# PHP & Java

## Inhalt:



## 1. Einleitung

Für PHP gibt es zahlreiche Erweiterungen (in Form von in C geschriebenen Bibliotheken, die mit entsprechenden Header-Files in den Interpreter eingebunden werden), deren Funktionsumfang im Regelfall keine Wünsche offen lässt. Trotzdem lässt sich nicht ausschließen, dass man in Situationen kommt, in denen man ein Problem mit den normalen PHP-Mitteln nicht ausreichend lösen kann (beispielsweise Image-Filter oder 3D-Rendering). Bevor man sich nun hinsetzt und mühsam eine eigene Bibliothek dafür schreibt, sei es in C oder in PHP selbst, gibt es in PHP, wie in vielen anderen Skriptsprachen auch, die Möglichkeit, andere Programmiersprachen einzubinden. In PHP existiert eine *Java-Extension*, mit der es möglich ist, beinahe die gesamte Funktionalität von Java auch in PHP nutzbar zu machen. Aus Platzgründen setzt dieses Tutorial fundierte Kenntnisse in der Java-Programmierung voraus und geht nur auf die Punkte ein, die bei der Interaktion von PHP und Java beachtet werden müssen.
Das Thema Java & PHP scheint sich einiger Beliebtheit zu erfreuen, was sich daran zeigt, dass eine neue Java-Bridge in Arbeit ist, die recht viel versprechend ist (wobei PHP 5 wohl die erste unterstütze Skriptsprache sein wird, dank der Bemühungen von Zend). Dennoch gibt es kritische Einwände: wenn schon Java auf dem Server bereitsteht, warum dann nicht gleich Servlets oder JSP verwenden? Zumal das PHP-Skript dank der externen Initialisierung der Java-VM manchmal schnarchig langsam läuft und die Installation der PHP4-Java-Bridge eine recht knifflige Angelegenheit ist (wie wir noch sehen werden). Zudem laufen einige Programme nur als CGI und nicht als Apache-Modul, wozu sich gerade beim Apache ein besonders Problem gesellt: die mit *fork* erzeugten Kindprozesse erzeugen jeweils eine neue Instanz der Java-VM, wodurch der Arbeitsspeicher sehr schnell aufgebraucht wird (ein Problem, das wir bei den Installationsanleitungen nochmal aufgreifen werden). Bei Servern mit echten Threads wie dem IIS tritt dieses Problem nicht auf.
Bei aller Kritik darf nicht vergessen werden, dass es durchaus sinnvolle Einsätze des Duos PHP/Java gibt. In kleineren Netzen oder bei wenig Zugriffen ist die Performance sicher ausreichend. Wohl noch wichtiger ist aber, dass die gewohnte Umgebung PHP nicht aufgegeben werden muss. Wenn ein Projekt bereits in PHP steht und die Mitarbeiter schon mit PHP vertraut sind, kann ein Java-Kundiger einfach die neuen Funktionen hinzufügen, ohne dass das gesamte Projekt in Servlets umgeschrieben werden muss

oder alle Mitarbeiter sich neues Wissen aneignen müssen. Und vielleicht will man ja auch nicht noch einen zusätzlichen Server (z.B. Tomcat) zu warten haben, wenngleich sich speziell Tomcat schön in Apache integrieren lässt. Und nicht zuletzt: der Spaß-Faktor...

## 2. Installation

Die von mir beschriebenen Installationen sollten auf den meisten Systemen ohne Probleme arbeiten. Dennoch können hier nicht alle Kombinationen von Betriebssystemen, HTTP-Servern und Java-Distributionen besprochen werden. Ich beschränke mich auf Windows, Linux, das Sun-JDK und den Apache-Server. Wer mit dem JDK eines anderen Herstellers arbeitet, den IIS oder einen anderen Server verwendet oder einfach trotz meiner Installationsanleitung noch Probleme hat, sollte einen Blick auf folgende Seite werfen (welcher sich übrigens immer lohnt): http://www.php.net/manual/de/ref.java.php

