



## ***Netzprogrammierung Serverseitige Verarbeitung im Web***

Prof. Dr.-Ing. Robert Tolksdorf  
Freie Universität Berlin  
Institut für Informatik  
Netzbasierte Informationssysteme  
mailto: [tolk@inf.fu-berlin.de](mailto:tolk@inf.fu-berlin.de)  
<http://www.robert-tolksdorf.de>



# Überblick

---

- Common Gateway Interface CGI
- Server Side Includes SSI
- PHP
- Java Server Pages JSP
- Servlets

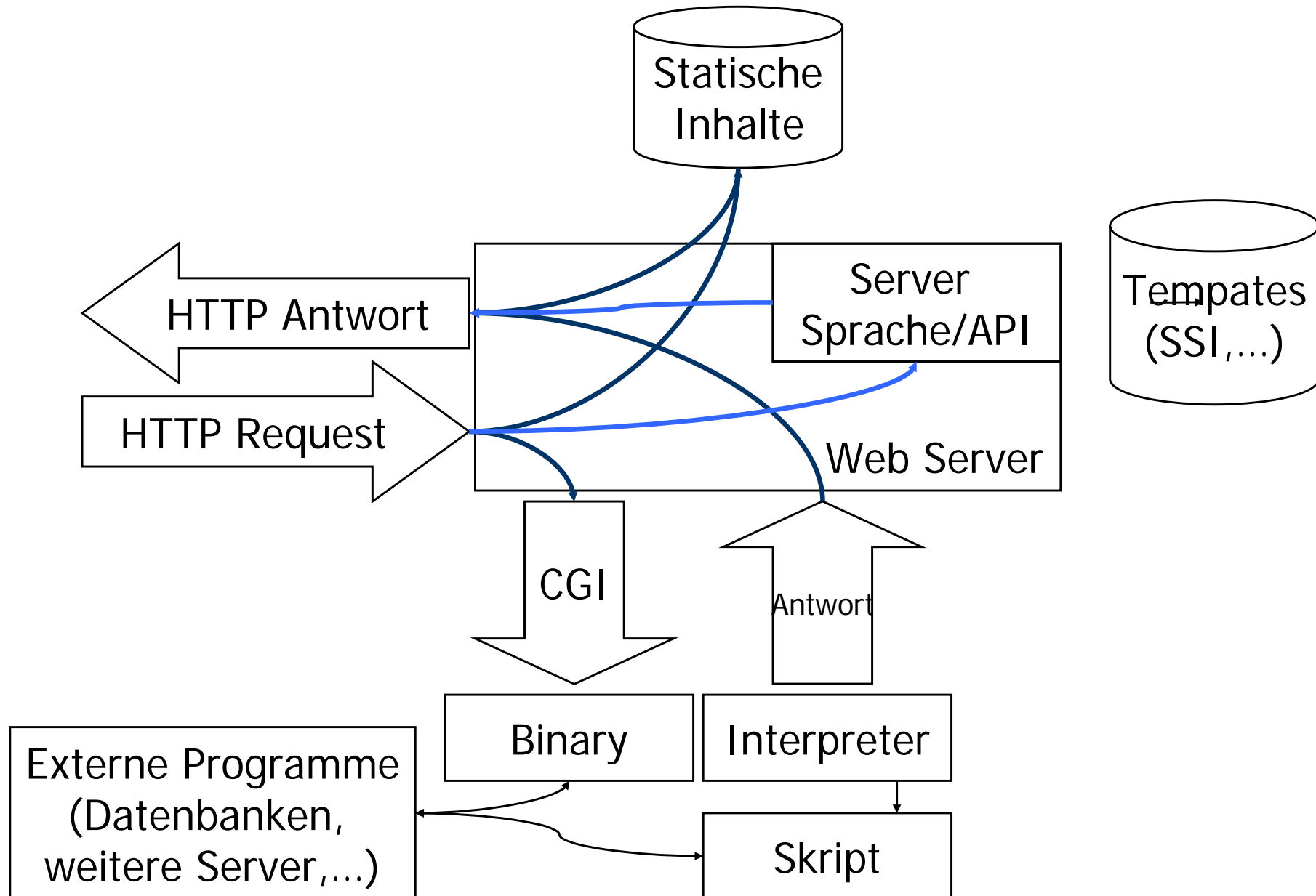


## Dynamische Webinhalte

# Dynamische Web Seiten

---

- Seiten können
  - *Statische Inhalte* liegen vorgefertigt auf dem Server und werden unverändert ausgeliefert und dargestellt
  - *Dynamische Inhalte* bewirken vor und bei der Auslieferung oder der Darstellung Effekte auf Inhalt oder Darstellung durch Ausführung von Programmen
- Aufgaben
  - Serverseitig
    - Vereinfachung der Inhaltserstellung
    - Erzeugung/Konvertierung von Inhalt
    - Anpassung/Personalisierung von Inhalt
    - Ergebnis anderer serverseitiger Transaktionen
  - Clientseitig
    - Erzeugung einer Darstellung
    - Anpassung von Darstellung
    - GUI Interaktion mit Nutzer





