



AdvSearch 1.0.0 pl

Date: 25/01/2012

Dynamic content search add-on that supports results highlighting, faceted search and search in custom packages

Author: Coroico <coroico@wangba.fr>

Query Hooks 1.2

Build your faceted search with query hooks !

A query hook is a snippet which transform the query runs by AdvSearch.

Through a unique query hook, you could manage:

- The filtering of results
- The modifying of the number of results per page
- The modifying of the sort of results
- The definition of the main class resource
- The definition of joined classes resources

Introduction

A Query Hook is a snippet so the Php writing rules should be applied.

Then you have only one single query hook per AdvSearch snippet call.

The structure of the query Hook is always:

```
<?php
```

```
... Some declarations ...
```

```
$qhDeclaration = array(  
    'qhVersion' => '1.2',  
    ... some declarations ...  
);
```

```
$hook->setQueryHook($qhDeclaration);  
return true;
```

A queryHook is a php snippet, so it should start with <?php

The queryHook should always be terminated by:

```
$hook->setQueryHook($qhDeclaration);  
return true;
```

For the management of further possible changes, we add a queryHook version number. This number (here 1.1) is linked with the version of advSearch.

The possible declarations are:

- 'andConditions'
- 'sortby'
- 'perPage'
- 'requests'
- 'joined'
- 'main'

Declaration syntaxes - andConditions

A andCondition declaration is writted as follow:

```
$andConditions = array(  
    'class.field:operator:pattern' => 'value : typeOfValue : filteredValues'  
);
```

```
$qhDeclaration = array(
    'qhVersion' => '1.2'
    'andConditions' => $andConditions
);
```

The **key** of a condition element is be as '**class.field:operator:pattern**'

Class:

For TVs we use 'tv' as alias for the class name.

For other classes like modResource or any other customs classes we will use the real name of the class.

Field:

For TV we use the TV name

For other classes, the field name or alias if defined

The field name (or alias) is mandatory.

Operator:

The allowed operators are the same as operators of xPDO conditions + some specific operators like MATCH, FIND

As example: > , >= , < , <= , IN, NOT IN, =, REGEXP, LIKE, MATCH, FIND

The default operator is '='.

Pattern:

The pattern is used with REGEXP and FIND operators.

The **value** of a condition element should be as '**value:typeOfValue:filteredValues**'

Value:

Value could be a constant ('string' or 'numeric') or a html tag name.

If it is a tag name, then typeOfValue should be 'request'

typeOfValue:

Could be 'request', 'string' or 'numeric'

'request' means that the real value will be provided by \$_REQUEST['value']

Could be omitted. In this case the values are supposed of 'request' type.

FilteredValues:

Represent a csv list of string values that will be filtered

This means for request type that if \$_REQUEST['value'] = filteredValue then no filtering is applied

Could be omitted (no values filtered)

For the management of further possible changes, we add a queryHook version number. This number (here 1.1) is linked with the version of advSearch.

And finally we add our queryHook by using:

```
$hook->setQueryHook($qhDeclaration);
```

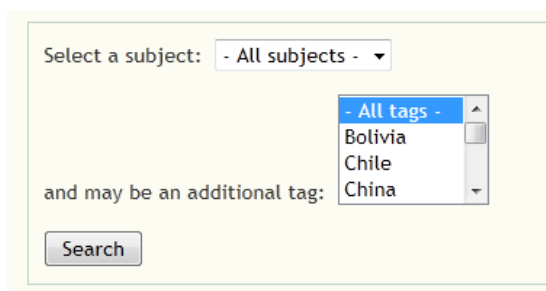
And at the end we return a 'true' Boolean value.

```
return true;
```

Ok, let's go with some examples ...

