OXID esales

OXID eSales

Konfiguration und Anwendung von Xdebug

30. März 2015

Copyright

Konventionen

In diesem Dokument werden die folgenden typographischen Konventionen verwendet:

`Grau hinterlegte Proportionalschrift`
: Für Benutzereingaben, Quellcode und URLs

Fett kursiv
: Für Dateinamen und Pfade und sonstige kursive Auszeichnungen

Fettschrift
: Für Eingabefelder und Navigationsschritte

Fettschrift dunkelrot
: Für Warnungen und wichtige Hinweise

Impressum

OXID eSales AG
Bertoldstraße 48
79098 Freiburg
Deutschland
Fon: +49 (761) 36889 0
Fax: +49 (761) 36889 29
Vorstand: Roland Fesenmayr (Vorstandsvorsitzender), Dr. Marcus Klosterberg
Vorsitzender des Aufsichtsrats: Michael Schlenk
Sitz: Freiburg
Amtsgericht Freiburg i. Brsg.
HRB 701648

Inhaltsverzeichnis

1. Einleitung

Debugger sind hilfreiche Werkzeuge bei der Programmierung. Zum einen können damit Fehler analysiert und behoben werden. In komplexen Softwaresystemen sind sie zudem hilfreich, um die Ausführung zu beobachten und so den Ablauf einer Anwendung besser zu verstehen. Dieser Artikel beschreibt Installation und Benutzung von Xdebug in Verbindung mit den beiden Entwicklungsumgebungen:

- PhpStorm (https://www.jetbrains.com/phpstorm/)
- Eclipse (http://www.eclipse.org/)

2. Installation

Zunächst muss Xdebug heruntergeladen und in der PHP-Konfiguration eingetragen werden. In den meisten Linux-Distributionen sollte sich Xdebug in den Software-Repositorys befinden und einfach installieren lassen. Für eine manuelle Installation gibt es ein Tool, zu finden unter http://xdebug.org/wizard.php, das den Output der phpinfo()-Funktion analysieren kann, um dann individuelle Installationsanweisungen zu generieren.
Nachdem Xdebug heruntergeladen wurde, muss die Konfiguration angepasst werden. Entweder steht die Konfiguration direkt in der *php.ini* oder es gibt spezielle ini-Dateien für alle Erweiterungen, unter Debian z.B. */etc/php5/mods-available/xdebug.ini*. Eine minimale Konfiguration könnte so aussehen:

```
zend_extension=/usr/lib/php5/20100525/xdebug.so

xdebug.remote_enable=1
```

Nach dem Neuladen der Konfiguration, z.B. durch einen Apache-Neustart, sollte in der Ausgabe der phpinfo()-Funktion folgender Abschnitt zu finden sein:

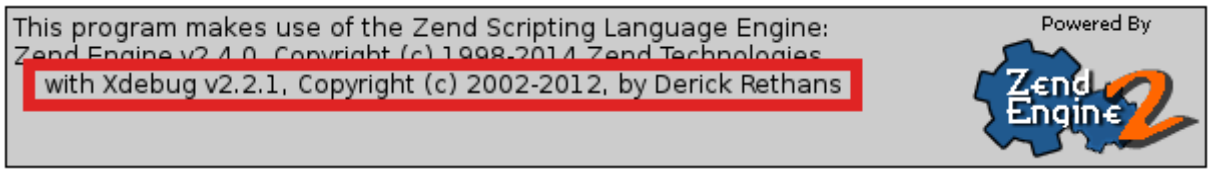

Abbildung 1: Auszug aus der phpinfo()

Für weitere Konfigurationsparameter sei auf folgende Dokumentation verwiesen:
http://xdebug.org/docs/all_settings.
Um diesen Beitrag nachvollziehen zu können, sollte aber die genannte Minimalkonfiguration ausreichen, da alle Einstellungen mit sinnvollen default-Werten belegt sind.

3. Entwicklungsumgebungen und Xdebug

Nachdem Xdebug auf dem Webserver installiert ist, muss die Entwicklungsumgebung eingerichtet werden. Beschrieben wird hier die Einrichtung unter den beiden populären Entwicklungsumgebungen Eclipse und PhpStorm.