## Common Gateway Interface CGI

# Rückblick: Parameter für Web-Skripte

- Zwei Arten der Übermittlung von Parametern an Skripte:
  - GET: Daten werden in URL kodiert
  - POST: Daten werden kodiert über Standardeingabe geliefert

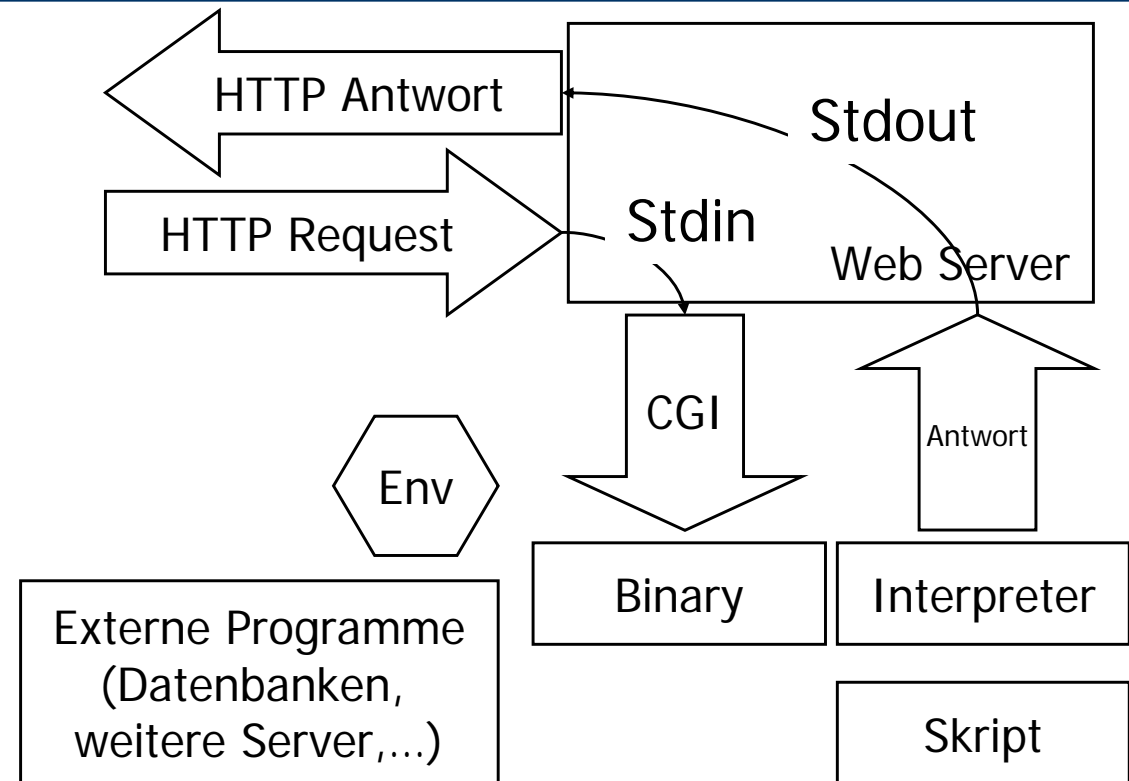
- HTML:

```
<html> <body>  
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi"  
  method="get"> <input name="Eingabe" type="text">  
  <input type="submit" value="Per GET">  
</form>  
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi"  
  method="post">  
  <input name="Eingabe" type="text">  
  <input type="submit"  
    value="Per POST">  
</form>  
</body> </html>
```

- CGI ist eine Schnittstelle zwischen Web-Server und Programm
  - URL bezeichnet Programm das über die CGI-Schnittstelle erzeugt wird und darüber kommuniziert
  - Server startet Programm als Prozess
  - Server übergibt Parameter der Anfrage und anderes in Umgebungsvariablen
  - Programm verarbeitet Anfrage
  - Kann Seiteneffekte haben, teilweise sehr große
  - Programm erzeugt eine HTTP-Antwort (mit HTML-Seite oder anderem)
  - Server leitet Antwort an Klienten weiter
- CGI-Programm ist etwas Ausführbares
  - Interpreter (der ein Skript interpretiert)
  - Compiliertes Programm

# Ablauf

1. Web Server erkennt, dass URL ein Skript bezeichnet
2. Prozess wird gestartet
3. Prozess werden Eingaben mitgeteilt
  1. Initialisierte Umgebungsvariablen
  2. Standardeingabe
4. Standardausgabe des Prozesses wird zum Web-Server gelenkt
5. Skript wird ausgeführt
6. Ausgaben werden an Klienten weitergeleitet



