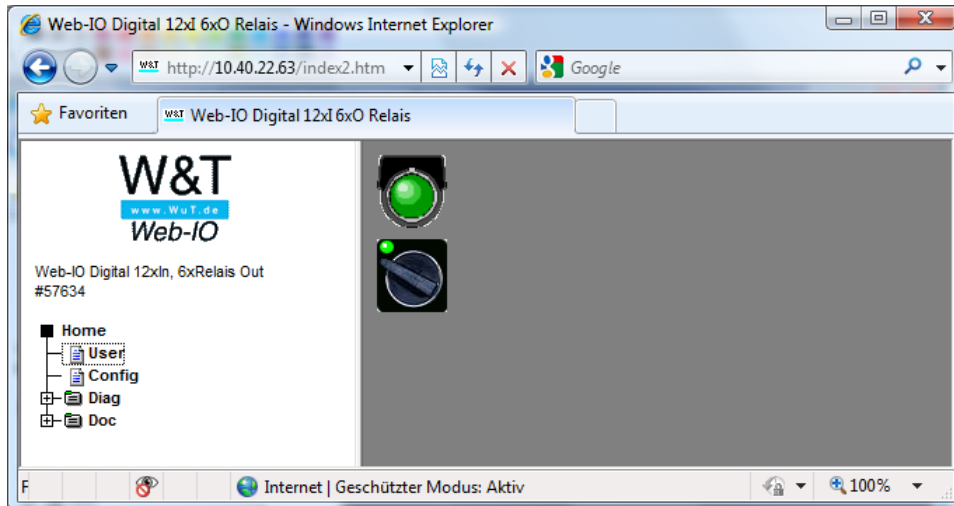
Die Funktion `setOutput` sendet über `DataRequest` das `outputaccess` Kommando zum Web-IO, wobei als Parameter `State=TOGGLE` angegeben ist, was zu einem Zustandswechsel des Outputs führt.

Über die an die Output-Anzeige angepasste Funktion `updateDisplay` wird die Antwort des Web-IO ausgewertet und der entsprechende Status angezeigt.

```
function updateDisplay(ReceiveStr)
{
  var HexVal
  var ReceiveData = ReceiveStr.split(';')
  if (ReceiveData[0].substring(0,1)=='o')
  {
    HexVal= HexToInt(ReceiveData[1])
    if ((HexVal & 1) == 1)
    {
      document.getElementById('switch').style.visibility = 'visible';
    }
    else
    {
      document.getElementById('switch').style.visibility = 'hidden';
    }
  }
}
```

Inputs und Outputs grafisch anzeigen

Als nächstes sollen sowohl ein Input als auch ein Output grafisch dargestellt werden.



Dazu müssen im Grunde nur die zwei vorangegangenen Beispiele kombiniert werden.

```
<user.htm>
<html>
  <head>
    <title>Schalten mit dem Web-IO</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <meta name="viewport" content="width=320" />
    <style type="text/css">
      *      { background-color:#808080 }
      .lamp  { position:absolute;
                left:10px;
                top:10px;
                border:none; }
      .switch { position:absolute;
                cursor:pointer;
                left:10px;
                top:70px;
                border:none; }

    </style>

    <script language="JavaScript" type="text/javascript">
      JavaScript-Teil
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/42-grafic.html

```

</script>
</head>
<body>
  
  
  <a href="javascript:setOutput(0)">
  
  </a>
  <a href="javascript:setOutput(0)">
    
  </a>
</body>
</html>

```

Im JavaScript-Teil werden die Funktionen *HexToInt*, *DataRequest* und *setOutput* unverändert übernommen.

Für die zyklische Abfrage der Inputs wird wieder eine Funktion *Commandloop* angelegt, die sich zeitversetzt selbst aufruft.

```

function Commandloop()
{
  DataRequest('input?PW=&');
  maintimer = setTimeout(„Commandloop()“, 500);
}

```

Die *updateDisplay* Funktion muss so angepasst werden, dass bei den vom Web-IO empfangenen Daten zunächst überprüft wird, ob es um den Input oder den Output geht.

```

function updateDisplay(ReceiveStr)
{
  var HexVal
  var ReceiveData = ReceiveStr.split(';')
  if (ReceiveData[0].substring(0,1)=='i')

```

```
{  HexVal= HexToInt(ReceiveData[1])
  if ((HexVal & 1) == 1)
  { document.getElementById('lamp').style.visibility = 'visible';
  }
  else
  { document.getElementById('lamp').style.visibility = 'hidden';
  }
}
if (ReceiveData[0].substring(0,1)=='o')
{  HexVal= HexToInt(ReceiveData[1])
  if ((HexVal & 1) == 1)
  { document.getElementById('switch').style.visibility = 'visible';
  }
  else
  { document.getElementById('switch').style.visibility = 'hidden';
  }
}
}
```

Am Ende des Scriptes wird einmal der Output-Status abgefragt in dem zeitversetzt das *output* Kommando an das Web-IO gesendet wird. Ferner wird die Funktion *Commandloop* ebenfalls zeitversetzt aufgerufen.

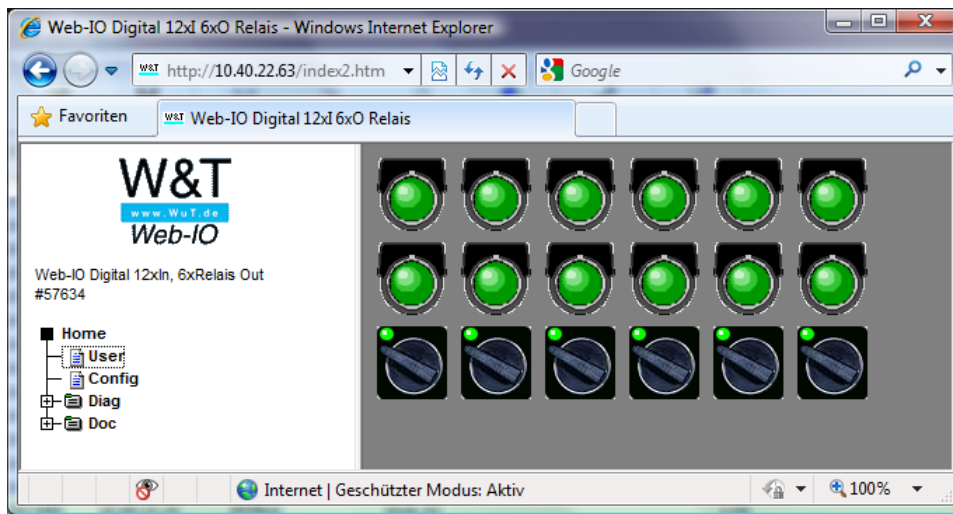
```
maintimer = setTimeout(„Commandloop()“, 500);
maintimer = setTimeout(„DataRequest('output?PW=&')“, 500);
```

Das zeitversetzte Aufrufen hat den Hintergrund, dass der JavaScript-Teil vom Browser sofort nach Einlesen startet. Es kann aber sein, dass zu dem Zeitpunkt der HTML-Teil noch nicht vollständig eingelesen wurde. Da im JavaScript-Teil auf Objekte des HTML-Teils (*id=“...“*) verwiesen wird, kennt JavaScript diese Objekte beim Start noch nicht und es würde eine Fehlermeldung geben.

Mehre IOs grafisch visualisieren

Mit relativ wenig Aufwand kann das vorangegangene Beispiel so erweitert werden, dass für alle Inputs eine Lampe und für alle Outputs ein Schalter dargestellt werden.

Das folgende Beispiel soll eine Visualisierung für ein Web-IO 12xDigital in, 6xRelais Out zeigen.



Dazu müssen die Style-Vorgaben angepasst bzw. erweitert werden. Darüberhinaus ist es nötig, die einzelnen Bilder im HTML-Teil anzulegen.

```
<user.htm>
<html>
  <head>
    <title>Schalten mit dem Web-IO</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <style type="text/css">
      *      { background-color:#808080 }
      .lamp0 { position:absolute;
                top:10px;
                border:none; }
      .lamp1 { position:absolute;
                top:70px;
                border:none; }
      .switch { position:absolute;
                cursor:pointer;
                top:130px;
                border:none; }
    </style>
    <script language="JavaScript" type="text/javascript">
      JavaScript-Teil
    </script>
  </head>
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/43-grafic.html

```
<body>
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
```

```





<a href="javascript:setOutput(0)">

</a>
<a href="javascript:setOutput(0)">
  
</a>
<a href="javascript:setOutput(1)">

</a>
<a href="javascript:setOutput(1)">
  
</a>
<a href="javascript:setOutput(2)">

</a>
<a href="javascript:setOutput(2)">
  
</a>
<a href="javascript:setOutput(3)">

</a>
<a href="javascript:setOutput(3)">
  
</a>
<a href="javascript:setOutput(4)">

```

```

</a>
<a href="javascript:setOutput(4)">
  
</a>
<a href="javascript:setOutput(5)">
  
</a>
<a href="javascript:setOutput(5)">
  
</a>
</body>
</html>
```

Im HTML-Teil hat jedes oben liegende Bild eine eindeutige *id*. Um im JavaScript-Teil eine einfache automatisierte Abarbeitung zu ermöglichen, wurden die *ids* fortlaufend durchnummeriert.

Als Hyperlink ist für die Schalter der Aufruf der Funktion *setOutput* angegeben, wobei als Parameter die Nr. des Outputs angegeben ist, der geschaltet werden soll.

```
function setOutput(Nr)
{ DataRequest('outputaccess' + Nr + '?PW=&State=TOGGLE&');
}
```

Die JavaScript Funktionen *HexToInt*, *DataRequest* und *Commandloop* werden wieder übernommen.

Die Funktion *updateDisplay* muss dahingehend geändert werden, dass alle 12 Inputs bzw. die 6 Outputs berücksichtigt werden.

```
function updateDisplay(ReceiveStr)
{ var HexVal
  var ReceiveData = ReceiveStr.split(';')
  if (ReceiveData[0].substring(0,1)=='o')
```

```

{  HexVal= HexToInt(ReceiveData[1])
  for (var i = 0; i < 6; i++)
  {  //alert(i);
    if ((HexVal & Math.pow(2,i)) == Math.pow(2,i))
      {  document.getElementById('switch'+i).style.visibility = 'visible';
        }
    else
      {  document.getElementById('switch'+i).style.visibility = 'hidden';
        }
  }
}

if (ReceiveData[0].substring(0,1)=='i')
{  HexVal= HexToInt(ReceiveData[1])
  for (var i = 0; i < 12; i++)
  {  if ((HexVal & Math.pow(2,i)) == Math.pow(2,i))
      {  document.getElementById('lamp'+i).style.visibility = 'visible';
        }
    else
      {  document.getElementById('lamp'+i).style.visibility = 'hidden';
        }
  }
}
}

```

Am Ende des Scriptes wird einmal der Output-Status abgefragt, in dem zeitversetzt das *output* Kommando an das Web-IO gesendet wird. Ferner wird die Funktion *Commandloop* ebenfalls zeitversetzt aufgerufen.

```

maintimer = setTimeout(„Commandloop()“, 500);
maintimer = setTimeout(„DataRequest('output?PW=&')“, 500);

```

HTML-Elemente automatisch mit JavaScript erzeugen

Schaut man sich den HTML-Teil des vorigen Beispiels an, ist der bereits recht ausladend, obwohl nur 18 Elemente dargestellt werden.

Nun ist die Anordnung der Elemente recht strukturiert (3 Zeilen à 6 Elemente).

In solchen Fällen bietet es sich an, die Elemente in einer For-Schleife mit der JavaScript-Funktion `document.write` anzulegen.

```
function createDisplay()
{
  for(i=0; i<6; i++)
  {
    document.write('');
    document.write('');
    document.write('');
    document.write('');
    document.write('<a href="javascript:setOutput(' +i+')" ">');
    document.write('');
    document.write('</a>');
    document.write('<a href="javascript:setOutput(' +i+')" ">');
    document.write('');
    document.write('</a>');
  }
}
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/44-grafic.html



Innerhalb von `document.write`-Anweisungen darf im realen Quelltext kein Zeilenumbruch sein.

Die Funktion `createDisplay` ersetzt den kompletten HTML-Teil des vorangegangenen Beispiels. Der Trick besteht darin, dass für jede der drei Zeilen die Elemente im Grunde gleich aufgebaut sind. Der einzige Unterschied ist die durch *left* vorgegebene horizontale Position und die *id*. Beides wird in der For-Schleife für jeden Durchlauf über *i* berechnet.

Natürlich muss die `createDisplay` Funktion zunächst aufgerufen werden.

```
createDisplay();
```

Der Rest des JavaScript-Teils kann unverändert übernommen werden.

Analoge Signale grafisch darstellen

Für die Anzeige digitaler Schaltzustände kommt man mit 2 Bildern aus, von denen je nach Zustand eins angezeigt wird.

Bei analogen Signalen kann der anzuzeigende Wert innerhalb bestimmter Grenzen jede beliebige Größe haben. Eine grafische Anzeige kann deshalb nur in Stufen erfolgen. Bei einem Signal zwischen 0V und 10V könnte man z.B. in 0,5V Schritten vorgehen.

Je nach gemessenem Wert wird dann ein Bild angezeigt, welches den entsprechenden Schritt wiedergibt.

In diesem Beispiel soll die Spannung, die von einem Web-IO Analog gemessen wird, als Zeigerinstrument angezeigt werden.

Dazu werden 21 Bilder benötigt, die den entsprechenden Zeigerausschlag zeigen.



Die Einbindung von 21 Bildern per HTML in eine Webseite würde wieder einen sehr ausladenden Quelltext mit sich bringen.

Deshalb schaffen wir eine JavaScript Funktion, die ein Grafikobjekt generiert, in dem alle benötigten Bilder vorgeladen werden. Diese Funktion sollte eine *Set*-Funktion beinhalten, über die bestimmt werden kann, welches der Bilder aktuell angezeigt werden soll.

Wir geben der Funktion den Namen *createMultipic*. Für eine einfache Handhabung sollen bei Aufruf der Funktion die folgenden, benötigten Informationen bzw. Parameter übergeben werden:

id

Eindeutige id über welche das Grafikobjekt später identifiziert werden kann.

img_name

Alle benötigten Bilder müssen den gleichen Namen gefolgt von einer fortlaufenden Nummer haben. Die Nummerierung beginnt bei 0.

Beispiel. volt0.gif, volt1.gif, volt2.gif, ,volt20.gif

img_name ist in diesem Fall *volt*.

img_count

Mit *img_count* wird die höchste Bildnummer übergeben. Für das Beispiel also die 20.

xpos

Pixelgenaue X-Postion, horizontal gemessen vom linken Rand des Browserfensters

ypos

Pixelgenaue Y-Postion, vertikal gemessen vom oberen Rand des Browserfensters

