# Embedded Webserver in Forth

# Die Anfrage vom Browser

GET / HTTP/1.1
User-Agent: Opera/9.80 (Windows NT 5.1)
Presto/2.12.388 Version/12.12
Host: 192.168.5.2
Accept: text/html, application/xml;q=0.9,
application/xhtml+xml, image/png, image/webp,
image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: de-DE,de;q=0.9,en;q=0.8
Accept-Encoding: gzip, deflate
Cache-Control: no-cache
Connection: Keep-Alive

# Die Antwort vom Server

```
HTTP/1.1  200  OK
Server: MiniModul/1.0 (Forth)
Content-Length:         69
Content-Language: de
Connection: Keep-Alive: timeout=5
Content-Type: text/html

<html> <body> <h1> Hello World </h1> </body> </html>
```

# statische Webseite

```
<HTML>
<body>
<h1>
  Hello  World
</h1>
</body>
</html>
```

# Scripting

```
<HTML>
<body>
<h1>
  Hello  World
</h1>
<br><br>
<$  @TIME  ADD.NUMBER  $>
</body>
</html>
```

# Ziel: Compilieren von HTML

In embedded Systemen gibt es nicht notwendigerweise ein Dateisystem,  deshalb soll HTML compiliert werden können.

```
: HELLO (  –  )
<HTML>
<body>
<h1>
  Hello  World
</h1>
<br><br>
<$  @TIME  ADD.NUMBER  $>
</body>
</html>
<$
;
```

# der Interpreter

```
: HELLO (   –   ) <HTML> <body> <h1> Hello World </h1> <br>....


….
BEGIN
  BL  WORD ( char – ^str )
  FIND ( ^str – ^str false | xt 1 | xt true )
  DUP
  IF          …..
  ELSE   DROP  ( ^str ) NUMBER? IF …. ELSE …. THEN
  THEN
  DONE?
UNTIL
….
```

# der modifizierte Interpreter

```
: HELLO (   –   ) <HTML> <body> <h1> Hello World </h1> <br>....

….
BEGIN
  BL  WORD ( char – ^str )
  vEX]  @  DUP  [']  NOOP  <>
  IF
    ( ^str  xt ) EXECUTE
  ELSE
    ( ^str  xt )  DROP  FIND  ( ^str – ^str  false | xt  1 | xt  true )
    DUP
    IF          …..
    ELSE   DROP  ( ^str )  NUMBER?  IF …. ELSE …. THEN
    THEN
  THEN
  DONE?
UNTIL
….
```

# HTML 'Compilieren'

**…. <br><br> <$ @TIME ADD.NUMBER $> ….**

```
:  >BUFFER        ( ^str --   )
\ >BUFFER fischt alle innerhalb einer Colon-Definition
\ gefundenen Wörter vor dem FIND weg und compiliert
\ sie als String in einen Puffer.
\ Es beendet sich selbst, wenn es auf <$ trifft.
DUP  COUNT  S"  <$"  COMPARE  0=
IF  \ das Wort war ein <$
  ( ^str ) DROP  $$>                  \ beende den String
  POSTPONE  ADD.STRING
  ['] NOOP  vEX]  !   \ entfernt sich selbst aus dem Interpreter
ELSE \ baue das Wort in den String ein
  ( ^str )  COUNT  1+  ( addr len )  +$$
THEN
;
```

# HTML 'Compilieren'

```
: HELLO (   –   ) <HTML> <body> <h1> Hello World </h1> <br><br>
<$  @TIME  ADD.NUMBER  $>  </body>  </html>  <$ ;
```

```
:  <HTML>     (   --   )   \ ab diesem Wort beginnt HTML
<$$                         \ initialisiere den Puffer für das HTML
S" <html> " +$$             \ und schreibe ein <html> hinein
[']  >BUFFER  vEX]  !       \ und behandle nachfolgende Wörter als HTML
;  IMMEDIATE

:  $>              (   --   )   \ ab diesem Wort beginnt wieder HTML
<$$                         \ initialisiere den Puffer für das html
[']  >BUFFER  vEX]  !
;  IMMEDIATE
```

# ...und das wird compiliert

Das ist der Quelltext...

```
: HELLO ( – )
<HTML> <body>
<h1> Hello World </h1>
<br><br>
<$ @TIME ADD.NUMBER $>
</body> </html>
<$
;
```

...und das wird compiliert:

```
: HELLO ( – )
S“ <html> <body> <h1> Hello World </h1> <br><br>“ ADD.STRING
@TIME ADD.NUMBER
S“ </body> </html>“ ADD.STRING
;
```

# Strings häppchenweise compilieren

**<$$ ( – )**
**legt einen namenlosen Puffer im Dictionary an und merkt sich den Pufferanfang**

**+$$ ( addr len – )**
**hängt den String ab addr hinten an den Puffer an und aktualisiert den DP**

**$$> compilation: ( – )**
**runtime: ( – buffer size )**
**schliesst den Puffer und compiliert ein DLITERAL mit Pufferanfang und Größe**