# Einfaches Beispiel

---

- Erzeugung einer einfachen Seite:

```
#!/usr/bin/perl
$temperatur=&messe_temperatur;
print "Content-type: text/html\n";  # Header
print "\n";                          # Leerzeile
print "<html><head>\n";                # Inhalt...
print "<title>Temperatur in Berlin</title>";
print "</head>\n";
print "<body>Temperatur: $temperatur</body>\n";
print "</html>\n";
```

# Umgebungsvariable

- Kommunikation Web-Server und CGI Programm über festen Satz von Umgebungsvariablen

```
#!/usr/bin/perl
print "Content-type: text/html\n";  # Header
print "\n";                          # Leerzeile
print "<html><head>\n";                # Inhalt...
print "<title>Ich erkenne Sie!</title></head>\n";
print "<body>Sie benutzen einen ";
print $ENV{'HTTP_USER_AGENT'};
print "</body>\n</html>\n";
```

- \$ENV ist in Perl ein assoziatives Feld der Umgebungsvariablen

# Umgebungsvariablen (zur Referenz) TODO

---

- **SERVER\_SOFTWARE**  
Server, der das Skript startet, als metaName/Version. Beispiel: NCSA/1.4.2.
- **SERVER\_NAME**  
Der Name des Servers, wie er in der URL des Skripts auftritt.
- **GATEWAY\_INTERFACE**  
Version der CGI-Spezifikation, der der Server folgt, als CGI/Version. Z.B.: CGI/1.1.
- **SERVER\_PROTOCOL**  
Das Protokoll, mit dem die Anfrage geschickt wurde, als Zeichenkette der Form Protokoll/Version. Z.B. HTTP/1.1
- **SERVER\_PORT**  
Die Nummer des (Unix-)Ports, über den die Anfrage geschickt wurde.
- **REQUEST\_METHOD**  
Methode, die bei der Anfrage verwendet wurde z.B. get oder post.
- **PATH\_INFO**  
Enthält den URL-Teil nach der eigentlichen Identifikation des Skripts.
  - CGI-Skript `www.info.berlin/cgi-bin/info`
  - Aufruf über `http://www.info.berlin/cgi-bin/info/tempelhof`
  - In der Umgebungsvariablen: `/tempelhof`
- **PATH\_TRANSLATED**  
PATH\_INFO mit vorangestelltem lokalen Pfadnamen des Dokumentenverzeichnis: `/usr/local/etc/httpd/htdocs/tempelhof`

# Umgebungsvariablen (zur Referenz)

---

- **SCRIPT\_NAME**  
Der Name des Skripts in der URL, z.B. /cgi-bin/info.
- **QUERY\_STRING**  
Die Anfrage
- **REMOTE\_HOST**  
Der Name des Rechners, von dem die Anfrage kam.
- **REMOTE\_ADDR**  
Die Internetadresse des Rechners, von dem die Anfrage kam.
- **REMOTE\_USER**  
Der Benutzername, für den das Passwort eingegeben wurde.
- **REMOTE\_IDENT**  
Nutzernamen des anfordernden Users, falls ermittelbar
- **AUTH\_TYPE**  
Name des Verfahrens, mit dem das Passwort kodiert ist, z.B. Basic
- **CONTENT\_TYPE**  
Bei Verwendung der POST-Methode steht hier der Medientyp des Inhalts, der an der Standardeingabe gelesen werden kann, z.B. application/x-www-form-encoded.
- **CONTENT\_LENGTH**  
Die Länge des Inhalts bei der POST-Methode.
- **HTTP\_ACCEPT**  
Die Medienarten, die der Browser akzeptieren will

# Umgebungsvariablen

- Real je nach Server auch mehr Variablen

```
#!/usr/bin/perl
print "Content-Type: text/plain\n\n";
foreach $var (sort keys %ENV) {
    $val = $ENV{$var};
    $val = ~ s|\n| |g;
    print "$var=\"$val\"\n";
}
```

- Ergibt für <http://www.inf.fu-berlin.de/~tolk/cgi-bin/env.cgi>:  
DOCUMENT\_ROOT="/export/local-1/www/page/htdocs"  
GATEWAY\_INTERFACE="CGI/1.1"  
HTTP\_ACCEPT="text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg, image/gif;q=0.2, \*/\*;q=0.1"  
HTTP\_ACCEPT\_CHARSET="ISO-8859-1,utf-8;q=0.7,\*;q=0.7"  
HTTP\_ACCEPT\_ENCODING="gzip,deflate"  
HTTP\_ACCEPT\_LANGUAGE="de,en-us;q=0.7,en;q=0.3"

