Homepage-Tipps & -Tricks

Franz Fiala

In einem Webverzeichnis gespeicherte Dateien wollen angezeigt werden, das ist die Grundfunktion von Webservern. Die meisten Seiten verhalten in ihren grundlegenden Eigenschaften der Beschreibungssprache HTML. Manchmal kann aber ein abweichendes Verhalten gewünscht sein, etwa ein Dokument zu verbergen oder nur in einem bestimmten Kontext anzuzeigen. In diesem Beitrag wird eine Auswahl solcher Techniken untersucht. Der Schwerpunkt liegt auf clientseitigen Skripts, die allen Betreibern von Websites zur Verfügung stehen. Ausgeklammert werden serverseitige Skripts.

Organisatorische Maßnahmen werden **am Server** selbst gesetzt, sie erfordern aber eine Zusammenarbeit mit dem Server-Operator und sind daher nur im Einzelfall verfügbar. In **HTML** selbst sind nur wenige Techniken verfügbar. Die meisten Möglichkeiten bietet **Javascript**, allerdings muss man sich mit Kompatibilitätsproblemen herumschlagen.

Alle Beispiele sind bei der Webversion dieses Artikels verfügbar.

Serverseitige Maßnahmen

Startdokument

Eine in der Adresszeile eingegebene Adresse (`http://` + `<Server>` + `<Pfad>` + `<Dokument>`) liefert dem Browser das Dokument. Fehlt `<Dokument>`, dann sucht der Server in dem Verzeichnis nach einem Startdokument. Startdokumente können durch den Serveradministrator beliebig definiert werden; auf Linux-Servern sind `index.html` oder `index.php`, auf Microsoft-Servern `default.htm`, `default.asp` oder `default.aspx` üblich.

Directory browsing

Wird kein Startdokument gefunden, hängt die weitere Reaktion von der Einstellung des *Directory browsing* ab. Es erlaubt einem Server, einen Verzeichnisinhalt zu zeigen, wenn es kein Startdokument gibt und wenn in der Adresse kein konkretes Dokument angegeben wurde. Ist *Directory browsing* verboten, erhält man eine Fehlermeldung wie *Directory browsing forbidden* oder *Document not found*, wenn das Verzeichnis nicht existiert oder eine *Friendly Error Message* vom Browser.

Normalerweise wird bei Servern das *Directory Browsing* ausgeschaltet, damit die User nicht versehentlich etwas zu sehen bekommen, was sie nichts angeht, Wenn es aber bei experimentellen Servern eingeschaltet sein sollte, muss man in jedem Verzeichnis ein Startdokument anlegen, wenn man die Sicht auf den Verzeichnisinhalt verhindern will.

Bei der neuen Clubverwaltung ***Helm*** kann das *Directory Browsing* unter *Website Settings* gesteuert werden. Beim alten Webserver ist es ausgeschaltet und muss vom Administrator für einzelne Anwendungen eingeschaltet werden.

Authentifizierung

Die häufigste Zugriffsform auf Webseiten erfolgt anonym und ohne Beschränkung.

Man kann aber einem Web oder auch nur einem Verzeichnis oder einer Datei über die Sicherheitseinstellungen des Servers besondere Zugriffsrechte verleihen, sodass nur eine bestimmte Gruppe von Usern nach Authentifizierung die Seiten sehen kann. Ein Beispiel sind die Memberseiten des CCC oder PCC. Diese Zugriffsform steht aber dem Besitzer eines Web nicht offen, da er im Allgemeinen User zu verwalten hat, die nicht alle auch User des serverseitigen Betriebssystems sind und daher dort nicht in einer Sicherheitsgruppe zusammengefasst werden können.

Daher bleibt diese Technik den Intranets vorbehalten, wo alle Web-User auch am Server als User definiert sind und in Gruppen Zugriffsrechte erhalten können.

Secure Folder

Wenn Sie Ihr Web auf dem Clubserver mit der Helm-Verwaltung betreiben, dann verfügen Sie über eine am Server implementierte Technik *secure folder*, die es Ihnen erlaubt, ein Verzeichnis nur bestimmten Usern zugänglich zu machen, die sie selbst definieren.

Technologie verbergen

Nehmen wir folgenden Link an: `http://meinedomain.at/anwendung.php`. Dieser Link referenziert die konkrete Datei `anwendung.php`. Die Seite ist erfolgreich und wird von Hunderten anderen Seiten verlinkt, natürlich auch von Suchmaschinen.

Jetzt schreiben wir die Anwendung um und nutzen dazu zum Beispiel die Dot-Net-Technologie. Daher muss der Dateiname geändert werden, der Link heißt jetzt `http://meinedomain.at/anwendung.aspx`. Alle bisherigen Links zu dieser Seite sind mit einem Schlag falsch und es dauert sehr lange, bis alle Nutzer dieser Seite ihre Links anpassen. Die sorgfältigen Programmierer lassen dann die alte Seite bestehen und versehen sie mit einem automatischen Umlenkung *(Redirect)*, kombiniert mit einem manuellen Link (Beispiele dazu im Abschnitt **HTML** und **Javascript**).

Solche Fehler sind such eine einfache organisatorische Maßnahme vermeidbar, die auch gleichzeitig Ordnung in große Webs bringt.

- Für jedes Dokument, das einen selbständigen Projektteil darstellt (man kann es auch grundsätzlich für alle Dokumente machen), wird ein gleichnamiges Verzeichnis erstellt. Das Dokument wird in dieses Verzeichnis kopiert (auch alle eventuellen weniger wichtigen Filialdokumente und Bilder) und in ein Startdokument, z.B. `default.php` umbenannt. (Startdokumente auf Microsoft-Servern sind `default.htm`, `default.asp`, `default.aspx` und `default.php...`, auf Linux-Servern sind es `index.htm`, `index.html`, `index.php...`)
- Alle Links zu diesem Dokument linken nicht auf den Dateinamen sondern auf den Ordnernamen. Beispiel:
- Vorher: `<a href="/anwendung.php">Link</a>` (Technologie sichtbar)
- Nachher: `<a href="/anwendung/">Link</a>`

Der Server startet selbsttätig das richtige Startdokument. Wird jetzt die Technologie geändert, ändert sich das Startdokument von `default.php` auf `default.aspx`, da dieses aber nicht direkt verlinkt ist, ist der Link weiterhin korrekt.

