Example for Using the PrestaShop Web Service : CRUD

This tutorial shows you how to use the PrestaShop web service with PHP library by creating a "CRUD".

Prerequisites:
- PrestaShop 1.4 installed on a server with "module rewrite" enabled for apache
- A XAMPP server under PHP5

What is CRUD?

CRUD is an acronym that stands for "Create, Read, Update, Delete".

These are the four basic operations for managing data in an application.

The PrestaShop web service uses REST architecture in order to be available on as many platforms as possible, because HTTP and XML protocols are present on countless platforms.

What is REST?

REST defines an architecture that represents a set of good methods to practice on the web. It defines several rules, including one that we will describe that is similar to CRUD.

In HTTP we find four main methods that can perform processing on data that are defined in the REST architecture. This correspondence can also be done with CRUD:

- GET	-> Retrieve
- POST	-> Create
- PUT	-> Update
- DELETE	-> Delete

Together we will look at how to create a small application to perform these four operations on our customers.

Chapters 1, 2 and 3 are mandatory.

In the next chapters you'll learn how to interact with the web service with each of the CRUD operations to give you the keys to make a full CRUD.

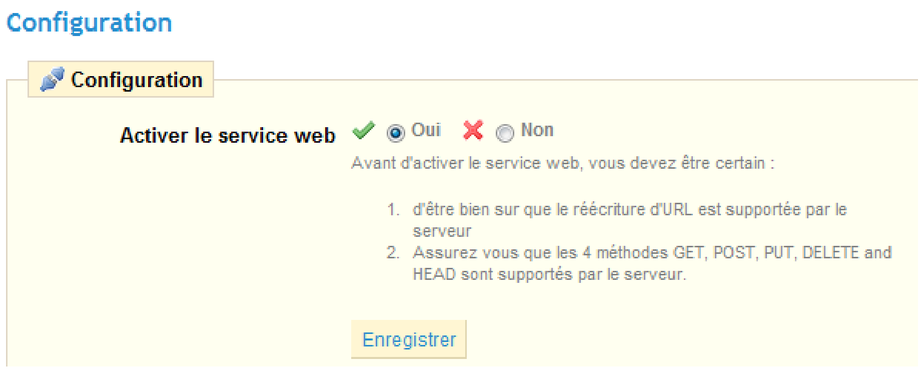
If you only want to retrieve data, for example in developing a web application to notify you of orders, you might only be interested in Chapter 4.

If you prefer to develop a more complete application, chapters 4 to 7 will interest you.

Contents

Chapter 1 - Introduction: Creating Access to Back Office

First we are going to create an access to the web service.
To do this, simply go into your Back Office and click on the tab Tools / Web Service.
Select "Enable Web Service" and click Save to activate the service.



Configuration

Configuration

Activer le service web ✓ [✓ Oui] ✗ Non

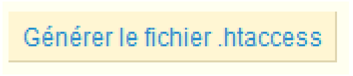
Avant d'activer le service web, vous devez être certain :

1. d'être bien sur que le réécriture d'URL est supportée par le serveur
2. Assurez vous que les 4 méthodes GET, POST, PUT, DELETE and HEAD sont supportés par le serveur.

Enregistrer

Generating a. htaccess file:

To get the web service to work, you need to generate / regenerate a. htaccess file.
While still in the Back Office, go to the Tools / Generators tab and click:

