Anhang C

PHP Beispiel

C.1 Grundlagen

Eine weit verbreitetet Sprache zur dynamischen Erstellung von Webseiten ist PHP. Es handelt sich dabei um eine Skript-Sprache, die in der Syntax stark an Perl angelehnt ist, aber wesentlich leichter zu erlernen ist. Ein PHP-Skript wird serverseitig interpretiert und die Ausgabe - in der Regel HTML-Code - wird an den Client geschickt. Im folgenden werden an einem Beispiel die Grundprinizipien dargestellt. Das übliche Einsteiger-Beispiel zeigt Bild C.1. Die Datei enthält ein HTML-Gerüst. Darin eingebettet ist ein PHP-Tag, der lediglich mit dem Befehl `echo` eine Zeichenkette ausgibt. Gibt man in einem Browser die Adresse (beispielsweise `http://localhost/hallo.php`) ein, so wird im Server der PHP-Teil ausgeführt, das Ergebnis mit dem umgebenden HTML-Text kombiniert und an den Client zurückgegeben. Man kann den HTML-Teil auch weglassen. Eine Datei mit dem Inhalt

```
<?php echo '<p>Hallo Friedberg</p>'; ?>
```

führt zum gleichen Ergebnis - allerdings nur mit der Pfadangabe als Seiten-Titel. PHP unterstützt die Eingabe von Werten über Formulare.

```
<html>
 <head>
  <title>PHP-Test</title>
 </head>
 <body>
 <?php echo '<p>Hallo Friedberg</p>'; ?>
</body>
</html>
```

Abbildung C.1: Einfache Begrüßung mit PHP

```
<html>
<head><title>Springer Spiel</title></head>
<body>
<h1>Eingabe einer Position</h1>
<form action="springer_save.php" method="GET">
  H&ouml;he: <input type="text" name="hoehe" /><br />
  Breite: <input type="text" name="breite" /><br />
  Datei: <input type="text" name="datei" /><br />
  <input type="submit" value="GO" />
</form>
</body>
</html>
```

Abbildung C.2: PHP Formular

C.2 Springerspiel

Wir wollen als Beispiel wieder die Erstellung einer Sicherungsdatei für das Springerspiel betrachten. Zur Vereinfachung wird nur die Höhe und Breite des Spielfeldes sowie optional der Name der Datei behandelt. In C.2 ist eine entsprechende PHP-Datei wiedergegeben. Mit

```
<form action="springer_save.php" method="GET">
```

wird ein Formular definiert. Die Auswertung übernimmt der als Attribut `action` eingetragene Skript `springer_save.php` und die Parameter werden mit dem HTTP-Befehl GET als Anhang der Adresse übertragen. Für die Eingabe sind drei Textfelder vorgesehen. Dabei ist jeweils noch der Name angegeben, unter dem später auf Informationen zugegriffen wird. Schließlich wird mit

```
<input type="submit" value="GO" />
```

ein Schalter zum Übermitteln der Formulardaten angelegt. Bild C.3 zeigt die resultierende Darstellung im Browser. Betätigt man den Übergabeschalter, so wird das angegebene Skript aufgerufen. Die Parameter werden als Attribut-Wert-Paare angehängt. Im Beispiel resultiert die neue Adresse

```
http://localhost/springer_save.php?hoehe=20&breite=11&datei=
```

Da kein Name für die Datei eingetragen wurde, bleibt dieses Attribut leer. Das aufgerufene Skript kann über ein Feld namens `$_GET` mit Assoziativadressierung auf die Übergabewerte zugreifen. Intern enthält das Feld die Attribut-Wert-Paare. Man kann direkt den Namen eines Attributes als Index für das Feld verwenden. Mit beispielsweise `$_GET['hoehe']` erhält man den Wert des Attributes `hoehe`.

