1 Sprachelemente und Syntax

In diesem Kapitel werden die wesentlichen Bestandteile und Strukturen der PHP-Programmierung vorgestellt. Da diese in allen Programmiersprachen in ähnlicher Form vorhanden sind, können Leser mit ausreichender Programmiererfahrung das Kapitel gerne überspringen, ohne den roten Faden zu verlieren. Dies ist aber nicht anzuraten, da doch immer wieder mit kleineren und größeren Abweichungen zwischen Programmiersprachen zu rechnen ist. Selbst erfahrenen Programmierern wird empfohlen, das Kapitel zumindest zu überfliegen. Für Leser ohne Programmiererfahrung ist es ohnehin eines der zentralen Kapitel des Buchs. Denken Sie jedoch daran, dass wir bewusst darauf verzichtet haben, in diesem Buch eine ausführliche Einführung zu geben, schließlich wollen wir Sie möglichst zielstrebig an die besonders schwierigen Themen der Programmierung heranführen und nicht die Online-Referenz nachahmen.

1.1 Codekonventionen

Sie sollten den folgenden Abschnitt nicht als Vorschriftensammlung sehen! Vielmehr sind die enthaltenen Informationen als Empfehlungen aufzufassen!

1.1.1 Was sind Codekonventionen?

Codekonventionen sind aus Sicht des PHP-Interpreters unverbindliche Vorschriften oder besser noch Empfehlungen zur Schreibweise von Variablen, Funktionen, Objekten usw. Grundsätzlich müssen die Namen nur den Anforderungen der Semantik genügen. Für die Lesbarkeit und Fehlersuche sind aber erweiterte Benennungsregeln empfehlenswert.

Neben den vielen Vorschriften, die eine Programmiersprache ausmacht, gibt es auch eine Reihe von Empfehlungen, die Programmierern helfen, gut lesbare und optimierte Programme zu schreiben. Die Codekonventionen sind solche Empfehlungen. Dabei geht es auch um die Pflege der Programme durch andere Programmierer oder später durch Sie selbst.

Hinweis: Nur wenige Programme werden ausschließlich vom ursprünglichen Programmierer gewartet. Codekonventionen verbessern dabei die Lesbarkeit und die Verständlichkeit. Selbst geringe Verbesserungen der Wartbarkeit rechtfertigen Anstrengungen in der Entwicklung.

Programme entstehen meist unter großem Zeitdruck, die nötige Kommentierung und Dokumentation wird nur oberflächlich betrieben. Die für jede Sprache geltenden Kon-

ventionen erleichtern dann das Einarbeiten in fremden oder eigenen Code. Trotzdem gibt es keine Prüfung oder keinen Zwang wie bei der Sprachsyntax, solche Empfehlungen einzuhalten. Wir wollen Ihnen lediglich nahe legen, sich mit den hier aufgeführten Konventionen etwas zu befassen. Es wird immer eine freiwillige Leistung des Entwicklers oder des Teams bleiben, guten Code zu schreiben.

Die Konventionen umfassen die folgenden Bereiche:

- Kommentarkonventionen
- Namenskonventionen für Variablen, Funktionen und Objekte
- Deklarationskonventionen
- Anweisungskonvention
- Textformatierung und Strukturierung

Die Empfehlungen entsprechen weitestgehend den Vorgaben aus der Informatik. In vielen anderen Programmiersprachen ist eine einheitliche Syntax bereits selbstverständlich. In PHP haben wir diesen Stand der Entwicklung noch nicht erreicht. Ein Standard ist daher noch nicht festgelegt. Vielleicht können wir jedoch mit diesem Abschnitt dazu beitragen, dass es einen Standard geben wird. Einige angemessene Codekonventionen können dabei nicht schaden.

1.1.2 Wie sollen Codekonventionen eingesetzt werden?

Die offiziellen Konventionen sollten als Basis dienen, sie funktionieren nur, wenn sie von jedem eingehalten werden. Es kann projektspezifische Erweiterungen geben.

