Izzy*Soft* präsentiert das

IMDBPHP v1.1.2

Handbuch

Table of Contents

Installation

Im Prinzip ist das Archiv einfach irgendwo innerhalb des *DOCUMENT_ROOT* zu entpacken - und man kann sofort mit der Anwendung beginnen (zumindest was die enthaltenen Demo-Seiten betrifft). Dann wäre zwar der Cache abgeschaltet, und die Bilder-Funktionalität wäre nicht gegeben - aber alles andere wäre sofort einsatzbereit. Doch spätestens wenn die API in der eigenen Applikation zum Einsatz kommen soll, ist ein wenig mehr "Finetuning" erforderlich.

Welche Installationsmethode wird empfohlen?

Das hängt ganz davon ab, auf welchem System IMDBPHP zum Einsatz kommen soll. Handelt es sich dabei um eine Linux-Distribution, die RPM- oder Debian-Pakete unterstützt (am besten noch mit APT oder YUM), sollte die Installation auch über diese Pakete erfolgen (mehr Details finden sich auf den Repository-Seiten). Handelt es sich um eine Distribution, die dies nicht unterstützt, lässt sich auch nach dem Entpacken des Archives eine Installation mit `make install` durchführen.

Diese drei Methoden haben einige Vorteile gegenüber anderen Alternativen:

- die Dateien landen automatisch am richtigen Platz
- der Cache und Bilderspeicher werden eingerichtet
- eine De-Installation ist ebenfalls automatisch möglich

Hinzu kommt im Falle der Installation über das Repository, dass auch Updates automatisch über das Paketsystem durchgeführt werden.

Kann keine dieser Varianten zum Einsatz kommen (etwa weil man Windows einsetzt), bleibt noch die manuelle Installation. Details folgen auf dieser Seite:

Welche Dateien werden benötigt?

Die API besteht aus den `*.class.php` Dateien, die gemeinsam im gleichen Verzeichnis installiert werden müssen:

- `browseremulator.class.php` - wird von der API zur Kommunikation mit anderen Webservern (zur "Informationsbeschaffung") genutzt
- `imdb_base.class.php` - *Basis-Klasse*, auf der die anderen Klassen aufbauen
- `imdb_config.class.php` - die Konfigurationsdatei
- `imdb.class.php` - API Klasse für die Filmdetails
- `imdb_charts.class.php` - API Klasse für *Top-Charts*
- `imdb_nowplaying.class.php` - was spielt gerade in US-Kinos?
- `imdb_person.class.php` - API-Klasse für Informationen zum "Staff" (Schauspieler, Regisseure, etc.)
- `imdb_request.class.php` - ein Wrapper für den Browser-Emulator
- `imdb_trailers.class.php` - API Klasse, um die tatsächlichen Trailer-URLs zu ermitteln

Bei den anderen Dateien handelt es sich entweder um Dokumentation oder Demo-Dateien, die von der API selbst nicht genutzt werden.

Wo sollten diese Dateien installiert werden?

Das hängt wiederum von der Nutzung ab. Soll die API fest in der eigenen Applikation verankert werden, werden sie wahrscheinlich innerhalb von deren Verzeichnis installiert. Andererseits bietet es sich auch hier an - insbesondere, wenn über das Paketsystem installiert wurde - das *außerhalb* der eigenen Applikations-Verzeichnisses befindliche Installationsverzeichnis in den PHP `include_path` aufzunehmen (entweder über die `php.ini`, oder z.B. die Apache-Anweisung `php_value include_path`). Somit kann man *IMDBPHP* einfach als Abhängigkeit angeben, und profitiert wiederum automatisch von etwaigen Updates. Seit Version 0.9.5 werden die `*.class.php` Dateien bei automatischer Installation (Paketmanagement bzw. `make install`) im Verzeichnis `/usr/share/php` abgelegt, welches bei den meisten Distributionen bereits im `include_path` der `php.ini` eingetragen ist (da ***PEAR*** hier ebenfalls installiert wird).

Was muss konfiguriert werden?

Prinzipiell sollte *IMDBPHP* "Out-of-the-Box" funktionieren. Dennoch mag das Bedürfnis bestehen, einige Dinge anzupassen. Daher soll die Konfiguration an dieser Stelle auch besprochen werden, die - wie zu erwarten - in der Datei `imdb_config.class.php` stattfindet.