3.1 Eclipse einrichten

Grundlage dieses Beitrags ist Eclipse Luna in der derzeit aktuellsten Version. In Eclipse lässt sich die Unterstützung von PHP und Xdebug leicht per Plugin (PHP Development Tools: PDT) nachrüsten. Oder man lädt direkt die Version für PHP-Entwickler herunter. Aufgrund eines Bugs in der Version 3.3.1 der PDT, die derzeit standardmäßig in Eclipse enthalten ist, (https://bugs.eclipse.org/bugs/show_bug.cgi?id=445680), ist die Benutzung von Xdebug sehr eingeschränkt. Deshalb empfiehlt es sich, direkt die PDT-Version des PDT-Projekts (derzeit 3.4.0) zu benutzen. Dazu klickt man in Eclipse auf Help → Install New Software... und trägt dort über den Button Add... die Projektseite von PDT ein.

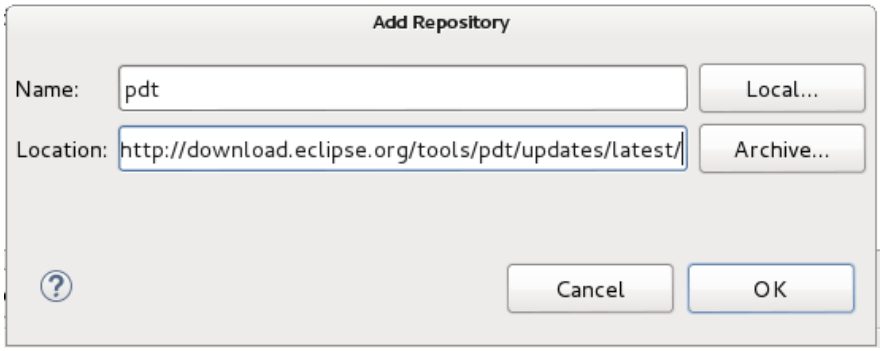

Abbildung 2: PDT Downloadseite

Anschließend kann die aktuellste Version der PDT ausgewählt und installiert werden. Danach kann noch eine Einstellung vorgenommen werden, damit Eclipse den Systembrowser zum Debuggen startet, der komfortabler zu benutzen ist, als der in Eclipse eingebaute Browser. Zu finden ist dies im Einstellungsmenü unter:
General → Web Browser → Use external web browser.

3.2 PhpStorm einrichten

In der aktuellen PhpStorm Version 8.0.3 muss keine weitere Software installiert werden, um mit Xdebug arbeiten zu können. Zunächst muss in den Einstellungen unter Languages & Frameworks → PHP der richtige Interpreter

eingestellt werden. Vermutlich ist hier zunächst noch *<no interpreter>* ausgewählt. Uber den Button '...' kann in dem sich öffnenden Fenster über das Plus-Symbol der PHP-Interpreter hinzugefügt werden.
Unter **Languages & Frameworks → PHP → Debug** sollte alles richtig eingestellt sein.
https://confluence.jetbrains.com/display/PhpStorm/Zero-configuration+Web+Application+Debugging+with+Xdebug+and+PhpStorm

4. Verwendung von Xdebug

Es gibt mehrere Möglichkeiten, um die Xdebug-Ausführung zu starten, z.B. durch einen an die URL angehängten Parameter oder durch das Setzen eines Cookies. Startet man das Debugging in Eclipse, öffnet Eclipse das Projekt im Browser und hängt den passenden Parameter an die URL.