Kommentarkonventionen

Mit Kommentaren wird Quelltext leichter lesbar und ist für die spätere Weiterbearbeitung wie auch bei der Fehlersuche leichter zu handhaben. Sie wissen bereits, dass Kommentare vom Interpreter ignoriert werden und damit auch für die Ausgabe keine Rolle spielen. Ob also 10 KByte oder 30 KByte ist einfach ausgedrückt nicht relevant! Nun einige Empfehlungen:

- Jede Funktion sollte mit einer kurzen Zweckbeschreibung beginnen.
- Schreiben Sie immer, wozu eine Funktion eingesetzt wurde, und nicht wie sie arbeitet.
- Nennen Sie alle globalen oder externen Variablen, Objekte oder andere Elemente, die Sie innerhalb der Funktion nutzen oder ändern.
- Beschreiben Sie die Struktur und den Wertebereich der Rückgabewerte bei Funktionen.

```
if (signal == TRUE) {
        lampe = true;          // Licht Ein
} else {
        lampe = false;         // Licht Aus
}
```

- Sie können nach der Kommentarzeichen // am Ende einer Befehlszeile einen Kommentar setzen. Nutzen Sie diese Technik, um wichtige Variablen am Ort Ihrer Deklaration zu beschreiben. Vermeiden Sie es jedoch, jede Hilfs- oder Zählvariable zu kommentieren, das führt zu unübersichtlichem Code.
- Denken Sie auch daran, dass Variablennamen selbsterklärend sein und zusätzliche Informationen nicht zur Regel werden sollten.

Am Beginn des Skripts selbst sollte ein kurzer Blick-Kommentar stehen. Er sollte

```
/*
  ...
*/
```

eine kurze Beschreibung der Funktionalität enthalten. Auf komplexe Algorithmen kann hingewiesen werden. Es ist besonders bei größeren Projekten sinnvoll, den Namen des Autors, das Datum der Freigabe, die Versionsnummer und die Versionsfolge zu beschreiben.

Es gibt einige allgemein übliche Schreibweisen zur Darstellung von wichtigen Kommentaren, die vor allem bei der Teamarbeit sehr nützlich sind.

`//:TODO:`

Zeigt an, dass hier noch etwas getan werden muss.

`//:BUG: [bugID]`

Thema bzw. Problem. Der Kommentar sollte den Fehler beschreiben und eine Fehlernummer enthalten.

`//:KLUDGE:`

Zeigt an, dass der Code nicht optimal ist, und symbolisiert damit einen Aufruf an die anderen Entwickler im Team, Verbesserungsvorschläge zu machen.

`//:TRICKY:`

Zeigt an, dass dieser Code recht komplex ist und daher bei der Bearbeitung vorab gut darüber nachgedacht werden sollte, welche Veränderungen vorgenommen werden.

Namenskonventionen

Die Groß-/Kleinschreibung ist für die Lesbarkeit enorm wichtig.

Beispiel:

```
$SPIELFIGUR  / $SpielFigur /  $spielfigur
```

Alle Bezeichner fangen mit Kleinbuchstaben an. Bezeichner sind Namen für Variablen, Funktionen, Objekte.

Beispiel:

```
$farbe / rennen() / $bild
```

Da in Bezeichnern keine Leerzeichen verwendet werden dürfen, sollte jedes neue Wort mit einem Großbuchstaben anfangen.

Beispiel:

```
setzeFarbe() / verwendeFarbe() / $initArray / $quizFrage
```

Hinweis: Eine Ausnahme gibt es bei der Bezeichnung die Objekte. Bei ihnen hat es sich durchgesetzt, sie mit einem Großbuchstaben zu beginnen.

Nachdem man sich in PHP eingearbeitet hat, sollte man auch in der Programmierung dazu übergehen, die Syntax in englisch zu halten. Da PHP auch in englisch geschrieben ist, ergibt sich dadurch ein einheitliches Sprachbild, damit fällt auch die Verwendung von Abkürzungen um einiges leichter. Die Konvention kann jedoch nach eigenem Ermessen in die Arbeit einbezogen oder ignoriert werden. Eines sollten Sie jedoch immer beachten: Verwenden Sie nie Umlaute!

Beispiel:

```
$Platzhalter (Besser: $dummy)
```

Hinweis: In diesem Buch werden Sie in den meisten Beispielen vor allem deutschsprachig formulierte Beispiele vorfinden, so dass auch diejenigen unter Ihnen, die sich mit der englischen Sprache noch nicht anfreunden konnten, ohne weiteres die Zusammenhänge verstehen.