Alle Konfigurations-Optionen in der `imdb_config.class.php` sind mit dem Präfix `$this->` versehen, da es sich hier um eine PHP-Klasse handelt. Der Übersichtlichkeit halber verzichten wir im Folgenden auf diesen Präfix, was die Angaben auch leichter lesbar macht:

Allgemeine Optionen

Option	Beschreibung
imdbsite	Die bevorzugte IMDB-Site. Die Voreinstellung ist hier `akas.imdb.com`, da diese auch ***alle*** sprachspezifischen Filmnamen und AKAs kennt. Diese Option kann natürlich auch zur Laufzeit geändert werden, wenn ein Serverwechsel stattfinden soll.
debug	Sollen Debug-Meldungen angezeigt werden (1) oder nicht (0)?

Cache Optionen

Option	Beschreibung
cachedir	In welchem Verzeichnis sollen gecachte Informationen gespeichert werden (sofern der Cache aktiviert ist). Es kann hier ein relativer oder auch absoluter Pfad angegeben werden.
usecache	Wenn eine angeforderte Seite im Cache verfügbar ist, sollen die gecachten Information dann genutzt (TRUE), oder soll die Seite erneut von Original-Server angefordert werden (FALSE)?
storecache	Wenn eine Seite von Original-Server bezogen wurde, soll sie für die spätere Nutzung im Cache abgelegt werden? Dies setzt voraus, dass der Webserver auf das Cache-Verzeichnis auch Schreibzugriff hat.
usezip	Soll der Cache komprimiert werden?
converttozip	Sofern bei einem Zugriff auf den Cache unkomprimierte Information entdeckt wird, soll sie komprimiert werden?
cache_expire	Für wie viele Sekunden ist eine gecachte Seite gültig? Ältere Informationen werden bei "Entdeckung" (d.h. beim nächsten Cache-Zugriff) automatisch gelöscht.

Image Optionen

Option	Beschreibung
photodir	Das Verzeichnis, in dem Bilddateien abgelegt werden sollen. Dies sollte sich innerhalb des `DOCUMENT_ROOT` befinden, sofern die Bilder direkt auf einer Webseite zur Anzeige kommen sollen - und natürlich benötigt der Webserver auch auf dieses Verzeichnis Schreibberechtigung.
photoroot	Die URL, die dem `photodir` entspricht - bei Angabe eines absoluten Pfades muss selbiger auf dem `DOCUMENT_ROOT` aufbauen: Ist das `DOCUMENT_ROOT` z.B. `/var/www`, und die Bilddateien werden in `/var/www/photos` gespeichert, wäre die zugehörige absolute URL `/photos/`.

Tweaking Optionen

Option	Beschreibung
maxresults	Hiermit lässt sich die Ergebnismenge für eine Filmsuche begrenzen. Ein Wert von "0" bedeutet "keine Einschränkung" - jeder andere Wert (natürlich größer als 0) gibt die Anzahl maximal erwünschter Treffer an.
searchvariant	Wer mit den Suchergebnissen nicht zufrieden ist, kann mit diesen Such-Varianten spielen - sie setzen jeweils unterschiedliche Optionen für die Filmsuche. Verfügbare Varianten sind `sevec` und `moonface`, der Default ist `izzy`.

Alternative Konfigurationsdatei

Ab Version 1.0.8 (bzw. Revision r170) lässt sich auch eine alternative Konfigurationsdatei verwenden - um z.B. verschiedene Projekt-bezogene Konfigurationen verwenden zu können. Diese sollte natürlich den gleichen Aufbau haben wie die mitgelieferte, und wird verwendet, wenn man **vor dem Einbinden der Datei `imdb_base.class.php`** deren Pfad mit der Konstante `IMDBPHP_CONFIG` definiert - z.B. so:

```
<?php
define('IMDBPHP_CONFIG','/var/www/myproject/imdb_config.php');
require_once(imdb_base.class.php);
// more code goes here
?>
```

Dokumentation

Dies ist nur eine kurze Dokumentation, die in die Benutzung der *IMDBPHP API* einführen soll. Es werden hier bei weitem nicht alle verfügbaren Methoden diskutiert (diese finden sich jedoch in der API Referenz) - vielmehr geht es hier um die generelle Nutzung der Klassen. Hat man diese verstanden, erschließt sich der Rest über die API Referenz.

Um in der Dokumentation zu navigieren, kann das *Inhaltsverzeichnis* am rechten Rand genutzt werden.

