Izzy*Soft* proudly presents the

IMDBPHP v1.1.2

Manual

Table of Contents

Installation and Configuration

Basically, you simply need to extract the files from the distribution archives main directory somewhere into your web tree, and could start straight away (at least with using the demo). Well, in that case the cache would be disabled, and images wouldn't work - but everything else would be ready. But at the lastest when you want to use the API with your applications, we would need some "fine tuning".

What is the recommended installation method?

That depends very much on your environment. If you are running a Linux distribution supporting `*.deb` or `*.rpm` packages (optionally with an APT or YUM repository), you should use these packages (for the repository, please visit http://apt.izzysoft.de/ to find more details). If you cannot use one of these two package types, the Tar-Archive contains a `Makefile` you can use by simply changing to the directory containing the unpacked archive and call `make install`.

There are a few advantages above mentioned installation methods have in common:

- all files will automatically be put to the right places
- caching and images will automatically be setup properly
- everything can be easily cleanly uninstalled

If none of these automatic installation methods can be used, you will have to install the software manually. Just read on for more details.

Which files are really needed?

The API consists of all the `*.class.php` files, which must reside together in the same directory:

- `browseremulator.class.php` - used to communicate with other web servers
- `imdb_base.class.php` - common methods for all classes
- `imdb_config.class.php` - the configuration file
- `imdb.class.php` - the API class for movie info
- `imdb_charts.class.php` - class for *Top-Charts*
- `imdb_nowplaying.class.php` - what is playing right now in US movie theaters
- `imdb_person.class.php` - the API class for staff info
- `imdb_request.class.php` - wrapper to the browser emulator
- `imdb_trailers.class.php` - class to retrieve the real trailer URLs

All other files are either documentation or demo files and are not needed to use the API.

Where should the files go?

This very much depends on the intended use. If you want to integrate the IMDBPHP API into your application, you will probably put them in some directory inside the app. Otherwise, if the API should be globally available, you should put them into some directory contained in your PHP include path (see your `php.ini` for the keyword `include_path`, or the web server configuration for the corresponding keyword - for Apache, this is e.g. `php_value include_path`). Important is that all these files are kept together to work without changing them. If you use one of the recommended installation methods, this will be done automatically: With v0.9.5 and up the API files will be placed into `/usr/share/php`, which is in the

include_path of the php.ini shipped with most distributions.

What needs to be configured?

Basically, it works "out of the box". But there are some things you may want to adjust. We will walk here through the configuration - which you already guessed correctly to be found in imdb_config.class.php - to discuss all issues.

All configuration options in imdb_config.class.php are prefixed by the term $this->, since we configure a class. The prefix $this-> will be omitted in the following description to make it easier to read:

General Options

Option	Description
imdbsite	The IMDB website of your preference. By default it is set to akas.imdb.com since that site will handle ***all*** movie names and AKAs known, so also localized movie names should be found. You can change this option even at runtime to switch the server.
debug	Show debug messages (1) or not (0)?

Caching Options

Option	Description
cachedir	Where to store the cache (if enabled). This directory must be read/write accessible for the web server. You can specify either relative or absolute path.
usecache	If a requested page is found in the cache, shall it be used to setup the data (TRUE) - or shall we ignore it and retrieve the page again from its origin (FALSE)?
storecache	If a page was retrieved from its origin, shall it be stored into the cache for later reference? Requires the cache directory to be set to an existing path with read/write permission for the web server.
usezip	Shall the cache files be stored in compressed form?
converttozip	If there are uncompressed pages in the cache, shall they be compressed when accessed?
cache_expire	For how many seconds shall the cache stay valid? Files older than this will be purged the next time the cache is probed.

Image Options

Option	Description
photodir	Directory where retrieved images should be stored. Must be inside the servers DOCUMENT_ROOT if you want this images to be displayed with your web pages, and as with the cachedir above, the web server needs permission to read and write here.
photoroot	URL equivalent for the photodir - i.e. it must start at the servers DOCUMENT_ROOT if you specify an absolute path: If your DOCUMENT_ROOT e.g. is /var/www, and your images are stored in /var/www/photos, the absolute URL would be /photos/.

Tweaking Options

Option	Description
maxresults	Limit the result set displayed on a movie search. A value of "0" means no limit, everything else sets the maximum results displayed.
searchvariant	

	If you are not happy with the results retrieved on a movie search, you may want to try one of the search variants (different options will be set on the movie search). Available variants are `sevec` and `moonface`, the default is `izzy`.

