crm-now/PS

REST Webservices API Tutorial

Version 1.5.1.0

Table Of Contents

OVERVIEW

The CRM system provides a simple, powerful and secure application programming interface (the API) to create, modify or delete CRM contents. This document is a tutorial for developers which intend to use REST based APIs to connect other applications with the CRM system.

API Call characteristics

- REST – the API is REST based. All communications between an application and the CRM server can happen over HTTP as GET or POST requests.
- JSON - JSON is used to encode response and request.
- Requests and Responses - The application prepares and submits a service request to the API, the API processes the request and returns a response, and the application handles the response.
- Committed Automatically - Every operation that writes to a CRM object is committed automatically. This is analogous to the AUTOCOMMIT setting in SQL.

API Response

Response Object

All responses will have the following format.

If the request is processed successfully,

```
<code>Response{
    success:Boolean=true
    result:Object //The Operation Result object
}
</code>
```

If there is a failure while processing the request,

```
<code>Reponse{
    success:Boolean=false
    error:ErrorObject
}
</code>
```

Error Object

```
<code>ErrorObject{
    errorCode:String //String representation of the error type
    errorMessage:String //Error message from the API.
}
</code>
```

errorCode is a string representation of the error type.

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 for the errorType, a single word description of the error, and errorMsg as a string containing a description of the error.

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

Requirements to run examples provided in this Tutorial

1. An installation of the CRM system.
2. PHP to run the example code
3. HTTP_Client which can be installed by doing a `pear install HTTP_Client`
4. Zend Json, you will need to download the minimal zend framework

Logging In

The API does not use the CRM users password to log in. Instead, the CRM provides an unique access key for each user. To get the user key for a user, go to the *My Preferences* menu of that user in the CRM. There you will find an Access Key field.

The Login procedure starts a client session with the CRM server, authenticates the user and returns a sessionId which will be used for all the subsequent communication with the CRM server.

Logging in is a two step process. In the first step the client obtains the challenge token from the server, which is used along with the user's access key for the login.

Get Challenge

Get a challenge token from the server. This must be a GET request.

URL format

```
<code>Request Type: GET
http://your_url/webservice.php?operation=getchallenge
    &username=[username]
</code>
```

Get Challenge Example

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

```
<code>//e.g.
```

```
//url path to CRM webservice: http://your_url/webservice.php
$endpointUrl = "http://localhost/ws/webservice.php";
//username of the user who is to logged in.
$userName="admin";

$httpc = new HTTP_CLIENT();
//getchallenge request must be a GET request.
$httpc-
>get("$endpointUrl?operation=getchallenge&username=$userName");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//check for whether the requested operation was successful or not.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('getchallenge failed:'.$jsonResponse['error']['errorMsg']);

//operation was successful get the token from the reponse.
$challengeToken = $jsonResponse['result']['token'];
</code>
```

The same procedure in .NET code:

Get Challenge Example

```
Dim httpreq1 As System.Net.HttpWebRequest =
System.Net.HttpWebRequest.Create(URL &
"/webservice.php?operation=getchallenge&username=" & username)
httpreq1.Method = "GET"

Dim response As System.Net.HttpWebResponse = httpreq1.GetResponse
Dim st As New System.IO.StreamReader(response.GetResponseStream)
```

st contains the JSON Object with the Challenge Token, which can be handled like a regular string.

GetChallengeResult

An object representing the response result of a getchallenge operation,

```
<code>GetChallengeResult{
    token:String //Challenge token from the server.
    serverTime:TimeStamp //The current server time.
    expireTime:TimeStamp //The time when the token expires.
}
</code>
```

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.

Login

Login to the server using the challenge token obtained in get challenge operation.

URL format

```
<code>Request Type:POST
http://your_url/webservice.php?operation=login
    &username=[username]
    &accessKey=[accessKey]
</code>
```

for accessKey create a md5 string after concatenating user accesskey from my preference page and the challenge token obtained from get challenge result.