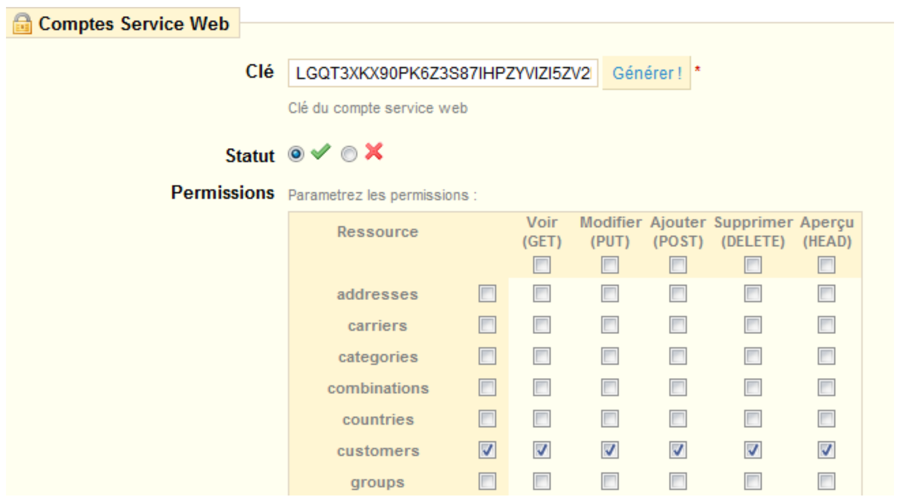


Creating access:

Return to Tools / Web Service

- Click "Add New ", and you'll access the "passkey" permission and definition page.
- Click on "Generate." This will generate an authentication key.

Thanks to this key, you will be able to access the web service.
Then you can create rights for each resource you want to access.



Comptes Service Web

Clé LGQT3XKX90PK6Z3S87IHPZYVIZI5ZV2 Générer ! *

Clé du compte service web

Statut [✓ ✓] ✗

Permissions Parametrez les permissions :

Ressource		Voir (GET)	Modifier (PUT)	Ajouter (POST)	Supprimer (DELETE)	Aperçu (HEAD)
		☐	☐	☐	☐	☐
addresses	☐	☐	☐	☐	☐	☐
carriers	☐	☐	☐	☐	☐	☐
categories	☐	☐	☐	☐	☐	☐
combinations	☐	☐	☐	☐	☐	☐
countries	☐	☐	☐	☐	☐	☐
customers	☑	☑	☑	☑	☑	☑
groups	☐	☐	☐	☐	☐	☐

In the list of permissions, the left button allows you to define all the rights for a given resource. Select the resources you need to manipulate from your application; in our case check the first check box in the "customers" row and then:

- Press "Save"

> Note: Please use the "Generate" button so that the key cannot be guessed.
> If you define your own passkey, make sure it is very secure and that these rights are limited.

Chapter 2 - Discovery: Testing access to the web service with the browser

To test if you have properly configured your access to the web service, go to the page **http://my passkey@mystore.com/api/** where "My passkey" is replaced by your key. Mozilla Firefox is the preferred browser to test access.

Note :

Another method is to go directly to the following page:

http://mystore.com/api/

The site should prompt you for a username and a password to enter. The ID is the authentication key and there is **no password**.

You will access the list of resources that you configured in your Back Office with all permissions granted.

Using "**XLink**", you'll be able to access your various resources.

What is "**XLink**" ?

XLink associates an XML file to another XML file via a link.

In the Customers tag, you should get these attributes:

```
<customers xlink:href="http://mystore.com/api/customers" get="true" put="true"
post="true" delete="true" head="true">
```

The get, put, post, and delete attributes have the value “true,” meaning that you have correctly configured the “customers” resource and that it is accessible.

You can now use the "XLink" that shows up on the URL "http://mystore.com/api/customers" and go to it.

Once the client list is displayed via “http://example.com/store/api/customers” you can access the XLink corresponding to each customer.

Example:

The XML file found on http://mystore.com/api/customers/1 whose link is found in the list of clients (see previous link) will give you the properties of customers that have ‘1’ as their ID.

You can also browse the web service to access all resources in XML.

Chapter 3 - First steps: Access the Web service and list client

Preparation :

Configure your PHP installation so that it has the CURL extension installed and activated:

With Windows: Place this line in your php.ini file: extension=php_curl.dll

Linux/Mac: install the CURL extension sudo apt-get install php5-curl

Copy the file provided by PSWebServiceLibrary.php to the root of your Web server. We will explain how to use this library in this tutorial.

Note :
You can do this tutorial on a local drive even while your store is on the internet.

Create a file list_the_clients.php at the root of the web server that you have chosen.
Specify where to find the web server in your file:

```
require_once('./ PSWebServiceLibrary.php');
```

Configured this way, your file should be found in the same folder as PSWebServiceLibrary.php.