### 2.1. PHP-Extension unter Windows

Will man die Java-Bridge unter einem Windows-Apache zum Laufen bringen, sollte PHP als CGI eingebunden sein und nicht als Apache-Modul eingebunden sein (in dem Falle würde beim 2. Aufruf einer Seite keine Java-VM mehr erzeugt werden können, womöglich ein Folge der schon genannten Apache-Subprozesse und des Pooling). Im Vergleich zu Linux ist die Installation nicht besonders schwer. Wir gehen von einem unter C:\j2sdk1.4.2 installierten JDK aus. Apache-Version für meinen Test war 1.3.31, PHP war in der Version 4.3.8 installiert.

1. Dieser Schritt ist nur notwendig, wenn PHP bereits als Apache-Modul läuft:

Kommentiere in **httpd.conf** die Zeilen wieder aus, mit denen das PHP-Modul eingebunden wurde, also:

```
# LoadModule php4_module        C:/pfad/zu/libphp4.so
# AddModule mod_php4.c
```

Der Eintrag des MIME-Types bleibt aber erhalten:

```
AddType application/x-httpd-php .php
```

Füge nun hinzu:

```
ScriptAlias /php/ "C:/php/"
Action application/x-httpd-php "/php/php.exe"
```

2. Bearbeite die PHP-Konfigurationsdatei **php.ini.** Entferne den Kommentar (also den Strichpunkt) vor der Zeile:

```
;extension=php_java.dll
```

Und füge bei der Sektion **[Java]** hinzu:

```
java.class.path = c:\php\extensions\php_java.jar
java.home = c:\j2sdk1.4.2\bin
java.library = c:\j2sdk1.4.2\jre\bin\server\jvm.dll
java.library.path = c:\php\extensions
```

## 2.2. PHP-Extension unter Linux

Die Installation unter Linux ist keine triviale Angelegenheit. In den meisten Fällen sollte meine Anleitung aber funktionieren (egal ob als Apache-DSO oder als CGI). Meine Installation habe ich getestet unter RedHat 9.0 und Slackware 9.1. Die Apache-Version ist 1.3.28 bzw. 2.0.47, die PHP-Version 4.3.3. Das JDK liegt in der Version 1.4.0 vor und wurde von mir unter "/usr/local/java" installiert.

1. Dieser Schritt ist *nur für Apache 2 notwendig*! Falls noch nicht geschehen, muss Apache mit der Option **--with-mpm=prefork** neu kompiliert und installiert werden.

2. In der Datei **/etc/ld.so.conf** müssen folgende Zeilen hinzugefügt werden:

```
/usr/local/java/jre/lib/i386
/usr/local/java/jre/lib/i386/server
```

Die Änderungen müssen anschließend wirksam gemacht werden (in der Shell):

```
/sbin/ldconfig
```

3. PHP wird mit der Option **--with-java=/usr/local/java** neu kompiliert und installiert (zusammen mit anderen gewünschten Erweiterungen).

4. Der Datei **php.ini** sind folgende Zeilen hinzuzufügen (wir gehen davon aus, dass **extension_dir** bereits auf "/usr/local/lib/php/extensions/no-debug-non-zts-20020429" gesetzt ist, wo sich die Datei **java.so** befindet):

```
java.class.path=/usr/local/lib/php/php_java.jar
java.library.path=/usr/local/lib/php/extensions/no-debug-non-zts-20020429
java.home=/usr/local/java/jre/bin
java.library=/usr/local/java/jre/lib/i386/libjava.so
extension=java.so
```

**java.library.path** zeigt übrigens auf das selbe Verzeichnis wie **extension_dir**!

5. Wechseln in das Verzeichnis der PHP-Extensions (also in diesem Fall "/usr/local/lib/php/extensions/no-debug-non-zts-20020429"), um folgenden Softlink namens "**libphp_java.so**" erstellen, der auf "**java.so**" zeigt:

```
ln -s java.so libphp_java.so
```

