

Table of Contents

<u>An Introduction to Vtiger Webservice</u>	1
<u>Requirements</u>	1
<u>Error Handling</u>	1
<u>Logging In</u>	1
<u>Getting Information</u>	2
<u>The output of the describeobject operation</u>	2
<u>CRUD Operations</u>	4
<u>Queries</u>	5

An Introduction to Vtiger Webservices

Vtiger provides a simple set of operations to work with entities stored in it.

The example code is written in Php and has two dependencies the Zend Json library and Http_Client.

Requirements

1. An installation of vtiger with webservices.
2. Php to run the example code
3. HTTP_Client which can be installing by doing a `pear install HTTP_Client`
4. Zend Json, you will need to download the [minimal zend framework](#)

Error Handling

The response for any webservice request is a json object. The object has a field success which will have the value true if the operation was a success and false otherwise. If success is false the response object will contain a field error which will contain json object. The error object will contain two fields errorType, a single word description of the error, and errorMsg, a string containing a description of the error.

Logging In

The api does not use the password to log in. Instead, Vtiger provides a unique access key for each user. To get the user key for a particular user, go to the *My Preferences* pane for that user. You will find a Access Key field.

Logging in is a two step process, Vtiger provides a challenge string which is used along with the user's access key to log in.

To get the challenge token you need to execute the following code.

```
//e.g.
$endpointUrl = "http://localhost/ws/webservice.php";
$username="admin";

$httpc = new HTTP_CLIENT();
$httpc->get("$endpointUrl?operation=getchallenge&username=$username");
$response = $httpc->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('getchallenge failed:'.$jsonResponse['error']['errorMsg']);

$challengeToken = $jsonResponse['result']['token'];
```

Now that you have the challenge token you can go ahead with the login. The access key field in the login operation is the md5 checksum of the concatenation of the challenge token and the user's own access key. The login operation, as opposed to getchallenge, is a post request.

```
//e.g.
$userAccessKey = 'dsf923rksdf3';
```

```

$generatedKey = md5($challengeToken.$userAccessKey);
$http->post("$endpointUrl",
    array('operation'=>'login', 'username'=>$userName,
        'accessKey'=>$generatedKey), true);
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('login failed:'. $jsonResponse['error']['errorMsg']);

$sessionId = $jsonResponse['result']['sessionId'];
$userId = $jsonResponse['result']['userId'];

```

The session id is used to identify the current session and will be a common parameter in all subsequent requests.

Getting Information

The api provides two operations to get information about available vtiger objects.

The first one is `listtypes`, which provides a list of available modules. This list only contains modules the logged in user has access to.

```

$http->get("$endpointUrl?sessionId=$sessionId&operation=listtypes");
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('list types failed:'. $jsonResponse['error']['errorMsg']);

$modules = $jsonResponse['result']['types'];

```

The second operation is `describeobject` which provides detailed type information about a vtiger object.

```

//e.g.
$moduleName = 'Contacts';

$params = "sessionId=$sessionId&operation=describeobject&elementType=$moduleName";
$http->get("$endpointUrl?$params");
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('describe object failed:'. $jsonResponse['error']['errorMsg']);

$description = $jsonResponse['result'];

```

The output of the *describeobject* operation

For example `print_r($description)` might display

```

Array
(
    [label] => Contacts
    [name] => Contacts
    [createable] => 1

```

```

[updateable] => 1
[deleteable] => 1
[retrieveable] => 1
[fields] => Array
  (
    [0] => Array
      (
        [name] => account_id
        [label] => Account Name
        [mandatory] =>
        [type] => Array
          (
            [name] => reference,
            [refersTo] => Array
              (
                "Accounts"
              )
          )
        [default] =>
        [nillable] => 1
        [editable] => 1
      )
    )
  )
)

```

The description consists of the following fields

1. *label* - The label used for the name of the module.
2. *name* - The name of the module.
3. *createable* - A boolean value specifying whether the object can be created.
4. *updateable* - A boolean value specifying whether the object can be updated.
5. *deleteable* - A boolean value specifying whether the object can be deleted.
6. *retrieveable* - A boolean value specifying whether the object can be retrieved.
7. *fields* - An array containing the field names and their type information.

Each element in the fields array describes a particular field in the object.

