

Resource Classes Doc

Classes

"A class--the basic building block of an object-oriented language such as Java--is a template that describes the data and behavior associated with instances of that class. When you instantiate a class you create an object that looks and feels like other instances of the same class."

Mary Campione and Kathy Walrath, The Java Tutorial: Object-Oriented Programming for the Internet, The Java Series (Reading, Mass.: Addison Wesley, 1996)

- Class Resource
 - Properties
- Examples
 - 1 - PHP
 - 2 - API request

Class Resource

```
class Resource {
    public int    $id;
    public string $name;
    public string $slug;
    public string $type;
    public int    $parent_id;
    public int    $category_id;

    protected array $attr  = array();
    protected bool  $loaded = FALSE;

    public object get_attr( string $key );
    public void   set_attr( string $key, object $value );
    public bool   loaded();
}
```

Properties

As you can see from the database layout, the public properties of the Resource class are all part of the main **resource** table. The two protected properties **attr** and **loaded** are created at runtime. There are many situations where only the core information is required. To improve performance, attribute data is ignored when it is not required. Attributes are stored in the database as longtext; non-primitive types (such as arrays) are serialized and stored as a string.

```
$attr
A key-value store of the attributes that exist in the resource_attr table.

$loaded
A boolean value which is used to indicate whether or not the attributes have been loaded.
```

Why do some attributes have names that start with an underscore?

This is the convention for storing metadata. Most attributes are for storing data that is created by the user and is available to be directly edited by the user. When we want to store system data, configuration options, or just data that isn't meant for human consumption - we store it as metadata. An attribute is identified as being metadata by the convention of starting the name/key of the attribute with an underscore character (e. g. `_meta`). If you are interfacing with the API, you will frequently come across metadata. You're welcome to modify the metadata of a resource (if you know what you're doing) or add metadata attributes for known metadata keys, but you shouldn't create your own attributes with keys that begin with an underscore. Future versions of ProVision will use new metadata keys without warning, and if there is a naming conflict, your data could be lost.

Examples

These examples show the different methods that can be used to find and load a Resource object. They also show different data structures that are used to represent the object.

1 - PHP

Internal code example

To help users better understand how ProVision works, some of the examples in this documentation are of internal processes. They can contain code that only works when used as part of the core system and thus is not applicable to 3rd party development. The API is currently the only way for external tools to integrate with ProVision. Any example that contains internal code should be clearly labeled. Some common characteristics of these examples are code that doesn't use the API and code written in PHP (most example code will be in JavaScript).

This example uses the ResourceQuery class to find a resource object and then prints the result. It is included to show the similarity between finding a resource via the API and what happens under the hood.

```
$params = array(
    'slug' => 'tlr'
);
$resourceQuery = new ResourceQuery();
$resource = $resourceQuery->query($params);

var_dump($resource);
/*
array (size=1)
  0 =>
    object(Resource)[27]
      protected 'id' => string '1' (length=1)
      protected 'name' => string 'TLR' (length=3)
      protected 'slug' => string 'tlr' (length=3)
      protected 'type' => string 'resource' (length=8)
      protected 'parent_id' => null
      protected 'category_id' => null
      protected 'attr' =>
        array (size=0)
          empty
      protected 'loaded' => boolean true
*/
```

2 - API request

This is a standard API request, the request data is urlencoded and the result is JSON

/api/v1/api.php?target=resource&action=get&slug=TLR

```
{
  "success": 1,
  "message": "Search successful",
  "data": [
    {
      "id": "1",
      "name": "TLR",
      "slug": "tlr",
      "type": "resource",
      "parent_id": null,
      "category_id": null,
      "attr": {}
    }
  ]
}
```