Filmsuche

Als erstes muss natürlich die entsprechende Klassen-Datei der API eingebunden werden. Dies kann auf vier verschiedene Arten geschehen - welche davon zum Einsatz kommen soll, obliegt dem Programmierer (und Details dafür finden sich in der PHP Dokumentation:

```
<?php
include("imdb.class.php");
include_once("imdb.class.php");
require("imdb.class.php");
require_once("imdb.class.php");
?>
```

`imdb.class.php` kümmert sich dann selbst darum, weitere benötigte Dateien der API einzubinden, die sich natürlich im selben Verzeichnis wie die oben eingebundene Datei befinden müssen.

Wie können wir jetzt nach einem Film suchen? Entweder haben wir den Namen (oder zumindest einen signifikannten Teil davon) - oder wir kennen bereits seine IMDB-ID. Im letzteren Fall entfällt natürlich die Suche, und man kann sich gleich an die Details machen: "Filmsuche" heißt hier nämlich "Suche nach der IMDB ID". Und so besteht das Suchergebnis dann schließlich auch aus einer "Auflistung" von Filmen mit der zugehörigen IMDB-ID.

Suche ausführen

Um die notwendigen Schritte zu beschreiben, nehmen wir am Besten ein Beispiel zur Hand: Sagen wir, wir suchen nach dem Film "Keine halben Sachen". Und sagen wir weiterhin, wir erinnern uns nicht an den richtigen Namen - wissen aber, dass der Titel "halbe Sachen" enthält:

```
<?php
$name = "halbe Sachen";          // der Name wird gewöhnlich dynamisch gesetzt
require_once("imdb.class.php"); // Klassendatei einbinden
$search = new imdbsearch();      // eine Instanz der Such-Klasse erstellen
$search->setsearchname($name);  // der Klasse mitteilen, wonach wir suchen
                                 // (keine Groß-kleinschreibungs-Unterscheidung)
$results = $search->results();  // Suche abschicken
?>
```

Ergebnisse parsen

Nachdem die Suche abgeschlossen ist, finden wir etwaige Ergebnisse in der Variable `$results`. Die API beschreibt den Rückgabewert der Methode `results()` als ein Array[0..n] von Objekten, wobei jedes Objekt eine Instanz der Klasse `imdb` ist (auf die wir bei den Film-Details noch näher eingehen werden) - im Prinzip können wir also jetzt bereits auf alle Details zugreifen. Was aber noch nicht sinnvoll ist - wir wissen ja

noch gar nicht, ob unser gesuchter Film überhaupt in der Ergebnismenge enthalten ist (oder wir uns nicht einmal an den Teil seines Namens richtig erinnert haben). Also wandern wir zunächst durch unser Array, um uns die wichtigsten "Erkennungsdaten" anzusehen:

```
<?php
foreach ($results as $res) {
  $mid  = $res->imdbid();
  $name = $res->title();
  $year = $res->year();
  // now do something with these data
  echo "$mid: $name ($year)<br>\n";
}
?>
```

Aus dieser Liste ist dann der richtige Film auch an seinem vollen Namen (und Erscheinungsjahr) recht klar zu erkennen (und steht in unserem Beispiel sogar an erster Stelle). Der Besucher unserer Seite kann also nun die passende IMDB-ID auswählen. Wir könnten es ihm auch einfacher machen, für einen detaillierteren Vergleich direkt auf die IMDB-Site zuzugreifen, indem wir ihm auch den passenden Link mit anbieten: `http://'".$search->imdbsite."/title/tt".$res->imdbid().`

Da wir die IMDB-ID nun kennen, können wir uns um die gewünschten Details kümmern. Wie das geht, ist hier beschrieben.

Details zum Film

Wie schon bei der Filmsuche, müssen wir auch hier zunächst wieder die Klassendatei einbinden. Um nun Details zu einem Film erhalten zu können, benötigen wir als erstes die IMDB-ID des gewünschten Films. Diese lässt sich mit der Filmsuche ermitteln, oder per Copy'n'Paste aus der URL der zugehörigen IMDB Seite entnehmen. Im Folgenden wird davon ausgegangen, dass die IMDB-ID bereits bekannt ist.

Allgemeines

Die IMDB Sites halten im allgemeinen eine ganze Reihe von Details zum Film bereit - so viele, dass sie selbst pro Film über mehrere Seiten verteilt sind. Eine Liste dieser Seiten findet sich auf der "Titelseite".