# Umgebungsvariablen

```
HTTP_CONNECTION="keep-alive"
HTTP_HOST="page.mi.fu-berlin.de"
HTTP_KEEP_ALIVE="300"
HTTP_USER_AGENT="Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
  rv:1.4) Gecko/20030624"
PATH="/usr/local/bin:/usr/bin:/bin"
QUERY_STRING=""
REMOTE_ADDR="160.45.114.204"
REMOTE_PORT="1559"
REQUEST_METHOD="GET"
REQUEST_URI="/~tolk/cgi-bin/env.cgi"
SCRIPT_FILENAME="/export/local-1/www/userpages/tolk/
  public_html/cgi-bin/env.cgi"
SCRIPT_NAME="/~tolk/cgi-bin/env.cgi"
SERVER_ADDR="160.45.117.11"
SERVER_ADMIN="webmaster@inf.fu-berlin.de"
SERVER_NAME="page.mi.fu-berlin.de"
SERVER_PORT="80"
SERVER_PROTOCOL="HTTP/1.1"
SERVER_SOFTWARE="Apache/1.3.33 Ben-SSL/1.55 (Debian GNU/Linux)
  PHP/4.3.10-2 mod_perl/1.29"
UNIQUE_ID="QgeKzaAtdQsAABj1HC4"
```

# Standardeingabe

---

- Auf der Standardeingabe steht der Inhalt der Anfrage-Mitteilung an
- GET/POST Unterscheidung, wo der Query-String erhältlich ist:

```
if ($ENV{'REQUEST_METHOD'} eq 'POST') {  
    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});  
} elsif ($ENV{'REQUEST_METHOD'} eq 'GET') {  
    $buffer=$ENV{'QUERY_STRING'};  
}
```

# Standardausgabe

- Ausgaben des CGI-Skripts werden als Antwort an den Klienten geschickt
- Notwendig: Content-Type-Header
- Möglich: Beliebige Medienarten als Antwort, z.B. auch Bilder

```
#!/usr/bin/perl
print("Content-Type: application/postscript\n\n");
print <<EO_POSTSCRIPT
%!PS-Adobe-3.0
%% see http://www.ntecs.de/old-hp/uu9r/lang/html/postscript.de.html
/Courier findfont
28 scalefont
setfont
0 0 moveto
(Hallo!) show
showpage
EO_POSTSCRIPT
```

# Standardausgabe

- Setzen des Statuscodes:  

```
#!/usr/bin/perl  
print "Content-Type: text/plain\n";  
print "Status: 405 Not Allowed\n\n";
```
- Ergebnis:  

```
>head http://www.inf.fu-berlin.de/~tolk/cgi-bin/405.cgi  
head http://www.inf.fu-berlin.de/~tolk/cgi-bin/405.cgi  
405 Not Allowed  
Connection: close  
Date: Mon, 07 Feb 2005 16:22:26 GMT  
Server: Apache/1.3.33 Ben-SSL/1.55 (Debian GNU/Linux)  
      PHP/4.3.10-2 mod_perl/1.29  
Content-Type: text/plain; charset=iso-8859-1
```
- Umleitung  

```
#!/usr/bin/perl  
@uni= ("http://www.fu-berlin.de",  
      "http://www.tu-berlin.de",  
      "http://www.hu-berlin.de");  
print("Location: $uni[rand(3)]\n\n");
```

# CGI bei inf.fu-berlin.de

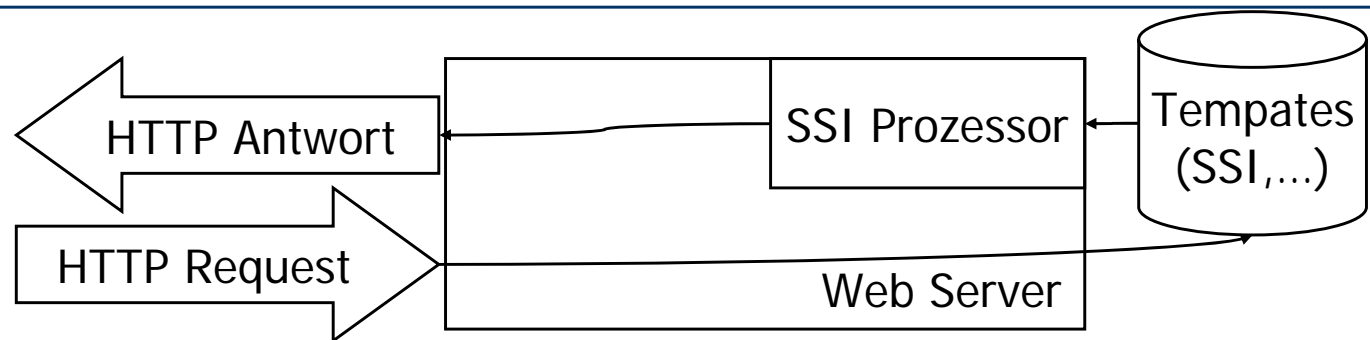
- Ihre Homepage hat die URL  
`http://www.inf.fu-berlin.de/~<nutzer>`
- Im Dateisystem bei
  - UNIX: `/web/page.mi.fu-berlin.de/web-home/<nutzer>`
  - MS-Windows-Rechner  
`\\web.mi.fu-berlin.de\page.mi.fu-berlin.de\<nutzer>`
- Darin gibt es zwei Verzeichnisse
  - `.../<nutzer>/data-rw`
    - Wird nicht nach außen geleitet
    - Skripte können darin schreiben
  - `.../<nutzer>/public_html`
    - Wird nach außen geleitet, ist Ihr Webspaces
    - Skripte können nicht darin schreiben
- `/web/page.mi.fu-berlin.de/web-home/<nutzer>/public_html/index.html` ist Ihre Homepage
- Details:  
`http://projects.mi.fu-berlin.de/w/bin/view/Tec/UserPages`