Abkürzungen sind erlaubt, wenn sie konsequent und sinnvoll eingesetzt werden. Bis auf wenige Ausnahmen sind einbuchstabige Bezeichner zwar schneller in der Ausführung, jedoch schlechter zu lesen.

Beispiel:

```
$punktestand statt nur $ps für eine Punktestand
```

Konstanten können durch Großbuchstaben hervorgehoben werden, da sich ihr Wert nicht ändert.

Beispiel:

```
MEINE_KONSTANTE
```

Bestehen die Namen aus mehreren Wörtern, werden die Wörter mit dem Unterstrich (_) voneinander getrennt und alle Buchstaben großgeschrieben. Eine weitere Alternative wäre, ein Präfix oder Postfix zur Kennzeichnung zu verwenden.

Beispiel:

```
// Konstante
ConMeineKonstante
MeineKonstanteCon
```

Hier eine Liste von Vorschlägen zu Variablen und Datentypen.

Datentyp	*Präfix*	*Beispiel*
Array	arr	$arrPersonen, $personen_arr
Boolean	bln	$blnSignal, $signal_bln
Byte	byt	$bytWert, $wert_byt
Date	dt	$dtMorgen, $morgen_dt
Double	dbl	$dblPreis, $preis_dbl
Error	err	$errWert, $wert_err
Integer	int	$intPunkte, $punkte_int
Long	lng	$lngAbstand, $abstand_lng
Object	obj	$objHaus, $haus_obj
Single	sng	$sngWert, $wert_sng
String	str	$strName, $name_str

Deklarationskonventionen

Es sollte möglichst nur eine Deklaration pro Zeile geben.

```
var $name;                  // Vor- und Nachname
var $straße;                // Straße mit Hausnummer
var $name, $straße;         // Möglichst vermeiden
```

Es dürfen keine

- Deklarationen unterschiedlichen Typs,
- Methodendeklarationen
- Attributsdeklarationen

in einer Zeile stehen.

```
var $kundennummer, getKundennummer();              // Vermeiden
var $name, $namensliste[];                         // Vermeiden
```

Deklarationen sollten immer am Anfang eines Blocks stehen.

Mit Variablendeklarationen sollte nicht bis zur ersten Verwendung gewartet werden. Lokale Variablen sollten bei Deklaration möglichst auch initialisiert werden.

Das Überdecken von Variablen in einer höheren Ebene sollte vermieden werden.

```
function meinFunktion () {
        var $zaehler;
        if (Bedingung) {
                var $zaehler;  // Vermeiden!
        }
}
```

Anweisungskonventionen

Eine Zeile sollte nur einen Ausdruck enthalten.

```
$i++; $j++;                              // Vermeiden!
$zahlEins++; $zahlZwei--;      // Vermeiden!
```

In Blöcken sollten öffnende Klammern am Zeilenende stehen.

```
if (Bedingung) {
        Anweisungen;
}

if (Bedingung) {
        Anweisungen;
} else {
        Anweisungen;
}
```

Textformatierung und Strukturierung

Um Programmstrukturen lesen zu können, hat sich bei fast allen Programmiersprachen die Strukturierung des Codes durch Einrückungen etabliert. Dies kann im PHP-Editorfenster im Expertenmodus mit der TAB-Taste manuell vorgenommen werden.

Der folgende Quelltext zeigt, wie eine gute Codestrukturierung aussehen könnte.

```
function setPosition($par) {
        return $par;
}
```

1.1.3 Benennung und Kommentare

Benennung

- Aussagekräftige Bezeichner ($punkteStand = 1000)
- Begriffe in Bezeichnern durch Großbuchstaben trennen (setPowerButton)
- Bezeichner-Präfixe, um den Typ oder die Klasse anzugeben ($txtFeld)
- Bezeichner-Postfixe aus demselben Grund ($feld_txt)
- Klassen großschreiben (RechnerKlasse)
- Klassenbezeichner-Suffix: »Class« oder Klasse (AutoKlasse)
- Praefix für Funktionsparameter ($p wie in $pVorname)
- Konstanten durchweg großschreiben ($VERSION = 1.0)

Kommentare

- Programmblöcke in Absätze mit // aufteilen
- Kommentare bei selbsterklärenden Codezeilen vermeiden

1.2 Debugging – Fehlersuche in PHP

Bevor Sie erfahren, was Debugging genau ist, werden Sie sich als Erstes mit dem PHP-Fehlerkonzept vertraut machen müssen, denn Fehler ist nicht gleich Fehler.