Example 1 – Filter1 - andConditions

Below the search form:



The screenshot shows a web form with a light yellow background. At the top, there is a label 'Select a subject:' followed by a dropdown menu currently showing '- All subjects -'. Below this, there is a label 'and may be an additional tag:' followed by a dropdown menu showing '- All tags -' with a list of tags: Bolivia, Chile, and China. At the bottom left of the form is a 'Search' button.

is implemented with the following snippet call:

```
[[!AdvSearchForm? & tpl=`filter1SearchForm`]]
```

With filter1SearchForm content as:

```
<form class="advsea-search-form" action="[[~[[+landing]]]]" method="[[+method]]">
  <fieldset>
    <input type="hidden" name="id" value="[[+landing]]" />
    <input type="hidden" name="asId" value="[[+asId]]" />
    <label>Select a subject:&nbsp;
      <select name="ctg" >
        <option value="all" selected="selected">- All subjects -</option>
        <option>Arts</option>
        <option>Countries</option>
        <option>Geography</option>
        <option>Litterature</option>
        <option>Music</option>
      </select>
    </label>
    <br /><br />
    <label>and may be an additional tag:&nbsp;
    <select name="tag[]" multiple="multiple">
      <option value="all" selected="selected">- All tags -</option>
      <option>Bolivia</option>
      <option>Chile</option>
      ....
      <option>settlements</option>
      <option>travel</option>
      <option>volcano</option>
    </select>
    </label>
    <br /><br />
    <input type="submit" name="sub" value="[[[%advsearch.search? &namespace=`advsearch` &topic=`default`]]]" />
  </fieldset>
</form>
[[+resultsWindow]]
```

Two tags are here important: 'ctg' and 'tag[]'. Ctg is the select name which hold the name or the category. And tag[] hold the possible selected tags. Notice that as tag is a multiple select, we use tag[] instead of tag. When a search should occur with all the categories, the value "all" will be sent through \$_GET. Same thing for the tags.

To implement the search with filtering by a category and tags we write the andConditions declaration as follow:

```
<?php

$andConditions = array(
  'tv.articleCategory:=' => 'ctg:request:all',
  'tv.articleTags:MATCH' => 'tag:request:all',
);

$qhDeclaration = array(
  'qhVersion' => '1.2'
  'andConditions' => $andConditions
);

$hook->setQueryHook($qhDeclaration);
return true;
```

Basically, all the search results should be filtered by two conditions, so we define an array named andConditions which contains two conditions:

The first element of the array handle the condition on the category and the second one on tags:

```
'tv.articleCategory:=' => 'ctg:request:all',
'tv.articleTags:MATCH' => 'tag:request:all',
```

Above we have 'tv.articleCategory:=' => 'ctg:request:all', Which means that the value of tv named 'articleCategory' should be equal to the value provided by an http request variable named 'ctg'. And this condition shouldn't applied if the value of \$_REQUEST['ctg'] = 'all' Easy, no ?

For tags this is pretty the same thing, we specify:

```
'tv.articleTags:MATCH' => 'tag:request:all',
```

Which means that one of the value of tv named 'articleTags' should match to the value provided by an http request variable named 'ctg'. For instance if the 'articleTags' content is "Chile | volcano" you could specify "volcano" as tag value.

MATCH provides an exact matching between word1 | word2 | word3 This is useful when you a TV defined as a Listbox (Multi-Select).

But here as we have a multiple select, AdvSearch We will applied the filter for all the possible values selected by the user.

To run correctly the TVs values 'articleCategory' and 'articleTags' should be present in the query so the advSearch snippet call for the results should be as follow:

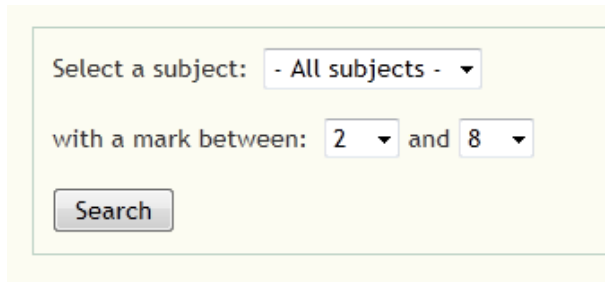
```
[[!AdvSearch? &queryHook=`Filter1QHook` &withTVs=`articleCategory, articleTags` ]]
```

To finalize the content of the queryHook we define a qhDeclaration array with our \$andConditions as 'andConditions' unique element:

Which is important to understand here is that the 'request' values are received, analyzed and propagated by AdvSearch. For instance if you look at the next or previous link for the pagination, you find again the request variable. They are propagated to the next/previous page. You have nothing to do more.