## Server Side Includes SSI

# Server Side Includes

- Server Side Includes (SSI) sind Anweisungen an den Web Server, die im HTML Code eingebunden sind



- Werden vom Server bei Auslieferung ausgeführt und zu HTML Code expandiert
- SSI Anweisung im statischen HTML-Code:  
`<div align="right" >`  
Letzte Änderung: `<!--#flastmod file=""-->`  
`</div >`
- Ergebnis nach Auslieferung  
`<div align="right" >`  
Letzte Änderung: `Wednesday, 08-Jan-2003 09:30:15 CET.`  
`</div >`

# SSI Anweisungen

- Als HTML-Kommentar eingebettet:  
`<!--#direktive Attribut1="Wert1" Attribut2="Wert2" ...-->`
- Direktiven:
  - `lastmod`: Fügt Änderungsdatum der im Attribut file genannten Datei ein
  - `filesize`: Fügt Größe der im Attribut file genannten Datei ein
  - `include`: Fügt Datei ein. Bezeichnung je nach Attribut:
    - `virtual`: Pfad im Web-Verzeichnisbaum  
`<!--#include virtual="/inst/ag-nbi/include/nbiheader" -->`
    - `file`: Pfad im (nur „unter“ aktueller Datei)
  - `exec`: Führt Kommando oder
    - `cgi`: Pfad im Web-Verzeichnisbaum (Apache 2.0: include virtual)
    - `cmd`: Ausführung im Datei-Verzeichnisbaum  

```
<pre>  
<!--#exec cmd="/usr/bin/lis -l"-->  
</pre>
```

- **echo**: Gibt Inhalt der im Attribut var bezeichneten Variablen aus
- SSI-Variablen
  - **DOCUMENT\_NAME**: Dateiname des Dokuments im Datei-Verzeichnisbaum
  - **DOCUMENT\_URI**: Dateiname des Dokuments im Web-Verzeichnisbaum
  - **QUERY\_STRING\_UNESCAPED**: Suchanfrage
  - **DATE\_LOCAL**: Aktuelles lokales Datum
  - **DATE\_GMT**: Aktuelles lokales Datum als GMT
  - **LAST\_MODIFIED**: Veränderungsdatum
- Beispiel:  
Hier ist es `<!--#echo var="DATE_LOCAL"-->`.

- CGI-Variablen

|                 |                 |                   |
|-----------------|-----------------|-------------------|
| SERVER_SOFTWARE | SERVER_NAME     | GATEWAY_INTERFACE |
| SERVER_PROTOCOL | SERVER_PORT     | REQUEST_METHOD    |
| PATH_INFO       | PATH_TRANSLATED | SCRIPT_NAME       |
| QUERY_STRING    | REMOTE_HOST     | REMOTE_ADDR       |
| AUTH_TYPE       | REMOTE_USER     | REMOTE_IDENT      |
| CONTENT_TYPE    | CONTENT_LENGTH  |                   |

- HTTP-Header der Form `HTTP_Headername`:
  - `HTTP_ACCEPT`, `HTTP_USER_AGENT`, ...

- Beispiel:

Wie geht es Ihnen bei

```
<!--#echo var="REMOTE_ADDR"--> mit Ihrem
```

```
<!--#echo var="HTTP_USER_AGENT"-->?
```

- **set**: Setzt den Inhalt der im Attribut var bezeichneten Variablen auf den im Attribut value angegebenen Wert
- Einfachste Ausdrücke bildbar

```
<!--#set var="salutation" value="Guten Tag"-->  
<!--#set var="remote"  
  value="{REMOTE_ADDR} ({REMOTE_HOST})"-->
```

```
<!--#echo var="salutation"--> at  
<!--#echo var="remote"-->!
```