4.1 Starten in Eclipse

Zunächst muss in Eclipse ein Projekt so angelegt werden, dass dessen Dateien durch den Webserver erreichbar sind. Dann wird z.B. eine Datei index.php mit folgendem Inhalt angelegt:

```
<?php
function getName()
{
     return 'OXID';
}
function hello($name)
{
     echo 'Hello ' . $name . PHP_EOL;
}
hello(getName());
```

Da Eclipse noch andere Debugger kennt und ggf. diese als Standard gesetzt sind, muss vorher noch der Standarddebugger auf Xdebug gesetzt werden. Das kann auch nachträglich geändert werden, aber das Setzen des Standarddebuggers sorgt dafür, dass dies später nicht vergessen wird. Die Einstellungen sind zu finden unter **Window → Preferences** und im sich dann öffnenden Fenster unter **PHP → Servers**. Editiert man den dortigen Standardserver, kann der Debugger auf Xdebug gestellt werden.
Per Rechtsklick auf die angelegte PHP-Datei startet mit **Debug As → 2 PHP Web Application** der Debugger. Es sollte sich ein Fenster öffnen, in dem die URL angegeben werden muss, über die die Datei abzurufen ist. Nach der

Bestätigung öffnet Eclipse den Browser und wechselt in die Debug-Perspektive, was man ggf. noch bestätigen muss. Eclipse ist standardmäßig so eingestellt, dass der Debugger die Ausführung in der ersten Zeile anhält.

4.2 Starten in PhpStorm

Eine weitere Möglichkeit Xdebug zu starten ist das Setzen von Cookies. Auf der Internetseite von PhpStorm kann man sich Bookmarklets generieren lassen, die per JavaScript entsprechende Cookies zum Starten oder Stoppen der Debug-Session setzen oder löschen (https://www.jetbrains.com/phpstorm/marklets/). Diese Bookmarklets können als Lesezeichen gespeichert werden. Um diese zu benutzen, ruft man im Browser die PHP-Datei auf, die analog zur Beschreibung zu Eclipse, diesmal in einem neuen Projekt in PhpStorm angelegt wird. Anschließend wird das Bookmarklet zum Starten des Debuggers aufgerufen, wodurch das Cookie gesetzt wird, das beim nächsten aktualisieren der Seite PHP sagen wird, dass Xdebug gestartet werden soll.
Damit PhpStorm auch auf den gestarteten Debugger reagiert, muss **Run → Start Listening for PHP Debug Connections** aktiviert werden. Aktiviert man zusätzlich noch **Run → Break at first line in PHP scripts**, hält auch PhpStorm den Debugger sofort in der ersten Zeile an.

4.3 Funktionen

Die Bedienung von Xdebug läuft in allen Entwicklungsumgebungen ähnlich, egal ob Eclipse oder PhpStorm. Die Toolbars enthalten dieselben Funktionen.

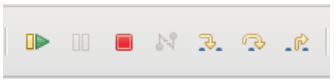

Abbildung 3: Eclipse Toolbar

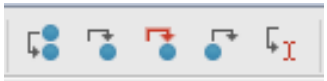

Abbildung 4: PhpStorm Toolbar

Die wichtigsten Funktionen sind:

- Step Over
- Step Into
- Step Out (bzw. Return)
- Resume

Nach der vorherigen Beschreibung sollte Xdebug gestartet und die Ausführung in der ersten Zeile angehalten worden sein. Jetzt ist es möglich, die Ausführung Zeile für Zeile zu beobachten. *Step Over* und *Step Into* gehen jeweils einen

Schritt in der Ausführung weiter. Steht in der auszuführenden Zeile z.B. ein Funktionsaufruf, wird *Step Into* in die Funktion springen und die Ausführung dort fortsetzen. *Step Over* springt in die nächste Zeile, egal was in der aktuellen Zeile ausgeführt werden wird und lässt so z.B. die Anzeige der Ausführung von Funktionen aus. In Abbildung 5 steht die Ausführung gerade vor Zeile 13. *Step Over* führt diese Zeile aus und da es die letzte Zeile ist, ist die Ausführung danach beendet. *Step Into* würde zunächst in die Funktion *getName()* hineinspringen und die Ausführung lässt sich ab Zeile 5 weiterverfolgen.

```
1  <?php
2
3  function getName()
4  {
5      return 'OXID';
6  }
7
8  function hello($name)
9  {
10     echo 'Hello ' . $name . PHP_EOL;
11 }
12
13 hello(getName());
```

Abbildung 5: Während der Ausführung

Steht die Ausführung aktuell in einer Funktion, wird *Step Out* die Funktion beenden und zur aufrufenden Stelle springen. Ist die Ausführung also z.B. in der Funktion *getName()* in Abbildung 5 wird *Step Out* zu Zeile 13 zurückspringen, auch wenn die Funktion aus weiteren Zeilen bestehen würde. *Resume* führt die Ausführung fort, bis die Ausführung wieder unterbrochen wird.