```
function createMultipic(id, img_name, img_ext, img_count, xpos, ypos)
{
    img_count++;
    var pics = new Array(img_count);
    for (var i=0; i<img_count; i++)
    {
        pics[i] = new Image();
        pics[i].src = img_name+i+'.'+img_ext;
    }
    this.Set = function(i_count)
```

```

    { document.getElementById(id).src = pics[i_count].src;
    }

    document.write(„“);
}

```



Innerhalb von document.write-Anweisungen darf im realen Quelltext kein Zeilenumbruch sein.

Die Funktion *createMultipic* erzeugt ein Array und verwaltet jedes verwendete Bild als Mitgliedvariable dieses Arrays. Die Zuweisung der einzelnen Bilder erfolgt über eine For-Schleife.

Mit dem *document.write* Kommando wird das erste Bild mit der vorgegebenen *id* als HTML-Objekt an der gewünschten Position angelegt.

über die eingebettete Funktion *set* kann später das gewünschte Bild ausgewählt werden.

Hier nun der komplette Quelltext für eine grafische Spannungsanzeige. Die Webseite ist für den Einsatz mit dem Web-IO Analog Typ 57662 vorgesehen:

```

<user.htm>
<html>
  <head>
    <title>Spannung messen mit dem Web-IO</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <meta name="viewport" content="width=320" />
    <script language="JavaScript" type="text/javascript">

      vmeter = new createMultipic('volt','http://www.wut.de/examples/volt','gif',20,10,10);

      function createMultipic(id, img_name, img_ext, img_count, xpos, ypos)
      { img_count++;
        var pics = new Array(img_count);
        for (var i=0; i<img_count; i++)
        { pics[i] = new Image();
          pics[i].src = img_name+i+'.'+img_ext;
        }
      }
    </script>
  </head>
  <body>
    <div id="volt" style="position: absolute; top: 10px; left: 20px;">
      
    </div>
  </body>
</html>

```

```

this.Set = function(i_count)
{ document.getElementById(id).src = pics[i_count].src;
}

document.write("");
}

function Commandloop()
{ DataRequest('single');
  maintimer = setTimeout("Commandloop()", 1000);
}

function DataRequest(SendString)
{ var xmlHttp;
  if( window.ActiveXObject )      // Internet Explorer
  { xmlHttp = new ActiveXObject( "Microsoft.XMLHTTP" );
  }
  else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
  { xmlHttp = new XMLHttpRequest();
  }
  if (xmlHttp)
  { xmlHttp.onreadystatechange = function()
    { if (xmlHttp.readyState == 4)
      { if (xmlHttp.status == 200)
        { updateDisplay(xmlHttp.responseText);
          xmlHttp=null;
        }
      }
    }
  }
  xmlHttp.open("GET", SendString, true);
  xmlHttp.setRequestHeader("Connection", "close");
  xmlHttp.setRequestHeader("If-Modified-Since", "Thu, 1 Jan 1970 00:00:00 GMT");
  xmlHttp.send(null);
}

function updateDisplay(ReceiveStr)
{ var ReceiveData = ReceiveStr.split(",");
  var value = ReceiveData[ReceiveData.length-2].split(' ')
  vmeter.Set(Math.round(parseFloat(value[0].replace(',','.'))*2));
}

```



Dieses Beispiel finden
Sie in der Software-
Sammlung zum Buch.
Download unter:
[www.wut.de/
examples/
50-grafic.html](http://www.wut.de/examples/50-grafic.html)

```

    maintimer = setTimeout(„Commandloop()“, 500);

    </script>
</head>
<body bgcolor="#000000">
    </body>
</html>

```

Zunächst wird über die *createMultipic* Funktion das Grafikobjekt *vmeter* generiert.

Das Abfragen des Spannungswertes erfolgt wie gehabt zyklisch über das Versenden des *single* Kommandos mittels der Funktion *Commandloop*.

In der Funktion *updateDisplay* wird die Antwort des Web-IO ausgewertet.

Das Web-IO antwortet mit

```
<IP-Adresse>;<Device-Name>;Wert1;Wert2
```

Beispiel

```
10.40.22.61;WEBIO-04D9D2;5,153 V;0,000 V
```

Über die *split*-Methode von JavaScript wird die Antwort des Web-IO in Blöcke zerlegt und in das Variablen-Array *ReceiveData* geschrieben.

In unserem Beispiel soll der Wert von Input1 angezeigt werden.

```
10.40.22.61;WEBIO-04D9D2;5,153 V;0,000 V
```

Der ist nach dem *split* im vorletzten Block von *ReceiveData* abgespeichert.

Über

```
var value = ReceiveData[ReceiveData.length-2].split(' ')
```

wird der Wert von der Einheit (hier V) separiert. (5,153 V).

Nun ist ein bißchen Rechenarbeit notwendig.

Der Wert ohne Einheit befindet sich nun als Zeichenkette (String) im Block1 des Variablen-Array *value*.

```
vmeter.Set(Math.round(parseFloat(value[0].replace(',', '.'))*2));
```

Um mit diesem Wert rechnen zu können, muss der String in eine Zahl gewandelt werden. In JavaScript erfolgt die Nachkommastellentrennung mit einem Punkt und nicht mit einem Komma, wie in der Antwort des Web-IO. Deshalb wird über *replace* zunächst das Komma durch einen Punkt ersetzt.

Die Wandlung von String auf Zahl (mit der man dann rechnen kann) erfolgt über die JavaScript-Funktion *parseFloat*

```
vmeter.Set(Math.round(parseFloat(value[0].replace(',', '.'))*2));
```

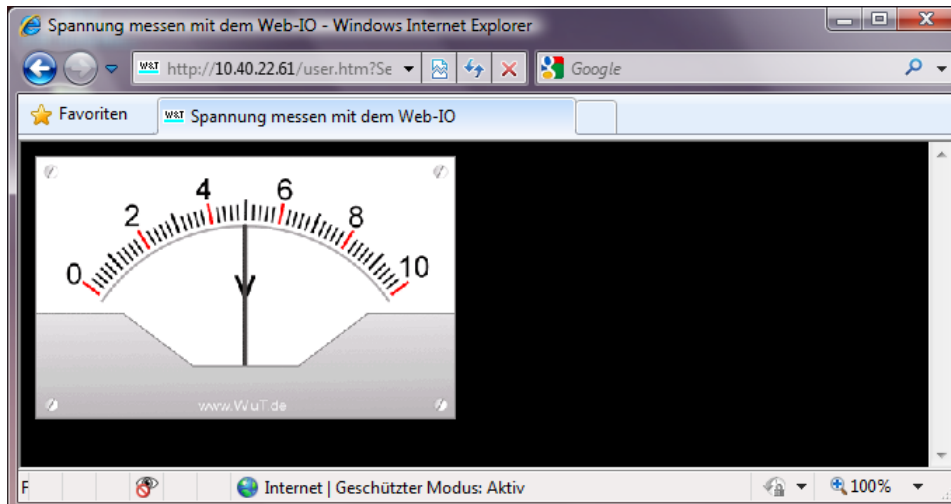
Da die Werte von 0 - 10V in Schritten a 0,5V angezeigt werden sollen (also Bild 0 bis Bild 20), wird der Zahlenwert mit 2 multipliziert.

```
vmeter.Set(Math.round(parseFloat(value[0].replace(',', '.'))*2));
```

Abschließend wird der Wert gerundet, da für die Nummerierung der Bilder ja nur Ganzzahlen zwischen 0 und 20 möglich sind. Dazu verwenden wir die Funktion *Math.round*

```
vmeter.Set(Math.round(parseFloat(value[0].replace(',', '.'))*2));
```

Mit so gewonnenen Ganzzahl als Parameter wird dann die *Set*-Funktion von *vmeter* aufgerufen. Diese Funktion setzt daraufhin das entsprechende Bild ein.



Wenn die Bilder in der richtigen Größe angelegt werden, wird so das iPhone schnell zum Voltmeter.



Mit überschaubarem Aufwand kann man auf diese Weise nahezu alles grafisch darstellen. Mit ein wenig Recherche findet

man im Internet Bilder und Grafiken zu allen Themen. Hierbei sollten allerdings die Urheberrechte beachtet werden.

Eigene Grafiken können mit den Programmen InScape , Gimp und in einfachen Fällen sogar mit Paint erstellt werden.

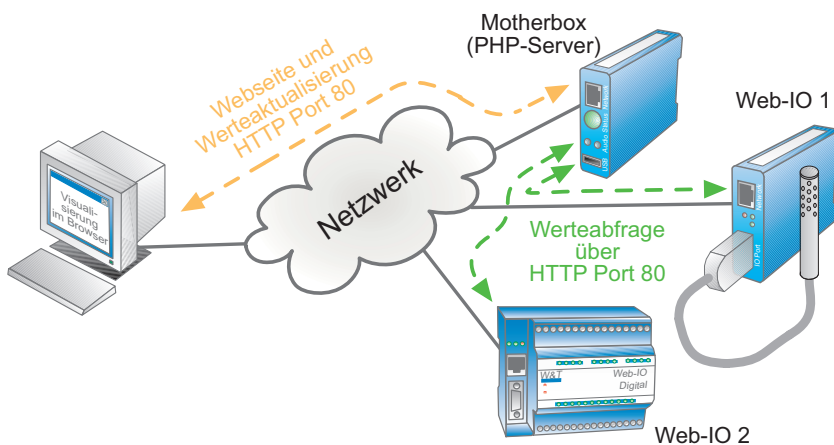
Eine Webseite für mehrere Web-IO

Die AJAX-Technik erlaubt ohne Laden eines zusätzlichen Applets das permanente Nachladen von Daten in eine bereits geladene Webseite.

Leider ist das Nachladen aber nur von dem Server möglich, von dem auch die Webseite geladen wurde. Damit ist die AJAX-Technik nicht ohne Weiteres einsetzbar, wenn es darum geht, die Status und Werte von mehreren Endgeräten auf einer gemeinsamen Webseite anzuzeigen.

Bereits bei kleineren Anwendungen macht grade das aber Sinn.

Abhilfe schafft hier ein PHP-fähiger Web-Server. Über ein kleines PHP-Script könnte der Server gegenüber dem Browser alle beteiligten Web-IO repräsentieren.



Der PHP-Server kann ein ohnehin vorhandener Server mit Webdiensten sein. Es kann aber genauso gut ein normaler PC mit installiertem XAMPP eingesetzt werden.

Eine Alternative zur Server oder PC-Lösung ist die W&T-Motherbox. Die Motherbox ist nicht größer als eine normale Web-IO Box, nimmt bei 24 Volt Versorgungsspannung gerade mal 5W auf und ist dabei ein vollwertiger PHP-Server.

PHP-Script als IP-Kommando-Gateway

Die Arbeitsweise der hier vorgestellten Lösung ist denkbar einfach.

Ein Beispiel:

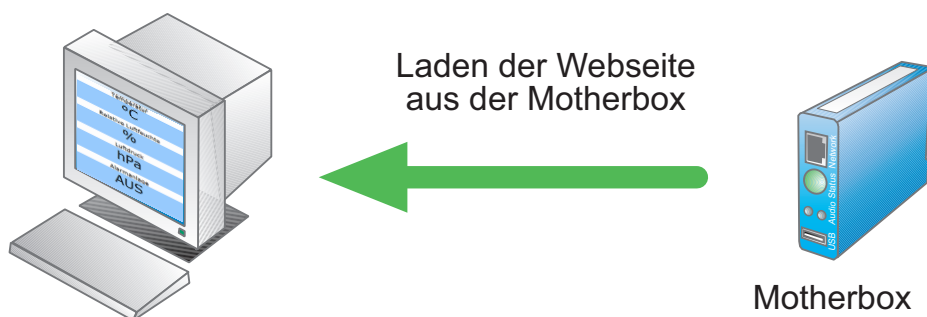
Eine Webseite, die z.B. die Klimadaten eines Raumes anzeigt, soll darüber hinaus anzeigen, ob die Alarmanlage scharf geschaltet ist.

Die Klimadaten werden von einem Web-Thermohygro-Barographen gemessen, der Alarmanlagenzustand wird von einem Web-IO Digital überwacht.

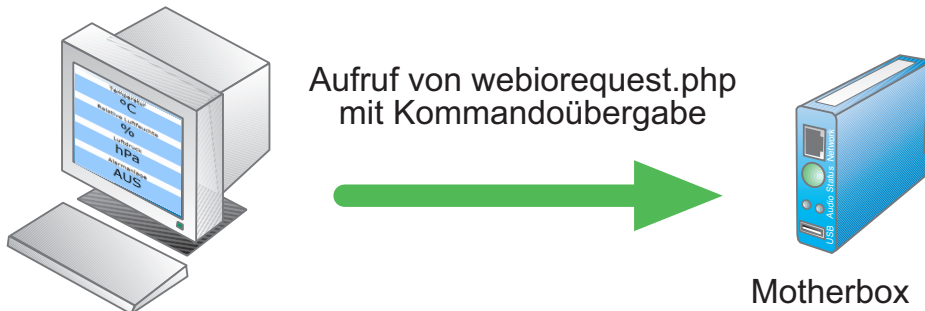
Die entsprechende Webseite wird auf dem PHP-Server abgelegt.

Ebenfalls auf dem PHP-Server befindet sich das PHP-Script zum nachträglichen Datenaustausch.

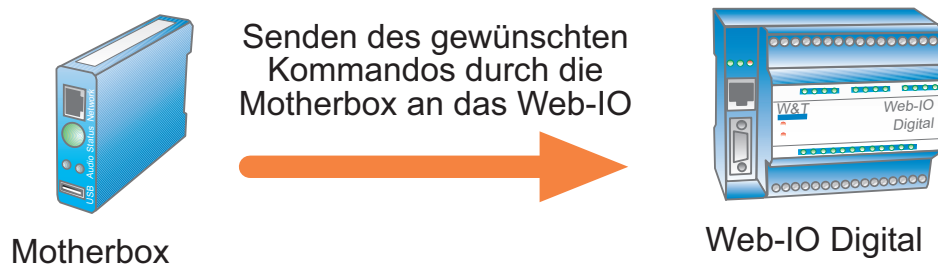
Im ersten Schritt wird die Webseite vom PHP-Server bzw. der Motherbox in den Browser geladen.



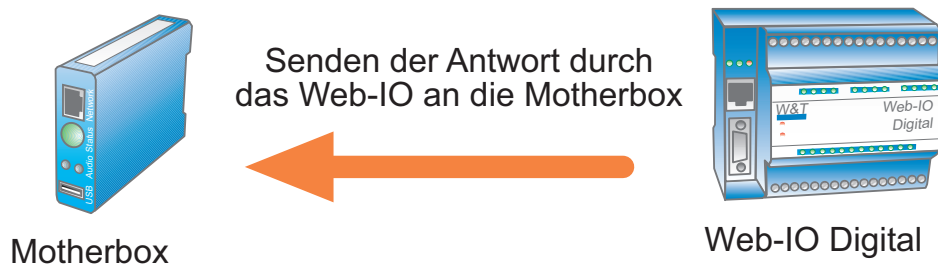
Per HTTP-Request wird das PHP-Script *webiorequest.php* aufgerufen und die gewünschten Parameter übergeben.



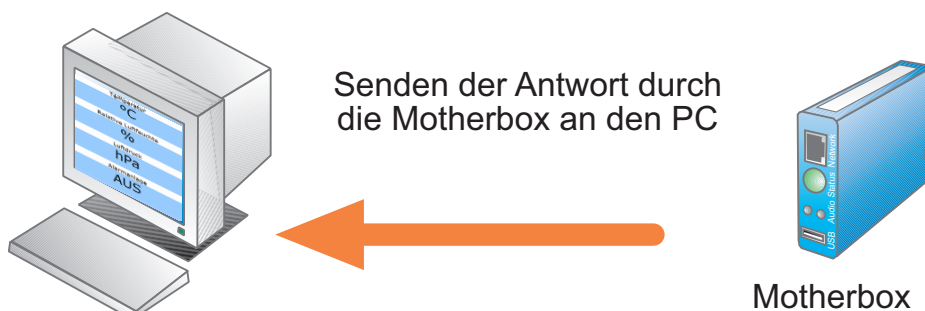
Der PHP-Server bzw. die Motherbox startet das PHP-Script, welches die mitgesendeten Parameter auswertet und das erkannte Kommando an das durch die IP-Adresse gewählte Web-IO sendet.



Das Web-IO sendet die Antwort an den PHP-Server bzw. die Motherbox.



Der PHP-Server bzw. die Motherbox empfängt die Antwort, wobei die empfangenen Daten vom `webiorequest.php` Script 1:1 an den Browser weitergeleitet werden.



Der Browser wertet die empfangenen Daten aus und aktualisiert entsprechend die Anzeige.

Das *webiorequest.php* Script hat folgenden Aufbau:

