aw2.query

This shortcode is one point *command* to access and interact with WordPress database objects like posts, taxonomies, comments and users. *aw2.query* behind the scene uses one of the WordPress functions like get_posts or WP_Query class depending on the main parameter.

It returns a result set, which can be accessed using *aw2.get* and *aw2.loop*. We pass parameters to aw2.query using JSON string with same key value pairs as respective WordPress function.

To save the results of a query in a variable use the set attribute

attributes:

- **main:**
  get_post | get_posts | get_post_terms | get_post_meta | insert_post | update_post | update_post_status | update_post_meta | delete_post_meta | add_non_unique_post_meta | delete_post | trash_post | set_post_terms | get_pages | wp_query | get_term_by | get_term_meta | insert_term | delete_term | get_terms | get_comment | get_comments | insert_comment | get_results | get_row | get_col | get_var | query | get_user_by | update_user_meta | get_user_meta | get_users | posts_builder | users_builder | delete_revisions
- **set:**
  name of variable where to save result of the query.

  It stores the result within the global 'stack' using the value specified.

Using posts_builder attribute with aw2.query

We use posts_builder attribute when we want to write conditional and complex wp_query. It requires additional attributes to be used with aw2.query. For details of various JSON parameters check WP_Query Class

Additional attributes

- **part**:
  start | tax_query | meta_query | date_query | run

  It allows us to break complex queries into parts. *part* attribute is used when we use '*posts_builder*' attribute. It gives us the ability to add or remove parts of a query based on conditions.

  When the part is *start* it destroys all existing parts and creates a new query, so it is important that when you are breaking a query into parts, *part=start* is first. Part *tax_query* indicates that JSON value is for running taxonomy query, similarly part *meta_query* indicates that JSON value is to be used for creating meta query and part *date_query* indicates date query.

  If the part is not used, then query executes immediately, else it only executes when it finds part *run.*

It returns a WP_Query object

```
[aw2.query posts_builder part=start]
  {
 "posts_per_page": 10,
 "post_type": "post",
 "post_status": "publish",
 "order": "DESC",
 "orderby": "date"
  }
[/aw2.query]
[aw2.query posts_builder part=tax_query]
  {
 "taxonomy": "category",
 "field": "slug",
 "terms": "nature"
  }
[/aw2.query]
[aw2.query posts_builder part=meta_query]
  {
 "key": "custom_value1",
 "value": "hello",
 "compare": "="
  }
[/aw2.query]
[aw2.query posts_builder part=run set=results /]
```

When you will dump *results*, you will get the WP_Query object, an example is shown below.

```
object(WP_Query)
[
    "query"                             =>  array(6)
    "query_vars"                        =>  array(67)
    "tax_query"                         =>  object(WP_Tax_Query)
    "meta_query"                        =>  object(WP_Meta_Query)
    "date_query"                        => bool(false)
    "request"                           => string(345) "SELECT SQL_CALC_FOUND_ROWS  wp_posts.ID FROM wp_posts  LEFT JOIN wp_term_relationships ON (wp_posts.ID = wp_term_relationships.obj
ect_id) WHERE 1=1  AND (
  wp_term_relationships.term_taxonomy_id IN (51)
) AND wp_posts.post_type = 'post' AND ((wp_posts.post_status = 'publish')) GROUP BY wp_posts.ID ORDER BY wp_posts.post_date DESC LIMIT 0, 10"
    "posts"                             =>  array(4)
    "post_count"                        => int(4)
    "current_post"                      => int(-1)
    "in_the_loop"                       => bool(false)
    "post"                              =>  object(WP_Post)
    "comment_count"                     => int(0)
    "current_comment"                   => int(-1)
    "found_posts"                       => string(1) "4"
    "max_num_pages"                     => float(1)
    "max_num_comment_pages"             => int(0)
    "is_single"                         => bool(false)
    "is_preview"                        => bool(false)
    "is_page"                           => bool(false)
    "is_archive"                        => bool(true)
    "is_date"                           => bool(false)
    "is_year"                           => bool(false)
    "is_month"                          => bool(false)
    "is_day"                            => bool(false)
    "is_time"                           => bool(false)
    "is_author"                         => bool(false)
    "is_category"                       => bool(true)
    "is_tag"                            => bool(false)
    "is_tax"                            => bool(false)
    "is_search"                         => bool(false)
    "is_feed"                           => bool(false)
    "is_comment_feed"                   => bool(false)
    "is_trackback"                      => bool(false)
    "is_home"                           => bool(false)
    "is_404"                            => bool(false)
    "is_embed"                          => bool(false)
    "is_paged"                          => bool(false)
    "is_admin"                          => bool(false)
    "is_attachment"                     => bool(false)
    "is_singular"                       => bool(false)
    "is_robots"                         => bool(false)
    "is_posts_page"                     => bool(false)
    "is_post_type_archive"              => bool(false)
```

```
    "thumbnails_cached"                  => bool(false)
    "query_array"                        =>  array(6)
    "private:WP_Query:query_vars_hash"    => string(32) "35376be83582715463b86da2fdff0568"
    "private:WP_Query:query_vars_changed" => bool(false)
    "private:WP_Query:stopwords"          => NULL
    "private:WP_Query:compat_fields"      =>  array(2)
    "private:WP_Query:compat_methods"     =>  array(2)
]
```