Ein sehr prominentes Beispiel für diese Technik Teile des ORF-Web. Als Demo können die Seiten `pcc.ac/seminare/`, `pcc.ac/anmeldung/`, `pcc.ac/seminare/seminare/`, `pcc.ac/seminare/angemeldet/` dienen.

HTML-Techniken

Die *Hypertext Markup Language* HTML ist eine Mischung zwischen Beschreibung der Struktur und Beschreibung des Aussehens von Texten. Es gibt nur wenige Steuerelemente (*Tags*), die auf die Anzeige eines Dokuments eingehen. Wenn das grundlegende Verhalten eines Tags geändert werden soll, benötigt man die Hilfe einer clientseitigen Skriptsprache, üblicherweise Javascript. Wenn der grundlegende Stil der Darstellung geändert werden soll, benötigt man die Konstruktion der Style-Sheets.

Browsereinstellungen

Während der Entwicklung der Webseite sollte man den Cache des Internet-Explorers ausschalten: ***Internetoptionen->Temporäre Internetdateien->Einstellungen->"Bei jedem Zugriff auf die Seite"***

Nicht verlinken

Die einfachste Möglichkeit ein Dokument zu verbergen, ist das Nicht-Verlinken mit anderen Dokumenten; dann entspricht der Dokumentenpfad einem zu erratenden Passwort.

Der Schutz ist aber nur dann gegeben, wenn die Adresse geheim gehalten wird. Wird die Adresse weitergegeben, kann man nicht mehr sicher sein, ob nicht andere einen Link zum Dokument legen und daher (ungewollt) publizieren und auch einer Suchmaschine zugänglich machen.

HTML selbst bietet keine Möglichkeiten eines Zugriffsschutzes. Eine Seite kann sich aber vor einer Indizierung durch Suchmaschinen schützen:

Keine Suchmaschine

Üblicherweise ist es erwünscht, eigene Inhalte in Suchmaschinen wiederzufinden. Bei Inhalten mit privatem Charakter ist das aber

franz@fiala.cc

nicht unbedingt erwünscht. Es empfiehlt sich daher, in jeder Seite, die von Suchmaschinen verschont bleiben soll, folgende Meta-Angabe im Head-Teil des Dokuments einzufügen:

```
<meta name="ROBOTS" content ="NOINDEX, NOFOLLOW">
```

Diese Zeile bewirkt, dass regelkonforme Robots beim Besuch der Seite diese weder indizieren noch die Links auf dieser Seite weiterverfolgen.

Jede Seite eines Web benötigt diesen Meta-Tag. Man kann aber das Verhalten eines Web gegenüber einem Robot auch in einer Datei `robots.txt` steuern, die man im Wurzelverzeichnis des Web ablegt. **[5]**, **[6]**

Erfahrungsgemäß funktionieren diese Angaben gut. Auf jeder Seite des **PCNEWS**-Web (alte Version) war diese Meta-Angabe enthalten und tatsächlich ist kaum ein Inhalt der **PCNEWS**-Seiten in Suchmaschinen zu finden. Dadurch ist auch das Verkehrsaufkommen in den letzten Jahren stark zurückgegangen.

TITLE verbirgt Pfad

Bei Dokumenten ohne `TITLE`-Tag wird in der Kopfzeile des Fensters der komplette Pfad des Dokuments angezeigt. Will man das aber vermeiden, dann genügt es, in jedem Dokument ein `TITLE`-Tag zu definieren. Das `TITLE`-Tag ersetzt den Pfad in der Kopfzeile des Fensters.

Adresse verbergen

Die Adresszeile verrät immer die aktuelle Seite und den Pfad dorthin. Auf Grund dieser Angaben, können neugierige User leicht weitere Inhalte finden, auch wenn diese nicht verlinkt sind.

Nehmen wir als Beispiel ein Verzeichnis mit Bildern. Der Besitzer will einige gelungene von 50 Bildern zeigen, die anderen speichern aber nicht anzeigen. Beispielsweise wird in der Adresszeile angezeigt:

`http://mydomain/pictures/img039.jpg`

Hier kann man raten, dass vielleicht auch `img000.jpg` oder `img040.jpg` exisitieren, wenn sie auch nicht verlinkt sind.

Die Adresszeile kann die wirkliche Adresse im Rahmen eines Frameset verstecken oder durch ein Fenster ohne Adresszeile gar nicht erst anzeigen (siehe Abschnitt Javascript).

Frame verbirgt Adresse

Verzeichnis `NoAddress`

Ein Frameset zeigt in der Adresszeile die Adresse des Framedokuments, nicht aber die Adressen der Einzeldokumente in den einzelnen Frames. Auf Grund der Fensterteilung erkennt man aber, dass es sich um eine Frametechnik handelt.

Wenn statt der Unterteilung eines Fensters in mehrere Frames nur ein einziges Frame verwendet wird, erkennt man nicht, dass es sich um ein Frameset handelt und verbirgt damit dennoch die Adresse des eigentlichen Dokuments.

`start.htm`

```
<frameset rows=100%,*>
  <frame src="start1.htm"
    frameborder="0" border="0" framespacing="0">
  <frame src=""
    frameborder="0" border="0" framespacing="0"
    scrolling=no noresize>
</frameset>
<noframes></noframes>
```

Die Datei `start1.htm` wird in der Adresszeile nicht angezeigt, `start.htm` bleibt stehen. Statt `start1.htm` kann natürlich ein Pfad oder eine beliebige Internetadresse stehen.