```
<?php
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");    // Datum aus Vergangenheit
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT"); // immer geändert
header("Cache-Control: no-store, no-cache, must-revalidate"); // HTTP/1.1
header("Cache-Control: post-check=0, pre-check=0", false);
header("Pragma: no-cache");                          // HTTP/1.0

parse_str($_SERVER['QUERY_STRING']);
if (!isset($_GET['PORT'])) {$PORT="80";}
if (!isset($_GET['NA'])) {$NA="OFF";}
if (!isset($_GET['PW'])) {$PW="";}
$fp=fsockopen($IP, $PORT, $errno, $error, 5);
if (!$fp)
{ printf("ERROR"); }
else
{ if ($COMMAND == "outputaccess")
  { IF ($MASK == "") {$MASK="0FFF";}
    IF (!isset($_GET['MASK'])) {$MASK="0FFF";}
    IF ($NA == "ON")
    { fputs($fp, "GET /".$COMMAND."?PW=".$PW."&Mask=".$MASK."&State=".$STATE."&NA=ON&");
    }
    else
    { fputs($fp, "GET /".$COMMAND."?PW=".$PW."&Mask=".$MASK."&State=".$STATE."&");
    }
  }
  else
  { if (substr($COMMAND,0,6) == "single")
    { fputs($fp, "GET /".$COMMAND);
    }
    else
    { fputs($fp, "GET /".$COMMAND."?PW=".$PW."&");
    }
  }
  do
  { $char=fgetc($fp);
    if($char!=chr(0))
    { echo $char;
```



Dieses Beispiel finden
Sie in der Software-
Sammlung zum Buch.
Download unter:
[www.wut.de/
examples/
webiorequest.php](http://www.wut.de/examples/webiorequest.php)

```
    }  
  }  
  while($char!=chr(0));  
fclose($fp);  
}  
?>
```

Dieses Script ist das Kernstück der hier beschriebenen Technik. Es wird bei Bedarf mittels eines AJAX-Requests vom Browser aus aufgerufen. Dabei werden über die URL alle benötigten Parameter übergeben:

IP

IP-Adresse des Web-IO, das angesprochen werden soll.

PORT

TCP-Port des Web-IO. Der Parameter *PORT* ist optional - ist er nicht angegeben, wird mit Port 80 gearbeitet.

COMMAND

Mögliche Kommandos sind: *output*, *input* oder *counter*, wobei dem eigentlichen Kommando noch die Nummer des Inputs oder Outputs folgen kann. Das Web-IO gibt den Status der Inputs oder Outputs zurück.

Als weiteres Kommando können mit *outputaccess* die Outputs eines Web-IO Digital oder Web-IO Analog gesetzt werden.

Für alle Web-Thermographen und das Web-IO Analog wird auch das Kommando *single* unterstützt.

PW

Administrator oder Operator Passwort des Web-IO (nicht in Verbindung mit dem *single* Kommando)

MASK

Gibt in hexadezimaler Schreibweise an, welche Outputs gesetzt werden sollen. *MASK* ist optional - bei nicht Angabe wird 0FFF eingesetzt, was bedeutet, dass alle Outputs geändert werden (*MASK* gibt es nur in Verbindung mit dem Kommando *outputaccess*).

STATE

gibt in hexadezimaler Schreibweise an, in welchen Zustand die Outputs gesetzt werden sollen (*STATE* gibt es nur in Verbindung mit dem Kommando *outputaccess*) .

Der Aufruf für die Abfrage der Inputs würde dann so aussehen:

```
http://webiorequest.php?IP=10.40.22.22&PORT=80&COMMAND=input&PW=test&
```

Als Antwort gibt das Script den Zustand der Inputs zurück:

```
10.40.22.22;devicename;input;0
```

Für das Setzen von Output 1 müsste dieser Aufruf genutzt werden:

```
http://webiorequest.php?IP=10.40.22.22&PORT=80&COMMAND=outputaccess&PW=test&MASK=2&STATE=2&
```

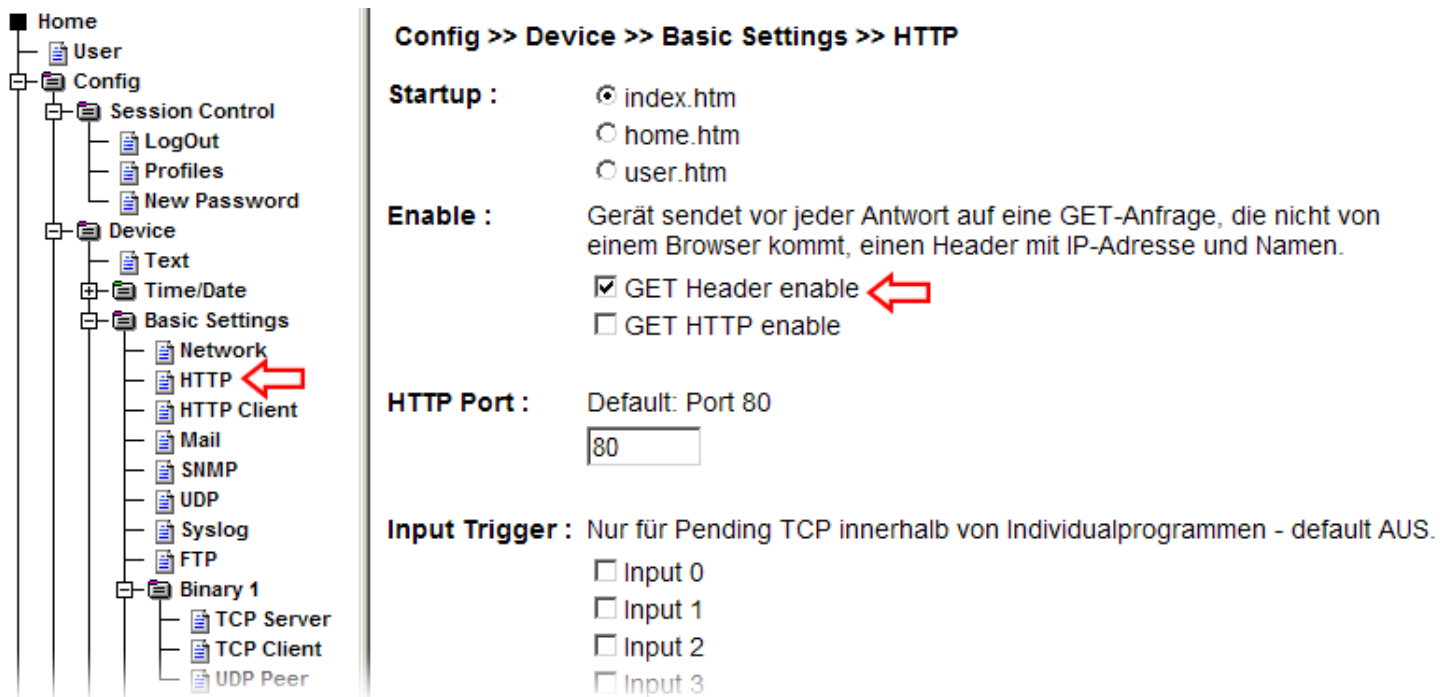
Als Antwort gibt das Script den Zustand der Outputs zurück:

```
10.40.22.22;devicename;output;2
```

Damit das Script einsetzbar ist, muss sichergestellt werden, dass die Antworten des Web-IO mit einem Header bestehend aus **IP-Adresse und Devicenamen** versehen sind.

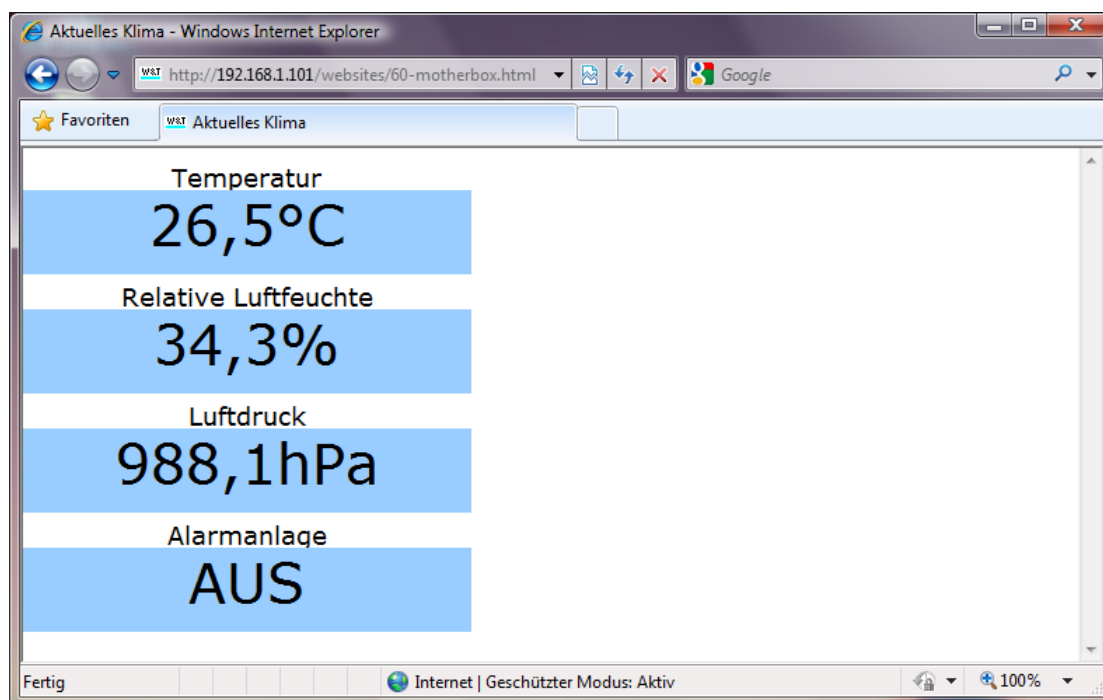
Um das zu erreichen, wählen Sie im Menübaum der beteiligten Web-IO *Config >> Device >> Basic Settings >> HTTP* und aktivieren Sie *GET Header enable*.

Abschließend müssen die Änderungen über *Logout* und *Speichern* übernommen werden.



Dass der Header vom Web-IO mitgesendet wird ist wichtig, damit das JavaScript im Browser identifizieren kann, von welchem Web-IO die Antwort gesendet wurde.

Zurück zu unserem konkreten Beispiel, bei dem eine Webseite die Klimadaten eines Raumes anzeigen soll und darüber hinaus wiedergibt, ob die Alarmanlage scharf geschaltet ist.



Der HTML-Teil hat folgenden Aufbau:

```

<user.htm>
<html>
  <head>
    <title>Aktuelles Klima</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <meta name="viewport" content="width=320" />
    <style type="text/css">
      * { font-family:Verdana, Helvetica; }
      .description { position:absolute;
                     text-align:center;
                     font-size:18px;
                     left:0px;
                     height:20px;
                     width:320px; }
      .value       { position:absolute;
                     background-color:#99CCFF;
                     text-align:center;
                     font-size:40px;
                     left:0px;
                     height:60px;
                     width:320px; }
    </style>

    <script language="JavaScript" type="text/javascript">
      JavaScript-Teil
    </script>
  </head>
  <body>
    <div class="description" style="top:10px; „>
      Temperatur
    </div>
    <div id="temperature" class="value" style="top:30px;">
      <w&t_tags=t1>°C
    </div>
    <div class="description" style="top:95px; „>
      Relative Luftfeuchte
    </div>
    <div id="humidity" class="value" style="top:115px; „>
      <w&t_tags=h1> %
    </div>
    <div class="description" style="top:180px; „>

```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/60-motherbox.html

```
        Luftdruck
    </div>
    <div id="pressure" class="value" style="top:200px;">
        <w&t_tags=pa>hPa
    </div>
    <div class="description" style="top:265px; ">
        Alarmanlage
    </div>
    <div id="alarm" class="value" style="top:285px;">
        AUS
    </div>
</body>
</html>
```

Auch für dieses Beispiel mit mehreren Web-IO, die über das PHP-Script angesprochen werden, kann die *DataRequest*-Funktion aus den anderen Beispielen verwendet werden. Das gilt ebenfalls für die *HexToInt*-Funktion.

Da für die einzelnen Web-IO Boxen eine direkte Adressierung nötig ist, sollten die zugehörigen IP-Adressen gleich im oberen Teil des JavaScript-Teils in Variablen geschrieben werden. So muss bei Änderung oder Erweiterung nicht an mehreren Stellen das JavaScript angepasst werden. Für einen einfachen Zugriff wird dazu ein Array genutzt.

Das Gleiche gilt für die verwendeten *ComandoString*-Zuweisungen. Über die Variable Applicationstep kann dann sehr einfach der benötigte Kommandostring ausgewählt werden.