# nochmal zu GET

Das sendet der Browser:
**GET / HTTP/1.1**
**User-Agent: Opera/9.80 (Windows.....**

```
: GET      (  –  )
BL WORD ( ^str )            \ URL evtl. mit Parameter
( ^str ) SEPARATE.PARAMETERS ( ^url )
( ^url ) SHOW.PAGE ;

: ANSWER.REQUEST   (  --  )  \ der Req. vom Browser steht im TIB
BL WORD ( ^str )
FIND ( xt -1 | xt 1 | ^str false )
IF \ GET, POST
    ( xt ) EXECUTE
ELSE ….
```

# hier wird endlich die Seite angezeigt

```
: SHOW.PAGE         ( ^str –   )
DUP  COUNT  S“  /“  COMPARE  0=
IF                              \ es war der Slash, zeige die Toppage an
  ( ^str ) DROP  $TOP-PAGE  ( ^str' )
THEN
( ^str )  FIND  (  xt  true | ^str  false )
IF  ( xt )  .RESPONSE-HEADER  ( xt )  EXECUTE
    >RESPONSE  @  START.CONTENT  @  -  INSERT.LEN
    RESPONSEBUFFER  @  >RESPONSE  @  TYPE
ELSE  ( ^str )                  \ es war kein Wort aus dem Dictionary
                                \ versuche es als Dateinamen
    COUNT  R/O  OPEN-FILE  ( fileid  ior ) \  0:ok  1:Fehler
    IF          sende Response-Header und ein 404 Not Found
    ELSE        sende Response-Header und die Datei
    THEN
THEN ;
```

# Formulare in HTML

```
….
<form action="auswahl" method="post" enctype="text/plain">

Vorname: <input name="Vorname" size="20" maxlength="30" ><br>
Nachname: <input name="Nachname" size="20" maxlength="30"
value="bitte ausfuellen">
<br><br>

<input type="submit" value="Absenden">
<input type="reset" value="Abbrechen">

</form>
….
```

Nach Drücken auf 'Absenden' kommt vom Browser:

**POST  /auswahl  HTTP1.1.....Content-Length: 30.....**
**Vorname=Hans  Nachname=Eckes**

# hier wird es hässlich

Das ist das leere Formular und steht auch so im Quelltext:
**… <input name="Vorname" size="20" maxlength="30" >....**
**….<input name="Nachname" value="bitte ausfuellen" size="20" maxlength="30" >....**

Nach dem Ausfüllen des Formulars sendet der Browser:
**Vorname=Hans Nachname=Eckes**

Das sendet der Server an den Browser zurück:
**…. <input name="Vorname" value="Hans" size="20" maxlength="30" >....**
**….<input name="Nachname" value="Eckes" size="20" maxlength="30" >....**

# Die Lösung

**Die <input ……..> Zeile darf nicht als String abgelegt werden.**

**Sie wird beim Compilieren in ihre einzelnen Attribute zerlegt und wird beim Anzeigen der Seite wieder zusammengebaut.**

**Vorteil:**

- **Values können aktualisiert werden**
- **Attribute wie** *checked* **oder** *selected* **können eingefügt werden**
- **Namen sind Forth-Worte, die sich um ihren Value und die Attributes selbst kümmern.**

# 'name' hat noch einen Nebenjob

Beim Anzeigen des Formulars:
**<input  name="Vorname"  value="Hans" size="20" maxlength="30" >**

Beim Übernehmen der Werte von POST:
**Vorname=Hans**

**30  $VARIABLE  $VORNAME**

```
: Vorname          ( ^str  true | <input  false --   )
IF ( ^str )                \ jetzt kommt ein neuer Value von POST
     $VORNAME $!
ELSE ( <input )        \ hier wird die Zeile <input ... > angezeigt
  $VORNAME  C@  0<>  IF  $VORNAME  !VALUE  ELSE  DROP  THEN
THEN
;
```

```
: ASSEMBLE.NAME     ( <input --   )
DUP  0x24  +  @  ( <input  ^name )
FIND ( <input  xt  true | <input  ^name  false )
IF ( <input  xt )
    OVER  FALSE  ROT ( <input  <input  flag  xt )  EXECUTE
    ( <input )
    DUP  0x24  +  @ ( <input  ^name )
THEN
( <input  ^name )
<#      \ baue  ein  name="Vorname" zusammen
    [CHAR]  "  HOLD
    ( <input ^name )  $HOLD
    [CHAR]  "  HOLD
    "  name="  $HOLD
    0.
#>  ( <input  addr  len )  ADD.STRING
( <input )  DUP  ASSEMBLE.VALUE
( <input )  DUP  ASSEMBLE.CHECKED
( <input )  DUP  ASSEMBLE.SELECTED
( <input )  DROP
;
```

# und was ist mit JavaScript?

```
<script
    src="test.js"  type="text/javascript">
</script>

<body background="textur.jpg"


….
<input  name="Calc"  type="button"  ….  onclick="Anzeigen();">
….

</body>
```