Eigentlich würde hier ein ein einziges Frame genügen, doch zeigen manche Versionen des Navigators Probleme bei der Anzeige des Inhalts, die durch das zweite leere Frame mit der Höhe 0 vermieden werden.

Ein größeres Online-Beispiel, das diese Technik verwendet, findet man unter *http://lehren.pcc.ac/*. In diesem Web wird immer nur diese Adresse angezeigt. Das oberste Frame kapselt ein weiteres darunter liegendes Frame, dessen Adresszeile je nach angezeigtem Inhalt immer einen anderen Inhalt hat und erst danach erfolgt die Bildschirmteilung in eine linke und rechte Hälfte. Wenn Sie den wirklichen Speicherort der angezeigten Dateien erfahren wollen, benutzen Sie die Taste [Kontextmenü] oder die ***rechte Maustaste->Eigenschaften***.

Das unsichtbare Frame kann noch eine weitere Aufgabe übernehmen: Auf einer Seite mit Javascript-Kode werden bei einem Reload der Seite oder beim Verzweigen auf eine andere Seite alle Variablen zurückgesetzt. Wenn man aber den Javascript-Kode in der unsichtbaren Seite unterbringt, kann man von allen folgenden Seiten immer auf diesen Kode und dessen Variablenzustand zurückgreifen.

Cache ausschalten

Wenn man an einer Internetseite arbeitet, ändert sich ihr Inhalt ständig, doch ein User bekäme immer nur den Inhalt einer früheren Version aus einem Cache zu sehen. Mit folgendem Meta-Tag im `HEAD`-Teil eines Html-Dokuments kann man erzwingen, dass das Dokument immer neu geladen wird:

```
<meta http-equiv="expires" content="0">
```

Regelmäßiges Update

Caches haben verschieden eingestellte Refreshintervalle, die man aber mit der folgenden Metaanweisung steuern kann:

```
<meta http-equiv="expires" content="Wed, 26 Nov 2004 08:21:57 GMT">
```

Redirect oder Reload

Verzeichnis `Redirect`

Man kann auch veranlassen, dass eine Seite nach einer einstellbaren Zeit automatisch neu geladen wird; im folgenden Beispiel wird die Seite `start1.htm` nach 3 Sekunden automatisch geladen.

`start.htm`

```
<meta http-equiv="refresh" content="3;URL=start1.htm">
```

Je nachdem, ob `pcnews.at` die aktuelle Seite oder eine andere Seite ist, handelt es sich um ein *Reload* oder ein *Redirect*. Da man am Browser das Auto-Redirect ausschalten kann, findet man für diesen Fall auf solchen Seiten immer auch den Text mit einem Link. Weitere Möglichkeiten, eine Seite neu zu laden finden sich im Abschnitt **Javascript**.

Darstellungsziel steuern

Verzeichnis `NewWindow`

Ein Link öffnet den neuen Inhalt `neuedatei.htm` im aktuellen Fenster. Die Datei `start.htm` enthält mehrere Demolinks zur Veranschaulichung des Linkverhaltens.

```
<a href="neuedatei.htm">Neue Datei</a>
```

Wenn der Inhalt des aktuellen Fensters aber erhalten bleiben soll, muss der neue Inhalt in einem neuen Fenster dargestellt werden. Dazu benutzt man das Attribut `target`:

```
<a target=neu href="neuedatei.htm">Neue Datei</a>
```

Immer, wenn in einem Link `target=neu` eingesetzt wird, erscheint der neue Inhalt im Fenster `neu`. Sollen es aber jeweils verschiedene Fenster sein, gibt es zwei Möglichkeiten: man verwendet immer einen anderen Namen (`neu`, `neu1`, `neu2`...) oder man benutzt den reservierten Wert `_blank`, mit dem immer ein neues Fenster geöffnet wird.

Besonders wichtig ist das `target`-Attribut bei Frames, weil der Inhalt dann jeweils in dem durch das `name`-Attribut bezeichnete Fenster öffnet.

Dieses grundlegende Verhalten des `a`-Tag kann mit Mitteln von Javascript verändert und erweitert werden.

Skripts

Alle weiteren Maßnahmen können nur mit Skriptprogrammierung implementiert werden. Skriptprogramme existieren als Quellkode und werden bei Aufruf durch einen Interpreter ausgeführt. Einerseits ist es die in HTML-Dokumenten verfügbare Skriptsprache JavaScript, anderseits die auf Servern verfügbare Skriptsprachen ASP/ASPX und PHP. Während Clientskripts in HTML-Dokumenten (Dateiendung `.htm` oder `.html` oder in inkludierten Dateien mit Endung `.js`) im Browser ausgeführt werden, werden Serverskripts am Server ausgeführt und nur mehr der reine HTML-Kode wird zum Client geschickt. Erkennbar sind die Serverskripts an den Dateiendungen `.ASP`, `.ASPX` und `.PHP`.

Serverskripts

Der Kode von Serverskripts ist für den User immer unsichtbar und daher auch die damit verbundenen organisatorischen Maßnahmen zum Anzeigen oder Verbergen von Seiten oder Inhalten. Da aber nicht jeder Webhoster auch automatisch Serverskripts zulässt, ist man in allen diesen Fällen auf die Möglichkeiten von Javascript angewiesen.

Bei unseren Clubservern sind die gängigen Serverskriptsprachen ASP, ASPX und PHP installiert und eingeschaltet.

Browsereinstellungen

Fehler, die bei Arbeiten an Serverscripts auftreten, werden in der Grundeinstellung des Internet-Explorers nicht angezeigt, man bekommt stattdessen die *"Friendly Error Pages"* zu sehen. Einzustellen ist

Internetoptionen*->*Erweitert*->*Kurze HTTP-Fehlermeldungen*->*nein

Clientskripts [1], [2], [4]

Bei allen Clientskripts muss man sich darüber im Klaren sein, dass der Programmkode dem Empfänger sichtbar ist und daher Rückschlüsse auf den Schutzmechanismus zulässt und damit eventuell umgangen werden kann.