3.1 - Access the web service

In this section we will see how to access the web service via the PHP library.

First, you must create a PrestaShopWebservice instance that takes 3 parameters in its constructor:

- The store's root path (ex: http://store.com/)
- The authentication key (ex: ZR92FNY5UFRERNI3O9Z5QDHWKTP3YIIT)
- A Boolean indicating whether the Web service must use its debug mode

If you do not understand the terms of object-oriented programming such as instance, method, or constructor, that's okay for the rest of the tutorial. Here's how you create a Web service call:

```
$webService = new PrestaShopWebservice('http://mystore.com/',
'ZR92FNY5UFRERNI3O9Z5QDHWKTP3YIIT', false);
```

Once the instance is created, you can access the following methods:
get (GET)
add (POST)
edit (PUT)
delete (DELETE)

We will develop the use of these methods in other parts of the tutorial.

3.2 - Handling errors

Learning error handling with the library is essential to begin. If you implement this verification directly, you immediately detect where the error comes from along with other information.

Error handling with the PHP library from the web service is done with the help of exceptions.

The principle: The treatments related to the PrestaShop web service must be within a try block which itself must be followed by a catch block to retrieve the errors and, if possible, to catch them.
Illustration:

```
try
{
        // Execution (stops and goes in the catch block if an error occurs)
}
catch
{
        // Error handling (tries to catch the error or the error display)
}
```

Example:

```
try
{
    // creating web service access
        $webService = new PrestaShopWebservice(
                                'http://maboutique.com/',
                                'ZR92FNY5UFRERNI3O9Z5QDHWKTP3YIIT',
                                false);
        // call to retrieve all clients
        $xml = $webService->get(array('resource' => 'customers'));
}
catch (PrestaShopWebserviceException $ex)
{
        $trace = $ex->getTrace();       // Retrieve all information on the error
        $errorCode = $trace[0]['args'][0]; // Retrieve the error code
    if ($errorCode == 401)
                echo 'Bad auth key';
        else
                echo 'Other error : <br />'.$ex->getMessage();
                                        // Shows a message related to the error
```

That means each creation or use of the library must be located within a "try" block. The “catch” block can then handle the error if it occurs during the execution of the try block.
Now we'll see how to list all customers via the web service, and then we will see the four CRUD methods.

3.3 - List clients

We will now look at how to view a full list of clients' IDs. We could display more information and customize it…but we'll take a look at that later in this tutorial.

To retrieve an XML file containing all the customers we have to use the "get" method whose array is as follows:

Key	Value
resource	customers

The value defines the resource that the web service will use in a future call. The value could be carrier types, countries or any other type of resource that can be found in the Web Service tab of the "Back Office".

Example:

```
$opt['resource'] = 'customers';
$xml = $webService->get($opt);
```

Calling the method will return us to a **SimpleXML** object containing all client IDs.
Now we need to access the tags that interest us in the XML file:

Structure:

```
<?xml>
<prestashop>
<customers>
      <customer>
          Client ID
      </customer>
...       Other client tags
</customers>
</prestashop>
```

By getting the return of "$ webService->get, " we are at the root of the document.
To access the fields of clients who are children from the Customers tag, we only need to retrieve all fields in an associative array in SimpleXML like this:

```
$resources = $xml->customers->children();
```

From there we can access client IDs easily. Here's an example with a path from identifiers:

```
foreach ($resources as $resource)
          echo $resource->attributes().'<br />';
```

Thanks to these elements, you create a table (HTML) containing all the customers' IDs before going on to the next chapter.
You can use the "customers" tab in the Back Office to find the IDs of all customers. If you encounter difficulties, please look at the file code "0-CustomersList.php" for the results you should get.

Chapter 4: Retrieve Data: Retrieving a Client

Objective: A Web application to list and display information from a customer
Difficulty: *
Problem: How to create a system that allows customers using IDs to retrieve customer records?

Preparation:

Duplicate the file list_the_clients.php from the previous step to a file named R-CRUD.php at the root of your Web server.
If you didn't succeed at the previous step, duplicate the file 0-CustomersList.php to a file named R-CRUD.php.