Gehen wir zunächst einmal davon aus, dass uns folgende Details interessieren: Filmtitel, Erscheinungsjahr, Laufzeit, Rating, und eine Liste von Trailers. Unsere Abfrage sähe dann so aus:

```
<?php
require_once("class.imdb.php");     // Klassen-Datei einbinden
$movie   = new imdb($mid);          // eine Instanz der `imdb` Klasse erstellen,
                                    // und dabei auch gleich die IMDB-ID übergeben
$title   = $movie->title();         // den Filmtitel abfragen
$year    = $movie->year();          // das Erscheinungsjahr abfragen
$runtime = $movie->runtime();       // die Laufzeit ermitteln (Filmlänge)
$rating  = $movie->mpaa();          // ein Array[country=>rating] verfügbarer "Ratings" abrufen
                                    // (auch als "FSK" bekannt)
$trailer = $movie->trailers();      // ein Array verfügbarer Trailer holen
?>
```

Wollen wir das gleich anschließend für (einen) weitere(n) Filme(e) wiederholen, müssen wir nicht jedes Mal eine neue Instanz der IMDB Klasse erstellen: Mit der Methode `$movie->setid($newid)` können wir die bestehende Instanz anweisen, sich ab sofort um einen anderen Film zu kümmern.

Für weitere Details sei an dieser Stelle auf die API Referenz verwiesen.

Technischer Hintergrund

Dem interessierten Leser sei noch kurz erläutert, was während der einzelnen oben beschriebenen Schritte "im Hintergrund" passiert:

Schritt	Ausführung
require_once("class.imdb.php");	Die Klassendatei wird eingebunden. Implizit werden auch weitere Klassendateien eingebunden, auf denen diese basiert - hier u.a. `imdb_base.class.php`, welche wiederum `imdb_config.class.php` einbindet.
new imdb($mid);	Hier wird eine Instanz der Klasse erstellt, und gleich mit der IMDB-ID initialisiert.
$movie title();	Es wird auf die "Titelseite" des Films zugegriffen. Da dies zuvor noch nicht geschehen ist, resultiert dies in einem Zugriff auf der IMDB-Site selbst. Ist der Cache aktiviert, wird die Seite in selbigem zwischengespeichert. Desweiteren wird die Seite zur Auswertung im Speicher gehalten, bis entweder die Ausführung des Skriptes beendet oder aber die Instanz mit `unset($imdb);` wieder gelöscht wird. Diese "Kopie" im Arbeitsspeicher wird sodann nach dem Filmtitel durchsucht, der wiederum in einer eigenen Variable innerhalb der Instanz abgelegt - und schließlich auch der (lokalen) Variable `$title` zugewiesen wird.
$movie year();	Es wird wiederum auf die "Titelseite" zugegriffen. Da wir diese sogar noch im Speicher gehalten haben, folgt daraus weder ein Zugriff auf den Cache, noch gar auf die IMDB-Site: Es wird einfach die im Speicher gehaltene Kopie nach dem Erscheinungsjahr durchsucht, welches dann wiederum in einer Klassen-Variable abgelegt und auch in unserer Variable `$year` gespeichert wird.
$movie year();	Würde dieser Aufruf hier nochmals erfolgen, bedeutete dies lediglich einen "Übertrag" aus der Klassen-Variable in die jeweilige lokale Variable.
$movie runtime();	Die Laufzeit wird ermittelt - analog dem ersten Aufruf von `$movie->year()`.
$movie mpaa();	Wiederum das gleiche, für die "FSK".
$movie trailers();	Hier wird eine neue Seite benötigt: Die Trailer-Information befindet sich nicht mehr auf der Titelseite. Der Ablauf ist hier also analog zu obigem `$movie->title()` Aufruf.

Das Beispiel zeigt bereits, dass wir so wenig wie möglich auf die (externe) IMDB-Site zugreifen. Selbst spätere Zugriffe auf dieselben Seiten würden zunächst vom Cache abgefangen (sofern er eingeschaltet ist). Außerdem holen wir auch nur die Seiten, die wir für die angefragte Information tatsächlich brauchen - aber das sollte sich von selbst verstehen. Auf diese Weise versucht die IMDB-API bereits von sich aus auf bestmögliche Performance zu achten.