Alternate Configuration File

Starting with version 1.0.8 (or revision r170), you can also define an alternate configuration file by defining a constant named `IMDBPHP_CONFIG` **before including `imdb_base.class.php`**:

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

This way it is possible to easily use different settings for different purposes, if needed.

Usage

This is just a basic documentation to get you started using the IMDBPHP Api. It will not discuss all the API methods available (for this, please see the API Reference) - but instead show you how to use it in general. Having understood the principles, you are able to use the API reference to extend your knowledge.

Please use the *Table of Contents* on the right hand side of this page to navigate through the chapters.

MovieSearch

The first thing you always need to do when you want to use the API is to include the class file. This can be done with one of the four commands:

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

as you do with other PHP files. `imdb.class.php` will then look for the other required files in the same directory it is located itself, and include them also.

Now, how you can search for a movie? You either have the name (or parts of it), or you already have the IMDB ID of the movie. In the latter case, you don't need to perform a search for the movie, but go directly to the MovieDetails. Consider the MovieSearch as a search for the IMDB ID - that is what it does: Listing up IMDB IDs matching to the (partially) movie name you specified.

Submit a search

To describe the necessary steps, let's take an example searching for the movie "The Whole Nine Yards". Assume we don't know the exact name, but just remember the "Nine Yards", this is what it would look like:

```
<?php
$name = "nine yards";           // the name will usually be dynamically set
require_once("imdb.class.php"); // include the class file
$search = new imdbsearch();     // create an instance of the search class
$search->setsearchname($name);  // tell the class what to search for (case insensitive)
$results = $search->results();  // submit the search
?>
```

Parse results

Now the search was submitted, and you can go to parse the results stored in the array `$results`. As the API reference describes, this is an array[0..n] of objects, where each object is an instance of the class `imdb` as described in MovieDetails - so basically you could go on and retrieve all movie details for all results now. Which generally is not that useful, since you do not yet know which of these are interesting for you. So normally you want to retrieve some information to identify the movie in question, so you will loop over the `$results` array e.g. as follows:

```
<?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";
}
?>
```

From the list created in that loop, the user then can select the IMDB ID of the movie in question. You could e.g. provide each item of this list with a link to the details page - `http://'".$search->imdbsite."/title/tt".$res->imdbid()` would provide a link to the movies page at the IMDB site.

With the IMDB ID known, you can proceed to retrieve the details as described on the MovieDetails page.

MovieDetails

As with the MovieSearch, you first must include the class file before using it. To retrieve movie details, you will need to provide the movies IMDB ID, which can be retrieved by a MovieSearch or by copy-and-paste from the IMDB page. However, this wiki page assumes you already know the IMDB ID.

General

Now, the IMDB sites provide a lot of details for a movie, which are spread over multiple pages. The according page will be retrieved on the first request for it (i.e. when you tried to retrieve data from that page), and kept in memory for subsequent accesses. If you have the cache enabled, it will even be stored for future "sessions", to give a better performance.

So let's assume you request the title, year, runtime, rating and a list of trailers, this would look as follows:

```
<?php
require_once("class.imdb.php");    // include the class file
$movie   = new imdb($mid);         // create an instance of the class and pass it the IMDB ID
$title   = $movie->title();        // retrieve the movie title
$year    = $movie->year();         // obtain the year of production
$runtime = $movie->runtime();      // runtime in minutes
$rating  = $movie->mpaa();         // array[country=>rating] of ratings
$trailer = $movie->trailers();     // array of trailers
?>
```

Going in a loop for multiple movies, you can simply reset the class using the `$movie->setid($newid)` method - no need to drop and recreate the class instance. For more details on available methods, and the structure of their returned values, please refer to the API documentation.

Technical background

For those interested in, here comes a short description of what would happen at each of the steps described above:

Step	Process
require_once("class.imdb.php");	The class file will be included into your current script. This implicitly includes all classes it depends upon, such as the `imdb_base.class.php`.

new imdb($mid);	This will create a new instance of the imdb class, and set the IMDB ID for the movie to process at the same time.
$movie title();	The movies title page will be accessed. Since it was not retrieved before, this will result in requesting the page from the configured IMDB site. If caching is enabled, the page will be stored in the cache. The page will for sure be held in a class variable. This variables content will be parsed for the movie title, which is then also kept in another variable - and returned to the calling process.
$movie year();	The movies title page will be accessed again. Since it is already found in the internal variable, the latter will be parsed for the year - which again is stored internally and returned to the calling process.
$movie year();	Assumed you call this a second time, it will simply return the stored value.
$movie runtime();	Same as with the first call to `$movie->year()`.
$movie mpaa();	Again the same.
$movie trailers();	This will access a new page, and thus be processed as `$movie->title()` above.