```
var Address = ['192.168.1.103','192.168.1.104'];
var CommandString = ['webiorequest.php?IP='+Address[0]+'&PORT=80&COMMAND=single&',
                    'webiorequest.php?IP='+Address[1]+'&PORT=80&COMMAND=input&PW=&'];
var Applicationstep = 0;
var Interval = 1000;

function HexToInt(HexStr)
{
    var TempVal;
    var HexVal=0;
    for( var i=0; i<HexStr.length;i++)
    {
        if (HexStr.charCodeAt(i) > 57)
            continue;
        TempVal = HexStr.charCodeAt(i) - 48;
        if (TempVal > 9) TempVal = TempVal - 10;
        HexVal = HexVal * 16 + TempVal;
    }
    return HexVal;
}
```

```

    { TempVal = HexStr.charCodeAt(i) - 55;
    }
    else
    { TempVal = HexStr.charCodeAt(i) - 48;
    }
    HexVal=HexVal+TempVal*Math.pow(16, HexStr.length-i-1);
  }
  return HexVal;
}

```

Für die wechselnde Abfrage der einzelnen Web-IO Zustände und Werte sorgt die Funktion *Commandloop*, die sich nach der in *Interval* festgelegten Zeit zyklisch selbst aufruft.

```

function Commandloop()
{ DataRequest(CommandString[Applicationstep]);
  Applicationstep++;
  if (Applicationstep>1)
  { Applicationstep = 0;
  }
  maintimer = setTimeout(„Commandloop()“, Interval);
}

function DataRequest(SendString)
{ var xmlHttp;
  if( window.ActiveXObject )      // Internet Explorer
  { xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
  }
  else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
  { xmlHttp = new XMLHttpRequest();
  }
  if (xmlHttp)
  { xmlHttp.onreadystatechange = function()
    { if (xmlHttp.readyState == 4)
      { if (xmlHttp.status == 200)
        { updateDisplay(xmlHttp.responseText);
          xmlHttp=null;
        }
      }
    }
  }
  xmlHttp.open(„GET“, SendString, true);
}

```

```
xmlHttp.setRequestHeader("Connection", "close");
xmlHttp.setRequestHeader("If-Modified-Since", "Thu, 1 Jan 1970 00:00:00 GMT");
xmlHttp.send(null);
}
}
```

In der Funktion *updateDisplay* wird zunächst wieder die Antwort der Web-IOs über *split* in Blöcke zerlegt. Im ersten Block steht die IP-Adresse der antwortenden Gerätes. Anhand dieser IP-Adresse entscheidet die Funktion, wie die Antwort weiter verarbeitet werden soll. In unserem Fall, wo die Werte bzw. der Zustand angezeigt werden sollen.

```
function updateDisplay(ReceiveStr)
{
    var ReceiveData = ReceiveStr.split(",");
    if (ReceiveData[0] == Address[0])
    {
        document.getElementById('temperature').innerHTML = ReceiveData[ReceiveData.length-3].
substring(0,ReceiveData[ReceiveData.length-3].length-2) + '°C';
        document.getElementById('humidity').innerHTML = ReceiveData[ReceiveData.length-2]
        document.getElementById('pressure').innerHTML = ReceiveData[ReceiveData.length-1]
    }
    if (ReceiveData[0] == Address[1])
    {
        var HexVal = HexToInt(ReceiveData[ReceiveData.length-1]);
        if (HexVal & 1 == 1)
        {
            document.getElementById('alarm').innerHTML = 'SCHARF';
        }
        else
        {
            document.getElementById('alarm').innerHTML = 'AUS';
        }
    }
}
```

Nach dem Laden der Webseite vom PHP-Server bzw. der Motherbox wird zunächst einmal zeitverzögert die Funktion Commandloop gestartet, um die zyklische Abfrage ins Rollen zu bringen.

```
maintimer = setTimeout("Commandloop()", Interval);
```

Bei Beachtung der richtigen Kommandosstruktur ist die Visualisierung mehrerer Web-IO Geräte über einen PHP-Server nicht komplizierter als mit nur einem Gerät.

Zustände mehrerer Geräte grafisch darstellen

Natürlich lässt sich dieselbe Technik genauso für die grafische Darstellung von Zuständen und Werten einsetzen.

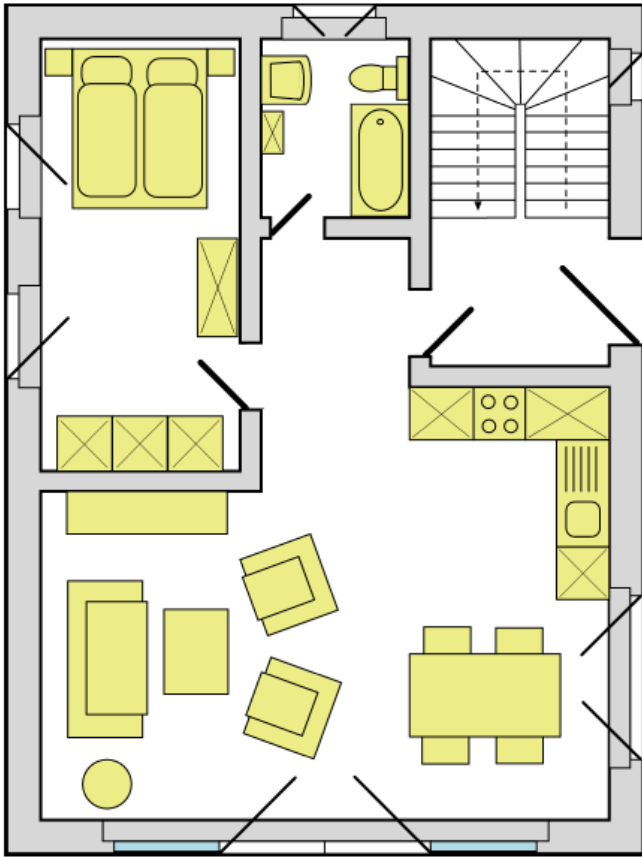
In diesem Beispiel soll der Grundriss eines Ferienhauses im Browser dargestellt werden. Die Inputs eines Web-IO Digital überwachen, ob Fenster und Türen geöffnet oder geschlossen sind. Über die Outputs des Web-IO sollen darüberhinaus die Lichter in den verschiedenen Räumen geschaltet werden können.

Außerdem werden die Temperatur, die Luftfeuchte und der Luftdruck von einem Web-Thermohygro-Barographen überwacht und sollen ebenfalls im Grundriss angezeigt werden.

Es wird also zunächst ein Bild des Grundrisses benötigt, in dem alle Türen und Fenster geschlossen sind.

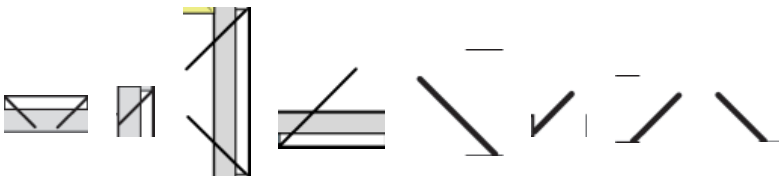


Im nächsten Schritt wird der Grundriss so verändert, dass alle Türen und Fenster geöffnet sind.



Aus diesem Bild müssen mit einem entsprechenden Grafikeditor (z.B. Gimp) die Bereiche, die eine einzelne offene Tür bzw. ein offenes Fenster zeigen, ausgeschnitten und als eigene Datei abgespeichert werden.

Hier ein paar Beispiele:



Nun müssen noch Grafiken für ein- und ausgeschaltetes Licht generiert werden:



Wenn alle benötigten Einzelgrafiken fertig sind, beginnt der Aufbau und die Positionierung der Grafiken im Browser.

Im Kopf der Seite werden Style-Merkmale, die den Grafiken gemeinsam sind, als Style-Vorgabe festgehalten.



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/61-motherbox.html

```
<html>
  <head>
    <title>Ferienhaus immer im Blick</title>
    <style type="text/css">
      * { font-family: Arial; font-size: 12px;}
      img { position:absolute; border:none; visibility:hidden;}
      div { position:absolute; border:none; float:left; }
    </style>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <script language="JavaScript" type="text/javascript">
      JavaScript-Teil
    </script>
  </head>
  <body>
  </body>
</html>
```

Im JavaScript-Teil wird zunächst festgelegt, an welcher XY-Position der Grundriss angezeigt werden soll.

```
var xpos = 50;
var ypos = 10;
```

Die IP-Adressen der beteiligten Web-IOs werden ebenfalls in Variablen festgehalten.

```
var IPio = '192.168.1.104';
var IPklima = '192.168.1.103';
```

Für jede Einzelgrafik der geöffneten Türen und Fenster wird innerhalb eines Variablen-Arrays je ein Array angelegt, in dessen Blöcke die Eigenschaften der Grafik festgelegt sind.

```
var img = new Array();
```

```
img[0] = ['in0', 'eg_fn1.gif', 0, 193];
img[1] = ['in1', 'eg_fw1.gif', 80, 0];
img[2] = ['in2', 'eg_fw2.gif', 193, 0];
img[3] = ['in3', 'eg_fs1.gif', 514, 143];
img[4] = ['in4', 'eg_fs2.gif', 514, 213];
img[5] = ['in5', 'eg_fo2.gif', 394, 384];
img[6] = ['in6', 'eg_fo1.gif', 30, 403];
img[7] = ['in7', 'eg_ta.gif', 157, 369];
img[8] = ['in8', 'eg_tf.gif', 191, 270];
img[9] = ['in9', 'eg_ts.gif', 228, 128];
img[10] = ['in10', 'eg_tb.gif', 125, 177];
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/61-motherbox.html

Die Blöcke stehen für *id*, *Dateiname*, *Y-Position* und *X-Position* .

Nachdem die Hauptgrafik (Grundriss) über *document.write* erstellt wurde, können die Einzelgrafiken über eine *For*-Schleife bequem und automatisch angelegt werden.

```
document.write("<img border='0' src='eg_main.gif' style ='position:absolute; visibility:visible; top:" + ypos + „px; left:" + xpos + „px'>");

for (i=0; i<img.length; i++)
{  document.write('');
}
```



Innerhalb von document.write-Anweisungen darf im realen Quelltext kein Zeilenumbruch sein.

Für die Darstellung der ein- und ausgeschalteten Lichter wird genauso verfahren.

```
var off = new Array();
var on = new Array();

off[0] = ['off0', 'light_off.gif', 149, 318];
off[1] = ['off1', 'light_off.gif', 178, 193];
off[2] = ['off2', 'light_off.gif', 65, 186];
off[3] = ['off3', 'light_off.gif', 156, 57];
off[4] = ['off4', 'light_off.gif', 298, 308];
off[5] = ['off5', 'light_off.gif', 461, 103];
```

```
on[0] = ['on0', 'light_on.gif', 149, 318];
on[1] = ['on1', 'light_on.gif', 178, 193];
on[2] = ['on2', 'light_on.gif', 65, 186];
on[3] = ['on3', 'light_on.gif', 156, 57];
on[4] = ['on4', 'light_on.gif', 298, 308];
on[5] = ['on5', 'light_on.gif', 461, 103];
```

Auch hier kommt wieder eine For-Schleife zum Einsatz.

```
for (i=0; i<off.length; i++)
{
  document.write('<a href="javascript:setOutput('+i+') ">');
  document.write('');
  document.write('</a>');
  document.write('<a href="javascript:setOutput('+i+') ">');
  document.write('');
  document.write('</a>');
}
```

Die Grafiken für die Lichter werden mit einem Hyperlink versehen, damit die Lichter durch Anklicken ein- und ausgeschaltet werden können.

Zuletzt muss noch die Anzeige der Klimawerte vorbereitet werden. Die Klimawerte sollen in 3 Div-Bereichen angezeigt werden, die ebenfalls in einem Array verwaltet werden.

```
var klima = new Array();

klima[0] = ['klima0', ' ', 325, 250];
klima[1] = ['klima1', ' ', 345, 250];
klima[2] = ['klima2', ' ', 365, 250];
```

Zur Positionierung kommt wieder eine For-Schleife zum Einsatz.

```
for (i=0; i<klima.length; i++)
```

```
{ document.write('<div ID="'+klima[i][0]+'"  
style="top:' + (ypos + klima[i][2]) + 'px; left:' + (xpos + klima[i][3]) + 'px">-</div>');  
}
```

Damit ist der grundsätzliche Bildschirmaufbau erst einmal abgeschlossen.

Die zyklische Abfrage der Web-IO Zustände bzw. Klimawerte erfolgt wieder über eine *Commandloop* Funktion.

Die benötigten Kommandos werden in einem Variablen-Array verwaltet. Die Abfolge, in der die Kommandos in der Funktion *Commandloop* abgearbeitet werden, steuert sich über die Variable *Applikationstep*. Der Zyklus zwischen zwei gesendeten Kommandos wird über die Variable *Interval* vorgegeben.

```
var Interval = 500;  
var CommandString = ['webiorequest.php?IP='+IPio+'&PORT=80&COMMAND=input&PW=&',  
                    'webiorequest.php?IP='+IPio+'&PORT=80&COMMAND=output&PW=&',  
                    'webiorequest.php?IP='+IPklima+'&PORT=80&COMMAND=single&'];  
  
maintimer = setTimeout(„Commandloop()“, Interval);  
  
function Commandloop()  
{ DataRequest(CommandString[Applicationstep]);  
  Applicationstep++;  
  if (Applicationstep>2)  
  { Applicationstep = 0;  
  }  
  maintimer = setTimeout(„Commandloop()“, Interval);  
}
```

Die Funktionen *DataRequest* und *HexToInt* werden wieder unverändert übernommen.

```
function HexToInt(HexStr)  
{ var TempVal;  
  var HexVal=0;  
  for( var i=0; i<HexStr.length;i++)  
  { if (HexStr.charCodeAt(i) > 57)  
    { TempVal = HexStr.charCodeAt(i) - 55;
```

```

    }
    else
    { TempVal = HexStr.charCodeAt(i) - 48;
    }
    HexVal=HexVal+TempVal*Math.pow(16, HexStr.length-i-1);
  }
  return HexVal;
}

function DataRequest(SendString)
{
  var xmlHttp;
  if( window.ActiveXObject )      // Internet Explorer
  {
    xmlHttp = new ActiveXObject( „Microsoft.XMLHTTP“ );
  }
  else if(window.XMLHttpRequest ) // Mozilla, Opera und Safari
  {
    xmlHttp = new XMLHttpRequest();
  }
  if (xmlHttp)
  {
    xmlHttp.onreadystatechange = function()
    {
      if (xmlHttp.readyState == 4)
      {
        if (xmlHttp.status == 200)
        {
          updateDisplay(xmlHttp.responseText);
          xmlHttp=null;
        }
      }
    }
  }
  xmlHttp.open(„GET“, SendString, true);
  xmlHttp.setRequestHeader(„Connection“, „close“);
  xmlHttp.setRequestHeader(„If-Modified-Since“, „Thu, 1 Jan 1970 00:00:00 GMT“);
  xmlHttp.send(null);
}
}

```

Die Funktion *updateDisplay* muss komplett überarbeitet bzw. neu erstellt werden.

```

function updateDisplay(ReceiveStr)
{
  var ReceiveData = ReceiveStr.split(„;“);
  var state;
  if (ReceiveData[0] == IPio)
  {
    if (ReceiveData[ReceiveData.length-2].substring(0,1)=='i')

```

```

{
  var HexVal = HexToInt(ReceiveData[ReceiveData.length-1]);
  for (var i = 0; i < img.length; i++)
  {
    if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
    {
      state = true;
    }
    else
    {
      state = false;
    }
    document.getElementById(img[i][0]).style.visibility
      = (!state) ? 'hidden' : 'visible';
  }
}

if (ReceiveData[ReceiveData.length-2].substring(0,1)=='o')
{
  var HexVal = HexToInt(ReceiveData[ReceiveData.length-1]);
  for (var i = 0; i < on.length; i++)
  {
    if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
    {
      state = true;
    }
    else
    {
      state = false;
    }
    document.getElementById(on[i][0]).style.visibility
      = (!state) ? 'hidden' : 'visible';
  }
}
}

if (ReceiveData[0] == IPklima)
{
  document.getElementById(klima[0][0]).innerHTML
    = ReceiveData[ReceiveData.length-3].
      substring(0,ReceiveData[ReceiveData.length-3].length-2) + '°C';
  document.getElementById(klima[1][0]).innerHTML = ReceiveData[ReceiveData.length-2]
  document.getElementById(klima[2][0]).innerHTML = ReceiveData[ReceiveData.length-1]
}
}

```

Zunächst wird die empfangene Antwort mittels *split* in Blöcken in die Array-Variable *ReveiveData* geschrieben.