Example 2 – Filter2 - andConditions

An another example:



Here the search form is implemented by

```
[[!AdvSearchForm? &amp;tpl=`filter2SearchForm` ]]
```

With as content for the chunk 'filter2SearchForm':

```
<form class="advsea-search-form" action="[[~[[+landing]]]" method="[[+method]]">
<fieldset>
<input type="hidden" name="id" value="[[+landing]]" />
<input type="hidden" name="asId" value="[[+asId]]" />
<label>Select a subject:&nbsp;
<select name="ctg">
<option value="all" selected="selected">- All subjects -</option>
<option>Arts</option>
<option>Countries</option>
<option>Geography</option>
<option>Litterature</option>
<option>Music</option>
</select>
</label>
<br /><br />
<label>with a mark between
<select name="min">
<option value="none" selected="selected"></option>
<option>0</option>
<option>1</option>
<option>2</option>
....
<option>8</option>
<option>9</option>
<option>10</option>
</select> and <select name="msup">
<option value="none" selected="selected"></option>
<option>0</option>
<option>1</option>
<option>2</option>
....
<option>8</option>
<option>9</option>
<option>10</option>
</select>
</label>
```

```
<br /><br />
<input type="submit" name="sub" value="[[%advsearch.search? &namespace=`advsearch` &topic=`default`]]" />
</fieldset>
</form>
[[+resultsWindow]]
```

The difference with previous example is that here we need to get two values (**minf** and **msup**) and compare these values to a TV value named 'articleMark'.

To do this, we create a snippet named 'Filter2Qhook' with the following content:

```
<?php

$andConditions = array(
    'tv.articleCategory:REGEXP' => 'ctg:request:all',
    'tv.articleMark:>=' => 'minf,request:none',
    'tv.articleMark:>=' => 'msup,request:none'
);

$qhDeclaration = array(
    'qhVersion' => '1.2'
    'andConditions' => $andConditions
);

$hook->setQueryHook($qhDeclaration);
return true;
```

The first part for the management of the filtering around the category is the same as the first example.

Regarding the mark two html tags are used **'minf'** and **'msup'**.

```
'tv.articleMark:>=' => 'minf,request:none',
'tv.articleMark:>=' => 'msup,request:none'
```

The type is always 'tv'.

The TV name is 'articleMark'

But now we need to manage two values 'minf' and 'msup'.

These values are provided through http request without exception, so by using default values we could write
'tv.articleMark:>=' => 'minf', instead of 'tv.articleMark:>=' => 'minf,request'

So a search result will not be filtered if the articleMark tv value is >= \$_REQUEST['minf'] AND <= \$_REQUEST['msup']
And we don't take care of the bound if \$_REQUEST['minf'] = 'none' or/and if \$_REQUEST['msup'] = 'none'

We have a condition on the lower bound and on the upper bound. Here we doesn't take into account of the value 'none'.

The rest of the Filter2QHook is similar as the filter1QHook snippet.

Example 3 – Dvd Shop – andConditions & requests

But sometimes it is a bit more complicated ... let's go with this example.

With this example we work with a custom package, but for the moment we focus only on the `andConditions` declaration.

The above search form is implemented with:

```
[[!AdvSearchForm? &landing=`151` &tpl=`dvd2SearchForm`]]
```