You can access these values using *aw2.get*.

One more detailed example of using posts_builder.

```
[aw2.query posts_builder part=start]
  {
 "posts_per_page": 10,
 "post_type": "post",
 "post_status": "publish",
 "order": "DESC",
 "orderby": "date"
  }
[/aw2.query]
[aw2.query posts_builder part=tax_query not_empty='{module.cat_slug}']
  {
 "taxonomy": "category",
 "field": "slug",
 "terms": "[aw2.get module.cat_slug /]"
  }
[/aw2.query]
[aw2.query posts_builder part=meta_query not_empty='{module.meta_f}']
  {
 "key": "custom_value1",
 "compare": "=",
 "value": "[aw2.get module.meta_f /]"
  }
[/aw2.query]
[aw2.query posts_builder part=run set=results]

<ul>
[aw2.loop results.posts]
  <li>[aw2.get item.post_title /]</li>
[/aw2.loop]
</ul>
```

Using users_builder attribute with aw2.query

Similar to posts_builder attribute, users_builder attribute is used to create complex queries for users. It requires additional attributes to be used with aw2.query. For details of various JSON parameters check WP_User_Query Class

It returns an array of IDs, stdClass objects, or WP_User objects, depending on the value of the 'fields' parameter.

fields (string|array) – Which fields to return. Defaults to all.

- 'ID' – Return an array of user id's.
- 'display_name' – Return an array of user display names.
- 'login' / 'user_login' – Return an array of user login names.
- 'nicename' / 'user_nicename' – Return an array of user nicenames.
- 'email' / 'user_email' – Return an array of user emails.
- 'url' / 'user_url' – Return an array of user urls.
- 'registered' / 'user_registered' – Return an array of user registered dates.
- 'all (default) or all_with_meta' – Returns an array of WP_User objects. Must pass an array to subset fields returned. *'all_with_meta' currently returns the same fields as 'all' which does not include user fields stored in wp_usermeta. You must create a second query to get the user meta fields by ID or use the __get PHP magic method to get the values of these fields.

Additional attributes

- **part**:
  start | meta_query | date_query | run

  It allows us to break complex queries into parts. *part* attribute is used when we use '*users_builder*' attribute. It gives us the ability to add or remove parts of a query based on conditions.

  When the part is *start* it destroys all existing parts and creates a new query, so it is important that when you are breaking a query into parts, *part=start* is first. Part *meta_query* indicates that JSON value is to be used for creating meta query and part *date_query* indicates date query it is applied to *user_registered* field.

  If the part is not used, then query executes immediately, else it only executes when it finds part=*run*.

```
[aw2.query users_builder part=start]
  {
 "role__in":[ "author","administrator" ],
  "search": "ami",
  "order" : "ASC",
  "orderby" : "display_name"
  }
[/aw2.query]
[aw2.query users_builder part=meta_query]
  {
 "key": "custom_value1",
 "value": "hello",
 "compare": "="
  }
[/aw2.query]
[aw2.query users_builder part=date_query]
  {
  "after" : "12 hours ago",
  "inclusive" : true
  }
[/aw2.query]
[aw2.query users_builder part=run set=users]
```