Details zur Person

Wie auch bereits bei der Filmsuche sowie den Filmdetails, muss auch hier natürlich als erstes die Klassendatei eingebunden werden. Um nun Details zu einer Person erhalten zu können, benötigen wir zunächst deren IMDB-ID - wie sie z.B. bei den Filmdetails mitgeliefert wird. Wir gehen hier also davon aus, dass sie bereits bekannt ist.

Allgemeines

Die IMDB-Sites bieten nicht nur massig Informationen zu den Filmen, sondern auch zum "Staff" sind diese so zahlreich, dass sie wiederum auf mehrere Seiten verteilt wurden. Das handelt die IMDB API natürlich in gleicher Weise, wie bereits bei den Filmdetails beschrieben.

Gehen wir für ein Beispiel also davon aus, dass wir zu einer IMDB-Person-ID den Namen, Spitznamen, das Geburtsdatum, die "Mini-Biografie", sowie eine Liste der Filme beziehen wollen, bei der die fragliche Person als Regisseur agierte:

```
<?php
require_once("class.imdb_person.php"); // Klassendatei einbinden
$person = new imdb_person($mid);       // eine Instanz der Klasse erstellen und ID übergeben
$name   = $person->name();             // den Namen erfragen
$birth  = $person->born();             // Geburtsdatum
$nick   = $person->nickname();         // Spitzname
$bio    = $person->bio();              // Array[0..n] an "Mini-Biographien"
$filmo  = $person->movies_director();  // Filme als Regisseur (Array)
?>
```

Auch hier gilt für einen etwaigen Loop über mehrere IMDB-IDs, dass diese einfach mit `$person->setid($newid)` gesetzt werden kann (anstatt jedesmal eine neue Instanz der Klasse anzulegen) - sowie dass weitere Methoden sich der API-Referenz entnehmen lassen.

Technischer Hintergrund

Diesen möchten wir auch hier dem interessierten Leser nicht vorenthalten:

Schritt	Ausführung
require_once("class.imdb_person.php");	Einbindung der Klassendatei - mit impliziertem Einbinden weiterer benötigter Dateien, wie `imdb_base.class.php` und wiederum implizit von dieser `imdb_config.class.php`.
new imdb_person($mid);	Erstellen einer Instanz der Klasse, sowie initialisieren mit der gewünschten IMDB ID.
$person name();	Es wird auf die "Titelseite" zugegriffen. Da dies der erste Zugriff ist, und die Seite sich auch noch nicht im Cache befindet, schlägt der Request bis zur IMDB Site durch. Die bezogene Seite wird natürlich im Cache abgelegt (sofern dieser aktiviert wurde), und ebenfalls in der Instanz selbst im Speicher gehalten. Hier wird sodann nach dem Namen gesucht, der wiederum in einer internen Variable und schließlich auch in der (lokalen) Variable `$name` gespeichert wird.
$person born();	Erneuter Zugriff auf die "Titelseite". Da diese noch innerhalb der Instanz gespeichert ist, resultiert dieser jedoch nicht mehr in einem Zugriff auf die IMDB Site (oder den Cache), sondern die "Kopie im

	Speicher" wird nach dem Geburtsdatum durchsucht - welches dann wiederum in einer Klassenvariable, und schließlich in der lokalen Variable `$birth` abgelegt wird.
$person born();	Ein erneuter Aufruf dieser Methode an dieser Stelle würde lediglich den in der Klassenvariable abgelegten Wert zurückgeben - kein Zugriff auf die IMDB Site oder den Cache, und auch kein erneutes "Parsen" der "Kopie im Speicher".
$person nickname();	Der Spitzname befindet sich nicht auf der Titelseite - also wird hier erneut eine Seite ("bio") von der IMDB Site geholt, im Cache und einer "Kopie im Speicher" abgelegt, letztere nach dem Spitznamen durchsucht, der wiederum in einer Klassen-Variable sowie der (lokalen) Variable `$nick` gespeichert wird. Im Prinzip also genau das gleiche wie beim Aufruf für `$person->name();`, nur für eine andere Seite.
$person bio();	Ähnlich dem ersten Aufruf von `$person->born();`, da sich die Daten auf der bereits bezogenen "bio" Seite befinden.
$person movies_director();	Hier greifen wir erneut auf die "Titelseite" zu. Da wir diese bereits zuvor im gleichen Lauf abgefragt haben, bedeutet dies auch nur ein erneutes Parsen der "Kopie im Speicher", sowie dem Ablegen des Ergebnisses in einer Klassenvariable sowie der lokalen Variable `$filmo`.