With as content for the chunk 'dvd2SearchForm':

```
<form class="advsea-search-form" action="[[~[[+landing]]]]" method="[[+method]]">
  <fieldset>
    <input type="hidden" name="id" value="[[+landing]]" />
    <input type="hidden" name="asId" value="[[+asId]]" />
    <label>Title or description:<br />[[+helpLink]]&nbsp;
    <input type="text" id="[[+asId]]_search" name="[[+searchIndex]]" value="[[+searchValue]]" />
  </label>
  <br /><br />
  <label><span>Category: </span>
  <select name="ctg" ><option selected></option>
    <option >Animation</option>
    <option >Movie</option>
    <option >Television</option>
  </select>
</label>
<br /><br />
<label><span>Genre: </span>
  <select name="genre" ><option selected></option>
    <option >Action</option>
    <option >Adventure</option>
    .....
    <option >Science Fiction</option>
    <option >Thrillers</option>
    <option >War</option>
    <option >Western</option>
  </select>
</label>
<br /><br />
<label><span>Studio: </span>
  <select name="studio" ><option selected></option>
    <option >20th Century Fox</option>
    .....
    <option >Universal Studios</option>
    <option >Warner Bros</option>
  </select>
</label>
<br /><br />
<label><span>Year of production: </span>
  <select name="year" ><option selected></option>
    <option >1960</option>
    <option >1961-1970</option>
    <option >1971-1980</option>
    <option >1981-1990</option>
    <option >1991-2000</option>
    <option >2001-2010</option>
    <option >>2011</option>
  </select>
</label>
<br /><br />
```

```

<label><span>Price range: </span>
<select name="price" ><option selected></option>
<option >05-10$</option>
<option >10-15$</option>
<option >15-20$</option>
<option >20-25$</option>
<option >>25$</option>
</select>
</label>
<br /><br />
<input type="submit" name="sub" value="[[%advsearch.search? &namespace=`advsearch` &topic=`default`]]" />
</fieldset>
</form>
[[+resultsWindow]]

```

In this example, the tags `ctg`, `genre` and `studio` are classical and they could be treated easily with:

```

$andConditions = array(
    'dvdCategories.name:=' => 'ctg',
    'dvdProducts.tags:=' => 'genre',
    'dvdProducts.studio:=' => 'studio'
);

```

But for the tags `'year'` and `'price'`, the life is not so easy ...

Inside the `dvdProducts` table the field `production_year` is stored as a timestamp. Not as a range like `'1961-1970'` or `'2011'`
 So this requires to work a bit. To solve this we write this short piece of code:

```

// production year
$tag = 'year';
$year = strip_tags($_REQUEST[$tag]);
if (!empty($year)) {
    if ($year == '1960<') {
        $andConditions['dvdProducts.production_year:<'] = '1960:numeric';
    }
    elseif ($year == '>2010') {
        $andConditions['dvdProducts.production_year:>'] = '2010:numeric';
    }
    else {
        list($yinf,$ysup) = explode('-', $year);
        $andConditions['dvdProducts.production_year:>='] = $yinf.':numeric';
        $andConditions['dvdProducts.production_year:<='] = $ysup.':numeric';
    }
}

// propagate the http request variable for pagination
$_requests['year'] = $year;
}

```

Several things should be explained:

1/ We catch inside the `queryHook` (not in `AdvSearch`) the value send by the http request:

```

$year = strip_tags($_REQUEST[$tag]);

```

And for security reasons, we sanitize through the `php strip_tags` function, the `$_GET['year']` value.

2/ depending the year range received, we add a `andConditions`

```

if ($year == '1960<') {
    $andConditions['dvdProducts.production_year:<'] = '1960:numeric';
}
elseif ($year == '>2010') {
    $andConditions['dvdProducts.production_year:>'] = '2010:numeric';
}
else {
    list($yinf,$ysup) = explode('-', $year);
    $andConditions['dvdProducts.production_year:>='] = $yinf.':numeric';
    $andConditions['dvdProducts.production_year:<='] = $ysup.':numeric';
}

```

3/ and finally, as we have get the `$_REQUEST` inside the `queryHook`, we propagate this value to `advSearch` by adding:

```
$requests['year'] = $year;
```

For the price we use the same workaround:

```
$tag = 'price';
$price = strip_tags($_GET[$tag]);
if (!empty($price)) {
    $vprice = substr($price,0,-1); // to clear the money unit
    if ($vprice == '>25') {
        $andConditions['dvdProducts.price:>'] = '25:numfield';
    }
    else {
        list($pinf,$psup) = explode('-', $vprice);
        $andConditions['dvdProducts.price:>='] = $pinf.':numfield';
        $andConditions['dvdProducts.price:<='] = $psup.':numfield';
    }

    // propagate the http request variable for pagination
    $requests['price'] = $price;
}
```

andConditions - Much more ...

custom class instead of tv

```
'dvdProducts.studio: =' => 'studio'
```

Here the field studio from the dvdProducts class should be equal to \$_REQUEST['studio'].