6. Die Datei **/usr/local/java/jre/lib/i386/libverify.so** weigert sich möglicherweise immer noch, gefunden zu werden. Das kann man überprüfen, wenn man in das Verzeichnis /usr/local/java/jre/lib/i386 wechselt und den Befehl "**ldd libjava.so"** aufruft. Wir schlagen dem ein Schnippchen, indem wir dem Apache-Steuerungs-Skript "**apachectl**" folgende

Zeile an geeigneter Stelle (bspw. in der Configuration Section) hinzufügen:

```
export LD_LIBRARY_PATH=:/usr/local/java/jre/lib/i386:/usr/local/java/jre/lib/i386/server
```

## 2.3. Das Servlet SAPI Modul

Mit diesem Beispiel ist es möglich, PHP-Skripte mit dem Tomcat-Server aus dem Apache-Projekt auszuführen. Der PHP-Prozessor wird von der Servlet-Engine ausgeführt. In der Theorie bringt dies einen Gewinn an Performance und Stabilität, in der Praxis funktioniert dies wieder nur in einer Thread-freien Umgebung. In einer Umgebung mit Threads (was mit Tomcat der Normalfall) wird Tomcat nach einigen Aufrufen eines PHP-Skripts abstürzen. Da ein Servlet für jeden Aufruf einen neuen Thread erzeugt, werden die Request- und Response-Objekte des Servlets überschrieben. Unter http://www.pelikan-it.com/download.html gibt es gepatchte Dateien für php4.3.4/sapi/servlet/**servlet.c** und php4.3.4/sapi/servlet/**servlet.java**, die dieses Problem beheben sollen, allerdings stürzte Tomcat in meinen Tests nach einigen Aufrufen einer PHP-Datei wie vorher ab.
Ich will dennoch eine Anleitung zur Installation geben, vielleicht haben andere etwas mehr Glück, insbesondere mit den gepatchten Dateien. Die Einbindung von PHP in Tomcat ist dennoch grundsätzlich interessant. Verwendet habe ich wieder wie oben ein JDK 1.4.0 von Sun, PHP 4.3.3 und Tomcat 4.1.27 installiert unter "/usr/local/tomcat".

1. Wir setzen, sofern noch nicht geschehen, die Umgebungsvariablen $JAVA_HOME (Verzeichnis des JDK, gebraucht von Tomcat beim Start), $CATALINA_HOME (Tomcat-Verzeichnis, bei uns also "/usr/local/tomcat"), $CLASSPATH (also die Verzeichnisse, von Java nach Klassenbibliotheken sucht) und $LD_LIBRARY_PATH (Pfad zu Shared-Object-Dateien). Den CLASSPATH müssen um die Datei **$CATALINA_HOME/common/lib/servlet.jar** ergänzen, der LD_LIBRARY_PATH muss das Verzeichnis "/usr/local/lib/php" enthalten (in diesem Verzeichnis wird die Datei **libphp4.so** nach dem Neukompilieren von PHP landen).

```
export JAVA_HOME=/usr/local/java
export CATALINA_HOME=/usr/local/tomcat
export CLASSPATH=$CLASSPATH:$CATALINA_HOME/common/lib/servlet.jar
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/php
```

2. PHP muss mit den Optionen **--with-servlet** und **--with-java=/usr/local/java** neu kompiliert und installiert werden

3. Die Datei **phpsrvlt.jar** muss ins Library-Verzeichnis von Tomcat kopiert werden:

```
cp /usr/local/lib/php/phpsrvlt.jar $CATALINA_HOME/common/lib
```

4. Die Dateien **$CATALINA_HOME/conf/web.xml** und **phpsrc/sapi/servlet/web.xml** müssen miteinander verschmolzen werden, indem die Inhalte von web.xml der PHP-Sourcen in die Datei web.xml der Tomcat-Konfiguration eingefügt werden, und zwar ohne die korrekte Reihenfolge der XML-Tags zu verletzten. Die Abschnitte <servlet> ... </servlet> der Datei phpsrc/sapi/servlet/web.xml sollten deshalb am Besten hinter die schon bestehenden <servlet>-Tags der Datei $CATALINA_HOME/conf/web.xml eingefügt werden, in jedem Fall aber vor den <servlet-mapping>-Tags. Mit den Abschnitten <servlet-mapping> ... </servlet-mapping> der Datei phpsrc/sapi/servlet/web.xml sollte analog verfahren werden. Zusätzlich können die Dateien index.php als Startseite eines