1.2.1 PHP-Fehlerkonzept

PHP unterscheidet vier Fehlerklassen, denen unterschiedliche Bitwerte zugeordnet sind.

Bitwert	*Beschreibung*
1	Fehler
2	Warnung
4	Parserfehler
8	Nachricht

Die Summe dieser Bitwerte ergibt den so genannten Fehlerstatus.

Treten in einem Skript Fehler auf, werden die entsprechenden Meldungen im Browserfenster ausgegeben, wenn ein zuvor festgelegtes Fehlerniveau erreicht wird. Diese Festlegung erfolgt in der PHP-Konfigurationsdatei *php.ini* mit Hilfe der Option *error_reporting=wert*. Standardmäßig ist diese Option auf den Wert 7 eingestellt, so dass mit Ausnahme von Nachrichten alle Fehler ausgegeben werden. Den Fehlerklassen entsprechen PHP-intern vordefinierte Konstanten mit den enstsprechenden Bitwerten als Werten. Zusätzlich werden von PHP so genannte Kernel-Fehler unterschieden.

Bitwert	*Fehlername*	*Beschreibung*
1	E_ERROR	Fehler im Programmablauf
2	E_WARNING	Warnungen
4	E_PARSE	Fehler in der Syntax
8	E_NOTICE	Nachrichten
16	E_CORE_ERROR	Fehler des PHP-Kernels
32	E_CORE_WARNING	Warnung des PHP-Kernels

Da der in *php.ini* vorgegebene Wert von *error_reporting* mit Hilfe der gleichnamigen Funktionen `error_reporting(int level)` auch zur Laufzeit des Skripts verändert werden kann, können auf diese Weise auch die vom PHP-Sprachkern gemeldeten Fehler zur Anzeige gebracht werden.

1.2.2 Syntaxanalyse

Nun den Programmiergrundlagen kommen wir zur Skriptanalyse. Diese Form der Qualitätskontrolle ist immens wichtig und vor allem bei größeren Projekten ausschlag-

gebend für einen möglichst fehlerfreien Ablauf. Dabei wird das Entfernen von Fehlern aus Skripts als Skriptanalyse (engl. debugging) bezeichnet.

Selbst wenn Sie schon Hunderte von Skripts geschrieben haben, machen Sie sicher ab und zu einen Fehler. Sie schreiben ein Wort falsch, oder Sie vergessen eine Funktion einzutippen. Deshalb kommt es trotz aller Sorgfalt beim Schreiben Ihres Skripts immer wieder vor, dass es nicht wie gewünscht funktioniert und fehlerfrei ausgeführt wird. Diese Probleme, die bewirken, dass ein Skript nicht oder nur fehlerhaft läuft, werden als Bugs, zu deutsch Ungeziefer, bezeichnet. Sie können jedoch beruhigt sein, Sie sind nicht allein. Jedes Programm enthält Fehler (Bugs), einschließlich der Betriebssysteme. Viele Fehler sind harmlos. Sie beeinträchtigen die korrekte Funktionsweise Ihres Skripts nicht direkt, sondern führen höchstens dazu, dass Ihr Projekt an der einen oder anderen Stelle langsamer läuft. Größere Fehler haben ernsthaftere Folgen. So könnte eine Endlosschleife Ihr gesamtes Projekt zum Erliegen bringen und damit eine Ausführung unmöglich machen.

Sie haben jedoch die Möglichkeit, Fehler zu vermeiden.

1.2.3 Fehlerprävention

Hier eine kurze Checkliste, um Fehler zu vermeiden. Sie soll Ihnen die Fehlerprävention erleichtern:

- Sie sollten allen Variablen, Arrays, Funktionen und Objekten eindeutige Namen zuordnen. Eine Ausnahme stellen lediglich die lokalen Variablen in Schleifen oder Funktionen dar.
- Verwenden Sie die Anweisung `foreach`, um die Eigenschaften von Objekten zu durchlaufen.
- Achten Sie darauf, dass `while`-, `do-while`-, `for`- und `foreach`-Schleifen korrekt beendet werden, um Endlosschleifen zu vermeiden.
- Prüfen Sie, ob alle Gültigkeitsbereiche stimmen.