Abbildung C.3: PHP Form

Der Name `hoehe` dient als Schlüssel für den Zugriff auf den dazugehörenden Wert. Diese Art der Adressierung ist in PHP sehr gebräuchlich und wird durch die Sprache gut unterstützt. So kann man alle Schlüssel-Wert-Paare in einem Feld durch eine spezielle foreach-Schleife in der Form

```
foreach ($_GET as $key=>$value) {
  echo $key." = " . $value . "</br>";
}
```

abarbeiten. Im Skript werden zur Vereinfachung zu Beginn die Werte in eigene Variablen kopiert. Der entsprechende Programmabschnitt ist:

```
$hoehe=$_GET['hoehe'];
$breite=$_GET['breite'];
$datei=trim( $_GET['datei'] );
```

Bei dem Dateinamen werden vorsorglich mit der Funktion `trim` eventuelle Leerzeichen entfernt. Ist kein Dateiname angegeben, so wird mit

```
if ( $datei == "" ) {
   $datei= "springer.xml";
}
```

ein Standardwert eingesetzt. Mit den übergebenen Werten soll eine neue Springerdatei generiert werden. Wir greifen dazu wieder auf das DOM-API zurück. Mit den folgenden Befehlen

```
$doc = new DOMDocument('1.0', 'ISO-8859-1');
$doc->formatOutput = true;
```

```
$doc->appendChild($doc->createProcessingInstruction(
   'xml-stylesheet', 'href="springer.css" type="text/css"'));
$doc->appendChild($doc->createComment(
   'Springerdatei aus Web-Eingabe') );
```

wird

- Ein neues Dokument mit der Versionskennung 1.0 und der Zeichenkodierung ISO-8859-1 angelegt
- Die Ausgabeformatierung eingeschaltet
- Ein Verweis auf eine CSS-Datei und ein Kommentar eingetragen

Dann wird ein Wurzelknoten mit

```
$root = $doc->createElement('Springerspiel');
$doc->appendChild($root);
```

angelegt. Die einzelnen Elemente werden dann mit weiteren Aufrufen der Methode `createElement` erzeugt. Die Inhalte werden durch Textknoten an die Elementknoten angehängt. Für die Spielfeldbreite lautet entsprechender Code

```
$breiteNode = $doc->createElement('Breite');
$breiteNode->appendChild( $doc->createTextNode( $breite ) );
$root->appendChild( $breiteNode );
```

Nach diesem Muster wird der DOM-Baum im Speicher aufgebaut. Die Speicherung in eine Datei übernimmt die Methode `save`:

```
$doc->save($datei);
```

Das vollständige Skript ist in Bild C.4 wiedergegeben. Im Browser ergibt sich die Darstellung C.5 mit einer kurzen Meldung und dem Verweis auf die neue Springerdatei.

C.3 GET oder POST?

Die Übermittlung der Parameter per GET hat in der vorliegenden Anwendung einen kleinen Nachteil. Sobald die URL mit den angehängten Werten aufgerufen wird, startet die Ausführung des Skriptes. Damit führt beispielsweise auch eine Aktualisierung im Browser dazu, dass die Springerdatei neu geschrieben wird. Dieses Verhalten ist dem Benutzer nicht unbedingt klar. Besser und transparenter wäre es, wenn nur auf ausdrücklichen Wunsch - Betätigen des entsprechenden Schalters - die Datei generiert wird. Um dies zu erreichen, muss man den Befehl POST zur Parameterübergabe verwenden. Dazu sind nur geringfügige Änderungen notwendig. In dem Formular wird dazu das Attribut `method` geändert:

```
<html>
 <head> <title>PHP-Test</title> </head>
 <body>
 <?php
$hoehe=$_GET['hoehe'];
$breite=$_GET['breite'];
$datei=trim( $_GET['datei'] );
if ( $datei == "" ) {
   $datei= "springer.xml";
}

echo '<p>Neues Springer-Spiel in Datei <a href="'. $datei .'">'
. $datei . '</a> gespeichert</p>';

$doc = new DOMDocument('1.0', 'ISO-8859-1');
$doc->formatOutput = true;
$doc->appendChild($doc->createProcessingInstruction(
   'xml-stylesheet', 'href="springer.css" type="text/css"'));
$doc->appendChild($doc->createComment( 'Springerdatei aus Web-Eingabe') );

$root = $doc->createElement('Springerspiel');
$doc->appendChild($root);

$breiteNode = $doc->createElement('Breite');
$breiteNode->appendChild( $doc->createTextNode( $breite ) );
$root->appendChild( $breiteNode );

$hoeheNode = $doc->createElement('Hoehe');
$hoeheNode->appendChild( $doc->createTextNode( $hoehe ) );
$root->appendChild( $hoeheNode );

$doc->save($datei);
?>
</body>
</html>
```