Verzeichnisses registriert werden, indem zwischen <welcome-file-list> und </welcome-file-list> die Zeile <welcome-file>index.php</welcome-file> eingefügt wird.
Jetzt kann Tomcat mit dem Skript **startup.sh** des Verzeichnisses "$CATALINA_HOME/bin" gestartet werden (das Skript zum Herunterfahren des Servers ist **shutdown.sh**) und eine Datei test.php, die im Verzeichnis "$CATALINA_HOME/webapps/ROOT" liegt, über die URL "http://localhost:8080/test.php" angesprochen werden (sofern der Server auf den Standardport 8080 lauscht).

## 2.4. PHP-Java-Bridge als Sourceforge-Projekt

*Ich persönlich rate dringend zu dieser Variante!* Da die mit PHP mitgelieferte PHP-Extension immer noch experimentell ist, Kinderkrankheiten hat und schwer zu installieren ist, gibt es eine PHP-Java-Bridge als *Sourceforge-Projekt* unter: http://sourceforge.net/projects/php-java-bridge
Diese Bridge ist als Ersatz für die offizielle Extension gedacht und läuft ergo nicht parallel mit ihr. Diese Bridge löst das Problem der Sub-Prozesse unter Apache und sorgt dafür, dass nur eine JVM läuft (dank Kommunikation über Socket). Die Performance scheint überhaupt besser zu sein. Außerdem ist sie leicht zu installieren. Im Folgenden gehen wir davon aus, dass wir ganz normal (sprich: ohne eine Java-Erweiterung und den oben erwähnten Schritten) Apache und PHP installiert haben. Getestet habe ich folgende Installation auf Slackware 9.1 mit Apache 1.3.28, PHP 4.3.3 und dem Sun-JDK 1.4.0 (installiert unter "/usr/local/java"). Die Version der PHP-Java-Bridge war 1.0.5.

1. Ins Verzeichnis der PHP-Java-Bridge wechseln und eine normale Installation durchführen (nicht vergessen die Bridge mit "**phpize**" auf die vorhandene PHP-Version einzustellen):

```
phpize
./configure --with-java=/usr/local/java
make CFLAGS="-DNDEBUG"
make install
```

2. Java-Extension in **php.ini** laden (die Angaben zum Loggen sind optional, **extension_dir** muss auf "/usr/local/lib/php/extensions/no-debug-non-zts-20020429" gesetzt sein):

```
extension=java.so
java.log_level=5
java.log_file=/tmp/java.log
```

3. Dieser Schritt ist *optional*: es wird empfohlen, die Java-VM in einem separaten Prozess zu starten. Dazu ist der Datei **php.ini** noch folgender Eintrag hinzuzufügen:

```
java.socketname=/tmp/.php_java
```

In der Shell muss dann immer, bevor Apache gestartet wird, folgender Befehl eingegeben werden (am Besten in Form eines Shell-Skripts, Pfadangaben je nach Installation natürlich anpassen, mehr dazu in der README-Datei):

```
JAVA_HOME=/usr/local/java /usr/local/java/bin/java \
-Djava.library.path=/usr/local/lib/php/extensions/no-debug-non-zts-2002049 \
-Djava.class.path=/usr/local/lib/php/extensions/no-debug-non-zts-20020429 \
-Djava.awt.headless=true JavaBridge /tmp/.php_java 5 /tmp/java.log
```

Es gibt auch ein Hilfs-Skript zum Starten der JVM im PHP-Java-Bridge-Verzeichnis:

```
./php-java-bridge
```