- Bedingte Abschnitte
  - <!--#if expr="Bedingung" -->
  - <!--#elif expr="Bedingung" -->
  - <!--#else -->
  - <!--#endif -->
- Bedingungen und Ausdrücke
  - String
  - String<sub>1</sub> != String<sub>2</sub>
  - String<sub>1</sub> < String<sub>2</sub>
  - String<sub>1</sub> <= String<sub>2</sub>
  - String<sub>1</sub> > String<sub>2</sub>
  - String<sub>1</sub> >= String<sub>2</sub>
  - ( Bedingung )
  - ! Bedingung
  - Bedingung<sub>1</sub> && Bedingung<sub>2</sub>
  - Bedingung<sub>1</sub> || Bedingung<sub>2</sub>



## PHP: Hypertext Preprocessor PHP

- SSI sind nicht wirklich eingebettete Programme
- PHP: HTML um „Skriptsprache“ erweitern die serverseitig ausgeführt wird und dessen Ergebnis in den HTML-Text eingebettet ist
- Einbettung im HTML-Code:

```
<html>  
<head><title>Das Einführungsbeispiel schlechthin  
...</title></head>  
<body>  
<?php echo "Brave New PHP World"; ?>  
</body>  
</html>
```
- Auch `<script language="php">...</script>` möglich

# PHP Elemente

- Schwach getypt: Boolean, Integer, Float, String, Array, Object, Resource, NULL
- Typ eines Werts ermittelbar: `is_int($int)`
- Variablen mit \$ gekennzeichnet: `$foo = 25;`
- Ausdrücke mit Operatoren gebildet
  - `., +, -, *, /, %, &, |, ~`
  - `+=, .=, -=, *=, /=, %=, &=, |=`
  - `<, <=, ==, ===, =>, $gt;, !=`
  - `$v++, ++$v, $v--, --$v;`
  - `&&, and, ||, or`
- Kontrollstrukturen
  - `if, else, elseif, switch`
  - `while, do..while, for, foreach, break, continue`

# PHP Elemente (zur Referenz)

- `$_SERVER` Feld enthält „übliche“ CGI-Variablen:
  - `DOCUMENT_ROOT` c:/programme/web/server/easyphp1-8/www
  - `HTTP_ACCEPT` text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5
  - `HTTP_ACCEPT_CHARSET` ISO-8859-1,utf-8;q=0.7,\*;q=0.7
  - `HTTP_ACCEPT_ENCODING` gzip,deflate
  - `HTTP_ACCEPT_LANGUAGE` de-de,en-us;q=0.7,en;q=0.3
  - `HTTP_CONNECTION` keep-alive
  - `HTTP_COOKIE` PHPSESSID=68017f63c4a74f81961eba78535eeec4
  - `HTTP_HOST` localhost
  - `HTTP_KEEP_ALIVE` 300
  - `HTTP_USER_AGENT` Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8) Gecko/20051111 Firefox/1.5
  - `REMOTE_ADDR` 127.0.0.1
  - `REMOTE_PORT` 1579
  - `SCRIPT_FILENAME` c:/programme/web/server/easyphp1-8/www/web-technologien/k\_13\_3\_2/server.php
  - `SERVER_ADDR` 127.0.0.1
  - etc.

# Beispiel

- Konfigurieren, welche Eingabefelder obligatorisch sind und Funktion zur Validierung definieren
- Danach mit HTML-Code beginnen

```
<?
```

```
// === Konfiguration ===
```

```
$obligatList = array( "user", "password" );
```

```
// === Funktionen ===
```

```
function validate( $data, $obligatList) {
```

```
    $faultList = array();
```

```
    foreach( $obligatList as $angabe ) {
```

```
        if( !isset( $data[$angabe] ) || $data[$angabe] == "" ) {
```

```
            $faultList[] = $angabe;
```

```
        }
```

```
    }
```

```
    return $faultList;
```

```
}
```

```
?>
```

```
<html>
```

```
<head><title>Anmeldung mit Validierung</title></head>
```

# Beispiel

- Bei Antwortgenerierung auf Submit: Wurden alle Eingabefelder ausgefüllt?
- Nein -> Formular erneut erzeugen, Teileingaben einfügen

```
<body>
```

```
<?
```

```
  $faultList = validate( $_GET, $obligatList );
```

```
  if( count( $faultList ) > 0 ) {
```

```
?>
```

```
<h3>Anmeldung</h3><br>
```

```
<form method="get">
```

```
<table border="0" cellspacing=0 cellpadding=0>
```

```
<tr>
```

```
<td><b>Benutzername: </b>&nbsp;&nbsp;&nbsp;</td>
```

```
<td><input type="text" name="user" size=12 maxlength=50  
value="<?=$_GET['user']?>"></td>
```

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

```
<td><b>Passwort: </b>&nbsp;&nbsp;&nbsp;</td>
```

```
<td><input type="password" name="password" size=12 maxlength=50  
value="<?=$_GET['password']?>"></td>
```