Sie sollten sich nochmals im Klaren darüber sein, dass niemand perfekt ist. Deshalb können Sie, selbst wenn Sie die oben aufgeführten Ratschläge beachten, keine hundertprozentige Garantie geben, dass Ihr Projekt fehlerfrei läuft. Selbst erfahrene Informatiker sind dagegen nicht gefeit. Fehler gehören zum Leben, wie Ungeziefer in den Keller. Sie werden es wohl kaum schaffen sie auszurotten, aber Sie können versuchen, so viele wie möglich zu vermeiden.

Wie kommt es eigentlich, dass Fehler als Bugs bezeichnet werden? Diese Frage soll hier kurz beantwortet werden, so dass Sie jederzeit bei einem netten Gespräch unter Entwicklern eine Anekdote parat haben.

Dafür müssen Sie zu den Anfängen des Computerzeitalters zurückkehren. Der erste Computer wurde mit mechanischen Relais betrieben, an elektronische Bauteile war noch nicht zu denken. Eines Tages blieb der Computer ohne ersichtlichen Grund stehen und ignorierte jegliche Eingabe. Die Entwickler prüften ihre Programme und mussten feststellen, dass sie eigentlich fehlerfrei funktionieren sollten. Die Stromversorgung war

ebenfalls gesichert und auch die Drähte im Computer waren ordnungsgemäß angeschlossen. Einer der Entwickler entdeckte jedoch, dass eine Wanze (engl. Bug) in einem der Relais zerquetscht worden war, was dazu geführt hatte, dass sich das Relais nicht mehr vollständig schließen konnte. Diese Wanze hatte den Computer lahm gelegt, und das ist auch der Grund, wieso seitdem Fehler als Bugs bezeichnet werden.

1.2.4 Fehlerarten

Fehler werden grundsätzlich in drei Gruppen eingestuft:

- Syntaxfehler
- Laufzeitfehler
- Logische Fehler

Syntaxfehler

Gerade als Programmierneuling sind Fehler oft unvermeidlich und tauchen immer dann auf, wenn man sie nicht erwartet. Die meisten Fehler beruhen auf einer fehlerhaften Syntax, deshalb werden sie auch als Syntaxfehler bezeichnet. Sie lassen sich recht einfach aus der Welt schaffen, indem Sie sich Ihre Codezeilen nach der Fehlerausgabe genauer betrachten. Hier einige Beispiele häufig vorkommender Syntaxfehler:

Fehlerhafte Bezeichner, welche nicht den Regeln entsprechen.

Beispiel:

```
$über = 10 ;
$erster Wert = 10;
```

Die Anweisungsblöcke von Funktionsdefinitionen bzw. Kontrollstrukturen sind nicht korrekt beendet.

```
// Ergebnis: Fehlermeldung
function meineFunktion () {
               $zahlEins += 10;
               $zahlZwei *= 10;

// Ergebnis: Fehlermeldung
if ($signal) {
       $aktiv = true;
else {
       $aktiv = false;
}
```

Fehler, die bei der Eingabe von Fließkommazahlen auftauchen, hier könnten Sie sagen: typisch deutsch.

Beispiel:

```
// Lösung: 1.95
$preis = 1,95;
```

Sollte Ihr PHP-Skript nach der Entfernung aller Syntaxfehler ausgeführt werden, wissen Sie, dass es keine Syntaxfehler mehr enthält. Nun müssen Sie sich den Laufzeit- und logischen Fehlern zuwenden.

Fehlende Klammern & Co.

Es wird Ihnen vor allem bei der Arbeit mit umfangreichen PHP-Skripts passieren, dass Sie Klammern eines Anweisungsblocks oder einer Schleife vergessen. In der Regel können Sie fehlende geschlossene Klammern daran erkennen, dass der PHP-Interpreter einen Fehler in der letzten Codezeile Ihres Skripts ausgibt.

Dies liegt daran, dass die abschließende Klammer nicht gefunden wird und der Fehler erst am Ende des Skripts vom PHP-Interpreter bermerkt wird. Sie müssen in diesem Fall Ihr Skript sorgfältig durchgehen und nach der vergessenen Klammer suchen. Um dieses Problem von Anfang an zu vermeiden, sollten Sie in solchen Fällen die Klammern von Anfang an öffnen und schließen und dann den benötigten Code einfügen.