## 3. Java-Klassen instanzieren

Ein Instanz einer Java-Klasse wird mit **new Java**("Klassenname") erzeugt, wobei "Klassenname" den vollen "Pfad", also Package und den eigentlichen Namen der Klasse, umfasst. Um z. B. eine Instanz der Klasse "Date" zu erzeugen, muss **new Java("java.util.Date")** aufgerufen werden. Sehen wir uns folgendes beliebtes Einstiegsbeispiel und Testprogramm an, in dem Angaben zu Java-Version, Betriebssystem usw. über die Methode **getProperties** der Klasse **java.lang.System** abgefragt werden. Die Datei **properties.php**:

```
<?php
$system = new Java("java.lang.System");
$properties = $system->getProperties();
foreach ($properties as $key => $value)
        echo $key . " = " .  $value . "<br />\n";
?>
```

Die Instanz der Java-Klasse wir an ein PHP-Objekt gebunden, so dass wir alle Methoden, die uns in Java zur Verfügung stehen würden, über das PHP-Objekt aufrufen können. Der Java-Kenner sollte jetzt einwerfen, dass die Klasse System nicht direkt instanziert werden kann, da der Konstruktor *private* ist und im Übrigen alle Methoden der Klasse (so auch "getProperties") *static* sind, daher nicht zu einer Instanz aufgerufen werden können. Auf diese Weise können in PHP auch Objekte beschafft werden, die in Java beispielsweise einer Factory-Methode bedürfen, und das sind in der Java-API nicht gerade wenige!
Sehen wir uns ein weiteres einfaches Beispiel an. Im Folgenden soll ein Stack mit drei Elementen gefüllt und diese Elemente dann nacheinander ausgelesen werden. Der Code dazu. Die Datei **stack.php**:

```
<?php
$stack = new Java("java.util.Stack");
$stack->push("Hallo");
$stack->push("Welt");
$stack->push("Chrstph");
while (!$stack->empty())
        echo $stack->pop() . "<br />\n";
?>
```

Folgende Dinge sind im Allgemeinen *zu beachten*:

- Bei Verwendung einiger Java-Klassen in PHP wird das Skript *nur mehr über CGI ausführbar*. Das scheinen nach meiner Erfahrung hauptsächlich solche zu sein, die grafische Elemente repräsentieren, wie z.B. das AWT. Im Zweifelsfall hilft einfach ausprobieren.

- Die *Zahlenwerte* von PHP und Java *sind nicht immer kompatibel*. Da Java nur vorzeichenbesetzte Zahlenwerte kennt und nicht automatisch konvertiert, kann es zu Abweichungen bei Berechnungen kommen. Ein Beispiel sind Prüfsummen und Message Digests: das Format der Ausgabe ist bei den entsprechenden PHP-Funktionen eine andere als bei den zugehörigen Java-Methoden. Hier müsste das Ergebnis erst auf ein einheitliches Format gebracht werden, was nicht immer einfach ist.
- *Strings sind beliebig austauschbar*. Zur Anschauung sei nur das Stack-Beispiel erwähnt: die Methode **push** nimmt nur Objekte, die Methode **pop** liefert nur Objekte zurück (**java.lang.String** ist ja wie alle Java-Klassen von **java.lang.Object** abgeleitet). Die direkte Bildschirmausgabe mit **echo** funktioniert trotzdem: wenn man den Datentyp, der von **pop** zurückgegeben wird, untersucht, so stellt man fest, dass es sich dabei um einen String und nicht um ein Objekt handelt.
- Ein besonders kniffliges Problem ergibt sich bei Arrays (hierfür habe ich noch keinen wirklichen Workaround gefunden): Arrays werden in der Bridge immer *by-value* an Methoden übergeben, in Java selbst gilt allerdings der Grundsatz, dass sie immer *by-reference* übergeben werden. Dadurch werden Java-Methoden, die ein by-reference übergebenes Array auffüllen (wie z.B. einige read-Methoden der Input-Streams) in PHP unbrauchbar. In der Regel kann man aber auf alternative Methoden ausweichen. Ansonsten sind Arrays fast beliebig austauschbar.
- Auch *Hashes* sind zwischen Java und PHP *fast beliebig austauschbar*. Es bleibt in PHP die Einschränkung, dass der Schlüssel nur vom Typ Integer oder String sein darf, während ein Java-Dictionary als Schlüssel beliebige Objekte akzeptiert. Werfen wir einen Blick auf unser erstes Beispiel: die Methode **getProperties** der Klasse **java.lang.System** liefert eine Instanz der Klasse **java.util.Properties** zurück. Diese ist von **java.util.Hashtable** abgeleitet. Und so können wir wie gewohnt in der foreach-Schleife ein PHP-Hash auflösen.
- Statische Member und Methoden werden über die selbe Syntax angesprochen wie Instanzvariablen und -methoden. Vergleiche die Aufrufe von **getProperties** der Klasse System (static) und **push** der Klasse Stack.