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

```
<td>&nbsp;</td><td><input type="submit" value="Anmelden"></td>  
</tr> </table></form><br>
```

# Beispiel

- Ja: Eingaben in einer Tabelle ausgeben

```
<?
  }
  else {
?>
<h3>Ihre Anmeldedaten</h3>
<br>
<table>
  <tr><td>Benutzername: </td><td><?=$_GET["user"]?></td></tr>
  <tr><td>Passwort: </td><td><?=$_GET["password"]?></td></tr>
</table>
<?
  }
?>
</body>
</html>
```

# Vergleich

---

- CGI
  - Externe Programme
  - Programme erzeugen HTML
- SSI
  - Server-Makros
  - Web-Server erweitert HTML
- PHP
  - Skript in HTML integriert
  - Web-Server führt Script aus



## Java Servlets

- Servlets:
  - CGI Programme könnten auch in Java geschrieben werden
  - Als „Interpreter“ müsste eine JVM gestartet werden
  - Wenn der Webserver selber auf einer JVM läuft, könnte er eine „CGI-Komponente“ nachladen und ausführen
  - Java Servlets sind solche Komponenten
- Unterschiede zu CGI und Applets
  - Parameterkommunikation läuft nicht über Umgebung und stdin/stdout sondern über Java-Schnittstelle
  - Applets als Komponente in JVM eines Browsers geladen
  - Servlets als Komponente in JVM eines Servers geladen
- Schnittstelle:
  - Erweitern von [HttpServlet](#)
  - Überschreiben von [doGet](#) und [doPost](#)

# Beispiel

```
// HelloServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
    public void doGet( HttpServletRequest req, HttpServletResponse res )
        throws ServletException, IOException {
        res.setContentType( "text/html" );
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head><title>HelloServlet</title></head>");
        out.println("<body>");
        out.println("<big>Hello</big>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

# Eingaben

---

- Über das `HttpServletRequest` Objekt kann man die auch in CGI vorhandenen Informationen erfragen:
  - `getServerName()`
  - `getServerPort()`
  - `getRemoteAddr()`
  - `getRemoteHost()`
  - `getQueryString()`
  - `getMethod()`
  - `getServletPath()`
- Header
  - `getHeaderNames()`
  - `getHeader(String name)`
- Query-String
  - `getParameter(String name)`
  - `getParameterValues(String name)`

```
// AnmeldenServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AnmeldenServlet extends HttpServlet {
    public void doGet( HttpServletRequest req, HttpServletResponse res )
        throws ServletException, IOException {
        res.setContentType( "text/html" );
        PrintWriter out = res.getWriter();
        String user = req.getParameter( "user" );
        String password = req.getParameter( "password" );
        out.println("<html><head><title>Ihre
Anmeldedaten</title></head><body>");
        out.println("<h3>Ihre Anmeldedaten</h3><br><table>");
        out.println(" <tr><td>Benutzername: </td><td>" + user + "</td></tr>");
        out.println(" <tr><td>Passwort: </td><td>" + password + "</td></tr>");
        out.println("</table></body></html>");
    }
}
```

# Ausgaben

- Statuscode und Header sind setzbar
  - setStatus(int no)
  - setHeader(String name, String value)
- Konfiguration von Servlets (z.B. Bindung eines Servlet-Objekts an ein URL):  
Siehe entsprechenden Webserver
  - Apache tomcat: <http://tomcat.apache.org/>

# Vergleich CGI / Servlets

---

- Portabilität
  - CGI führt weitere Sprache ein, oft extra Bibliothek nötig
  - Servlets sind innerhalb des Java Rahmens und sind einfacher strukturiert
- Leistungspotential
  - CGI ist eine Teillösung für Server Skripte
  - Servlets haben Anschluss an die gesamte Java Welt, einschließlich Sicherheitsarchitektur
- Effizient
  - CGI's werden jedes Mal als Prozess gestartet (Ausnahme: Fast CGI Mechanismus)
  - Servlets bleiben in JVM geladen



## Java Server Pages

- Java Server Pages
  - In SSI gibt es rudimentäre Ausdrücke
  - Man könnte auch komplexere Programmfragmente mit HTML-Code mischen, wie bei PHP
  - Java Server Pages erlauben die Mischung von HTML-Code mit Java Fragmenten
  - Aus ihnen werden automatisch Servlets generiert und ausgeführt
    - HTML-Code nach Ausgabe schreiben
    - Java-Fragmente in Servlet Rahmen einbetten
- JSP bestehen aus
  - Scriptlets
  - JSP Ausdrücken
  - Deklarationen
  - JSP Anweisungen
  - HTML

# Beispiel für Scriptlets und JSP Ausdrücke