Clientskripts kämpfen darüber hinaus mit dem Problem, dass über einen gemeinsamen Grundvorrat von Objekten, Eigenschaften und Methoden jeder Browser weitergehende und möglicherweise verschieden implementierte Möglichkeiten hat, was zur Folge hat, dass ein im Grunde einfaches Programm in mehreren Versionen gespeichert und dann durch eine Abfrage der Browserversion umgeschaltet werden muss.

Die nachfolgenden Skripts sind auf das unbedingt notwendige Minimum beschränkt, wurden mit dem Internet-Explorer getestet und müssen für andere Browser angepasst werden. Im Internet gibt es zahlreiche Sammlungen von Clientscripts, mit deren Hilfe diese hier angegebenen Beispiele verbessert werden können. Eine weitere wichtige Quelle sind alle Internet-Seiten, die selbst Javascript verwenden; wenn Sie ein praktisches Feature entdecken, nutzen Sie das Kontext-Menü, um über ***Quelltext anzeigen*** den Mechanismus herauszufinden.

Die Standardsprache am Client ist Javascript, es ist aber auch möglich, eine andere, dem betreffenden Browser verständliche Sprache einzuschalten, was aber nur für lokale Anwendungen Bedeutung haben kann.

Browsereinstellungen

Fehler, die bei Arbeiten an Clientskripts auftreten, werden in der Grundeinstellung des Internet-Explorers nicht angezeigt. Einzustellen ist ***Internetoptionen*->*Erweitert*->*Skriptfehler anzeigen*->*ja***. Ob man ***Skriptdebugging einschalten*** aktiviert, hängt davon ab, ob man den Skriptdebugger installiert hat. **[7]**

Einschränkungen

Betrachter von Webinhalten können Javascript in ihrem Browser ausschalten oder einen Browser benutzen, der Javascript nicht versteht, daher sind die hier vorgestellten Skripts stark browserabhängig.

Die Unmöglichkeit, diese Maßnahmen so formulieren zu können, dass sie nicht durch findige User entdeckt werden könnten, zeigt, dass man um serverseitige Skripts nicht herumkommt, wenn es um Fragen der Sicherheit geht. Das Gros der User wird aber auch schon durch diese einfache Taktiken auf den gewünschten Inhalt gelenkt (oder davon abgelenkt).

Schützen kann man nur Dokumente, die über eine Skriptsprache verfügen wie zum Beispiel HTML oder PDF. Daher bleibt für alle passiven Dateien (Bilder, Sounds, Videos) nur folgende Möglichkeiten

- Nicht-Verlinken oder
- das Verwenden von Datenbanken, die sowohl Bilder aber auch Texte speichert und über eine eigene Zugangskontrolle verfügt oder
- serverseitiger Zugriffsschutz (*secure folder* bei Clubwebspace)

Nur HTML-Seiten und PDF-Seiten können daher hinsichtlich ihres Abrufs selbständige Entscheidungen treffen.

Javascript

Syntaktische Hinweise

In allen folgenden Beispielen wird nur der Programmkode angegeben. Jedes Beispiel ist aber durch folgende Zeilen einzugrenzen. Die Demoversion der Skripts bei des Webversion dieses Artikels enthält den vollständigen Konstrukt:

```
<script language=Javascript1.2>
<!--
...
//-->
</script>
```

Die Kommentarzeichen am Beginn und am Ende des Skriptabschnitts verhindern eine Anzeige des Kodes bei nicht-skriptfähigen Browsern, die Sprachangabe mit Versionsnummer präzisiert den erforderlichen Sprachumfang.

Browser und Javascript-Version

Verzeichnis `Browser`

Wenn es auch in allen hier gezeigten Beispielen aus Gründen der Übersichtlichkeit nicht verwendet wird, muss man doch bei vielen Skripts zwischen den verschiedenen Browsertypen unterscheiden. Das nachfolgende Skript erkennt den verwendeten Browser:

browser.htm

```
var browser = navigator.appName
var version = parseInt(navigator.appVersion)
document.write('Dein Browser ist: ' + navigator.appName + ', Version: ' +
version + '<br>')
```

Die unterstütze Javascript-Version erfährt man aber nur mit einem Trick. In diesem Beispiel ist das `language`-Attribut im `script`-Tag wesentlich. Es werden alle Skriptabschnitte bis zu der aktuellen Version abgearbeitet, danach wird die Variable `version` nicht mehr verändert.

version.htm

```
<script language="JavaScript">
var version = 1;
</script>
<script language="JavaScript1.1">
var version = 1.1;
</script>
<script language="JavaScript1.2">
var version = 1.2;
</script>
<script language="JavaScript1.3">
var version = 1.3;
</script>
<script language="JavaScript1.4">
var version = 1.4;
</script>
<script language="JavaScript">
document.write('Dein Browser unterstützt JavaScript ' + version);
</script>
```

Pfad verbergen, Bedienung verhindern

Verzeichnis `NoMenu`

Das Standard-Layout des Browsers erlaubt es, über ***Ansicht*->*Quelltext*** den Programmkode des HTML-Dokuments anzuschauen und auszuwerten. Will man das verhindern, darf man die Menüzeile nicht anzeigen.

Die Adresszeile der Browsers gibt Aufschluss über den Speicherort des gerade angezeigten Dokuments, daher kann ein neugieriger User versuchen, ein durch Nicht-Verlinken verstecktes Dokument durch Ausprobieren herauszufinden. Erschwert wird diese Suche, wenn die Adresszeile im Browser nicht sichtbar ist.

Die Eigenschaft `menubar` eines `window`-Objekts kann im Netscape-Navigator ausgeschaltet werden:

```
self.menubar.visible=false;
```

nicht aber im Internet-Explorer.

Man kann aber beim Einstig in eine Website erzwingen, dass ein neues Browserfenster geöffnet wird, welches keine Menüleiste enthält und dann alle folgenden Inhalte in dem neuen Fenster anzeigt; das funktioniert bei allen Browsern.