## 4. Praxis-Beispiele

Wir wollen uns im folgenden zwei etwas komplexere Beispiele ansehen, die mit normalen PHP-Mitteln kaum oder gar nicht zu bewerkstelligen sind. Sie sollen Anregungen geben, wo der Einsatz der Java-Extension sinnvoll sein kann.

## 4.1. ZIP-Files erstellen

Das folgende Beispiel erstellt ein ZIP-Archiv mittels der Klassen des Packages **java.util.zip**. Das Skript ist allerdings sehr langsam, bei größeren Archiven kann es gar zum Server-Timeout kommen. Zu jedem Eintrag im ZIP-Archiv wird hier nur der CRC32-Wert gesetzt, normalerweise werden aber auch noch die Zeit der Modifizierung (Methode **setTime** der Klasse **ZipEntry**), die unkomprimierte Größe (Methode **setSize**) und die komprimierte Größe (Methode **setCompressedSize**) hinzugefügt. Mit den Klassen **JarOutputStream** und **JarEntry** des Packages **java.util.jar** lassen sich übrigens JAR-Archive erstellen, auf diese Weise auch von PHP aus!
Werden übrigens keine absoluten Pfadangaben beim ZIP-Archiv und den zu packenden Dateien angegeben, wird ihr Name relativ zum Ort des PHP-Interpreters interpretiert. Der Quelltext der Datei **javazip.php**:

```
<?php

$files = array("test1.txt", "test2.gif");
// Öffne ZipOutputStream
$zip = new Java("java.util.zip.ZipOutputStream", new Java("java.io.FileOutputStream",
"/usr/local/htdocs/test.zip"));
// Kompression Stufe 1
$zip->setLevel(1);
// Eine CRC32-Prüfsumme erzeugen
$crc32 = new Java("java.util.zip.CRC32");

foreach ($files as $entry)
{
        // Neuen ZIP-Eintrag erstellen...
        $zipentry = new Java("java.util.zip.ZipEntry", $entry);
        // ...und dem Archiv hinzufügen
        $zip->putNextEntry($zipentry);
        // zu zippende Dateien auslesen und in den ZipOutputStream schreiben
        $bufin = new Java("java.io.BufferedInputStream", new
Java("java.io.FileInputStream", "/usr/local/htdocs/".$entry));
        while (($byte = $bufin->read()) != -1)
        {
                $crc32->update($byte);   // Prüfsumme aktualisieren
                 $zip->write($byte);
        }
        // Prüfsumme dem Archiv-Eintrag hinzufügen
        $zipentry->setCrc($crc32->getValue());
        $crc32->reset();
        // ZIP-Eintrag abschließen und InputStream schließen
        $zip->closeEntry();
        $bufin->close();
}

// Kommentar setzen und ZipOutputStream schließen
$zip->setComment("ZIP-File zum Testen");
$zip->finish();
$zip->close();

?>
```