In the XML file containing the list of clients, we will find all the XLinks providing access to customer information.
Example:

```
<customers>
<customer id="1" xlink:href="http://example.com/store/api/customers/1" />
</customers>
```

Here we see that the xlink for the "customer" tag with ID 1 is:
"http://mystore.com/api/customers/1"
This link leads to an XML file containing information about the client with ID 1.

For this tutorial, in order to manage access to different clients you will proceed by associating page identifiers with customers via a GET parameter named "id".

Example:
The "http://mystore.com/R-CRUD.php?id=1" we will display the customer 1's file.
Modify the table created in the previous chapter to add a link to future customer files.
You'll have to isolate the display from the display list of a particular customer.

In order to do this you must isolate the display of your list by verifying, using isset, that the GET "id" parameter property is not present when viewing your list.

Calling the web service is exactly the same as for displaying the list, except for if you need to add the 'id' element to the table whose value is the id of a client.
At this moment, we are using "customers"or "client" resources. If we'd been trying to change the "countries " resources, this id would have been a country id.

```
$opt['resource'] = 'customers';
$opt['id'] = $_GET['id'];
$xml = $webService->get($opt);
```

Tip: Use "isset" before setting an ID allows you to easily carry out everything in this chapter.

Accessing resources is performed as above for displaying the list because the tags that interest us are children of the "customers " tag.

```
$resources = $xml->customers->children();
```

This path is done in another way (here in an HTML table):

```
foreach ($resources as $key => $resource)
        echo 'Name of field : '.$key.' - Value : '.$resource.'<br />';
```

You now have everything needed to create a script to both list and display information for a particular client.

Try creating this script "R-CRUD.php". If you encounter any difficulties, follow the example from the file"1-Retrieve.php" which corresponds to the result you should get.

We will see in another tutorial how to filter, sort and limit the number of items displayed in the list.

If you're in a hurry to implement these features, you can find more information in Chapter 8.