Bis hierhin wurde die Ausführung in der ersten Zeile des PHP-Skripts angehalten und die Kontrolle an den Debugger übergeben. Normalerweise möchte man aber nicht direkt an der ersten Zeile die Ausführung anhalten, sondern gezielt an einer Stelle stoppen, die genauer untersucht werden soll. Dazu wird im Quelltext einfach links neben die Zeile geklickt. Das funktioniert sowohl in Eclipse als auch in PhpStorm. Dort wird ein Breakpoint gesetzt. Läuft die Ausführung, weil sie z.B. über *Resume* fortgesetzt wurde, und es wird ein solcher Breakpoint in der Ausführung erreicht, hält die Ausführung an, und der Benutzer kann die Kontrolle übernehmen.

Mit den bis hierhin beschriebenen Mitteln lässt sich der Ausführungspfad verfolgen. In der Abbildung 6 ist PhpStorm während der Ausführung zu sehen, wobei ein Breakpoint in die Funktion *\oxArticle::getPictureUrl()* gesetzt wurde, so dass man im Bild die unterbrochene Ausführung an dieser Stelle sieht. Auf der linken Seite sieht man alle zu diesem Zeitpunkt aktiven Funktionsaufrufe, auf der rechten Seite ist der Inhalt aller Variablen zu diesem Zeitpunkt zu sehen. In Eclipse sieht diese Anzeige ähnlich aus (Abbildung 7). Dadurch lässt sich nicht nur der Ablauf beobachten, sondern es lassen sich auch wichtige Informationen zum aktuellen Zustand auslesen.