Wie wir bereits gesehen haben, versucht die API auch hier so effizient wie möglich vorzugehen: Auf die IMDB Site wird nur zugegriffen, wenn es sich nicht vermeiden lässt, und alles wird dynamisch zwischengespeichert.

imdbXML

Die **imdbXML Klasse** kann benutzt werden, um IMDB Objekte im XML Format zu exportieren: Alle Objekt-Attribute werden dabei in XML Tags konvertiert. Da diese Klasse auf der *XML_Serializer* Klasse von *PEAR* basiert, muss *PEAR* natürlich auch installiert sein.

Das folgende Code-Fragment stellt eine beispielhafte Benutzung dar. Speichert man diesen in eine `*.php` Datei und ruft diese dann im Browser auf, gibt sie eine XML-Datei mit den Filmtitel, Jahr und der Plot-Outline für die als Parameter übergebene IMDB-ID aus:

```
<?php
require_once("imdbXML.php");

$imdb = new imdb($_REQUEST['id']);
$imdb->title();
$imdb->year();
$imdb->plotoutline();

$obj = new imdbXML($imdb);
$xml = $obj->getSerializedData();

header ("content-type: text/xml");
header("Content-Length: ".strlen($xml));
echo $xml;
?>
```

Tuning Hinweise

Hier ein paar Hinweise, wie man eine bessere Performance erreichen/faken kann:

- Nie alle möglichen Details auf einer Seite konzentrieren - die Ladezeiten wären zu hoch. Besser ist es, die Informationen über mehrere Seiten zu verteilen
- Wer dies ignorieren möchte, sollte zumindest ab und an ein `flush();` Statement in seinen Code einbauen, sodass der Besucher der Seite zumindest sieht, dass etwas passiert (siehe `imdb.php` Beispielseite)
- Den Cache benutzen: Dies beschleunigt folgende Zugriffe auf die gleichen (IMDB) Seiten. Die *Cache Expiry* sollte dabei auf einen sinnvollen Wert eingestellt sein: Ist der Wert zu groß, erhält der Besucher u.U. "veraltete" Informationen (und außerdem wird mehr Platz auf der Platte gebraucht) - ist er zu klein, leidet die Performance. Einen absoluten idealen Wert gibt es natürlich nicht - ein guter Wert liegt jedoch etwa zwischen einer Stunde und einem Tag.
- Soweit möglich, sollten die Informationen einer Seite sich auf eine bis wenige IMDB Seiten beschränken. So müssen weniger "Requests" an die IMDB Site geschickt werden. Außerdem werden einmal ermittelte Informationen in der Klasse auch (für die Laufzeit des Skriptes) zwischengespeichert, um entsprechende *Parse*-Aufrufe zu minimieren.
- Werden gerade bezogene Informationen auf einer Seite mehrfach benötigt werden, kann man sie auch in einer (lokalen) Variable speichern. Werden sie für weitere Aufrufe erneut benötigt, bietet sich auch ein Speichern in Session-Variablen oder einer (lokalen) Datenbank an. Damit ist der Zugriff zumeist schneller als bei einem erneuten Parsen - und mit Sicherheit dann, wenn der Cache abgeschaltet ist.

Testseiten

Um möglichst einfach feststellen zu können, ob irgend eine Methode z.B. durch Änderungen auf der IMDB site "kaputt gegangen" ist, verfügt IMDBPHP auch über ein "Test-Modul". Dieses ist mit Datensätzen konfiguriert, von denen wir wissen, dass die IMDB Sites die entsprechenden Daten bereithalten - sodass wir zumindest erwarten können, auch ein Ergebnis zu bekommen (auch wenn der Inhalt einer Beschreibung sich durchaus ändern kann). Dieses "Test-Modul" befindet sich im Unterverzeichnis `test/`, und sollte dort auch einfach mit dem Browser aufrufbar sein (sofern sich das ganze Paket innerhalb des `DOCUMENT_ROOT` des Webservers befindet). Eventuell muss die URL noch um den Dateinamen `index.php` erweitert werden, falls diese nicht aufgrund der Webserver-Konfiguration automatisch verwendet wird. Nun sollten die verfügbaren Methoden aufgelistet werden - zusammen mit einem grünen "PASS" (wenn sie funktioniert) bzw. einem roten "FAIL" (im Fehlerfall).