Compare with constant values (numeric)

```
'tv.articleMark:>=' => '5:numeric'
```

Here a search result will not be filtered if the articleMark value is >= 5

Compare with constant values (textfield)

```
'tv. articleCategory:LIKE' => '%Music%:string'
```

Here a search result will not be filtered if the articleCategory value looks like %Music%

Define your own REGEXP (textfield)

```
'tv. articleCategory:REGEXP: %s[0-9]*' => 'ctg:string'
```

Here a search result will not be filtered if the articleCategory value doesn't end by a numeric between 0 and 9.

Find In Set (textfield)

```
'tv. articleTags:Find:|'| => 'tags:string'
```

Here a search result will not be filtered if the tag value can't be found inside the articleTags value.
articleTags content is a list of tags separated by | |

```
'tv. articleTags:Find' => 'tags:string'
```

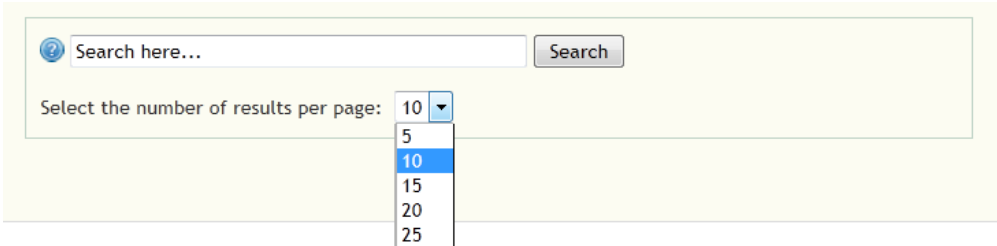
Here a search result will not be filtered if the tag value can't be found inside the articleTags value.
articleTags content is a list of tags separated by a comma.

Declaration syntaxes - perPage

To allow the user to change the number of results displayed, use the 'perPage' declaration.

Simply add: 'perPage' => \$perPage as array element of \$qhDeclaration. Where \$perPage contains the tag name which hold the perPage value.

Example 4 – Filter4 - perPage



Here the search form is implemented by

```
[[!AdvSearchForm? &amp;tpl=`filter4SearchForm` ]]
```

With as content for the chunk 'filter4SearchForm':

```
<form class="advsea-search-form" action="[[~[[+landing]]]" method="[[+method]]">
<fieldset>
<input type="hidden" name="id" value="[[+landing]]" />
<input type="hidden" name="asId" value="[[+asId]]" />
[[+helpLink]]
<input type="text" id="[[+asId]].search" name="[[+searchIndex]]" value="[[+searchValue]]" />
<input type="submit" name="sub" value="[[%advsearch.search? &namespace=`advsearch` &topic=`default`]]" />
<br /><br />
<label>Select the number of results per page:&nbsp;  
<select name="ppage" >
<option selected="selected">5</option>
<option>10</option>
<option>15</option>
<option>20</option>
<option>25</option>
</select>
</label>
</fieldset>
</form>
[[+resultsWindow]]
```

To catch and manage the 'ppage' html tag we write the following queryHook (named here: Filter4QHook):

```
<?php
$perPage = 'ppage'; // 'ppage' is the html tag name used to catch the number of results to display

$qhDeclaration = array(
    'qhVersion' => '1.2'
    'perPage' => $perPage
);

$hook->setQueryHook($qhDeclaration);
return true;
```

And the inside the AdvSearch snippet call we add: &queryHook=`Filter4QHook` :