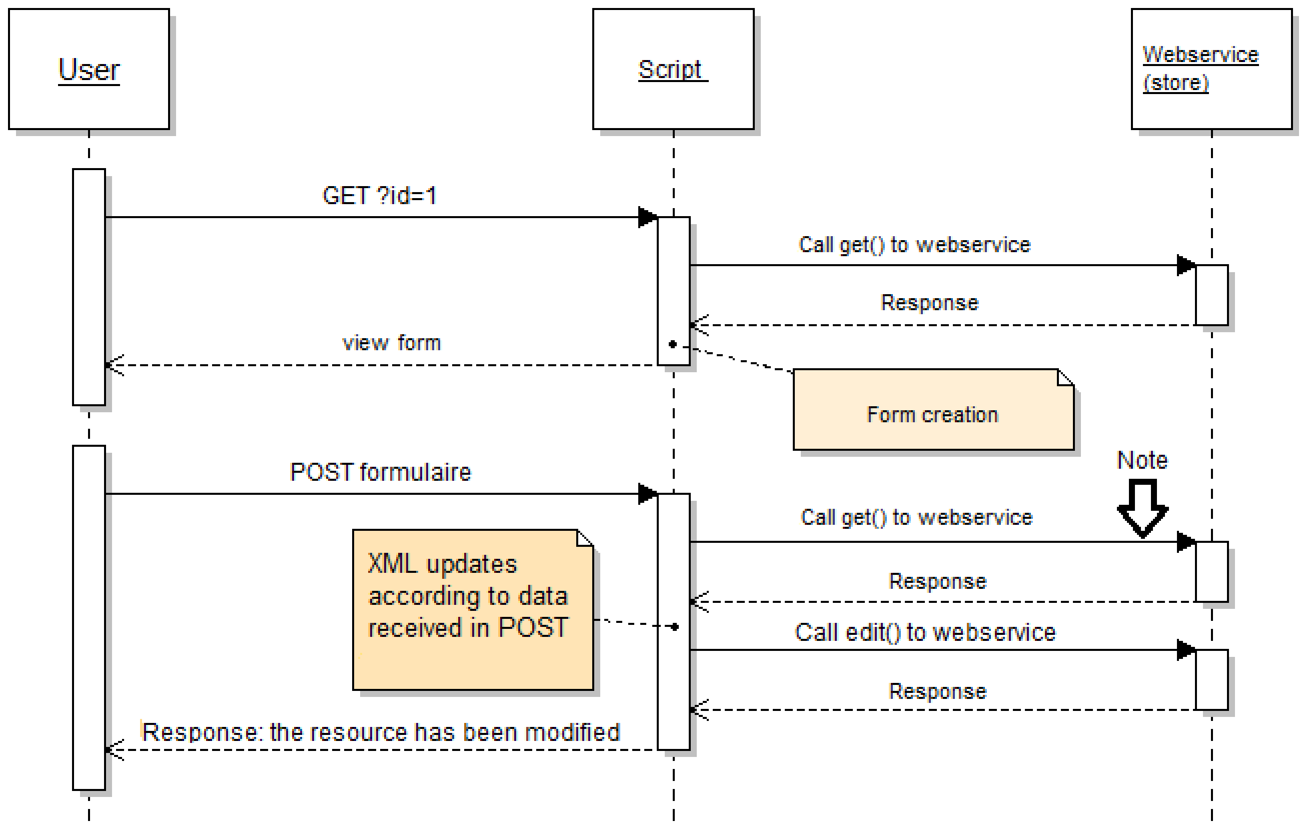
Chapter 5 - Modification: Update client

Objective: A Web application to list and update customer information.
Difficulty: ***

Preparation:

Duplicate file list_the_clients.php from Section 3.3 to a file named U-CRUD.php at the root of your Web server.

Updating resources via the web service is complex, so we will first explain its operation.
Sequence diagram representing how a resource is updated:



We can see that the diagram is divided into 2 stages:
- Getting the resource to a defined id (1 in the diagram) and creating of the form.
- Update resource.

Note (Down arrow on the diagram):
The arrow points to a "get", which corresponds to a resource getting.
This step is important because we need to get the XML file back in order to match it with the data sent by the form before we can call "edit" to update the resource.
Note that we could have modified it otherwise by sending an XML using Javascript, and thus have not used "get" in this process.

Step 1: Getting data and form creation

Retrieving the XML file and display the form:

```
    // Define the resource
        $opt = array('resource' => 'customers');
        // Define the resource id to modify
        $opt['id'] = $_GET['id'];
        // Call the web service, recuperate the XML file
        $xml = $webService->get($opt);
        // Recuperate resource elements in a variable (table)
        $resources = $xml->children()->children();
        // client form
```

Here, the call is similar to data getting. It is this call that will allow us to create the form.

We will generate the automatic update form.

For now, use HTML tags "input" as having as its "name" the name of the attribute, and as its "value" the value of the attribute.

In order not to lose the id for the second step according to the diagram, the form will show up as:

```
?id = "Customer Id"
```

Thus we will get it like this:

```
$_GET['id'];
```

We could have done this differently, such as by passing this ID in POST, but you will see that this method will simplify the processing that follows.

Step 2: Update the resource

Initially, as you can see from the "Note" arrow in the diagram, we will retrieve the XML file. For this, you will carry out the same call as you did when you created the form.

If you have specified, as indicated above, the form destination with an id, your call should already be done and the form will be redisplayed.

Help for creating a form:

```
foreach ($resources as $key => $resource)
{
        echo '<tr><th>'.$key.'</th><td>';
        echo '<input type="text" name="'.$key.'" value="'.$resource.'"/>';
        echo '</td></tr>';
}
```

Once the XML file is retrieved we need to modify the new data with data received by POST.

Path of the keys in the XML file and update values:

```
foreach ($resources as $nodeKey => $node)
{
        $resources->$nodeKey = $_POST[$nodeKey];
}
```

We now have an XML file updated. Now we just need to send it.