It returns the WP_User_Query object

```
 object(WP_User_Query)
[
    "query_vars"                         =>  array(26)
    "meta_query"                         =>  object(WP_Meta_Query)
    "request"                            => string(88) "SELECT SQL_CALC_FOUND_ROWS wp_users.* FROM wp_users WHERE 1=1 ORDER BY display_name ASC "
    "query_fields"                       => string(30) "SQL_CALC_FOUND_ROWS wp_users.*"
    "query_from"                         => string(13) "FROM wp_users"
    "query_where"                        => string(9) "WHERE 1=1"
    "query_orderby"                      => string(25) "ORDER BY display_name ASC"
    "query_limit"                        => NULL
    "private:WP_User_Query:results"      =>  array(12)
    "private:WP_User_Query:total_users"   => string(2) "12"
    "private:WP_User_Query:compat_fields" =>  array(2)
]
```

Using get_post attribute

By using get_post attribute you can quickly retrieve a single post. It requires few additional attributes

Additional attributes

- **post_id**:
  ID of post you want to get

  By specifying post_id you can quickly get the single post.

- **post_slug**:
  slug of the post you want to get

  If you don't have post_id you can specify post_slug to get the single post.

- **post_type**:
  post type of post you want to retrieve

  When you provide post_slug, you need to specify the post_type as well to get the single post. If you specify post_id then you don't need to use this.

Get a single post using ID

```
[aw2.query get_post post_id=1 set='module.single_post' /]
```

Get single post using post slug

```
[aw2.query get_post post_slug=hello post_type=post set='single_post' /]
post title= [aw2.get single_post.post_title /]
```

It returns single WP_Post object. Here is the sample WP_Post object, you can access these parameters using aw2.get

```
object(WP_Post)
[
    "ID"                    => int(12335)
    "post_author"           => string(1) "4"
    "post_date"             => string(19) "2015-09-06 07:01:23"
    "post_date_gmt"         => string(19) "2015-09-06 07:01:23"
    "post_content"          => string(1809) "Welcome,
     We are happy to announce our 'Awesome Studio Framework' for WordPress to the world.
     We are currently in a closed beta, and if you want to check it out,
     do <a href="http://getawesomestudio.com/#register">apply for our beta</a> program.
     "
    "post_title"            => string(12) "Hello world!"
    "post_excerpt"          => string(0) ""
    "post_status"           => string(7) "publish"
    "comment_status"        => string(4) "open"
    "ping_status"           => string(4) "open"
    "post_password"         => string(0) ""
    "post_name"             => string(13) "hello-world-2"
    "to_ping"               => string(0) ""
    "pinged"                => string(0) ""
    "post_modified"         => string(19) "2015-09-06 07:01:23"
    "post_modified_gmt"     => string(19) "2015-09-06 07:01:23"
    "post_content_filtered" => string(0) ""
    "post_parent"           => int(0)
    "guid"                  => string(32) "http://getawesomestudio.com/?p=1"
    "menu_order"            => int(0)
    "post_type"             => string(4) "post"
    "post_mime_type"        => string(0) ""
    "comment_count"         => string(1) "1"
    "filter"                => string(3) "raw"
]
```

Examples

Get a single post using ID

```
[aw2.query get_post post_id=1 set='module.single_post' /]
```

Get single post using post slug

```
[aw2.query get_post post_slug=hello post_type=post set='single_post' /]

post title= [aw2.get single_post.post_title /]
```

Get terms of single post

```
[aw2.query get_post_terms post_id=1 taxonomy=category <any args> set='b' /]
[aw2.query get_post_terms post_id=1 taxonomy=category set='b']
 args json onject
[/aw2.query]

[aw2.query get_post_terms post_id=1 taxonomy=category fields=names set='b']
[/aw2.query]
```

Using get_posts attribute with aw2.query

It is just a wrapper to get_posts WordPress function, to use it you pass the JSON object with parameters. You can also pass these args as attributes as well.

```
[aw2.query get_posts set=result]
  {
 "posts_per_page": 10,
 "post_type": "post",
 "post_status": "publish",
 "order": "DESC",
 "orderby": "date"
  }
[/aw2.query]
```

Here is an example how to pass the JSON args as attributes as well.

```
[aw2.query get_posts set=result posts_per_page=20]
  {
  "post_type":"post"
  }
[/aw2.query]
```

Using insert_comment attribute of aw2.query

You can insert comments using *insert_comment* attribute of *aw2.query*. It accepts following additional attributes

Additional attributes

- **post_id**:

  ID of the post on which the comment is being added.

- **author_name**:

  Name of the author, this will be dispalyed when comment is shown

- **author_email**:

  Email address of the comment author.

- **author_url**:

  URL of the comment author, if any.