Anhand des ersten Blocks, in dem die IP-Adresse des Web-IO steht, wird entschieden, wie die Daten weiterverarbeitet werden.

Wird die IP Adresse des Web-IO Digital (*IPio*) erkannt, unterscheidet das Script durch Prüfen des vorletzten Blocks, ob die Antwort einen Input oder einen Output betrifft.

Im letzten Block der Antwort befindet sich die binäre Entsprechung der Inputs bzw. Outputs in hexadezimaler Schreibweise. Über die *HexToInt* Funktion wird daraus ein Zahlenwert berechnet, der in die Variable *HexVal* geschrieben wird.

In einer *For*-Schleife werden dann die einzelnen Bits ausmaskiert und auf ihren Status geprüft.

Abhängig davon, ob ON oder OFF erkannt wurde, setzt das Script *visibility* auf *visible* oder *hidden*.

Erkennt das Script die IP-Adresse des web-Thermohygro-Barographen, werden die übergebenen Werte in die entsprechenden Div-Elemente eingetragen.

Damit ist die dynamische Anzeige der grafischen Elemente abgeschlossen.

Nun fehlt nur noch die Bedienung der Lichter.

Die für die Lichter angezeigten Grafiken sind ja mit einem Hyperlink versehen, der die JavaScript-Funktion *setOutput* aufruft.

Der Funktion *setOutput* wird dabei die Nummer des Outputs übergeben, dessen Zustand geändert werden soll.

```
function setOutput(nr)
{
  var Mask = Math.pow(2,nr).toString(16);
  var SendString = `webiorequest.php?IP='+IPio+'&PORT=80&
    COMMAND=outputaccess&PW=&MASK='+Mask+'&STATE=TOGGLE&`;
  DataRequest(SendString);
}
```

Aus der Nr. des Outputs muss zunächst eine hexadezimale Zahl generiert werden, die dem Bit entspricht, welches den Output repräsentiert.

Das erfolgt über folgendes Kommando:

```
var Mask = Math.pow(2,nr).toString(16);
```

Dazu wird die Zweierpotenz mit der Nr. als Exponent gebildet.

```
var Mask = Math.pow(2,nr).toString(16);
```

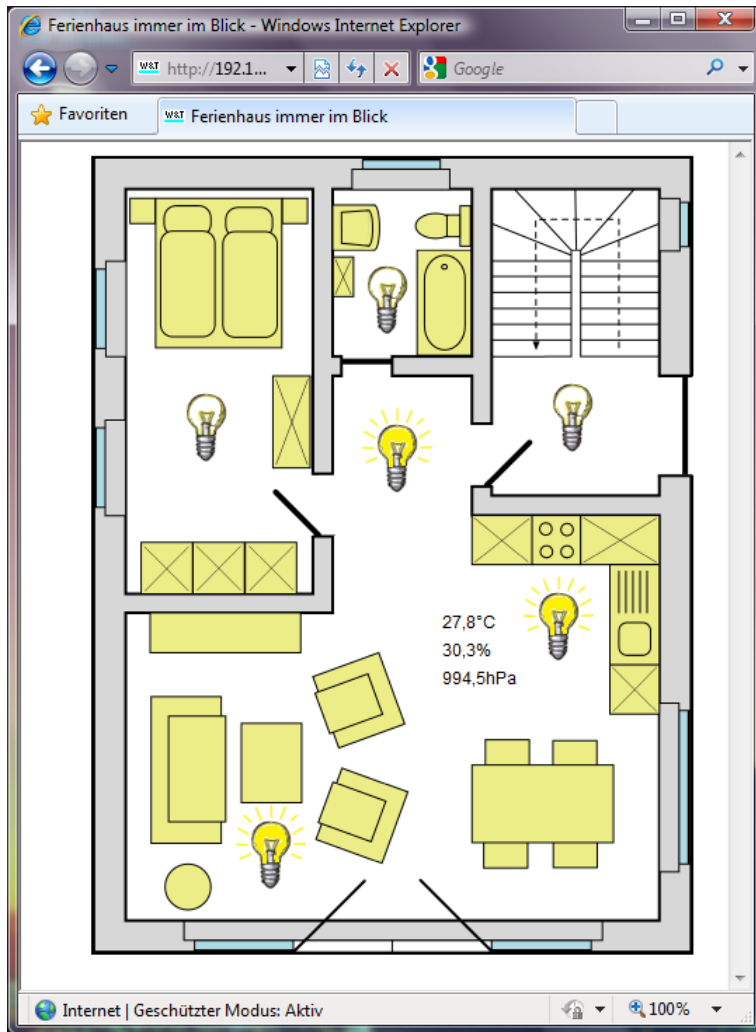
Der daraus resultierende Wert wird dann über das Kommando *toString* mit dem Parameter *16* in einen String gewandelt, der die hexadezimale Zahl wiedergibt.

Dieser String wird in die Variable *Mask* gespeichert.

Mask wird dann in den Kommandostring zum Setzen des Outputs eingesetzt.

```
var SendString = 'webiorequest.php?IP='+IPio+'&PORT=80&  
COMMAND=outputaccess&PW=&MASK='+Mask+'&STATE=TOGGLE&';
```

Als Parameter für *State* wird *TOGGLE* eingesetzt, was zu einem Zustandswechsel des Outputs führt, der über *MASK* vorgegeben ist.



Damit haben wir mit einer überschaubaren Menge an Quelltext eine grafische Visualisierung geschaffen, die durch ihren Aufbau sehr flexibel erweiterbar ist.

Serielle Daten mit PHP in den Browser bringen

Nicht alle Geräte, die Daten und Zustände zur Verfügung stellen, haben ein Web-Interface, wie es bei den Web-IO Boxen genutzt werden kann.

Vor allem ältere Geräte wie z.B. Wetterstationen, Messgeräte oder Barcodescanner geben ihre Daten oft über eine serielle RS232 Schnittstelle aus.

Mit dem W&T Com-Server lassen sich solche Daten recht einfach ins Netzwerk bringen - allerdings ohne echten Browserzugang - diese Lücke kann ein kleines PHP-Script

schließen, das wie in den letzten beiden Beispielen über AJAX bzw. HTTP-Requests aufgerufen werden kann.

Das *csrequest.php* Script nimmt folgende Parameter entgegen:

IP

IP-Adresse des Com-Servers

PORT

Port auf dem der Com-Server eine Verbindung annimmt. Standard ist 8000. Dieser Parameter ist optional. Wird er nicht angegeben, nimmt das Script den Standard-Port.

DATA

Mit *DATA* können Daten übergeben werden, die über den Com-Server zum seriellen Gerät gesendet werden sollen. Hier kann z.B. ein Kommando für Daten-Polling angegeben werden. *DATA* ist optional. Ist *DATA* nicht gesetzt, werden keine Daten zum seriellen Endgerät gesendet.

Beispiel: nur lesender Aufruf des Scriptes für den Com-Server mit der IP 10.40.22.130:

```
csrequest.php?IP=10.40.22.130&PORT=8000
```

Beispiel: An das Gerät, welches an Com-Server 10.40.22.130 angeschlossen ist, soll der String „GetValue“ gesendet werden:

```
csrequest.php?IP=10.40.22.130&PORT=8000&DATA=GetValue
```

Das Script ist recht einfach aufgebaut.

```
<?php
```

```
if( !isset( $_GET["IP"] ) ) die( 'ip' );
if( !isset( $_GET["PORT"] ) ) $_GET["PORT"] = 8000;
if( !isset( $_GET["DATA"] ) ) $_GET["DATA"] = '';
```

```
$hComServer = @fsockopen( $_GET["IP"], intval( $_GET["PORT"] ), $iErrorNr, $sErrDescription,
3 );
if( $hComServer )
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/csrequest.php

```

{ socket_set_blocking( $hComServer, false );
  if ( $_GET["DATA"] ) fputs( $hComServer, $_GET["DATA"] );
  $sCSdata = '';
  $iStart = microtime(true);
  do
  { $cRead = fgetc( $hComServer );
    if( $cRead !== false )
    { $sCSdata .= $cRead;
      $iStart = microtime(true);
    }
    else
      usleep( 1000 );
  }
  while(( microtime(true) - $iStart ) < 0.3 );      // Timeout 0,3s = 300ms
  fclose( $hComServer );
  echo $sCSdata;
}
else
  die( 'connect' );