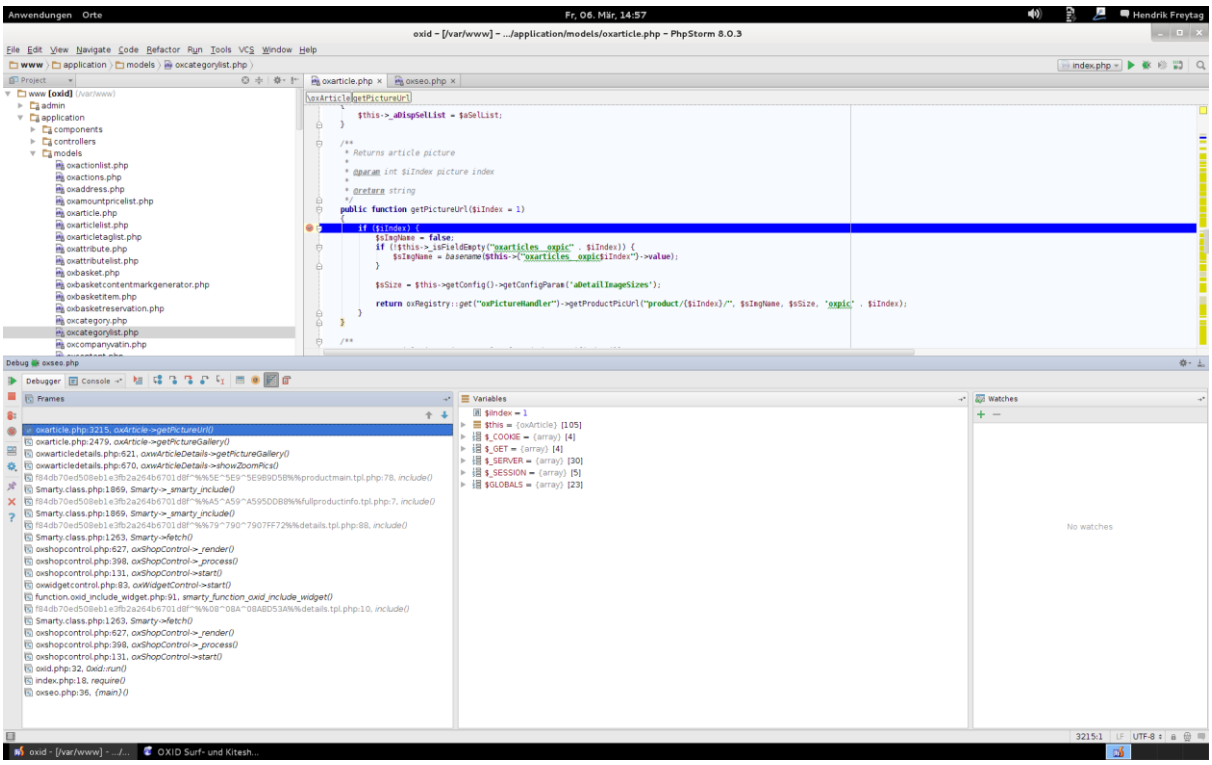

Abbildung 6: Während der Ausführung (PhpStorm)

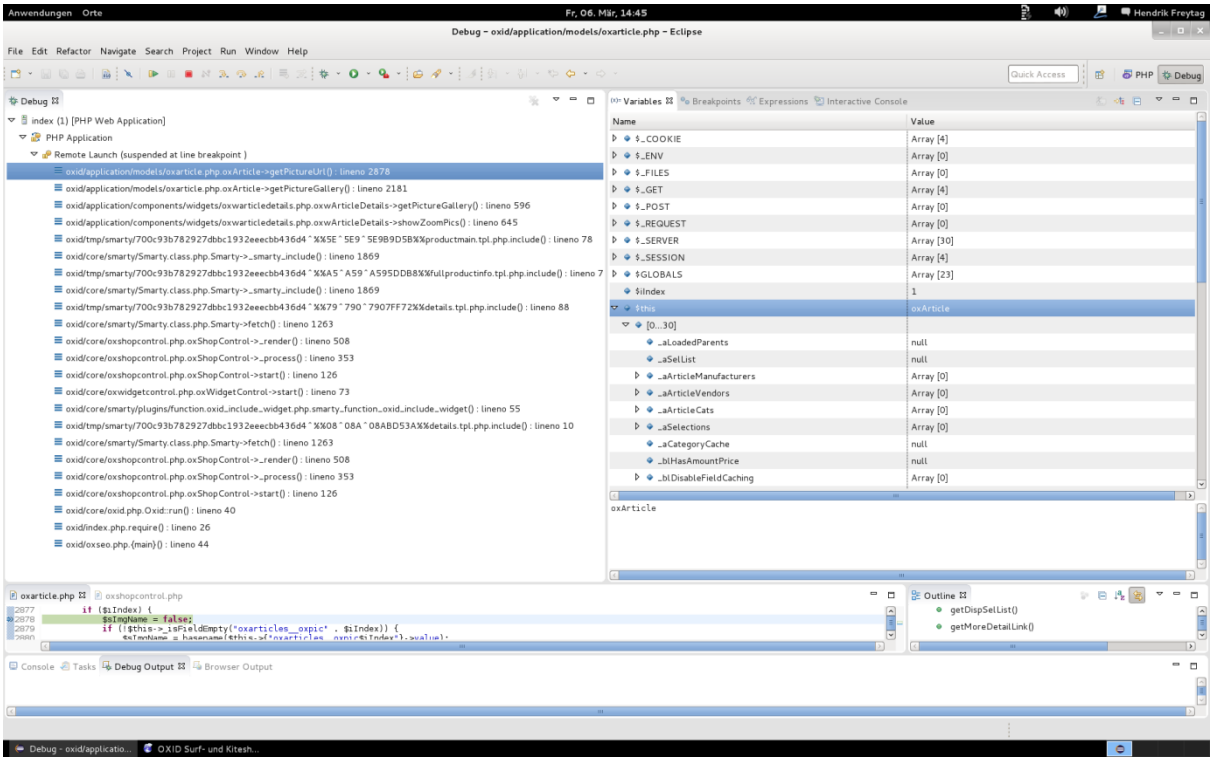

Abbildung 7: Während der Ausführung (Eclipse)