Cache

Um sicherzustellen, dass eventuelle Probleme nicht etwa mit dem lokalen Caching zu tun haben, kann man diesen explizit an- oder abschalten, indem man der URL die entsprechenden Parameter mitgibt: `?cache=off` schaltet ihn aus, `?cache=on` aktiviert ihn explizit.

Tests einschränken

Es ist auch möglich, die Tests auf einzelne "Sektionen" einzuschränken: `?check=movie` würde z.B. nur die Methoden der Hauptklasse ausführen, während sich `?check=name` auf die Methoden der Klasse `imdb_person` beschränken würde. Natürlich lässt sich dies auch mit der o.g. Cache-Anweisung verbinden, z.B. als `?check=movie;cache=off` oder `?check=name&cache=on`. Natürlich muss als Options-Trenner das verwendet werden, was in der `php.ini` konfiguriert wurde - die beiden Beispiele zeigen die meistverwendeten Trennzeichen, das ";" und das "&".

Folgende Werte sind für die Variable `check` möglich:

Wert	Bedeutung
movie	Methoden der Hauptklasse (movie methods)
name	Methoden der Klasse `imdb_person`
charts	Methoden der Klasse `imdb_topcharts`
nowplay	Methoden der Klasse `imdb_nowplaying`
trail	Methoden der Klasse `imdb_trailers`

Anhang

Diese Seite fasst verschiedene Themen zusammen:

- IMDBPHP — kurzer Überblick
- FAQ — Fragen und Antworten zu IMDBPHP
- ProjectsUsingThis — einige Referenzen
- IzzySoft — Wer steht hinter dem Projekt?

IMDBPHP

Was ist IMDBPHP?

IMDBPHP stellt eine API (Schnittstelle) zu den Daten von IMDB.COM dar, und richtet sich in erster Linie an Programmierer, welche Informationen von dort in ihr Programm/ihre Website einbinden wollen. Wie der Name schon andeutet, ist diese "Library" in PHP geschrieben - und daher vor allem für PHP Anwendungen interessant. So wird es auch bereits in verschiedenen Anwendungen genutzt - einige Beispiele finden sich u.a. hier aufgelistet.

Features

IMDBPHP bietet die Suche nach einem Film (mit Auflistung aller Treffer) an. Die Trefferliste enthält sowohl Links direkt zur IMDB site, als auch zu einer eigenen Detailseite. Ist die IMDB-ID erst einmal bekannt, lassen sich mit der Library gezielt einzelne Details zum Film abfragen: Darsteller, Regisseur, Sprachen, Ratings u.a.m.. Ebenfalls kann man gezielt Bilder zum Film, sofern auf der IMDB Site vorhanden und verlinkt, laden. Welche Details sonst noch verfügbar sind, kann man in der Demo erkunden.

Entwicklung

Weiterentwicklung heißt bei der IMDB Site in erster Linie, mit den ständigen und zahlreichen Änderungen auf dem Laufenden zu bleiben, welche den Code sonst oftmals mehrfach kurz hintereinander unbrauchbar machen. Wenn daneben noch Zeit bleibt, bauen wir auch noch was ein - was meistens bedeutet, mehr Details aus den IMDB Seiten herauszuholen.

Vielleicht habe (oder bekomme) ich noch mehr Ideen - aber vielleicht hat ja auch noch jemand anders welche? In dem Fall: einfach ein Ticket aufmachen, und wer weiß: Mit einer der nächsten Versionen von IMDBPHP sind sie vielleicht bereits umgesetzt!

IMDBPHP FAQ

Do I have to install/compile imdbphp?

You might have been confused by the `doc/install.txt` file, describing the installation with `make`. Even though `make` can be used to install (and uninstall) IMDBPHP, there is nothing to compile - since, as you know, PHP is a script language which "compiles" at runtime. So you may as well use the "old-fashioned" way described in `doc/README`, and simply unpack the archive. Most likely you will prefer this way when your website provider didn't grant you `ssh` access, and your only way is called `ftp`. For details on how to install IMDBPHP, please see the Installation page.

Caching seems not to work

This is usually caused by not reading the installation instructions. You probably only downloaded the `.tar.gz` variant of the distribution, unpacked it, and then just started using it. Please check that you configured the cache directory correctly (see Caching Options for more details).

There are no images displayed

Same issue as with the cache above. Please make sure your image directory is configured correctly (see Image Options for details).