This example shows that we try to access the IMDB site as little as possible. Processing the same page with the same IMDB ID again would access the pages from the cache (if enabled) instead of retrieving them again from the IMDB site. Moreover, we also only retrieve the pages needed for our requests - so in our above example, e.g. the */fullcredits* page is not even touched since none of its details have been asked for. This is the best we can do to increase the performance. If you, however, go to retrieve all possible details, the API needs to retrieve (and parse) all pages necessary - which may (and probably will) cause a delay for data delivery. So you better don't put all details on one page.

PersonDetails

As with the MovieSearch and MovieDetails, you first must include the class file before using it. To retrieve details on staff members, you will need to provide the persons IMDB ID, which is retrieved e.g. with the details of some movies parsed with the MovieDetails. You also can obtain it by copy-and-paste from the IMDB page. However, this wiki page assumes you already know the persons IMDB ID.

General

Now, the IMDB sites provide a lot of details for staff members, which are spread over multiple pages. The according page will be retrieved on the first request for it (i.e. when you tried to retrieve data from that page), and kept in memory for subsequent accesses. If you have the cache enabled, it will even be stored for future "sessions", to give a better performance.

So let's assume you request the name, birth date, nickname, mini biography and directors filmography, this would look as follows:

```
<?php
require_once("class.imdb_person.php"); // include the class file
$person = new imdb_person($mid);       // create an instance of the class and pass it the IMDB ID
$name   = $person->name();             // retrieve the name
$birth  = $person->born();             // obtain the birthday data array
$nick   = $person->nickname();         // get the nickname
$bio    = $person->bio();              // array[0..n] of mini biographies
$filmo  = $person->movies_director();  // array of movies (filmography as director)
?>
```

Going in a loop for multiple persons, you can simply reset the class using the `$person->setid($newid)` method - no need to drop and recreate the class instance. For more details on available methods, and the structure of their returned values, please refer to the API documentation.

Technical background

For those interested in, here comes a short description of what would happen at each of the steps described above:

Step	Process
require_once("class.imdb_person.php");	The class file will be included into your current script. This implicitly includes all classes it depends upon, such as the `imdb_base.class.php`.
new imdb_person($mid);	This will create a new instance of the class, and set the IMDB ID for the staff member to process at the same time.
$person name();	The persons name (main) page will be accessed. Since it was not retrieved before, this will result in requesting the page from the configured IMDB site. If caching is enabled, the page will be stored in the cache. The page will for sure be held in a class variable. This variables content will be parsed for the persons name, which is then also kept in another variable - and returned to the calling process.
$person born();	The persons name (main) page will be accessed again. Since it is already found in the internal variable, the latter will be parsed for the birthday data - which again will be stored internally and returned to the calling process.
$person born();	Assumed you call this a second time, it will simply return the stored value.
$person nickname();	Since the nickname is not stored on the persons main page, we need to access another one ("bio"). It was not retrieved before, so it will be grabbed from the server, stored in the cache, parsed - and finally the nickname is retrieved. Basically the same as with our call to `$person->name();`, just for a different page.
$person bio();	Similar to the first call to `$person->born();` - data is taken from the already retrieved "bio" page.
$person movies_director();	This again comes from the main page, which already was retrieved before. So it will be parsed for the filmography of this person as director, and the corresponding array will be (stored internally and) returned.

This example shows that we try to access the IMDB site itself as little as possible. Processing the same page with the same IMDB ID again would access the pages from the cache (if enabled) or from the internal variable (if already parsed during the same process) instead of retrieving them again from the IMDB site. Moreover, we also only retrieve the pages needed for our requests - so in our above example, e.g. the */awards* page is not even touched since none of its details have been asked for. This is the best we can do to increase the performance. If you, however, go to retrieve all possible details, the API needs to retrieve (and parse) all pages necessary - which may (and probably will) cause a delay for data delivery. So you better don't put all details on one page in one run.

imdbXML

The **imdbXML class** can be used to convert an imdb object to XML: All object attributes will be converted to XML tags, including their values. Since the class extends PEARs XML_Serializer class, you will need to have PEAR installed for this.