Abbildung C.4: PHP Generierung einer neuen Springer-Datei

Abbildung C.5: Darstellung nach Ausführung des Skriptes `springer_save.php`.

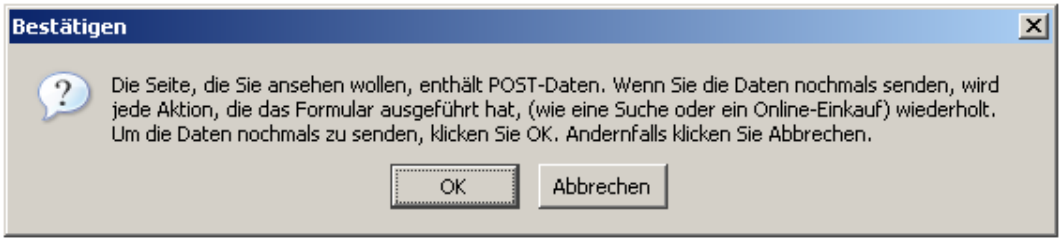

Abbildung C.6: Abfrage bei Aktualisierung von POST-Daten

```
<form action="springer_save.php" method="POST">
```

Die übergebenen Werte stehen dann im Feld `$_POST` und können von dort nach dem gleichen Prinzip geholt werden. Ansonsten bleibt der Ablauf unverändert. Bei der Aktualisierung der Seite erhält der Anwender jetzt allerdings einen Hinweis (Bild C.3).

Allgemein gilt die Regel, dass man GET nur verwenden soll, wenn mit der Ausführung keine Seiteneffekte verbunden sind. So sind Abfragen unkritisch. Man kann auch problemlos eine URL inklusive der Daten weitergeben oder als Lesezeichen aufheben. Sobald aber mit einem Skript Änderungen an Daten oder neue Einträge in eine Datenbank veranlasst werden, sollte man POST verwenden. Eine weiteres Kriterium ist der Umfang der Daten. Bei GET gilt eine Beschränkung von 256 Zeichen. Für größere Mengen muss man auf POST wechseln.

C.4 Cookies

Wie in Abschnitt 11.3 bereits erwähnt, kann man in so genannten Cookies Informationen dauerhaft auf dem Client-System speichern. PHP unterstützt diese Möglichkeit durch entsprechende Methoden und ein Feld `$_COOKIE$`. Der Mechanismus soll anhand eines Zähler dargestellt werden. Bild C.7 zeigt den entsprechenden PHP-Codeabschnitt im Kopf der HTML-Seite. Dabei wird geprüft, ob im Feld `$_COOKIE$` bereits ein Wert für `zaehler` enthalten ist. Falls ja, wird der Wert erhöht, ansonsten mit Eins initialisiert. Dann wird mit der Methode

```
 <head>
<title>PHP-Test</title>
 <?php
if( $_COOKIE['zaehler'] ) {
  ++$_COOKIE['zaehler'];
 } else {
  $_COOKIE['zaehler'] = 1;
 }
  setcookie( "zaehler", $_COOKIE['zaehler'] );
?>
</head>
```

Abbildung C.7: Zähler in einem Cookie

`setcookie` der Wert in das Cookie geschrieben. Der aktuelle Wert kann in der Form

```
echo '<p>'. $_COOKIE['zaehler']
. '. Springer-Spiel in Datei <a href="'. $datei .'">'
. $datei . '</a> gespeichert</p>';
```

als Zähler ausgegeben werden.