API is suddenly broken

The API worked perfectly for months - and now it suddenly is broken? Oh well, the code lives I mean, the code of the IMDB sites. Since IMDBPHP parses the HTML source of those pages, changes to the IMDB site often has more or less heavy side-effects to its results. So after most site changes, we have to check and fix our API. This is nothing to worry about too much - if you tap into such a trap, simply open a new ticket here and report (but please check before that exactly this case has not been reported just a few minutes before , since the authors may not always be aware.

IMDBPHP seems to return the wrong main genre

Yepp - agreed, it does. Sometimes. Or even most times. But we cannot do anything against this: IMDB.COM just gives a list of genres associated with the movie, and has this list in alphabetical order. So the genre() method simply returns the first mentioned genre - chances that this is the real main genre are about 1:3 to 1:5 (depending on whether there are 3 or 5 genres associated). If somebody has an idea how to retrieve the real main genre, we are open to new ideas!

Why doesn't the API reference list properties?

The API reference only lists methods, but no properties - with the exception of the configuration class. The reason why the properties are not mentioned there is simple: You should not reference them directly - consider them all to be private, and use the listed methods to retrieve the data. The policy not only in this project is: Public methods/properties (i.e. those explicitly listed in the API reference without being marked private) should always be downwards compatible when a new version is released, while private methods/properties may be subject to change. So to avoid problems on a future update, stick to the public stuff.

Is there an Update Notifier for new releases?

So you don't want to miss a release? No problem. You have several options:

- Using a Debian or RPM based Linux distribution, you can include the APT/YUM Repository in your APT/YUM configuration. More information can be found following the link.
- If you are a Freshmeat member, you can subscribe to the IMDBPHP project there, and receive an email on every release. Please, don't forget to rate the project!
- Similar steps apply to SourceForge: You can subscribe to the IMDBPHP project there, and an email will be sent to you on every update
- You can subscribe to the IzzySoft RSS Newsfeed to have the always latest release information for all IzzySoft projects

- You can subscribe to the projects Timeline RSS feed to see all changes (including releases)
- Not enough? Version junkie? You also can checkout the latest development code from the SVN repository (see below), and create a scheduler (e.g. Cron) job to update that copy regularly

How can I check out the latest development code?

Sometimes you may not want to wait for the next release to appear - either since there is a new feature you immediately want to use, or some minor bugfix which affects you and is not yet released. Or you are simply a version junkie Never mind: You always can check out the latest development code anonymously with Subversion:

```
svn co http://projects.izzysoft.de/repos/imdbphp/trunk
```

If you want the latest code in order to supply YOUR changes - i.e. you want to take place in development, you better contact the author as mentioned in the documentation - since the above mentioned link to the repository is read-only…

Projects using IMDBPHP

Applications and AddOns

Application	Description
phpVideoPro	PHP application to manage your collection of DVDs, VideoCDs, VHS tapes, etc.
AS3 API	ActionScript API
IMDBPlugIn	WordPress PlugIn to retrieve IMDB movie info on-demand (see also here for a page using it)

Online Services

Service	Description
http://www.divxforever.in/	Movie information and ed2k links
MTDb	The *Movie Trends Database* (a project at the MIT)

Your project is missing here? Then drop a message to the project admin, providing an URL plus a short description!

IzzySoft

IzzySoft wurde 1995 neben meinem Informatik-Studium gegründet - zum einen, um der "drögen Theorie" auch etwas Praxis zur Seite zu stellen; zum anderen aber natürlich auch, um das Studium mit zu finanzieren. Anfangs lag der Schwerpunkt auf Web-Entwicklungen sowie Installation, Konfiguration und Wartung von Computern und kleineren Netzwerken.

Das änderte sich etwa 2002/2003, als meine letzte Festanstellung endete - und ich mich dafür entschied, zu 100% freiberuflich zu arbeiten. Seitdem konzentriert sich IzzySoft hauptsächlich auf Oracle Support, was sowohl Tätigkeiten vor Ort (Installation, Konfiguration, Tuning, etc.) als auch die Entwicklung einiger kleiner Tools (die schließlich unter die GPL gestellt und veröffentlicht wurden) beinhaltet.

Weitere Information

- IzzySoft Homepage
- IzzySoft Oracle Entwicklungen und Dokumente
- IzzySoft Software
- IzzySoft *.deb/*.rpm Repositories