Daher zwingt man zunächst den User, dass er das Informationsangebot über eine einheitliche Einstiegsseite sehen kann (Startdokument). Danach öffnet man eines neues Fensters mit den gewünschten Eigenschaften.

Das folgende Skript `start.htm` öffnet ein neues Fenster, das keine Adresszeile besitzt (und das auch in anderen Merkmalen andere Eigenschaften hat als das Standardfenster) und in diesem neuen Fenster wird das eigentliche Dokument `start1.htm` angezeigt.

Da jetzt zwei Fenster geöffnet sind, kann man der Ordnung halber versuchen, das erste Hilfsfenster mit `opener.window.close()` zu schließen, was prinzipiell möglich ist aber von einem Popup-Fenster mit Warnhinweis begleitet wird, sodass der User diesen Vorgang unterbinden kann.

start.htm

```
Hauptfenster=window.open("start1.htm","Homepage",
  "alwaysLowered=0, alwaysRaised=0, " +
  "dependent=0, directories=0, " +
  "height=480, width=640, " +
  "hotkeys=0, location=0, " +
  "innerHeight=480, innerWidth=640, " +
  "menubar=0, resizable=0, " +
  "screenX=0, screenY=0, " +
  "scrollbars=0, status=0, " +
  "titlebar=0, toolbar=0"
)
```

Die lange Parameterliste ist nicht unbedingt notwendig, wenn man sich mit den Grundeinstellungen begnügen kann. In diesem Beispiel

ist die Liste vollständig angegeben worden, um sie einfacher anpassen zu können.

```
start1.htm
...
window.opener.close()
```

Der Effekt dieser Maßnahme ist, dass der User das Dokument start1.htm in einem Fenster ohne Bedienungsmöglichkeiten sieht und daher auch nicht über den Menüpunk ***Ansicht->Quellkode*** verfügt.

Da das Schließen des ersten Fensters nicht ohne Warnung erfolgt, könnte man das Schließen unterlassen und statt dessen den Fokus auf das neu geöffnete Fenster richten und im ersten Fenster eine allgemeine Information über das Web zu belassen und auch ein Inhaltsverzeichnis mit einer Linkliste, deren Ziel immer das zweite Fenster ohne Bedienungsmöglichkeit ist.

User, die mit den Tastaturkodes eines Browsers vertraut sind, können aber die Bedienungselemente durch Drücken der Taste F11 jederzeit wieder sichtbar machen. Außerdem gibt es ja noch das Kontext-Menü (oder die rechte Maustaste), mit dem man über ***Eigenschaften*** die Adresse herausfinden kann. Aber auch für diese Fälle gibt es Gegenmittel:

Rechte Maustaste ausschalten

Verzeichnis NoRightClick

Wenn man den Eventhandler für das Ereignis `onMouseDown` durch eine eigene Funktion ersetzt, kann man durch den Rückgabewert `false` die weitere Behandlung unterbinden.

```
start.htm
message = "Danke für deinen Besuch";
function NoRightClick(b) {
   if ( event.button > 1)    {
     alert(message)
     return false
   }
}
document.onmousedown = NoRightClick
```

Weiter bleibt immer noch die Taste Kontextmenü (zwischen AltGr und Strg), die nur durch die sehr im Vordergrund stehende Mausbedienung leicht in Vergessenheit gerät.

Tasten ausschalten [3]

Verzeichnis NoKeys

Zunächst muss man wissen, dass jeder Tastendruck drei Ereignisse auslöst `onkeydown`, `onkeyup` und `onkeypress` und dabei eine Zahl (Tastaturkode) in der Eigenschaft `window.event.keyCode` hinterlässt. Das Ereignis kann man mit einer eigenen Funktion `document.onkeydown` abfangen. Das folgende Beispiel zeigt, wie man den Tastaturkode jeder Taste erfragen kann:

```
keycode.htm
function document.onkeydown() {
  alert (window.event.keyCode)
}
```

Zum Beispiel kann man durch Erweiterung dieser Funktionen die Großschreibung erzwingen oder Tasteneingaben ganz ausschalten. Leider funktionieren diese Mechanismen nicht bei besonderen Tasten wie F11 und Kontextmenü, da die Tastaturkodes dieser Tasten (122 und 93) noch vor der Ausführung der Event-Handler abgearbeitet werden, man muss daher mit Tricks arbeiten.

Kontextmenü ausschalten

Wenn der Handler onkeydown verwendet wird, daher wird beim Drücken der Taste Kontextmenü zwar zuerst das Kontextmenü gezeigt aber unmittelbar danach durch das Alert-Fenster im onkeydown-Handler geschlossen, so dass es dem User letztlich verborgen bleibt.

```
start.htm
function document.onkeydown() {
  if (window.event.keyCode==93)
    alert ("Kein Kontextmenü")
}
```

F11 ausschalten

Die Taste F11 schaltet für jede aktive Explorer-Seite jedenfalls das Menü im Vollbildmodus ein, auch wenn alle Bedienungselemente mit Javascript ausgeschaltet worden sind. Dagegen kann man sich zwar mit Javascript nicht schützen, doch kann man darauf reagieren, indem man den Inhalt des aktuellen Fensters durch eine leere Seite (oder andere belanglose Information) ersetzt:

```
startF11.htm
function document.onkeydown() {
  if (window.event.keyCode==122)
    window.location.href="blank.htm"
}
```

Zurück-Button ausschalten

Ein ähnlich schwieriger Fall ist der Back-Button am Browser. Man kann ihn nicht ausschalten. Man kann aber die *History* manipulieren und damit einen anderen Rückweg markieren.

Version 1

Man schreibt auf jene Seite, zu der man nicht zurückkehren möchte:

```
nohistory1.htm
window.history.forward(1)
```

Der Zurück-Button ist zwar aktiv, doch gelangt man auf dieselbe Seite, die man gerade sieht.