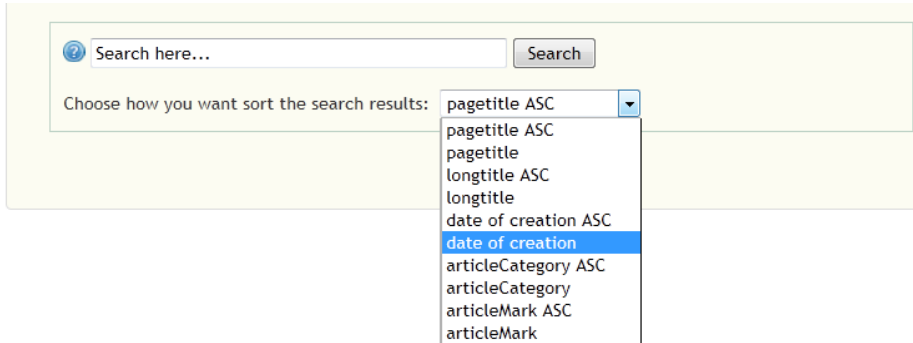
```
[[!AdvSearch? &queryHook=`Filter4QHook` ]]
```

Declaration syntaxes - sortby

To allow the user to change the sort of results displayed, use the 'sortby' declaration.

Simply add: 'sortby' => \$sort as array element of \$qhDeclaration. Where \$sort contains the tag name which hold the csv list of fields to use for the sort.

Example 5 – Filter5 - sortedby

A screenshot of a web search interface. At the top, there is a search bar with a placeholder text "Search here..." and a "Search" button. Below the search bar, there is a label "Choose how you want sort the search results:" followed by a dropdown menu. The dropdown menu is open, showing a list of sorting options: "pagetitle ASC", "pagetitle", "longtitle ASC", "longtitle", "date of creation ASC", "date of creation" (which is highlighted in blue), "articleCategory ASC", "articleCategory", "articleMark ASC", and "articleMark".

Here the search form is implemented by

```
[[!AdvSearchForm? &amp;tpl=`filter5SearchForm` ]]
```

With as content for the chunk 'filter5SearchForm':

```
<form class="advsea-search-form" action="[[~[[+landing]]]" method="[[+method]]">
<fieldset>
<input type="hidden" name="id" value="[[+landing]]" />
<input type="hidden" name="asld" value="[[+asld]]" />
[[+helpLink]]
<input type="text" id="[[+asld]].search" name="[[+searchIndex]]" value="[[+searchValue]]" />
<input type="submit" name="sub" value="[[%advsearch.search? &namespace=`advsearch` &topic=`default`]]" />
<br /><br />
<label>Choose how you want sort the search results:&nbsp;  
<select name="sort" >
<option value="pagetitle ASC" selected="selected">pagetitle ASC</option>
<option>pagetitle</option>
<option value="longtitle ASC">longtitle ASC</option>
<option value="longtitle">longtitle</option>
<option value="createdon ASC">date of creation ASC</option>
<option value="createdon">date of creation</option>
<option value="articleCategory ASC">articleCategory ASC</option>
<option value="articleCategory">articleCategory</option>
<option value="articleMark ASC">articleMark ASC</option>
<option value="articleMark">articleMark</option>
</select>
</label>
</fieldset>
</form>
```

To catch and manage the 'sort' html tag we write the following queryHook (named here: Filter5QHook):

```
<?php
$sortby = sort; // 'sort' is the html tag name to catch the array of fields used to sort results

$qhDeclaration = array(
    'qhVersion' => '1.2'
    'sortby' => $sortby
);

$hook->setQueryHook($qhDeclaration);
return true;
```

And the inside the AdvSearch snippet call we add: &queryHook=`Filter4QHook` :

```
[[!AdvSearch? &queryHook=`Filter4QHook` ]]
```

Declaration syntaxes - Main

As explained with the example 3, we could use custom package instead of the default modResource package. But as modResource is the main package, you need to choose and declare a new main package. For this we use the 'main' declaration.

The elements of the main declaration are the followings:

- **package** - The name of the schema Package to add.
- **packagePath** - The path to the model/ directory where the package is located. Use {core_path} as generic variable.
- **class** - The class name of the table you want to search.

and optionally (these parameters could be passed thru the snippet call):

- **withFields** - A comma-separated list of column names where to search.
- **fields** - A comma-separated list of column names to display.
- **ids** - A comma-separated list of primary keys to filter where the search should occurs.
- **where** - criteria or Array of criteria. Column names should be prefixed by class name and wrapped with backticks.
- **sortBy** - csv list of couple 'columnName ASC [[DESC]'. Column names should be prefixed by class name and wrapped with backticks.