Tipp: Sollten Sie eine Fehlermeldung zusammen mit einer Zeilennummer erhalten, bei der Sie sich sicher sind, dass sich dort kein Fehler befinden kann, dann betrachten Sie auch die Zeilen darüber und kontrollieren Sie diese. Vor allem bei nicht geschlossenen Klammern geht der PHP-Interpreter das gesamte PHP-Skript durch und gibt dann erst die letzte Zeile als Zeilennumer des Fehlers aus.

Laufzeitfehler

Ein Laufzeitfehler kommt immer dann vor, wenn Ihr Skript Daten erhält, mit denen es nichts anfangen kann. Diese Fehlerart ist um einiges subtiler als Syntaxfehler. Ihr Skript kann eine Vielzahl von Laufzeitfehlern haben, von denen Sie erst etwas merken, wenn Sie Ihr Skript ausführen. Ein Beispiel für einen Laufzeitfehler in einem Skript finden Sie in folgendem Beispiel:

```
function steuer() {
       $steuerSatz = $steuerSchulden/$einkommen;
}
```

Die Funktion führt normalerweise Ihre Anweisung fehlerfrei aus, solange das Einkommen größer ist als 0. Bei einem Einkommen von 0 gibt es ein Problem. Da die Division durch 0 nicht möglich ist, kommt es zu einem Fehler bzw. einer fehlerhaften Zuweisung. Um Laufzeitfehler zu entdecken, müssen Sie Ihr PHP-Skript mit jeder möglichen Eingabe testen, angefangen vom Drücken einer falschen Taste bis hin zu einer negativen Zahl, die ein Anwender als sein Einkommen eingibt. Dies bezeichnet man häufig als DAU-Sicherung (DAU = Dümmster Anzunehmender User). Auch wenn diese Bezeichnung für den Anwender möglichst die heiligen vier Wände des Entwicklers nicht verlassen sollte.

Sie werden jedoch eines sehr schnell feststellen: Da die Anzahl der Dinge, die schief gehen können, ins Endlose geht (Murphy's Gesetz), verstehen Sie sicher, warum jedes Programm Bugs hat.

Logische Fehler

Neben den Syntax- und Laufzeitfehlern gibt es noch die logischen Fehler. Diese Fehlerart tritt auf, wenn ein Skript deshalb nicht ordnungsgemäß funktioniert, weil Sie ihm die falschen Anweisungen oder die Anweisungen in der falschen Reihenfolge gegeben haben. Sie fragen sich, wie das sein kann, wie Sie in einem Skript falsche Anweisungen ablegen können, wenn Sie doch selbst das Skript geschrieben haben. Glauben Sie es ruhig, es kommt vor!

Natürlich gehen Sie davon aus, dass Sie in all Ihren Skripts des Projekts die richtigen Anweisungen abgelegt haben. Sie haben zunächst keine Ahnung, warum Ihr Projekt nicht einwandfrei läuft. Jetzt gilt es die Stelle in den Skripts zu finden, an der Ihre Anweisungen nicht genau genug waren. Wenn Ihr Projekt recht umfangreich ist, kann das bedeuten, dass Sie Zeile für Zeile durchgehen müssen.

Wie Sie sehen ist das kein Zuckerschlecken, daher sollten Sie immer versuchen, den Überblick zu wahren. Ein Beispiel für einen logischen Fehler darf nicht fehlen.

Beispiel:

```
// Logischer Fehler (bei Schleifen)
// Ergebnis: Endlos Schleife
for ($i = 0;$i <= 100;$i--) {
        $summe += $i;
}
```

1.2.5 Debugging

Vor allem die logischen Fehler können entweder sehr einfach oder sehr schwer zu beheben sein. Eines haben jedoch alle Fehler gemeinsam: Sie müssen erst einmal gefunden werden, damit man sie anschließend beheben kann. Genau hierin liegt die Schwierigkeit!