Version 2

```
nohistory.htm
<script>
function go(url) {
    if (document.images)
        location.replace(url);
    else
        location.href = url;
}
</script>
<A HREF="javascript:go('back.htm')">Die nächste Seite hat keine
Vorgeschichte</A>
```

Button als Hyperlink

Verzeichnis Redirect

Wenn eine Weiterleitung nicht über einen Link, sondern über einen Button erfolgen soll, muss man das grundlegende Verhalten des Button ändern. Im folgenden Beispiel wird die Seite start1.htm durch Drücken des Button ***"Redirect"*** geladen:

```
startj.htm
<form>
<input value="Redirect" type="button"
   onClick="window.location.href='start1.htm'">
</form>
```

Redirect oder Reload

Verzeichnis Redirect

Da man am Browser das *Auto-Redirect* ausschalten kann, ist eine Javascript-Version gefragt, mit der man dieses HTML-Tag nachbilden kann.

```
startjauto.htm
function AutoRefresh() {
    location.href = 'start1.htm';
}
setTimeout('AutoRefresh()',3000);
```

Je nachdem, ob man die aktuelle Seite oder eine andere Seite als Ziel einsetzt, handelt es sich um einen *Reload* oder *Redirect*.

Back-Button implementieren

Verzeichnis Redirect

Auf manchen Webseiten findet man den Hinweis, dass man den "Back-Button" des Browsers benutzen soll. Dabei ist es ganz einfach, den Button in der eigenen Seite nachzubilden

```
start1.htm
Hier gehts zurück zur Startseite:
<a href="javascript:history.back()">Go</a>
```

Alle Beispiele im Verzeichnis Redirect verzweigen zur Seite seite1.htm. Um zur richtigen Vorgängerseite zurückkehren zu können, müsste man diese Seite vielfach ausführen oder eben den Hinweis anbringen, man solle den "Back-Button" des Browsers verwenden oder aber - wie im obigen Beispiel - man benutzt die Methode history.back(), die den "Back-Button" simuliert. Man kehrt daher von der Ziel-Datei start1.htm immer zu der rufenden Datei zurück.

Linkverhalten ändern

Verzeichnis AskLink

Ein Link in einem Text veranlasst den Browser jedenfalls den aktuellen Fensterinhalt durch ein neues Dokument zu überschreiben. Will man dem User die Chance geben, das noch einmal zu überlegen (etwa, wenn er im Begriff ist das eigene Web zu verlassen), eignet sich folgender Kode:

```
start.htm
<script>
function Bestaetigung() {
  return window.confirm ("'OK' um fortzufahren")
}
</script>
```

```
<A NAME=b2 HREF="start1.htm" onclick="return Bestaetigung()">Next</A>
```

Zusatzinformationen

Verzeichnis `Info`

Will man eine weitergehende Information zum Text in einer HTML-Seite anbringen aber nicht direkt beim Text (etwa, weil man um Übersicht bemüht ist), gibt es mehrere Möglichkeiten. Einfach ist die Verwendung eines Frame, wo in einem kleinen Teil des aktuellen Rahmens die Zusatzinformation angezeigt wird oder eben der Rahmen für die Zusatzinformation leer bleibt. Störend ist dabei, dass der Platz für die Zusatzinformation auch dann reserviert ist, wenn man gar nichts anzeigt.

Eine weitere Möglichkeit ist ein Filialfenster, das mit einem Link geöffnet und mit einem weiteren Link geschlossen wird.

Die Datei `start.htm` enthält alle folgende Beispiele in einer einzigen Datei.

start.htm Variante 1

```
<script>
var Hilfefenster = null
function OpenHelp() {
  Hilfefenster=window.open("hilfe.htm","Hilfe",
    "directories=0, height=200, width=200, "+
    "screenX=10, screenY=10, toolbar=0, " +
    "location=0, menubar=0, resizable=0, " +
    "scrollbars=0, status=0, titlebar=1")
  Hilfefenster.focus()
}
function CloseHelp() {
  if (Hilfefenster!=null) {
    Hilfefenster.close()
    Hilfefenster=null
  }
}
</script>
<a href="javascript:OpenHelp()">Hilfefenster öffnen</a>
<a href="javascript:CloseHelp()">Hilfefenster schließen</a>
```

Das Filialfenster kann problemlos (und ohne Warnung) geschlossen werden, weil man es selbst mit dem Skript geöffnet hat.

`hilfe.htm`

```
Hilfetext, erscheint in einem eigenen Fenster.
<a href="javascript:self.close()">Fenster schließen</a>
```

Das Fenster `Hilfefenster` wird in der Datei `start.htm` geöffnet, dort die Datei `hilfe.htm` geladen und kann sowohl aus dem rufenden Fenster als auch vom gerufenen Fenster geschlossen werden.

Unelegant ist, dass die beiden Links `"Hilfefenster öffnen"` und `"Hilfefenster schließen"` gleichzeitig angezeigt werden. Besser wäre es, wenn bei geschlossenem Hilfefenster nur der Text `"Hilfefenster öffnen"` und nach dem Öffnen nur `"Hilfefenster schließen"` stehen würde. Das gelingt mit einem gemeinsamen Link, der gleichzeitig Text und auch Funktion durch den Zustand des Hilfsfensters ändert.

start.htm Variante 2

```
<script>
function ToggleWindow() {
  if (Hilfefenster==null) {
    OpenHelp()
    Text.innerHTML="Hilfefenster schließen"
  }
  else {
    CloseHelp()
    Text.innerHTML="Hilfefenster öffnen"
  }
}
</script>
<a href="javascript:ToggleWindow()"><span id=Text></span></a>
<script>
Text.innerHTML="Hilfefenster öffnen"
</script>
```

Hier wird die Möglichkeit benutzt, dass man jedes Element eines Dokuments über das `ID`-Attribut ansprechen kann. Der Text, den das `SPAN`-Element einschließt, hat den Variablennamen `innerHTML`.