```
<html>
<head> <title>Anmeldung</title> </head>
<body>
<%
String us = request.getParameter( "user" );
String pw = request.getParameter( "password" );

if( us == null || "".equals( us ) || pw == null || "".equals( pw ) ) {
%>
<h3>Anmeldung</h3>
<br>
<form method="get">
  <table border="0" cellspacing=0 cellpadding=0>
    <tr>
      <td><b>Benutzername: </b>&nbsp;&nbsp;&nbsp;</td>
      <td><input type="text" name="user" size=12
        maxlength=50 value="<%out.print( us != null ? us : "" );%>"></td>
    </tr>
```

# Beispiel für Scriptlets und JSP Ausdrücke

```
<tr>
  <td><b>Passwort: </b>&nbsp;&nbsp;&nbsp;</td>
  <td><input type="password" name="password" size=12
    maxlength=50 value="<%out.print( pw != null ? pw : "" );%>"></td>
</tr>
<tr>
  <td>&nbsp;</td>
  <td><input type="submit" value="Anmelden"></td>
</tr>
</table>
</form>
<br>
<%
  }
  else {
%>
```

# Beispiel für Scriptlets und JSP Ausdrücke

```
<h3>Ihre Anmeldedaten</h3>
<br>
<table>
  <tr><td>Benutzername: </td><td> <%=request.getParameter( "user" )%>
    </td></tr>
  <tr><td>Passwort: </td><td> <%=request.getParameter( "password" )%>
    </td></tr>
</table>
<%
  }
%>
</body>
</html>
```

# Beispiel Deklarationen

```
<html>
<head><title>Anmeldung</title></head>
<body>
<%!
private String[] obligatList = { "user", "password" };
private boolean validate( HttpServletRequest request ) {
    for( int i = 0; i < obligatList.length; i++ ) {
        String param = request.getParameter( obligatList[i] );
        if( param == null || "".equals( param ) )
            return false;
    }
    return true;
}
%>
boolean valid = validate( request );
if( !valid ) {
%>
<h3>Anmeldung</h3>
<br><form method="get"> <table border="0" cellspacing=0 cellpadding=0>...
```

# Beispiel JSP Anweisungen

---

```
<html>
<head><title>Include.jsp</title></head>
<body>
<h2>Include.jsp</h2>
<p>Hier steht der spezifische Seiteninhalt.
<br><br><br><br><br><br><br><br><br>
<%@ include file="foot.jsp" %>
</body>
</html>
```

# Scriptlets und JSP Ausdrücke

---

- Scriptlets
  - Syntax `<% Java code %>`
  - Beliebiger Java Code, wird in generierte Methode `_jspService()` entsprechend incompiliert
- JSP Ausdrücke
  - Syntax `<%= Java Ausdruck %>`
  - Wird zur Laufzeit evaluiert
  - Ergebnis wird als Zeichenkette in die Ausgabe geschrieben
- Deklarationen
  - Syntax `<%! Java Code %>`
  - Wird in Klasse incompiliert
- JSP Anweisungen
  - Werden bei Übersetzungszeit interpretiert

# Vergleich SSI / JSP

---

- Ausführungsort
  - PHP beim Server interpretiert oder on-the-fly kompiliert
  - JSP zu Servlet kompiliert und in JVM ausgeführt
- Ausdrucksfähigkeit
  - PHP im PHP-Rahmen
  - JSP volle Java-Mächtigkeit
- Erstellung
  - Beide sind eigentlich HTML Erweiterungen
  - JSP ist aber eigentlich Programmierung



## Zusammenfassung

- CGI
  - Verarbeitung von Eingabe
  - Beliebige Sprache
  - Kommunikation vom Server-Prozess mit Umgebungsvariablen, stdin/ stdout
- SSI
  - Vom Web-Server evaluierte zusätzliche Tags
  - „Makros“
- PHP
  - Vom Web-Server evaluierte zusätzliche Script-Abschnitte
  - „Skriptsprache“
- Servlets
  - Rahmenwerk für „CGI-Komponenten“ in Java
  - Kommunikation über Schnittstellen
- JSP
  - Mischung von HTML- und Java-Code
  - Compiliert zu Servlets

# Referenzen

---

- CGI Specification. <http://hoohoo.ncsa.uiuc.edu/cgi/>
- NCSA HTTPd Development Team. *Server Side Includes (SSI)*. <http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html>
- Apache HTTP Server Documentation Project. *Apache Tutorial: Introduction to Server Side Includes*.  
<http://httpd.apache.org/docs/howto/ssi.html>  
<http://httpd.apache.org/docs-2.0/howto/ssi.html>
- Diverse Autoren. PHP Handbuch.  
<http://www.php.net/manual/de/>
- Sun Microsystems, Inc. *JavaServer Pages*.  
<http://java.sun.com/products/jsp/>
- Heiko Wöhr. Web-Technologien, Konzepte - Programmiermodelle – Architekturen. dpunkt.verlag 2004.