## 4.2. Ein raffiniertes Logo

Zuletzt werden wir Gebrauch machen von den Möglichkeiten der Packages **java.awt** und **java.awt.image**, die die Fähigkeiten der mit PHP gebundelten GD-Bibliotheken schlicht alt aussehen lassen. Wir erstellen ein Logo aus einer Schrift, die mit einem Farbverlauf gefüllt wird und einen Schatten wirft. Zuletzt wenden wir auf das ganze Bild einen Weichzeichner an und speichern es als JPEG. Das fertige Bild wird bei relativen Pfadangaben wieder im Verzeichnis des PHP-Interpreter gespeichert. Das Skript läuft nur als CGI. Der Quelltext der Datei **javalogo.php**:

```
<?php

$message = "mad-teaparty.com";
$fontName = "Monospace";
$fontSize = 80;
// Matrix für Weichzeichner
$filterArr = Array(0.1, 0.1, 0.1, 0.1, 0.2, 0.1, 0.1, 0.1, 0.1);

// dicke Monospace-Schrift in Größe 80 erzeugen
$font = new Java("java.awt.Font", $fontName, 1, $fontSize);
// Dummy-Image, um Font-Metrics zu holen
$tempimg = new Java("java.awt.image.BufferedImage", 1, 1, 1);
$tempg2d = $tempimg->createGraphics();
$metrics = $tempg2d->getFontMetrics($font);

// Größe des Bildes aus Länge des Textes berechnen
$messageWidth = $metrics->stringWidth($message);
$imgWidth = ($messageWidth+2*$fontSize)-15;
$imgHeight = $fontSize*2;

// Farben: weißer Hintergrund, grauer Schatten, Farbverlauf von blau nach schwarz
$bgCol = new Java("java.awt.Color", 255, 255, 255);
$shadowCol = new Java("java.awt.Color", 192, 192, 192);
$startCol = new Java("java.awt.Color", 0, 0, 255);
$endCol = new Java("java.awt.Color", 0, 0, 0);

// Buffered Image erzeugen
$img = new Java("java.awt.image.BufferedImage", $imgWidth, $imgHeight, 1);
$g2d = $img->createGraphics();
// Hintergund
$g2d->setPaint($bgCol);
$g2d->fillRect(0, 0, $imgWidth, $imgHeight);

// Schatten zeichnen
$g2d->setFont($font);
$g2d->translate(35, $imgHeight*8/10);          // Koordinatenursprung verschieben
$g2d->setPaint($shadowCol);
$origTransform = $g2d->getTransform();
$g2d->shear(-1.0, 0);                // um 45 Grad nach rechts kippen
$g2d->scale(1, 1.5);                 // um 50% vergrößern
$g2d->drawString($message, 0, 0);
$g2d->setTransform($origTransform);          // Transformation anwenden

// eigentlichen Text mit Farbverlauf schreiben
$g2d->setPaint(new Java("java.awt.GradientPaint", 0, 0, $startCol, $imgWidth-
$messageWidth/10, $imgHeight, $endCol));
$g2d->drawString($message, 0, 0);

// neues Bild zur Anwendung des Filters
$img2 = new Java("java.awt.image.BufferedImage", $imgWidth, $imgHeight, 1);
$g2d = $img2->createGraphics();
$g2d->setPaint($bgCol);
$g2d->fillRect(0, 0, $imgWidth, $imgHeight);
```

```
// Weichzeichner anwenden
$kernel = new Java("java.awt.image.Kernel", 3, 3, $filterArr);
$convolveOp = new Java("java.awt.image.ConvolveOp", $kernel);
$convolveOp->filter($img, $img2);

// JPEG speichern
$out = new Java("java.io.DataOutputStream", new Java("java.io.FileOutputStream",
"test.jpg"));
$codec = new Java("com.sun.image.codec.jpeg.JPEGCodec");
$encoder = $codec->createJPEGEncoder($out);
$param = $encoder->getDefaultJPEGEncodeParam($img2);
$param->setQuality(1.0, false);          // beste Qualität
$encoder->setJPEGEncodeParam($param);
$encoder->encode($img2);

?>
```

Christoph Bichlmeier
E-Mail: chris 'at' bichlmeier 'dot' info
Website: http://www.bichlmeier.info
erstellt am: 17.11.2004