Example of an update:

```
$opt = array('resource' => 'customers');      // Resource definition
$opt['putXml'] = $xml->asXML();               //XML file definition
$opt['id'] = $_GET['id'];  // Definition of ID to modify
// Calling asXML () returns a string corresponding to the file
$xml = $webService->edit($opt);     // Call
```

Now, in your "U-CRUD.php" script, try to modify a customer with an ID defined in the code, then do it for all customers.

Check using "R-CRUD.php" that the information has been changed and then process the Client ID.

If you have trouble, look at the code for 2-update.php.

Chapter 6 - Creation: Remote Online Form

Objective: A Web application to list and create a new customer.
Difficulty: **

Preparation:

Copy the file list_the_clients.php from Section 3.3 to a file named C-CRUD.php at the root of your Web server.

The addition of resource can be likened to an upgrade from an empty element.

But how do we retrieve an XML formatted as an empty client?

In the web service, there is a method to retrieve an empty XML. It is accessible via a URL formatted as follows:

```
http://mystore.com/api/[resource name]?schema=blank
```

Note :
It is possible to replace the parameter scheme value "blank" with "synopsis" in order to gain more information about the resource fields.

As we saw in Section 3.3 (List customers) it is possible make an array of parameters for "get ", "resource," and "id." It is also possible to specify only one URL this way:

```
$xml = $webService->get(array('url' =>
'http://mystore.com/api/customers?schema=blank'));
```

Here, we get the entire XML variable from an empty client.

Beginning of the XML file retrieved:

```
<prestashop>
<customer>
<id_default_group/>
etc...
```

We can then, thanks to the many fields we have, create an associated form.

Getting of all fields:

```
$resources = $xml->children()->children();
```

Path of all fields and part of the dynamic creation of form fields in a table:

```
foreach ($resources as $key => $resource)
{
        echo '<tr><th>'.$key.'</th><td>';
        echo '<input type="text" name="'.$key.'" value=""/>';
        echo '</td></tr>';
}
```

Once the data is passed in POST, we combined the data sent with the blank XML file, which is the same technique used for updating data.

```
foreach ($resources as $nodeKey => $node)
{
        $resources->$nodeKey = $_POST[$nodeKey];
}
```

Calling the web service is similar to what we have seen previously:

```
$opt = array('resource' => 'customers');
$opt['postXml'] = $xml->asXML();
$xml = $webService->add($opt);
```

Now create a script that adds a client. Remember that some fields are mandatory, so do not forget to fill them out.

If you have trouble, look at the code for "3-Create.php.

Chapter 7 - Removal: Remove customer accounts from the database

Objective: A Web application for listing and deleting customers.
Difficulty: *

Preparation:

Duplicate file list_the_clients.php from Section 3.3 to a file named D-CRUD.php at the root of your Web server.

For this last part, we will learn how to delete a resource.
Here is the complete, detailed call you need to remove a client:

```
try
{
        $webService = new PrestaShopWebservice('http://mystore.com/',
        'ZR92FNY5UFRERNI3O9Z5QDHWKTP3YIIT', false); // Create an instance
        $opt['resource'] = 'customers';         // Resource to use
        $opt['id'] = 3;                         // ID to use
        $webService->delete($opt);              // Delete
        echo 'Client '.3.' supprimé avec succès !';     // if we can see this message then
        // that means that we have not left the try block
}
catch  (PrestaShopWebserviceException $ex)
{
        $trace = $ex->getTrace();               // Recuperate all info on this error
        $errorCode = $trace[0]['args'][0];      // Recuperate error code
        if ($errorCode == 401)
                echo 'Bad auth key';
        else
                echo 'Other error : <br />'.$ex->getMessage();
                                                // Display error message
}
```

This code allows you to remove a customer whose id is 3. As you can see, deleting the customer differs only slightly from retrieving a resource. In fact the only thing difference in the code lies in the method called.

We will no longer call this method "get" but instead simply "delete”!
You must now replace the customer ID by an ID defined dynamically.

Now create all the script that will display a list of customer IDs and delete a customer of your choice.

Again, if you have trouble, look at the code for "4-delete.php.”

Chapter 8 - Advanced Use

Rendering Options

Include all fields from the "products" resource.