Fehlersuche mit dem Maguma Studio Debugger

Die Werte von Variablen können Sie mit Hilfe des im Maguma Studio integrierten *Debuggers* auf einfache Weise überwachen. Der wesentliche Vorteil gegenüber der PHP-Interpreter-Ausgabe besteht darin, dass die Überwachung in Echtzeit erfolgt und die Aktualisierung automatisch während der Ausführung des Skripts durchgeführt wird. Der Debugger arbeitet in diesem Fall mit Hilfe von Breakpoints (Haltepunkten), die im Skript festgelegt werden können.

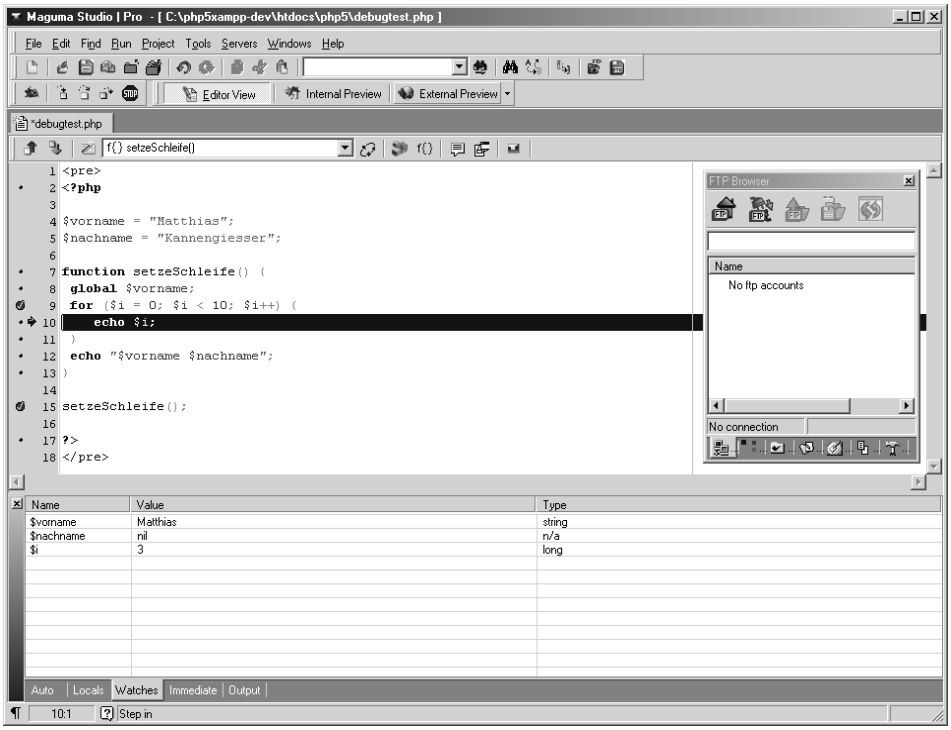

Bild 1.1: Maguma – Debugger samt Breakpoints

Tipp: Wir empfehlen Ihnen, das Maguma Studio oder einen vergleichbaren Editor einzusetzen, da dies die Analyse und Fehlersuche in umfangreichen PHP-Projekten wesentlich erleichtert.

1.2.6 Fehler suchen und Fehler finden

In diesem letzten Abschnitt soll Ihnen gezeigt werden, wie sich die Fehlersuche darstellt. Die Fehlersuche und Beseitigung kann in drei Phasen unterteilt werden:

- Erkennen, dass ein Fehler vorliegt.
- Suchen und Finden des Fehlers.
- Beseitigen des Fehlers.

Erkennen, dass ein Fehler vorliegt

Der beste Weg, Fehler in Ihrem Projekt und Ihren Skripts zu entdecken, besteht darin, freiwillige Testpersonen mit Ihrem Projekt arbeiten zu lassen. Dies wird in der Entwicklung als Testphase bezeichnet. In der Softwareindustrie ist dies auch als Betatest bekannt.

Beobachtung von Variablen zur Laufzeit

Sie können hierzu den Befehl `echo()` verwenden, indem Sie die Inhalte verdächtiger Variablen im Browser ausgeben. Oft sind nicht nur die aktuellen Werte der Variablen von Interesse, sondern auch andere Informationen über die Variablen, zu deren Abfrage in PHP eine Reihe von Funktionen zur Verfügung stehen.