- **approved**:
  1 | 0

  It is used to indicated whether comment is approved or not. Is the comment approved? 1 for yes and 0 for “awaiting moderation”

- **type**:

  If you want to separate comments by types, you can set it here.

- **parent**:

  To establish parent child relationship between comments. To make a comment child of another comment you can specify the comment id of the parent.

- **user_id**:

  You can specify the user_id for a logged-in user.

An example of using *insert_comment* attribute.

```
[aw2.query insert_comment
   post_id="12"
   author_name="Amit"
   author_email="amit@wpoets.com"
   author_url="http://www.wpoets.com"
   approved="1"
   type=""
   parent="0"
   user_id="1"
   set=reply ]
 this is new comment
 that will work with "double" quotes
[/aw2.query]
```

It return the new comment ID on success and false on failure.

Using get_terms attribute with aw2.query

You can use *get_terms* to get all the terms of a taxonomy. For detailed parameters check

Additional attributes

- **taxonomies**:

  Specify the taxonomy you want to get the terms from.

- **hide_empty**:
  1|0

  If you want to get terms with no posts, you can specify 0 with hide_empty

An example of using *get_terms*

```
[aw2.query get_terms taxonomies=category set=results hide_empty=0]
  {}
[/aw2.query]
```

Another example

```
[aw2.query get_terms taxonomies=category set=results hide_empty=0]
 {
   "orderby":"name",
   "order":"desc"
 }
[/aw2.query]
```

Using get_post_terms attribute with aw2.query

get_post_terms is used to get the terms of a taxonomy associated with a post, it’s a wrapper to wp_get_post_terms function. To use *get_post_terms* you need to provide following additional parameters

Additional attributes

- **post_id**:

  ID of the post you want to retrieve the associated terms from.

- **taxonomy**:

  Slug of the taxonomy you want to get terms from.

- **fields**:
  all|names|ids

  If you set fields value as *all*, it return the WP_Term Object. If it is set to names, it return the array of term names, while 'ids' return the term_id .

You can pass the oderby and order etc as JSON object to the shortcode.

Here is an example of WP_term Object

```
Array
(
    [0] => WP_Term Object
        (
            [term_id] => 145
            [name] => Example Category
            [slug] => example-cat
            [term_group] => 0
            [term_taxonomy_id] => 145
            [taxonomy] => adcpt_categories
            [description] =>
            [parent] => 0
            [count] => 2
            [filter] => raw
        )

)
```

Few examples of using get_post_terms with aw2.query

```
[aw2.query get_post_terms post_id=1 taxonomy=category fields=names set='b']
[/aw2.query]
```

```
[aw2.query get_post_terms post_id=1 taxonomy=category fields=names set='b']
{
 "orderby":"name",
 "order":"ASC"
}
[/aw2.query]
```

Using get_post_meta with aw2.query

You use *get_post_meta* when you need to fetch meta values of a post directly, it acts as a wrapper for get_post_meta function of WordPress. It needs following additional attributes to work

Additional attributes

- **post_id**:

  ID of the post you want to retrieve the meta values from.

- **key**:

  Use *key* attribute to provide the meta key whose value you want to fetch. If you do not specify the key, it will return all the meta key/value pair as an array

- **single**:
  true|false

  By default it is set to true, and return the single value of the meta key. If it is set to false it will return the array with all the value for the specified key.

Below example will return all the meta values associated with post_id 1.

```
[aw2.query get_post_meta post_id=1 set='meta' /]
```

Below example will return the value of meta key 'custom_value2'

```
[aw2.query get_post_meta post_id=1 key='custom_value2' set='meta' /]
```

Using get_term_by attribute with aw2.query

By using *get_term_by* you can get all the details of a term based on ID or slug or name. It is a wrapper to WordPress function get_term_by. It supports following additional attributes

Additional attributes

- **field**:
  slug | name | id | term_taxonomy_id

  Field specifies they type of value we want to use for finding the specific term.

- **value**:

  Search for this field value

- **taxonomy**:

  Taxonomy name. Optional, if $field is 'term_taxonomy_id'.

- **output**:
  OBJECT | ARRAY_A | ARRAY_N

  It is an optional field, if not specifed it return an array.

- **filter**:

default is raw or no WordPress defined filter will applied.

Here are some examples

```
[aw2.query get_term_by field="id" value="12" taxonomy="category" set="result" /]
```

```
[aw2.query get_term_by field="slug" value="general" taxonomy="category" set="result" /]
```