URL :

```
« Store URL » /api/products/?display=full
```

PHP :

```
$opt = array('resource' => 'products', 'display' => 'full');
```

Include only the ID of all carriers "carriers"

URL :

```
« Store URL » /api/products/
```

PHP :

```
$opt = array('resource' => 'products');
```

Only include the "name" and "value" fields from the "configurations" resource.

URL :

```
« Store URL » /api/configurations/?display=[name,value]
```

PHP :

```
$opt = array('resource' =>'configurations', 'display' => '[name,value]');
```

Rendering Filters

Only include the first and last names of customers "customers" whose ids are between 1 and 5

```
« Store URL » /api/customers/?display=[firstname,lastname]&filter[id]=[1|5]
```

PHP :

```
$opt = array('resource' =>'customers', 'display' => '[firstname,lastname]', 'filter[id]' => '[1|5]');
```

Only include the first and last names of customers “customers” whose ids are between 1 and 10

```
« Store URL » /api/customers/?display=[lastname]&filter[id]=[1,10]
```

PHP :

```
$opt = array(‘resource’ =>’customers’, ‘display’ => ‘[lastname]’, ‘filter[id]’ => ‘[1,10]’);
```

Only include the birthday of clients whose name is “John” and whose last name is “Doe”

```
« Store URL »
/api/customers/?display=[birthday]&filter[firstname]=[John]&filter[lastname]=[DOE]
```

PHP :

```
$opt = array(‘resource’ =>’customers’, ‘display’ => ‘[birthday]’, ‘filter[firstname]’ => ‘[John]’,
‘filter[lastname]’ => ‘[DOE]’);
```

Only include the names of manufacturers “manufacturers” whose name begins with “Appl”

```
« Store URL » /api/manufacturers/?display=[name]&filter[name]=[appl]%
```

PHP :

```
$opt = array(‘resource’ => ‘manufacturers’, ‘display’ => ‘[name]’, ‘filter[name]’ => ‘[appl]%’);
```

Sorting Filters

Filter the customers “customers” in alphabetical order according to last name

```
« Store URL » /api/customers?display=full&sort=[lastname_ASC]
```

PHP :

```
$opt = array(‘resource’ => ‘customers’, ‘display’ => ‘full’, ‘sort’ => ‘[lastname_ASC]’);
```

Filters to limit rendering

Only include the first 5 states “states”

```
« Store URL » /api/states/?display=full&limit=5
```

PHP :

```
$opt = array(‘resource’ => ‘states’, ‘display’ => ‘full’, ‘limit’ => ‘5’);
```

Only include the first 5 elements starting from the 10th element from the states resource “states”

```
« Store URL » /api/states/?display=full&limit=9,5
```

PHP :

```
$opt = array(‘resource’ => ‘states’, ‘display’ => ‘full’, ‘limit’ => ‘9,5’);
```

Cheat Sheet: Concepts Outlined in this Tutorial

Method

To help you get started with web service, here's a little memo of techniques used in this tutorial.

			Method parameters				
	REST	Method	url	resource	id	postXml	putXml
C	POST	add	X	X		X	
R	GET	get	X	X	X		
U	UPDATE	edit	X	X	X		X
D	DELETE	delete	X	X	X		

If the URL parameter is specified, no other setting can be used and vice versa.

Options

Key	Key suffix		prefix	Value	Suffix	Description
display				[field1,field2 …]		Only display fields in brackets
display				full		Display all fields

Key	Key suffix		prefix	Value	Suffix	Description
filter	[field]			[value1|value2]		Filter "field" with value between 'value1 " and "value2"
filter	[field]			[value]		Filter field with the value “value”
filter	[field]			[value1,value2…]		Filter fields for values specified between brackets
filter	[field]		%	[value]	%	Filer “columns” for values containing “value”

Key	Key suffix		prefix	Value	Suffix	Description
sort				[field1_ASC,field2_DESC,field3_ASC]		Sort by field with the suffix _ASC _DESC or in the order
sort				full		show all fields

Key	Key suffix		prefix	Value	Suffix	Description
limit				Number		Limit the result to « Nombre »
limit				Starting index, Nombre		Limit the result to "Number" from the "Index"