The following code gives you an example of how to use the class. Saved to a file and called, it will return XML with title, year and plotoutline tags:

```
<?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 Hints

Here are some hints on how to gain (or fake) some performance:

- Never include all details possible into one page - stick to necessary details and/or split up information to multiple pages
- If you ignore this, you can at least put a *flush();* statement once upon a while - so the user sees some progress (see sample `imdb.php`)
- Use the cache - this speeds up subsequent requests for the same movie. Set the cache expiry to something useful - a too large value will fill your disk and ignore possible updates at the IMDB site, a too low value will be bad for the performance. A suitable value is very subjective, but can probably be anything between one hour and one day.
- If possible, group requests by their respective pages - so they can use the already set-up internal variables instead of reparsing. As for performance, if on the first "click" you retrieve items from different pages, and on a second "click" again, this will be slower as if sticking to groups for each "click" (ahw, how to describe that?!?)
- If you will need the just retrieved information on a later page, store them locally (e.g. in your own database, in a PHP session variable, or the like). Accessing that information will probably be faster than a re-parse (for sure if you turned the cache off).

Test pages

In order to detect broken things, the distribution contains a set of test pages. They are set up with data we know are available on the IMDB sites - so despite the fact that things may be updated, we at least know we can expect some response here. These tests are located in the `test/` subdirectory of the distribution, and you usually can simply point your browser to that directories URL (you may have to add `index.php` to the URL if you get an error about directory browsing not being allowed). Now the available methods should list up and be marked with either a green "PASS" or a red "FAIL" - and you should know what is broken.

Make sure whether caching is involved

To figure out what the cache could have to do with it, simply add `?cache=off` to the end of the URL to turn it off - or `?cache=on` to force it being used.

Limiting tests to sections

It is also possible to only check a given part of the API: `?check=movie` restricts the run to the movie methods, while `?check=name` restricts it to the person methods. Of course you can combine these with the cache advisor, e.g. with `?check=movie;cache=off` or `?check=name&cache=on`. Just use the delimiter your PHP installation understands (you may have noticed one example using the ";" while the other uses "&"). Available "restrictors" include:

Value	Description
movie	methods from the main class (movie methods)
name	methods from the `imdb_person` class
charts	methods of the `imdb_topcharts` class
nowplay	methods of the `imdb_nowplaying` class
trail	methods of the `imdb_trailers` class

Appendix

This page gives an overview on several issues:

- IMDBPHP — short overview
- FAQ — Questions and Answers on IMDBPHP
- ProjectsUsingThis — some references
- IzzySoft — who's behind it

IMDBPHP

What is IMDBPHP?

IMDBPHP provides an API (Application Programmers Interface) to the movie information stored at the IMDB.COM site. As this and the name suggests, it is primarily targeted at PHP programmers who want to extend their programs or their site with these movie information.

IMDBPHP offers the search for a movies ID. Using this ID, it can either provide a link to the original site - or retrieve details stored their directly and specifically. This includes a list of actors, directors, languages and the like - plus much more (see the Demo for details). Moreover, IMDBPHP is e.g. used in phpVideoPro - so there you can see an example of integration into another software project.

Development

Development of IMDBPHP primarily means to keep the code up-to-date with the many and frequent changes on the IMDB sites - which render the whole API useless otherwise within sometimes short intervals. If there's any time left, we also look to implement new features - which mostly means to retrieve more details.

Maybe I have more ideas - but also maybe you have? Feel free to add your ideas to the ticket section (feature requests as well as bug reports) and let me know what ***you*** would like to see in IMDBPHP. Chances are high that you convince me to implement your idea!

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 was founded in 1995 as a sideline to the owners study of computer sience (and, of course, to fund it somehow). In the early years, IzzySoft mainly served web development and configuration/installation issues for computers and small networks.

The focus changed in 2002/2003, when the last "normal employment" ended and the founder decided to start freelancing for 100%. Since then, IzzySoft concentrates on Oracle support. This includes the service on site (installation, configuration, tuning, etc.) as well as some pieces of software provided to the public under GPL - like this project you are watching right now.

More information

- IzzySoft Homepage
- IzzySoft Oracle stuff
- IzzySoft Software page
- IzzySoft *.deb/*.rpm repositories