?>

```

Zunächst überprüft das Script, welche Parameter übergeben wurden. Fehlt z.B. die IP-Adresse, wird das Script sofort mittels *die* beendet und der Fehler-Code *ip* zurückgegeben.

Wurde eine IP-Adresse angegeben, baut das Script mit *fsockopen* eine Verbindung zum Com-Server auf.

Mit *if(\$hComServer)* wird geprüft, ob der Verbindungsaufbau erfolgreich war.

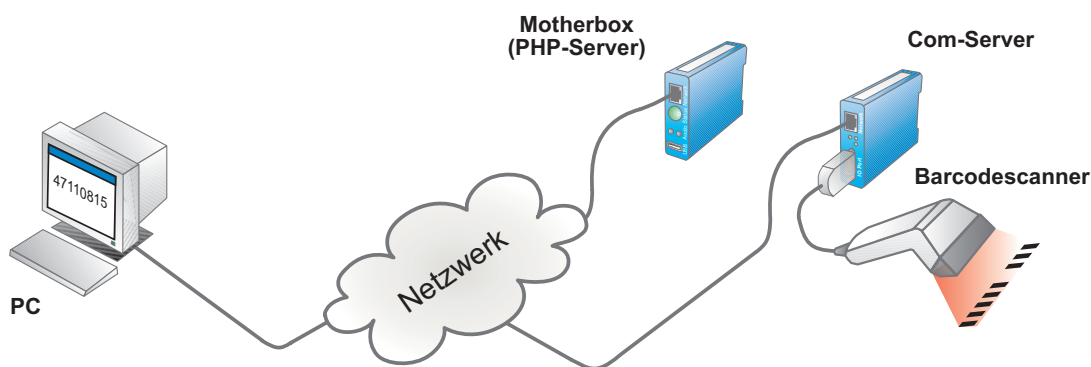
Wenn ja, wird überprüft, ob der Parameter *DATA* Sendedaten beinhaltet, die dann mittels *fputs* an den Com-Server gesendet werden.

Daten, die von Com-Server Seite empfangen werden, schreibt das Script zunächst in die Variable *\$sCSdata*. Vor dem Empfang des ersten Zeichens wird ein Zeitstempel in die Variable *\$iStart* geschrieben. Mit jedem empfangenen Zeichen wird dieser Zeitstempel neu gesetzt.

Der Zeichenempfang wird in einer do/while-Schleife abgewickelt, wobei die Endebedingung ein Timeout von 0,3s ist. Wird für diese Zeit kein weiteres Zeichen empfangen, werden die bis dahin entgegengenommenen Daten als Antwort auf den Script-Aufruf an den Browser gesendet. Abschließend wird die Verbindung zum Com-Server beendet und das Script geschlossen.

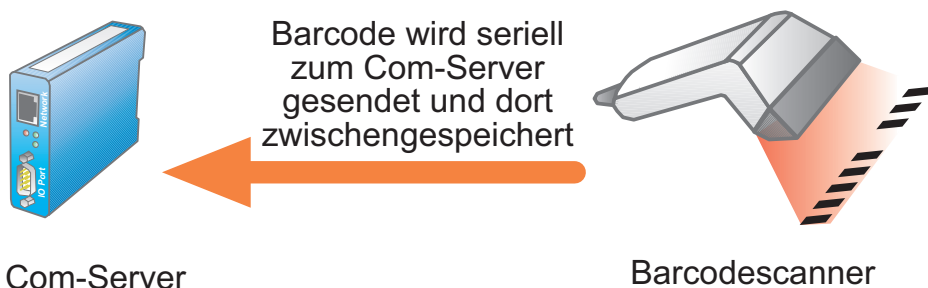
Gescannte Barcodes im Browser anzeigen

Im ersten Beispiel sollen die Daten eines Barcodescanners im Browser ausgegeben werden.



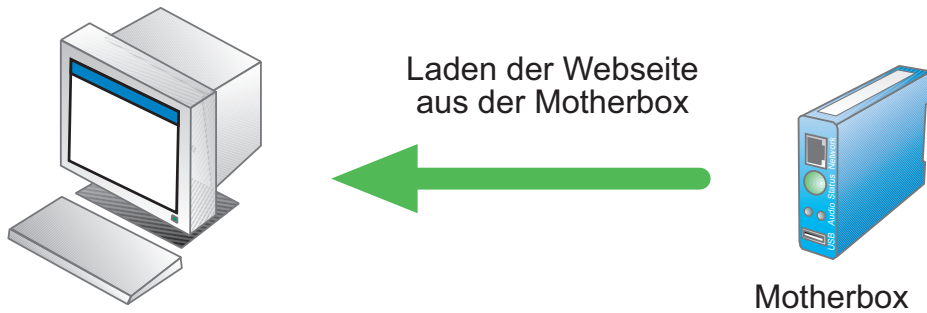
Hier die einzelnen Schritte im Detail:

Der Barcodescanner sendet den Barcode über die serielle RS232-Schnittstelle an den Com-Server. Im Com-Server wird der Barcode zwischengespeichert, bis eine TCP-Verbindung zustande kommt.

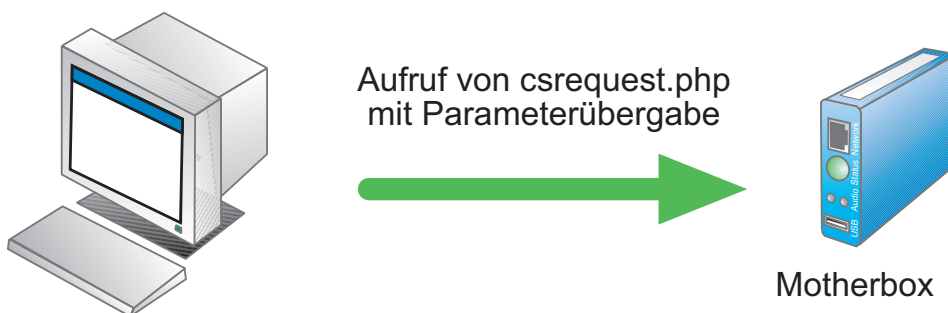


Der Com-Server muss dazu im Bereich *Setup Port 0 >> Setup TCP/IP >> System Options >> Flush Buffer* von 0 auf 1 gesetzt werden.

Im nächsten Schritt wird die Webseite vom PHP-Server bzw. der Motherbox in den Browser geladen.



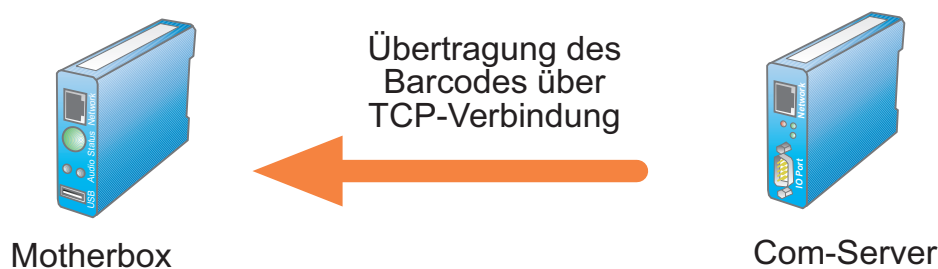
Per HTTP-Request wird das PHP-Script *csrequest.php* aufgerufen und die gewünschten Parameter übergeben.



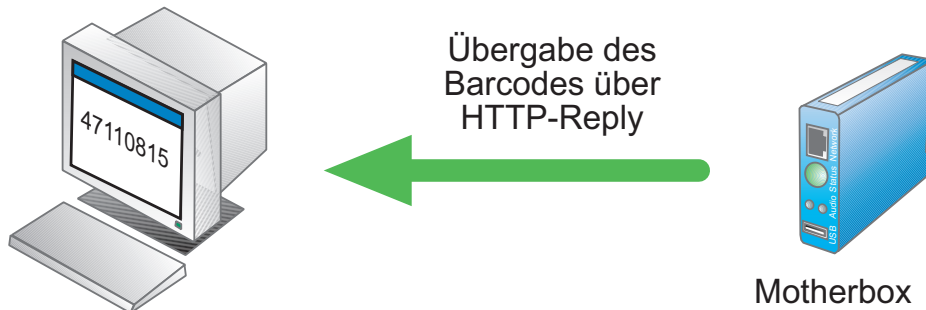
Der PHP-Server bzw. die Motherbox startet das PHP-Script, welches die mitgesendeten Parameter auswertet und eine TCP-Verbindung zum Com-Server mit der übergebenen IP-Adresse aufbaut.



Der Com-Server sendet nach Zustandekommen der Verbindung alle zwischengespeicherten Daten an den PHP-Server bzw. die Motherbox.



Der PHP-Server bzw. die Motherbox empfängt die Daten, wobei diese vom *csrequest.php* Script 1:1 an den Browser weitergeleitet werden.



Der Browser wertet die empfangenen Daten aus und aktualisiert entsprechend die Anzeige.

Die entsprechende Webseite hat als einziges Element ein Textausgabefeld.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="de" xml:lang="de">
```

```
<head>
```

```
<title>Barcode-Scanner</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
<script type="text/javascript">
```

```
var sIP = '10.40.22.130';
```

```
var iPort = 8000;
```

```
var sCommandstring = csrequest.php?IP=' + sIP + '&PORT=' + iPort;
```

```
function DataRequest( SendString )
```

```
{ var xmlhttp;
```

```

if( window.ActiveXObject )           // Internet Explorer
{ xmlHttp = new ActiveXObject( 'Microsoft.XMLHTTP' );
}
else if( window.XMLHttpRequest )     // Mozilla, Opera und Safari
{ xmlHttp = new XMLHttpRequest();
}
if( xmlHttp )
{ xmlHttp.onreadystatechange = function()
  { if( xmlHttp.readyState == 4 )
    { if( xmlHttp.status == 200 )
      { updateDisplay( xmlHttp.responseText );
        xmlHttp = null;
      }
    }
  }
}
xmlHttp.open( 'GET', SendString, true );
xmlHttp.setRequestHeader( 'Connection', 'close' );
xmlHttp.setRequestHeader( 'If-Modified-Since', 'Thu, 1 Jan 1970 00:00:00 GMT' );
xmlHttp.send( null );
}
}

function updateDisplay( sBarcode )
{ iTimeout = 250;
  if( sBarcode != '' )
  { sText = '';
    switch( sBarcode )
    { case 'ip':
      sText    = 'Keine IP-Adresse angegeben\n';
      iTimeout = 0;
      break;
      case 'connect' :
        sText = 'Verbindungsfehler\n';
        iTimeout = 5000;
        break;
      default :
        sText = sBarcode;
        break;
    }
    document.getElementById( 'display' ).value += sText;
  }
}

```



*Dieses Beispiel finden
Sie in der Software-
Sammlung zum Buch.
Download unter:
[www.wut.de/
examples/
70-motherbox.html](http://www.wut.de/examples/70-motherbox.html)*

```
        if( iTimeout > 0 )
            window.setTimeout( "DataRequest(sCommandstring )", iTimeout );
    }

    maintimer = setTimeout( "DataRequest( sCommandstring )", 250 );

</script>
</head>
<body>
    <div>
        <textarea id="display" cols="50" rows="10" readonly="readonly"></textarea>
    </div>
</body>
</html>
```

Der eigentliche HTML-Teil der Webseite besteht nur aus einem Textanzeigefeld, in dem später die gescannten Barcodes ausgegeben werden.

Im ersten Teil des JavaScriptes werden zunächst IP-Adresse und Port in entsprechende Variablen geschrieben. Im Anschluss wird der Kommandostring zusammengesetzt, der zum Abrufen der Barcodes zum PHP-Server bzw. zur Motherbox gesendet werden soll.

Die DataRequest Funktion wurde 1:1 aus den vorangegangenen Beispielen übernommen.

Die updateDisplay Funktion unterscheidet über eine *case/switch* Abfrage, ob die Daten, die vom PHP-Server bzw. der Motherbox zurückgegeben wurden, tatsächlich gescannte Barcodes sind. Dazu wird geprüft, ob die empfangenen Daten einen der im PHP-Script festgelegten Fehlercodes enthält (*ip*=keine IP-Adresse oder *connect*=Verbindungsfehler).

Wird ein Fehler erkannt, wird anstelle des Barcodes eine entsprechende Fehlermeldung ausgegeben. Außerdem wird im Fehlerfall das Abfrageintervall von sonst 250ms auf 5000ms hochgesetzt.

Für die Ausgabe in das Textanzeigefeld wird wie gewohnt mit *document.getElementById* gearbeitet.

Nach Empfang der Daten wird - um das Abfrageintervall zeitversetzt - erneut der Kommandostring zum PHP-Server bzw. zur Motherbox gesendet.

Das erste Senden des Kommandostrings erfolgt am Ende des JavaScriptes, ebenfalls zeitversetzt, um der gesamten Webseite die nötige Zeit zum vollständigen Laden zu geben.

Datenaustausch mit seriellen Geräten über den Browser

Im letzten Beispiel wurde nur das Empfangen von seriellen Daten unterstützt.

Um eine echte bidirektionale Datenkommunikation mit einem seriellen Endgerät zu realisieren, müssen HTML-Teil und JavaScript des Beispiels nur geringfügig erweitert werden.



The image shows a web browser window containing a form. At the top, there is a single-line text input field followed by a button labeled 'Send'. Below these, there is a large, multi-line text area (textarea) with a vertical scrollbar on the right side.

Oberhalb des Textanzeigefeldes wird ein Texteingabefeld und ein Button eingefügt.

Hier der entsprechende HTML-Teil dazu:

```
<body>
  <form>
    <input type="text" name="tb_sendtext">
    <input type="button" value="Send" onclick="sendText();"><br>
    <textarea id="display" cols="50" rows="10" readonly="readonly"></textarea>
  </form>
</body>
```

Im JavaScript-Teil muss die Funktion *sendText* hinzugefügt werden.

```
function sendText()
{
  var sSendstring = document.forms[0].tb_sendtext.value;
  if (sSendstring != '')
  {
    DataRequest(sCommandstring + '&DATA=' + sSendstring);
  }
}
```



Dieses Beispiel finden Sie in der Software-Sammlung zum Buch. Download unter: www.wut.de/examples/71-motherbox.html

Durch Anklicken des *Send*-Buttons kann der User diese Funktion aufrufen. Zunächst wird der Text aus dem Eingabefeld in die Variable *sSendstring* geschrieben. Anschließend prüft das Script mittels *if (sSendstring != '')*, ob überhaupt Text eingegeben wurde. Ist das der Fall, wird dieser Text als *DATA*-Parameter an den Kommandostring angehängt, der dann mittels *DataRequest*-Funktion zur Motherbox bzw. zum PHP-Server gesendet wird.

Darüber hinaus muss die *updateDisplay* Funktion angepasst werden.

```
function updateDisplay( sReceivestring )
{
  if( sReceivestring != '' )
  {
    sText = '';
    switch( sReceivestring )
    {
      case 'ip':
        sText = 'Keine IP-Adresse angegeben\n';
        break;
      case 'connect' :
        sText = 'Verbindungsfehler\n';
        break;
      default :
        sText = sReceivestring;
        break;
    }
    document.getElementById( 'display' ).value += sText;
  }
}
```

Da die Datenabfrage in diesem Beispiel vom User ausgelöst wird, entfällt der zyklische Aufruf des *csrequest.php* Scriptes. Damit sind auch die für den Fehlerfall angepassten Timeout-Zeiten nicht mehr nötig.

Auch der erste Aufruf des *csrequest*-Scriptes am Ende des JavaScriptes kann gelöscht werden.

Damit ist die Webseite bereits so angepasst, dass serielle Endgeräte gezielt aus dem Browser abgefragt werden können.

Anhang zur Web-IO Programmierung

Character Sets

Wie die einzelnen Zeichen im Browser dargestellt werden, wird im Kopf einer Webseite über ein Meta-Tag vorgegeben.

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

Welche Möglichkeiten es gibt, zeigen die folgenden Tabellen:

ISO Character Sets

Character Set	Description	Covers
ISO-8859-1	Latin alphabet part 1	North America, Western Europe, Latin America, the Caribbean, Canada, Africa
ISO-8859-2	Latin alphabet part 2	Eastern Europe
ISO-8859-3	Latin alphabet part 3	SE Europe, Esperanto, miscellaneous others
ISO-8859-4	Latin alphabet part 4	Scandinavia/Baltics (and others not in ISO-8859-1)
ISO-8859-5	Latin/Cyrillic part 5	The languages that are using a Cyrillic alphabet such as Bulgarian, Belarusian, Russian and Macedonian
ISO-8859-6	Latin/Arabic part 6	The languages that are using the Arabic alphabet
ISO-8859-7	Latin/Greek part 7	The modern Greek language as well as mathematical symbols derived from the Greek
ISO-8859-8	Latin/Hebrew part 8	The languages that are using the Hebrew alphabet
ISO-8859-9	Latin 5 part 9	The Turkish language. Same as ISO-8859-1 except Turkish characters replace Icelandic ones
ISO-8859-10	Latin 6 Lappish, Nordic, Eskimo	The Nordic languages
ISO-8859-15	Latin 9 (aka Latin 0)	Similar to ISO 8859-1 but replaces some less common symbols with the euro sign and some other missing characters
ISO-2022-JP	Latin/Japanese part 1	The Japanese language
ISO-2022-JP-2	Latin/Japanese part 2	The Japanese language
ISO-2022-KR	Latin/Korean part 1	The Korean language

UTF Character Sets

Character-set	Description
UTF-8	A character in UTF8 can be from 1 to 4 bytes long. UTF-8 can represent any character in the Unicode standard. UTF-8 is backwards compatible with ASCII. UTF-8 is the preferred encoding for e-mail and web pages
UTF-16	16-bit Unicode Transformation Format is a variable-length character encoding for Unicode, capable of encoding the entire Unicode repertoire. UTF-16 is used in major operating systems and environments, like Microsoft Windows 2000/XP/2003/Vista/CE and the Java and .NET byte code environments

HTML-Tags - die Wichtigsten Tags im Überblick

Übersicht der wichtigsten HTML-Tags			
Tag	Attribut	Wert	Beschreibung
<!-- //-->	-	-	Kommentar im Quellcode
<A> 	href	URL	Verweis Gibt Ziel Adresse an
		mailto:e-mail	Öffnet Mailprogramm des Servers mit e-mail Adresse
		news:URL	Öffnet Newgroup
	cc	e-mail	Gibt Adresse für Kopieehalter an
	bcc	e-mail	Gibt Adresse für 2.Kopieerhalter an
		text	Gibt Text in Betreffzeile vor
	name	text	Erzeugt Verweisziel (Sprungmarke)
	target	name	Legt fest, in welchem Fenster das neue Dokument geöffnet werden soll
		_blank	Öffnet Dokument im neuen Fenster
		_parent	Öffnet Dokument im festgelegten Hauptframe
		_self	Öffnet Dokument im selben Fenster
		_top	Vernichtet alle Frames und öffnet Dokument im ganzen Fenster
	accesskey	Tastaturzeichen	Legt Tastaturtaste fest, mit der die Verknüpfung ohne Klick zustande kommt
	shape	circle, poly, rect, default	Definiert Verweisanklickfläche
<APPLET> </APPLET>	codebase	URL	Lädt Java-Applet in Dokument
	alt	text	Gibt URL des Applets an
	align	left, right, top, middle, bottom	Alternativtext falls nicht zu laden Ausrichtung des Applets
	height	n	Höhe des Applets
	width	n	Breite des Applets
	hspace	n	Freier Raum links & rechts des Applets
	vspace	n	Freier Raum oben & unten des Applets
	name	text	Gibt den Namen des Applets an
	param	text	Lässt Parameter zu
	archive	scripte	Öffnet genannte Scripte vor dem Applet
<AREA> </AREA>	shape	rect, circle, poly, default	Definiert verweissensitive Fläche in Image-Map Bestimmt Aussehen der Verweisfläche
	coords	x1, x2, y1, y2 oder: x, y, radius	Definiert Verweisfläche
	href	URL	Gibt Zieladresse an
	alt	text	Gibt Alternativetext an
	target	siehe auch <A>	Bestimmt wo Dokument geöffnet werden soll
	accesskey	Tastaturzeichen	Legt Tastaturtaste fest, mit der die Verknüpfung ohne Klick zustande kommt
 	-	-	Fettschrift
<BASEFONT> </BASEFONT>			Bestimmt die Standardschrift des Dokumentes
	size		Legt Größe fest
	face	Schriftart	Legt Schriftart fest
	color	#xxxxxx, color	Legt Farbe fest