Funktionen	*Bedeuteung*
empty()	Ist die Variable leer?
gettype()	Ermittelt des Datentyps
is_array()	Ist Variable ein Array?
is_double()	Datentyp double?
is_float()	Datentyp float?
is_int()	Datentyp integer?
is_object()	Datentyp object?
is_real()	Datentyp real?
is_string()	Datentyp string?
isset()	Variable defniert?

Behandlung von Laufzeitfehlern

Wie Sie bereits erfahren haben, stellen vor allem die Laufzeitfehler oftmals ein größeres Problem dar. Diese Fehler gilt es mit Hilfe von entsprechenden Fehlerbehandlungsroutinen abzufangen. Dies ist jedoch eher eine Frage des sauberen Programmierstils als des Debugging.

Eine einfache Möglichkeit, auf Laufzeitfehler zu reagieren, besteht darin, die Skriptausführung kontrolliert abzubrechen. Hierzu stehen in PHP die Sprachkonstrukte `exit` und `die()` zur Verfügung.

Während `exit` die Skriptausführung sofort abbricht, gibt `die()` vor dem Abbruch noch eine Meldung an den Browser aus.

Beispiel:

```
<?php
$file = "htdocs/daten.txt";
$fp = fopen($file, "r") or die("Datei nicht gefunden: $file");
$dat = fread($fp, filesize($file));
fclode($fp);
?>
```

Ausgabe:

```
Warning: fopen(htdocs/daten.txt) [function.fopen]: failed to open stream:
No such file or directory in C:\php5xampp-dev\htdocs\php5\debugtest.php on
line 3
Datei nicht gefunden: htdocs/daten.txt
```

Sollte die mit der Variablen `$file` referenzierte Datei nicht gefunden werden, wird die Skriptausführung beendet und zuvor noch eine entsprechende Meldung ausgegeben. Um die Interpreter-eigene Warnung zu unterdrücken, steht, wie Sie ja wissen, das @-Zeichen zur Verfügung.

Beispiel:

```
<?php
$file = "htdocs/daten.txt";
$fp = @fopen($file, "r") or die("Datei nicht gefunden: $file");
$dat = fread($fp, filesize($file));
fclode($fp);
?>
```

Ausgabe:

```
Datei nicht gefunden: htdocs/daten.txt
```

Suchen und Finden des Fehlers

Wenn Sie festgestellt haben, dass ein Fehler vorliegt, müssen Sie ihn suchen und finden. Einen Fehler zu finden ist häufig der schwierigste Teil. Die einfachste, aber auch zeitaufwendigste Methode besteht darin, Ihr Projekt auszuführen und Ihre Skripts Zeile für Zeile zu untersuchen. Zu dem Zeitpunkt, an dem der Fehler auftaucht, wissen Sie genau, welche Codezeile ihn verursacht. Diese Methode eignet sich jedoch lediglich bei kleineren Projekten. So vorzugehen, um ein komplexes und großes Projekt nach Fehlern zu durchsuchen, wäre abwegig.

Die Alternative, die wesentlich schneller zum Ziel führt, wäre folgende: Sie sollten nur in Teilen des Projekts nach dem Fehler suchen, in denen Sie diesen vermuten. Wenn in Ihrem Projekt beispielsweise die Formularüberprüfung für ein in Ihrem Projekt angelegtes Formular nicht richtig durchgeführt wird, steckt der Fehler wahrscheinlich in dem Skript, in dem Sie die Methoden zur Überprüfung definiert haben.

Tipp: Bei der Fehlersuche stehen Ihnen in PHP selbst nicht sehr viele Hilfsmittel zur Verfügung, jedoch bietet Ihnen das Maguma Studio ein hervorragendes Werkzeug, nämlich den integrierten Debugger. Dieser kann Ihnen beim Auffinden der Fehler nur dann behilflich sein, wenn Sie ihn auch einsetzen.

Sobald Sie den fehlerhaften Teil isoliert haben, müssen Sie herausfinden, was den Fehler verursacht.

Beseitigen Sie den Fehler

Wenn Sie den Fehler gefunden haben, können Sie die Ursache und damit den Fehler beseitigen. Gehen Sie dabei vorsichtig vor! Oft genug kommt es vor, dass die Fehlerkorrektur, unbeabsichtigt, neue Fehler hervorruft. Daher ist es meist einfacher, einen großen Anweisungsblock neu zu schreiben, als zu versuchen, einen Fehler darin zu korrigieren.