As example for the dvdProduct class we write:

```
$main = array(  
    'package' => 'dvd',  
    'packagePath' => '{core_path}components/dvd/model/',  
    'class' => 'dvdProducts',  
    'fields' => 'code , category_id , name , tags , price , weight , image , image_thmb , in_stock , studio , production_year , length , description',  
    'withFields' => 'name , tags , studio , description', // where we do the search  
    'sortBy' => 'dvdProducts.studio DESC, dvdProducts.name ASC'  
);
```

And we simply add the 'main' declaration with:

```
$qhDeclaration = array(  
    'qhVersion' => '1.2', // version of queryHook - to manage futures changes  
    'main' => $main,  
);
```

Some remarks:

By choosing a class you choose the main object from which you will get the information. This is will drive the query.

- withFields define the list of fields where to do the search or which field could be filtered
- fields define the list of fields that we be available as placeholders inside the result template as [[+fieldname]]
- ids is a comma separated list of primary key. This is could be a string rather an integer, but in any case you should be obliged to manage the link that will be set in the result template.

When the primary key is a modResource id. The link is usually set by:

```
<a href="[~[[+id]]]" title="[+longtitle]">[[+pagetitle]]</a>
```

For instance:

```
<a href="index.php?id=8" title="Villarica">Villarica</a>
```

If your primary key is not a document you probably need to have

```
<a href="[~235]&id=[[+id]]" title="[+longtitle]">[[+pagetitle]]</a>
```

where 235 is the resource id that should manage the displaying of the resource which as a primary key = [[+id]].

For the DVD example, this means for instance to get from the database all the information regarding the DVD and then display them.

Declaration syntaxes - Joined

To the main class (modResource or custom resource) you could join some others classes. To do it “the joined” declaration is our friend. Hereafter are the elements of a joined class. You could have several joined classes. But all should be joined to the maintables. It's not possible (for the version 1.1 of queryHook) to join a joined class to an another joined class.

- **package** - The name of the schema Package to add.
- **packagePath** - The path to the model/ directory where the package is located. Use {core_path} as generic variable.
- **class** - The class name of the table you want to search.

and optionally:

- **withFields** - A comma-separated list of column names where to search.
- **fields** - A comma-separated list of column names to display. By default same fields name as withFields. An alias could be provided.
- **joinCriteria** - The SQL condition to join the table you want to search and an other table.
- **where** - criteria or Array of criteria. Column names should be prefixed by class name and wrapped with backticks.

As example for the dvdCategories class is writted as follow:

```
$joined = array(
    array(
        'package' => 'dvd',
        'class' => 'dvdCategories',
        'packagePath' => '{core_path}components/dvd/model/',
        'fields' => 'name , description',
        'withFields' => 'name , description',
        'joinCriteria' => 'dvdCategories.id = dvdProducts.category_id'
    )
);
```

And we simply add the ‘joined’ declaration with:

```
$qhDeclaration = array(
    'qhVersion' => '1.2',
    'joined' => $joined,
);
```

QueryHook - Changelog

QueryHook 1.2:

- New operators added: FIND, MATCH
- REGEXP with pattern

QueryHook 1.1: initial version

PostHook

A posthook is a snippet which transform the displaying of search results.

To be completed

Demos

All the examples presented in this document are available on the AdvSearch demo site: www.revo.wangba.fr
As I am not a native English speaker, don't hesitate to propose updates or correction for this document.

On the documentation page you could download the last release of this document and a zip of the files presented here.
Enjoy Query Hook!

Support AdvSearch

Thanks to support AdvSearch : <http://www.revo2.wangba.fr/advsearch-donate.html>

Donate - Support AdvSearch & Gmaps Developments

I hope you enjoy **AdvSearch** and **Gmaps** and find the demos of this site helpful. If you are using (or would like) one of these add-ons, please consider making a donation to say thanks for the **time required to develop and maintain this code**. By using these components you and/or your company have saved lot of time and be your site, or your customer sites, more professional.

Donations can also be made to encourage fixing for a specific integration problem, or develop new features. In this case don't hesitate to use comments or the contact form, if you need some specific developments.

Thanks for your **support** and **encouragements**.