Tag	Attribut	Wert	Beschreibung
<BGSOUND> </BGSOUND>	src	URL	Öffnet Musikdatei
	loop	n , infinite	Anzahl der Wiederholungen
<BIG> </BIG>	-	-	relative größere Schrift
<BLINK> </BLINK>	-	-	Blinkschrift
<BLOCKQUOTE> </BLOCKQUOTE>	-	-	Markiert langes Zitat
<BODY> </BODY>	text	#xxxxxx, color	Leitet Hauptteil ein
	background	URL	Definiert Textfarbe im Dokument
	bgcolor	#xxxxxx, color	Fügt Hintergrundgrafik ein
	link	#xxxxxx, color	Gibt Hintergrundfarbe an
	alink	#xxxxxx, color	Farbe eines Links
	vlink	#xxxxxx, color	Farbe eines gedrückten Links
	leftmargin	n	Farbe eines besuchten Links
	topmargin	n	Gibt Abstand zu linkem Rand an
	bgproperties	fixed	Gibt Abstand zu oberem Rand an
 	-	-	Wasserzeichen , Hintergrund fixiert
<BUTTON> </BUTTON>	-	-	Zeilenumbruch
	name	text	Erzeugt Eingabefeld
	value	text	Weist Button Namen zu
	type	submit	Weist Button Wert/Zieladresse zu
		button	Erzeugt Schaltfläche zum Absenden von Formularen
		reset	Erzeugt Schaltfläche, die mit Script funktioniert
<CAPTION> </CAPTION>	align	top, bottom	Erzeugt Schaltfläche um alle Eingaben des Users zu Löschen
			Legt Tastaturzeichen fest, mit der die Verknüpfung ohne Klick zustande kommt
<CENTER> </CENTER>	-	-	Legt Tastaturtaste fest, mit der die Verknüpfung ohne Klick zustande kommt
<CODE> </CODE>	-	-	Definiert Tabellenüber- oder unterschift - je nach Wert. (Vor <TR>-Tag einsetzen)
<COL> </COL>	-	-	Mittige Textzentrierung
<COL> </COL>	width	n o. %	Markiert Quellcode
	span	n	Definiert Format von Tabellenspalte(n)
	align	left, center, right, justify, char	Breite der Spalten
	charoff	n	Gibt die Anzahl der Spalten an
	valing	top, middle, bottom, baseline	Richtet den Text waagerecht aus
			Abstand zwischen den Zeichen
<DIR> </DIR>	compact	-	Richtet Text horizontal aus (baseline = wie vorherige Zellen)
<DIV> </DIV>	align	left, center, right	Markiert Verzeichnislisten (kompakt)
<FIELDSET> </FIELDSET>	-	-	Legt Textausrichtung fest
 	face	Schriftart	Fasst Formulare zu Gruppe zusammen
<FORM> </FORM>	size	1-7 o. (-2) - (+4)	Definiert Schriftart
	color	#xxxxxx, color	Definiert Schriftgröße (absolut & relativ)
	action	URL, e-mail	Definiert Schriftfarbe
	method	post, get	Umschließt ein Formular
	target	siehe <A>	Art der Übermittlung
			Gibt an in welchem Fenster geöffnet werden soll

Tag	Attribut	Wert	Beschreibung
<FRAME> </FRAME>	name	name	Definiert Frame Weist Frame Namen zu
	frameborder	1	Dreidimensionaler Rahmen (Ja, Nein)
	marginheight	n	Textabstand zum oberen & unteren Rand
	marginwidth	n	Textabstand zum linken & rechten Rand
	noresize	-	User kann Größe des Fensters nicht ändern
	scrolling	yes, no	Gibt an ob Leiste erlaubt ist oder nicht
	src	URL	Legt URL zu Dokument fest, das in Fenster soll
<FRAMESET> </FRAMESET>			Fasst Frames zu Gruppe zusammen
	border	n	Gibt Rahmendicke an
	bordercolor	#xxxxxx, color	Gibt Rahmenfarbe an
	frameborder	1	Dreidimensionaler Rahmen (Ja oder Nein)
	framespacing	n	Gibt Abstand zwischen Frames in Pixeln an
	cols	n o. %	Bestimmt die Breite die die Frames in die Horizontale einnehmen mit Kommata getrennt
<HEAD> </HEAD> <	rows	n o. %	Bestimmt die Höhe die Frames in die Senkrechte einnehmen mit Kommata getrennt
<H1-6> </H1-6>	align	left, center, right	Überschriftsebene 1-6 (Ausrichtung)
<HR> </HR>			Erzeugt Trennlinie
	size	n	Legt Höhe in Pixeln fest
	width	n o. %	Legt Breite in Pixeln o. Prozent fest
	align	left, center, right	Ausrichtung
	color	#xxxxxx, color	Legt Farbe fest
	noshade	noshade	Verhindert 3D-Effekt
<HTML> </HTML>	-	-	Umschließt HTML-Dokument
<I> </I>	-	-	Italic oder Kursivschrift
<IFRAME> </IFRAME>			Definiert eingebetteten Frame in Dokument
	name	name	Weist Frame Namen zu
	frameborder	1	Dreidimensionaler Rahmen (Ja, Nein)
	marginheight	n	Textabstand zum oberen & unteren Rand
	marginwidth	n	Textabstand zum linken & rechten Rand
	align	left, center, right, top, middle, bottom	Legt Ausrichtung des Frames fest
	scrolling	yes, no	Gibt an ob Leiste erlaubt ist oder nicht
	src	URL	Legt URL zu Dokument fest, das in Fenster soll
	width	n o. %	Legt Breite in Pixeln o. Prozent fest
	height	n o. %	Legt Höhe in Pixeln o. Prozent fest

Tag	Attribut	Wert	Beschreibung
 	src	URL	Fügt Grafik ein (Adresse Grafik)
	alt	text	alternativer Text, falls Grafikanzeige nicht möglich
	align	left, right, top, middle, bottom	Angabe, wo sich die Grafik befinden soll
	height	n	Gibt die Höhe der Grafik in Pixeln an
	width	n	Gibt die Breite der Grafik in Pixeln an
	border	n	Erstellt Rahmen mit Breite
	hspace	n	Gibt den horizontalen freien Raum um die Grafik in Pixeln an
	vspace	n	Gibt den vertikalen freien Raum um die Grafik in Pixeln an
	controls	-	Erstellt Schalterleiste bei Videos
	dynsrc	URL	Verweist anstelle von src auf einen Videoclip
<INPUT> </INPUT>	type	text	Texteingabefeld
		password	Feld zur Passworteingabe
		radio	Radioknopf
		checkbox	Kontrollkästchen
		reset	Reset-Button
		submit	Submit-Button (Abschicken)
		button	Schaltfläche
	disabled	-	Deaktiviert Fläche
<INS> </INS>	alle Attribute wie 	-	Markiert in neueingefügten Text
<KBD> </KBD>	-	-	Markiert über die Tastatur einzugebenden Text
<LEGEND> </LEGEND>	align	left, center, right, justify	Weist Fieldset Namen zu (Ausrichtung der Daten)
	accesskey	Tastaturzeichen	Legt Tastaturtaste fest, mit der die Verknüpfung ohne Klick zustande kommt
 	-	-	Listeneintrag
<LINK> </LINK>	alle Attribute wie <A>	-	Lädt andere Dateien in Quellcode (z.B. CSS)
<LISTING> </LISTING>	-	-	Markiert Auflistung
<MAP> </MAP>	-	-	Umschließt Image-Map
<MARQUEE> </MARQUEE>	loop	n, infinite	Laufzeit
			Anzahl der Scrollings
			Bewegungsverhalten
			läuft durch
	behavior	scroll	läuft hin und her
		alternate	läuft rein, bleibt stehen
		slide	Legt Laufrichtung fest
	direction	left, right, up, down	Legt Hintergrundfarbe fest
	bgcolor	#xxxxxx, color	Höhe des Marquees in Pixeln o. %
	height	n o. %	Breite des Marquees in Pixeln o. %
	width	n o. %	Pixelabstand zwischen Abschnitten
	scrollamount	n	Millisekunden Pause zwischen 2 Animationsschritten
	scrolldelay	n	

Tag	Attribut	Wert	Beschreibung
<META> </META>	siehe Lexikon	-	Erzeugt Bereich für Zusatzinformationen
<NOBR>	-	-	Verhindert Zeilenumbruch
<NOFRAMES> </NOFRAMES>	-	-	Umfasst Alternativetext für frameunkompatible Browser
<NOSCRIPT> </NOSCRIPT>			Umfasst Alternativetext für scriptunkompatible Browser
<OBJECT> </OBJECT>	declare archive align codebase codetype data height width hspace vspace shapes standby type usemap	- Scripte left, center, right, baseline, middle, right, textbottom, textmiddle, texttop URL codetype URL n n n n - text type URL	Fügt beliebiges Objekt in Dokument ein Wandelt Objekt in Definition um Gibt Scripte an, die vor Objekt geladen werden sollen Legt Ausrichtung des Objekts fest Gibt die Code-Basis des Objekts an Gibt Codetype des Objekts an Gibt Daten an, die zum Objekt gehören Bestimmt die Höhe des Objektes Bestimmt die Breite des Objektes Definiert Freiraum links & rechts des Objekts Definiert Freiraum oben & unten des Objekts Gibt an, ob das Objekt Verweise enthält Gibt Text an, der während des Ladevorgangs des Objekts angezeigt werden soll Gibt den Type des Objekts an Gibt die Image-Map an, die mit Objekt benutzt werden soll
 	type start compact	1, a, A, i, I n -	Leitet sortierte Liste ein Gibt Aufzählungsart an Legt Startzahl der Auzählung fest Stellt Liste kompakt dar
<OPTION> </OPTION>	value selected disabled	text - -	Auswahlfeld im Select-Tag Gibt Ziel an Legt Feld als Vorausgewählt fest Deaktiviert Ziel
<P> </P>	align	left, center, right, justify	Definiert einen Absatz Ausrichtung
<PARAM> </PARAM>	name value	text	Passt Java-Applet Wünschen an Name des Parameters Wert für Parameter
<PLAINTEXT> </PLAINTEXT>	-	-	gleich Pre-Tag
<PRE> </PRE>	-	-	übernimmt vorformatierten Text
<Q> </Q>	src	URL	Markiert Zitat mit URL-Angabe
<S> </S>	-	-	Durchgestrichener Text
<SAMP> </SAMP>	-	-	Markiert Beispiel
<SCRIPT> </SCRIPT>	language	Sprache	Umfasst Scripte, mit Sprachangabe

Tag	Attribut	Wert	Beschreibung
<SELECT> </SELECT>			Erstellt Pop-up Menü
	size	n	Gibt an, wieviele Felder angezeigt werden sollen
	multiple	-	Erlaubt mehrere Auswahlmöglichkeiten
	disabled	-	Deaktiviert Menü
<SMALL> </SMALL>	-	-	relative kleiner Schrift
 	-	-	Zum Strukturieren mit CSS
<STRIKE> </STRIKE>	-	-	Durchgestrichener Text
 	-	-	Markiert zu betonenden Text
<STYLE> </STYLE>			Leitet Stilvorlagenbereich ein (z.B. für CSS)
	type	text	Bestimmt Sprache der Stilvorlage
	media	screen, print, projection, braille, aural, all	Gibt an, wofür das Dokument geeignet ist (normal, Drucken, für Projektoren, für Blindenschrift, Sprachausgabe, alles)
	-	-	Tiefgestellter Text
	-	-	Hochgestellter Text
<TABLE> </TABLE>			Leitet Tabelle ein
	bgcolor	#xxxxxx, color	Option der Hintergrundfarbe
	bordercolordark	#xxxxxx, color	Option der oberen Rahmenfarbe (3D-Effekt)
	bordercolorlight	#xxxxxx, color	Option der unteren Rahmenfarbe (3D-Effekt)
	background	URL	Fügt Hintergrundgrafik ein
	border	nichts, n	Fügt Rahmen ein (Wert kann dicke angeben)
	align	left, center, right, justify<	Gibt an wie der Text fließen soll
	cellpadding>	n	Legt den Abstand des Inhalts zu den Gitternetzlinien fest
	cellspacing	n	Legt die Dicke der Gitternetzlinien fest
	width	n o. %	Option der Breite in Pixeln o. Prozent
	hspace	n	Gibt Freiraum links & rechts der Tabelle an
	vspace	n	Gibt Freiraum oben & unten der Tabelle an
	frame	above, below, border, box, hspace, lhs, rhs, void, vsides	Hiermit lässt sich einstellen, um welche Teile der Tabelle ein Rahmen gezogen werden soll (vsides wäre z.B. nur vertikal)
	rules	all, cols, group, none, rows	Gibt an, welche Gitternetzlinien angezeigt werden sollen
	summary	text	Gibt Erklärung zum Inhalt der Tabelle
<TBODY> </TBODY>	alle Attribute wie <THEAD>	-	Tabellenzeilen werden Hauptteil der Tabelle zugewiesen
<TD> </TD>	alle Attribute wie <TR>	-	Leitet neue Tabellenzeile ein
	width	n o. %	Gibt Breite in Pixeln o. Prozent an
	colspan	n	Gibt an, wieviele Spalten die Zelle übergreift
	rowspan	n	Gibt an, wieviele Zeilen die Zelle übergreift
	background	URL	Fügt Hintergrundgrafik ein
	abbr	text	Legt Abkürzung für Tabellenkopf fest
	headers	?	?
	axis		(zu aufwendig zum Erklären)

Tag	Attribut	Wert	Beschreibung
<TEXTAREA> </TEXTAREA>			Mehrzeiliges Texteingabefeld
	rows	n	Legt Anzahl der Zeilen fest
	cols	n	Legt Anzahl der Breite fest
	wrap	soft	Zeilenumbruch
<TEXTFLOW> </TEXTFLOW>	-	-	Umfasst Alternative Text, falls ein Applet nicht angezeigt werden kann
<TFOOT> </TFOOT>	alle Attribute wie <THEAD>	-	Tabellenzeilen werden Fussteil der Tabelle zugewiesen (bei Ausdruck der Tabelle auf jeder Seite unten)
<TH> </TH>	alle Attribute wie <TD>	-	Leitet Überschriftszelle ein
	scope	-	?
<THEAD> </THEAD>			Tabellenzeilen werden Kopfteil der Tabelle zugewiesen (bei Ausdruck der Tabelle auf jeder Seite oben)
	align	left, center, right, justify	horizontale Ausrichtung der Daten
	valign	top, middle, bottom, baseline	vertikale Ausrichtung der Daten
	charoff	-	Definiert Abstand zwischen Zeichen
<TITLE> </TITLE>			Titel der Seite
<TR> </TR>			Leitet neue Tabellenzeile ein
	align	left, center, right	Option des horizontalen Textflusses
	valign	top, middle, bottom	Option des vertikalen Textflusses
	height	n	Gibt die Höhe der Zeile an
	bgcolor	#xxxxxx, color	Gibt die Hintergrundfarbe an
	charoff	-	Definiert Abstand zwischen Zeichen
<TT> </TT>	-	-	Teletype, Fernschreiber, unproportional
<U> </U>	-	-	Unterstrichener Text
 			Leitet unsortierte Liste ein
	type	square, circle, disc -	
	compact	Stellt liste kompakt dar	
<WBR> </WBR>	-	-	Darf auch in Nobr-Zone getrennt werden

Style-Informationen - Unterschiede CSS und JavaScript

Die Schreibweise der Style-Parameter unterscheidet sich zum Teil zwischen CSS/HTML und JavaScript.