Allzu viele Fenster können rasch unübersichtlich werden. Daher können kleinere Hilfstexte auch einfacher eingefügt werden:

Infofenster

start.htm Variante 3

```
<span title="Hilfetext, erscheint, wenn man drüber fährt.">Um Hilfetext
zu zeigen, hier die Maus bewegen.</span>
```

Text aufdecken

Das zunächst leere `SPAN`-Element `hi` wird durch die Funktion `ShowText()` mit dem Text gefüllt; gleichzeitig wird das Element `h` verändert.

start.htm Variante 4

```
<script>
function ShowText() {
  if (hi.innerHTML=='') {
    h.innerHTML='verbergen'
    hi.innerHTML='Hier ist der Hilfetext'
  }
  else {
    h.innerHTML='zeigen'
    hi.innerHTML=''
  }
}
</script>
<a href="javascript:ShowText()">Hilfetext <span
id=h>zeigen</span></a><br>
<span id=hi></span><br>
```

Überlappende Texte

Die universellste Methode dürfte das Überlappen zweier unabhängiger Texte sein, denn damit kann der Hilfetext aus beliebigen Elementen bestehen (Texte, Bilder, Links...).

start.htm Variante 5

```
<script>
function ShowTextFloating() {
  Hilfe.style.display=''
  Hilfe.style.position='absolute'
  //Hilfe.style.top='100px'
  //Hilfe.style.left='100px'
}
</script>
<SPAN
  onMouseOver="ShowTextFloating()"
  onMouseOut="Hilfe.style.display='none'"
>
Hilfetext zeigen</SPAN>
<DIV id=Hilfe STYLE="display:none; background-color:lightgreen">
<table width=200><tr><td>
Das ist der eigentliche Hilfetext<br>
er kann auch sehr vielfältig sein und <br>
kann auch Links enthalten. <br>
Man kann den Text absolut positionieren oder relativ;
mit einer Positionsangabe kann der Text irgendwo am Bildschirm sein
(auch bezogen auf die aktuellen Mauskoordinaten), ohne
Positionsangabe ist der Text bei dem aktuellen Element.
</td></tr></table>
</DIV>
```

iframe

Ein besonders elegantes Konstrukt ist ein `iframe`. Er erlaubt die Einfügung einer zweiten Datei in den Verlauf einer HTML-Seite. Der `iframe` kann auf ein bestimmtes Maß eingestellt werden und rüstet sich selbst mit einer Scroll-Leiste aus, wenn der Text zu lang sein sollte.

`start.htm Variante 6`

```
<iframe src="hilfe1.htm" width=200 height=50>
</iframe>
```

Passwortschutz

Verzeichnis `NoAccess`

Da Javascript den gesamten Kode zum Client schickt, würde ein Formular zu Eingabe des Passworts im Rahmen der Prüfroutine auch das Passwort selbst verraten.

Daher verwenden Javascript-Programmierer folgenden Trick: sie verwenden als Startdatei eine Datei deren Name das Passwort ist. Voraussetzung ist natürlich, dass *Directory Browsing* nicht möglich ist und der Zugang zu der Homepage über ein Startdokument erfolgt (z.B. `index.html` oder `default.htm`).

start.htm

```
<script>
  function go() {
    window.location.href = "" +
        document.PWform.PW.value + '.htm';
    return false;
  }
</script>
<form name="PWform" onSubmit="return go()">
Passwort: <input type="password" name="PW" value="" size=8>
</form>
```

12345.htm

```
<title>Das Startdokument</title>
<p>Diese Datei ist das eigentliche Startdokument.</p>
```

Wichtig ist das `TITLE`-Tag, damit in der Kopfzeile der Text `"Das Startdokument"` steht und nicht der Dateiname, der ja das Passwort darstellt, damit ein anderer Beobachter den Dateinamen nicht in der Titelzeile sieht.

Frames abwerfen (alle)

Verzeichnis NoFrames/FramesAlle

Nehmen wir an, dass es zu verhindern gilt, dass ein Pirat eine Seite in seinen eigenen Frameseiten inkludiert.

Dazu muss man die Adresse des einschließenden Frame durch die Adresse der eigenen Homepage ersetzen. Das klingt plausibel, birgt aber ein wichtiges Problem: Die Javascript-Security ist so ausgelegt, dass es einem Skript nicht möglich ist, die Inhalte von Seiten zu analysieren, wenn sie nicht vom eigenen Server kommen und dann auch noch deren `location` zu ändern. Das heißt, dass das folgende Skript nicht funktioniert, wenn man sich in einem fremden Frame befindet. Bereits die erste Zeile meldet einen Zugriffsfehler von Javascript.

```
if (top.location.href != self.location.href)
  top.location.href = self.location.href;
```

Das Problem der Bedingung ist, ob der Browser das Auslesen der Adresse der Frames erlaubt (die von Hersteller und Version abhängt) und das Problem der Anweisung ist, ob man die `location` des einschließenden Frame ändern kann (was im Allgemeinen möglich ist). Beide Punkte erfordern daher eine besondere Formulierung, weil die Ansprache der Eigenschaft `location.href` so einfach nicht möglich ist. Erfolgreich war folgende Variante:

```
if (top != self) top.location.href = self.location.href;
```

Man vergleicht daher nicht den Inhalt der Adresszeilen, sondern prüft nur, ob das äußerste Frame (`top`) gleich dem eigenen Fenster (`self`) ist.