1. *name* - The name of the field, as used internally by vtiger.
2. *label* - The label used for displaying the field name.
3. *mandatory* - This is a boolean that specifies whether the field is mandatory, mandatory fields must be provided when creating a new object.
4. *type* - An map that describes the type information for the field.
5. *default* - The default value for the field.
6. *nillable* - A boolean that specifies whether the field can be set to null.
7. *editable* - A boolean that specifies whether the field can be modified.

The type field is of particular importance as it describes what type of the field is. This is an map that will contain at the least an element called name which is the name of the type. The name could be one of the following.

1. *string* - A one line text field.
2. *text* - A multiline text field.
3. *integer* - A non decimal number field.
4. *double* - A field for floating point numbers

The output of the describeobject operation

5. *boolean* - A boolean field, can have the values true or false.
6. *time* - A string of the format hh:mm, format is based on the user's settings time format.
7. *date* - A string representing a date, the type map will contain another element called format which is the format in which the value of this field is expected, its based on the user's settings date format.
8. *datetime* - A string representing the date and time, the format is base on the user's settings date format.
9. *autogenerated* - These are fields for which the values are generated automatically by vtiger, this is usually an object's id field.
10. *reference* - A field that shows a relation to another object, the type map will contain another element called refersTo which is an array containing the name of modules of which the field can point to.
11. *picklist* - A field that can hold one of a list of values, the map will contain two elements, picklistValues which is a list of possible values, and defaultValue which is the default value for the picklist.
12. *multipicklist* - A picklist field where multiple values can be selected.
13. *phone* - A field for storing phone numbers
14. *email* - A field for storing email ids
15. *url* - A field for storing urls
16. *skype* - A field for storing skype ids or phone numbers.
17. *password* - A field for storing passwords.

CRUD Operations

The api provides operations to create, retrieve, update and delete crm entity objects.

You can create a contact using the create operation, in this case the mandatory fields are lastname and assigned_user_id.

```
//e.g.
$contactData = array('lastname'=>'Valiant', 'assigned_user_id'=>$userId);
$objectJson = Zend_JSON::encode($contactData);
$moduleName = 'Contacts';

$params = array("sessionName"=>$sessionId, "operation"=>'create',
    "element"=>$objectJson, "objectType"=>$moduleName);
$http->post("$endpointUrl", $params, true);
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('create failed:'.$jsonResponse['error']['errorMsg']);
$savedObject = $jsonResponse['result'];
$id = $savedObject['id'];
```

\$savedObject is a map containing the fields of the new object including an id field which can be used to refer to the object.

To retrieve an object using its id use the retrieve operation. You can retrieve the contact created in the create example.

```
$params = "sessionName=$sessionId&operation=retrieve&id=$id";
$http->get("$endpointUrl?$params");
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);
```

```

if($jsonResponse['success']==false)
    die('retrieve failed:'.$jsonResponse['error']['errorMsg']);

$retrievedObject = $jsonResponse['result'];

```

To update a retrieve or newly created object you can use the update operation. You can add a first name to the contact.

```

$retrievedObject['firstname'] = 'Prince';
$objectJson = Zend_JSON::encode($retrievedObject);

$params = array("sessionName"=>$sessionId, "operation"=>'update',
    "element"=>$objectJson);
$http->post("$endpointUrl", $params, true);
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('update failed:'.$jsonResponse['error']['errorMsg']);

$updatedObject = $jsonResponse['result'];

```

To delete an object use the delete operation.

```

$params = array("sessionName"=>$sessionId, "operation"=>'delete', "id"=>$id);
$http->post("$endpointUrl", $params, true);
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('delete failed:'.$jsonResponse['error']['errorMsg']);

```

The delete does not have a result. The success flag is enough to find out if the operation was successful.

Queries

Vtiger provides a simple query language for fetching data. This language is quite similar to select queries in sql. There are limitations, the queries work on a single Module, and does not support joins. But this is still a powerful way of getting data from Vtiger.

```

$query = "select * from Contacts where lastname='Valiant';";
$queryParam = urlencode($query);
$params = "sessionName=$sessionId&operation=query&query=$queryParam";
$http->get("$endpointUrl?$params");
$response = $http->currentResponse();
$jsonResponse = Zend_JSON::decode($response['body']);

if($jsonResponse['success']==false)
    die('query failed:'.$jsonResponse['errorMsg']);

$retrievedObjects = $jsonResponse['result'];

```

This will return a array containing arrays representing the fields of each object that matched the query.