Style-Informationen		
CSS-Name	JScrip-Name	Kurzbeschreibung
background	background	Hintergrunddefinition
background-attachment	backgroundAttachment	Hintergrundbild
background-color	backgroundColor	Hintergrundfarbe
background-image	backgroundImage	Hintergrundbild
background-position	backgroundPosition	Hintergrundpositionierung
background-position-x	backgroundPositionX	Hintergrund Position horizontal
background-position-y	backgroundPositionY	Hintergrund Position vertikal
border	border	Rahmendefinition
border-bottom	borderBottom	Rahmen unten
border-bottom-color	borderBottomColor	Rahmenfarbe unten
border-bottom-style	borderBottomStyle	Rahmenstil unten
border-bottom-width	borderBottomWidth	Rahmenbreite unten
border-collapse	borderCollapse	Rahmenbehandlung
border-color	borderColor	Rahmenfarbe
border-left	borderLeft	Rahmen links
border-left-color	borderLeftColor	Rahmenfarbe links
border-left-style	borderLeftStyle	Rahmenstil links
border-left-width	borderLeftWidth	Rahmenbreite links
border-right	borderRight	Rahmen rechts
border-right-color	borderRightColor	Rahmenfarbe rechts
border-right-style	borderRightStyle	Rahmenstil rechts
border-right-width	borderRightWidth	Rahmenbreite rechts
border-style	borderStyle	Rahmenstil
border-top	borderTop	Rahmen oben
border-top-color	borderTopColor	Rahmenfarbe oben
border-top-style	borderTopStyle	Rahmenstil oben
border-top-width	borderTopWidth	Rahmenbreite oben
border-width	borderWidth	Rahmenbreite oben
bottom	bottom	Abstand unten
clear	clear	Positionierung
clip	clip	Anzeigebereich
color	color	Farbe
cursor	cursor	Cursor
direction	direction	Leserichtung
display	display	Anzeigemodus
float	styleFloat	Darstellung
font	font	Schriftdefinition

Style-Informationen

CSS-Name	JScript-Name	Kurzbeschreibung
font-family	fontFamily	Schriftart
font-size	fontSize	Schriftgröße
font-style	fontStyle	Schriftstil
font-variant	fontVariant	Schriftvariation
font-weight	fontWeight	Schriftstärke
height	height	Höhe
layout-flow	layoutFlow	Überschneidung
layout-grid	layoutGrid	Layoutgitter
layout-grid-char	layoutGridChar	Zeichengittergröße
layout-grid-line	layoutGridLine	Elementgitter
layout-grid-mode	layoutGridMode	Layoutgitter-Modus
layout-grid-type	layoutGridType	Gittertyp
left	left	Abstand links
letter-spacing	letterSpacing	Buchstabenabstand
line-break	lineBreak	Behandlung des Zeilenumbruchs
line-height	lineHeight	Zeilenhöhe
list-style	listStyle	Listenstil
list-style-image	listStyleImage	Listenbild
list-style-position	listStylePosition	Listenposition
list-style-type	listStyleType	Listentyp
margin	margin	Außenabstand
margin-bottom	marginBottom	Abstand unten
margin-left	marginLeft	Abstand links
margin-right	marginRight	Abstand rechts
margin-top	marginTop	Abstand oben
min-height	minHeight	Mindesthöhe
overflow	overflow	Überschneidung
overflow-x	overflowX	Überschneidung horizontal
overflow-y	overflowY	Überschneidung vertikal
padding	padding	Innenabstand
padding-bottom	paddingBottom	Innenabstand unten
padding-left	paddingLeft	Innenabstand links
padding-right	paddingRight	Innenabstand rechts
padding-top	paddingTop	Innenabstand oben
page-break-after	pageBreakAfter	Seitenumbruch
page-break-before	pageBreakBefore	Seitenumbruch
right	right	Abstand rechts

Style-Informationen

CSS-Name	JScript-Name	Kurzbeschreibung
scrollbar-3dlight-color	scrollbar3dLightColor	Scrollbar
scrollbar-arrow-color	scrollbarArrowColor	Scrollbar
scrollbar-base-color	scrollbarBaseColor	Scrollbar
scrollbar-darkshadow-color	scrollbarDarkShadowColor	Scrollbar
scrollbar-face-color	scrollbarFaceColor	Scrollbar
scrollbar-highlight-color	scrollbarHighlightColor	Scrollbar
scrollbar-shadow-color	scrollbarShadowColor	Scrollbar
scrollbar-track-color	scrollbarTrackColor	Scrollbar
table-layout	tableLayout	Tabellenanzeige
text-align	textAlign	Textausrichtung
text-align-last	textAlignLast	Textausrichtung
text-autospace	textAutospace	Textdarstellung
text-decoration	textDecoration	Textverzierungen
text-indent	textIndent	Zeileneinschub
text-justify	textJustify	Textausrichtung
text-kashida-space	textKashidaSpace	Kashida
text-overflow	textOverflow	Textüberlauf
text-transform	textTransform	Textveränderung
text-underline-position	textUnderlinePosition	Position der Unterstreichung
top	top	Abstand oben
vertical-align	verticalAlign	Vertikale Ausrichtung
visibility	visibility	Sichtbarkeit
white-space	whiteSpace	Leerzeichenbehandlung
width	width	Breite
word-break	wordBreak	Worttrennung
word-spacing	wordSpacing	Wortabstand
word-wrap	wordWrap	Zeilenumbruch
writing-mode	writingMode	Schriftmodus
z-index	zIndex	Z-Position
zoom	zoom	Zoom

Darstellung von Sonderzeichen innerhalb von HTML

HTML Umlaute

Zeichen	Beschreibung	HTML Entsprechung	HTML Unicode
Ä	A Umlaut	Ä	Ä
ä	a Umlaut	ä	ä
Ë	E Umlaut	Ë	Ë
ë	e Umlaut	ë	ë
Ï	I Umlaut	Ï	Ï
ï	i Umlaut	ï	ï
Ö	O Umlaut	Ö	Ö
ö	o Umlaut	ö	ö
Ü	U Umlaut	Ü	Ü
ü	u Umlaut	ü	ü
ÿ	Y Umlaut	Ÿ	Ã
ÿ	y Umlaut	ÿ	ÿ

Häufig verwendete Sonderzeichen

Zeichen	Beschreibung	HTML Entsprechung	HTML Unicode
	Erzwungenes Leerzeichen	 	
ß	scharfes S	ß	ß
©	Copyright-Zeichen	©	©
®	Registriermarke-Zeichen	®	®
™	Trademark	™	™
«	angewinkelte Anf.zeichen links	«	«
»	angewinkelte Anf.zeichen rechts	»	¬
•	Bullet (größerer, dickerer Punkt)	•	•
·	Mittelpunkt	·	·
¨	Pünktchen oben	¨	¨
...	Ellipsis, Auslassungspunkte	…	…
¡	umgekehrtes Ausrufezeichen	¡	¡
¿	umgekehrtes Fragezeichen	¿	¿
	durchbrochener Strich	¦	¦
§	Paragraph-Zeichen	§	§
º	Ordinal-Zeichen männlich	º	º
ª	Ordinal-Zeichen weiblich	ª	ª
-	kurzer Trennstrich	­	­
—	Überstrich	¯	¯
¶	Absatz-Zeichen	¶	¶
†	einfaches Kreuz (Dagger)	†	†
‡	doppeltes Kreuz	‡	‡
°	Grad-Zeichen	°	°

Währungszeichen

Zeichen	Beschreibung	HTML Entsprechung	HTML Unicode
¢	Cent-Zeichen	¢	¢
€	Euro-Zeichen	€	€
£	Pfund-Zeichen	£	£
¤	Währungs-Zeichen	¤	¤
¥	Yen-Zeichen	¥	¥
f	Florin, Gulden	ƒ	ƒ

Mathematische Sonderzeichen

Zeichen	Beschreibung	HTML Entsprechung	HTML Unicode
±	Plusminus-Zeichen	±	±
÷	Divisions-Zeichen	÷	÷
×	Mal-Zeichen	×	×
√	Quadratwurzel	√	√
∫	Integral	∫	∫
¹	Hoch-1-Zeichen	¹	¹
²	Hoch-2-Zeichen	²	²
³	Hoch-3-Zeichen	³	³
´	Acute-Zeichen	´	´
¸	Häkchen unten	¸	¸
‰	Promille	&permill;	‰
μ	Mikro-Zeichen	µ	µ
¬	Verneinungs-Zeichen	¬	¬
½	ein Halb	½	½
¼	ein Viertel	¼	¼
¾	drei Viertel	¾	¾

Zeichen für andere Sprachen

Zeichen	Beschreibung	HTML Entsprechung	HTML Unicode
À	A mit Accent grave	À	À
Á	A mit Accent acute	Á	Á
Â	A mit Circumflex	Â	Â
Ã	A mit Tilde	Ã	Ã
Ä	A mit Ring	Å	Å
Æ	Æ-Ligatur	Æ	Æ
Ç	C mit Häkchen	Ç	Ç
È	E mit Accent grave	È	È
É	E mit Accent acute	É	É
Ê	E mit Circumflex	Ê	Ê
Ë	E mit Tilde	&Etilde;	Ë
Ì	I mit Accent grave	Ì	Ì
Í	I mit Accent acute	Í	Í
Î	I mit Circumflex	Î	Î
Ð	Eth (isländisch)	Ð	Ð
Ñ	N mit Tilde	Ñ	Ñ
Ò	O mit Accent grave	Ò	Ò
Ó	O mit Accent acute	Ó	Ó
Ô	O mit Circumflex	Ô	Ô
Õ	O mit Tilde	Õ	Õ
Ø	O mit Schrägstrich	Ø	Ø
Œ	Œ-Ligatur	Œ	Œ
Ù	U mit Accent grave	Ù	Ù
Ú	U mit Accent acute	Ú	Ú
Û	U mit Circumflex	Û	Û
Ý	Y mit Accent acute	Ý	Ý
Þ	THORN (isländisch)	Þ	Þ
à	a mit Accent grave	à	à
á	a mit Accent acute	á	á
â	a mit Circumflex	â	â
ã	a mit Tilde	ã	ã
ä	a mit Ring	å	å
æ	æ-Ligatur	æ	æ
ç	c mit Häkchen	ç	ç
è	e mit Accent grave	è	è
é	e mit Accent acute	é	é
ê	e mit Circumflex	ê	ê
ë	e mit Tilde	&etilde;	ë
ì	i mit Accent grave	ì	ì
í	i mit Accent acute	í	í
î	i mit Circumflex	î	î
ï	i mit Tilde	ĩ	ï
ð	eth (isländisch)	ð	ð
ñ	n mit Tilde	ñ	ñ
ò	o mit Accent grave	ò	ò
ó	o mit Accent acute	ó	ó
ô	o mit Circumflex	ô	ô
õ	o mit Tilde	õ	õ
ø	o mit Schrägstrich	ø	ø
œ	œ-Ligatur	œ	œ
ù	u mit Accent grave	ù	ù
ú	u mit Accent acute	ú	ú
û	u mit Circumflex	û	û
ý	y mit Accent acute	ý	ý
þ	thorn (isländisch)	þ	þ
Š	S mit Hacek	Š	Š
š	s mit Hacek	š	š

Griechisches Alphabet

Zeichen	Beschreibung	HTML Entsprechung	HTML Unicode
A	großes griechisches Alpha	Α	Α
B	großes griechisches Beta	Β	Β
Γ	großes griechisches Gamma	Γ	Γ
Δ	großes griechisches Delta	Δ	Δ
E	großes griechisches Epsilon	Ε	Ε
Z	großes griechisches Zeta	Ζ	Ζ
H	großes griechisches Eta	Η	Η
Θ	großes griechisches Theta	Θ	Θ
I	großes griechisches Iota	Ι	Ι
K	großes griechisches Kappa	Κ	Κ
Λ	großes griechisches Lambda	Λ	Λ
M	großes griechisches Mu	Μ	Μ
N	großes griechisches Nu	Ν	Ν
Ξ	großes griechisches Xi	Ξ	Ξ
O	großes griechisches Omicron	Ο	Ο
Π	großes griechisches Pi	Π	Π
P	großes griechisches Rho	Ρ	Ρ
Σ	großes griechisches Sigma	Σ	Σ
T	großes griechisches Tau	Τ	Τ
Υ	großes griechisches Ypsilon	Υ	Υ
Φ	großes griechisches Phi	Φ	Φ
X	großes griechisches Chi	Χ	Χ
Ψ	großes griechisches Psi	Ψ	Ψ
Ω	großes griechisches Omega	Ω	Ω
α	kleines griechisches alpha	α	α
β	kleines griechisches beta	β	β
γ	kleines griechisches gamma	γ	γ
δ	kleines griechisches delta	δ	δ
ε	kleines griechisches epsilon	ε	ε
ζ	kleines griechisches zeta	ζ	ζ
η	kleines griechisches eta	η	η
θ	kleines griechisches theta	θ	θ
ι	kleines griechisches iota	ι	ι
κ	kleines griechisches kappa	κ	κ
λ	kleines griechisches lambda	λ	λ
μ	kleines griechisches mu	μ	μ
ν	kleines griechisches nu	ν	ν
ξ	kleines griechisches xi	ξ	ξ
ο	kleines griechisches omicron	ο	ο
π	kleines griechisches pi	π	π
ρ	kleines griechisches rho	ρ	ρ
ς	kleines griechisches finales sigma	ς	ς
σ	kleines griechisches sigma	σ	σ
τ	kleines griechisches tau	τ	τ
υ	kleines griechisches ypsilon	υ	υ
φ	kleines griechisches phi	φ	φ
χ	kleines griechisches chi	χ	χ
ψ	kleines griechisches psi	ψ	ψ
ω	kleines griechisches omega	ω	ω