Zum Testen dieses Skripts wurden im Ordner NoFrames/FramesAlle vier Dateien angelegt:

Dummy.htm	Hilfsdatei, die in allen nicht benutzen Frames angezeigt wird
FrameEigen.htm	Frame-Datei, die in allen Frames nur die Datei Dummy.htm inkludiert; sie zeigt, wie das Frameset aufgebaut ist.
FrameFremd.htm	wie FrameEigen.htm aber im Hauptframe wird die Datei MeineSeite.htm inkludiert
MeineSeite.htm	wird diese Seite in einem Frameset inkludiert, dann schüttelt sie das Frameset ab

Fremde Frames abwerfen

Verzeichnis NoFrames/FramesFremd

Im vorigen Beispiel wird jede Inklusion einer Seite in einem Frame abgeschüttelt. Das verhindert aber auch die Anwendung eigener Frames. Dafür muss man zwischen fremden und eigenen Frames unterscheiden. Das Beispiel im Verzeichnis NoFrames/FramesFremd verwendet den Inhalt des Title-Tag, um zwischen dem eigenen und einem fremden Frame zu unterscheiden.

```
if ((top.document.title!="MeinFrame") && (self!=top))
    top.location.href = self.location.href
```

Frames annehmen

Verzeichnis NoFrames/FrameShow

Der Vorteil von Frames ist die Möglichkeit der Trennung zwischen feststehenden Inhaltsverzeichnissen und dem eigentlichen Inhalt. Inhalte können daher unabhängig vom Inhaltsverzeichnis erstellt werden. Für die volle Navigationsmöglichkeit muss aber immer der ganze Frame gezeigt werden und nicht die Inhaltsseite allein. Oft findet man in Suchmaschinen nur die Inhaltsseite aber nicht den einschließenden Frame. Daher wird ein Mechanismus gesucht, der bei Aufruf der Seite immer auch den zugehörigen Frame lädt.

```
if (self==top)
  top.location.href = "FrameEigen.htm"
else
  if (top.document.title!="MeinFrame")
    top.location.href = self.location.href
```

Wenn man die Seite MeineSeite.htm über eine direkte Adresse im Browser aufruft, erscheint sie immer in dem Frameset MeinFrame.htm.

Be Linked

Verzeichnis NoLink

Die Seiten eines Webs sind entsprechend den Regeln des HTTP-Protokolls zusammenhanglos. Es ist gleichgültig, ob man die Startseite eines Webs aufruft und dann über Links zu einem gewünschten Inhalt navigiert oder, ob man den Inhalt direkt über den Text in der Adresszeile anzeigen lässt. Das ist bei statischen Inhalten auch weiter kein Problem.

Will man aber eine bestimmte Lesereihenfolge eingehalten wissen (zum Beispiel, damit der Leser keinesfalls die Nutzungsbedingungen überspringen kann), muss man bei direktem Aufruf einer bestimmten Seite sicherstellen, dass nur der Weg von einer gewünschten Vorgängerseite zulässig ist. Die zugehörige Information liefert der so genannte **Referrer**, Eigenschaft des `document`-Objekts. Allerdings ist `document.referrer` nur auf einem Webserver definiert. Der Kode funktioniert daher nicht, wenn man die Dateien lokal auf einem Desktop testet.

start.htm

```
<a href="start1.htm">Nächste Seite</a>
```

Die Seite start.htm ist jene Seite, von der aus die Seite start1.htm erreichbar sein soll. In diesem Beispiel enthält sie nur diesen Link.

start1.htm

```
<script>
Fehlerseite = 'dummy.htm'
GerufenVon = 'start.htm'
if (document.referrer.indexOf(GerufenVon)==-1)
  self.location.href=Fehlerseite
</script>
<p>Diesen Text sehen Sie nur, wenn die Seite von einem Server
aufrufen. Das hier eingebaute Skript verhindert, dass jemand die
Seite direkt aufruft oder von einer Seite aufruft (verlinkt), die auf
einem fremden Server liegt.</p>
```

Die Zielseite start1.htm prüft, ob im Text des Referrers der Text "start.htm" enthalten ist. Man kann auch auf den gesamten Textstring überprüfen, dann kann man aber die Dateien nicht mehr auf einen anderen Server verschieben, ohne auch den Kode zu ändern. Wenn die Seite richtig gerufen wird, wird der nachfolgende Text angezeigt, wenn nicht, bekommt der User die Fehlerseite dummy.htm zu sehen.

Zusammenfassung aller Maßnahmen

Wenn es gilt, die hier vorgestellten Maßnahmen auf allen Webseiten einer Homepage anzuwenden, bewährt sich die Möglichkeit, Javascript-Kode in externen Dateien auslagern zu können. Kodeänderungen in dieser externen Datei wirken sich dann automatisch in allen Seiten aus, die die Datei inkludieren.

```
<SCRIPT src="myjs.js"></SCRIPT>
```

Ein neugieriger Leser dieser Datei sieht dann den Kode nicht unmittelbar, kann ihn aber leicht finden, indem er den Dateinamen myjs.js statt dem Namen der HTML-Seite eingibt oder indem er die Seite lokal speichert durch Eingabe von ***Datei -> Speichern unter -> Webseite komplett***. Es wird eine Datei mit dem Titel der Webseite angelegt und alle Zusatzdateien dieser Seite (Bilder, Sounds, Scripts, Stylesheets...) werden in einem gleichnamigen Unterverzeichnis mit der Endung _dateien (oder _files) gespeichert.

Beispieldateien

Alle Beispiele finden sich bei der Webversion dieses Artikel. Man kann sowohl die einzelnen Dateien anwählen oder über ein Menü jede einzelne Version ausprobieren. Die Webversion findet man so: *http://pcnews.at?Id=pcn89*.

Literatur

[1]	Javascript-Materialien	*http://lehren.pcc.ac/* ->Computersprachen -> Javascript
[2]	Javascript FAQ	*http://developer.irt.org/script/*
[3]	Tastaturkodes	*http://www.js-examples.com/beginners/key_codes.php3*
[4]	(D)HTML-Dokumentation	*http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/dhtml_node_entry.asp*
[5]	Robots	*http://www.robotstxt.org/wc/robots.html*
[6]	Robots	*http://www.searchengineworld.com/robots/robots_tutorial.htm*
[7]	Skript-Debugger NT/2000/XP	*http://www.microsoft.com/downloads/details.aspx?FamilyID=2f465be0-94fd-4569-b3c4-dffdf19ccd99&DisplayLang=en*

http://demo.pcc.ac/ -> Homepage-Tipps