Login Example

```
<code>//e.g.
//access key of the user admin, found on my preferences page.
$userAccessKey = 'dsf923rksdf3';

//create md5 string concatenating user accesskey from my preference
page
//and the challenge token obtained from get challenge result.
$generatedKey = md5($challengeToken.$userAccessKey);
//login request must be POST request.
$httpc->post("$endpointUrl",
    array('operation'=>'login', 'username'=>$userName,
        'accessKey'=>$generatedKey), true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('login failed:'.$jsonResponse['error']['errorMsg']);

//login successful extract sessionId and userId from LoginResult to
it can used for further calls.
$sessionId = $jsonResponse['result']['sessionName'];
$userId = $jsonResponse['result']['userId'];

</code>
```

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

The same procedure in .NET code:

Login Example

```
Dim md5Hasher As System.Security.Cryptography.MD5 =
System.Security.Cryptography.MD5.Create
Dim data As Byte() =
md5Hasher.ComputeHash(System.Text.Encoding.Default.GetBytes(token &
accesskey))
```

```
Dim sBuilder As New System.Text.StringBuilder()
Dim i As Integer
For i = 0 To data.Length - 1
sBuilder.Append(data(i).ToString("x2"))
Next i
Dim hash As String = sBuilder.ToString
Dim httpreq2 As System.Net.HttpWebRequest =
System.Net.HttpWebRequest.Create(url)
httpreq2.Method = "POST"
httpreq2.ContentType = "application/x-www-form-urlencoded"
Dim postData As String = "operation=login&username=" & username &
"&accessKey=" & hash
Dim urlbytes As Byte() = System.Text.Encoding.UTF8.GetBytes(postData)
httpreq2.ContentLength = urlbytes.Length
Dim dataStream As System.IO.Stream = httpreq2.GetRequestStream
dataStream.Write(urlbytes, 0, urlbytes.Length)
dataStream.Close()
response = httpreq2.GetResponse
Dim st2 As New System.IO.StreamReader(response.GetResponseStream)
```

st2 contains the JSON Object with the Session ID and user ID, which can be handled like a regular string.

LoginResult

An object representing the response result of a login operation,

```
<code>LoginResult{
    sessionId:String //Unique Identifier for the session
    userId:String //The CRM id for the logged in user
    version:String //The version of the webservices API
    vtigerVersion:String //The version of the CRM.
}
</code>
```

Getting Information

The API provides two operations to get information about available CRM objects.

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

List Types

List the names of all the CRM objects available through the API.

URL format

```
<code>Request Type: GET
```

```
http://your_url/webservice.php?operation=listtypes
    &sessionName=[session id]
</code>
```

Returns a map containing the key 'types' with the value being a list of names of CRM objects.

List Types Example

```
<code>//listtypes request must be GET request.
$httpc-
>get("$endpointUrl?sessionName=$sessionId&operation=listtypes");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('list types failed:'.$jsonResponse['error']['errorMsg']);
//Get the List of all the modules accessible.
$modules = $jsonResponse['result']['types'];
</code>
```

The second operation is called ‘describe’ which provides detailed type information about a CRM object.

The same procedure in .NET code:

List Types Example

```
Dim httpreq3 As System.Net.HttpWebRequest =
System.Net.HttpWebRequest.Create(URL &
"/webservice.php?operation=listtypes&sessionName=" & sessionid)

httpreq3.Method = "GET"

Dim response As System.Net.HttpWebResponse = httpreq3.GetResponse
Dim st3 As New System.IO.StreamReader(response.GetResponseStream)
```

Describe

Get the type information about a given CRM Object.

URL format

```
<code>Request Type: GET
http://your_url/webservice.php?operation=describe
    &sessionName=[session id]
    &elementType=[elementType]
</code>
```

Describe Example

```
<code>//e.g.
```

```
//CRM Object name which need be described or whose information is
requested.
$moduleName = 'Contacts';

//use sessionId created at the time of login.
$params =
"sessionName=$sessionId&operation=describe&elementType=$moduleName";
//describe request must be GET request.
$httpc->get("$endpointUrl?$params");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('describe object
failed:'.$jsonResponse['error']['errorMsg']);
//get describe result object.
$description = $jsonResponse['result'];
</code>
```

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 the CRM.
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 for floating point numbers.
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* - Thes are fields for which the values are generated automatically by the CRM, 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 a 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.
18. *owner* - A field for defining the owner of the field. which could be a group or individual user.

DescribeResult

```
<code>DescribeResult{
    label:String //label of the module.
    name:String  //name of the module, as one provided in request.
    createable:Boolean //true if the logged-in user is allowed create
records of type and false otherwise.
    updateable:Boolean //true if the logged-in user is allowed update
records of type and false otherwise.
    deleteable:Boolean //true if the logged-in user is allowed delete
records of type and false otherwise.
    retrieveable:Boolean //true if the logged-in user is allowed
retrievealbe records of type and false otherwise.
    fields:Array //array of type Field.
}
</code>
```

Field

```
<code>Field{
<pre class="_fck_mw_lspace">   name:String //name of the field.
   label:String //label of the field.
   mandatory:Boolean //true if the needs to be provided with value
while doing a create request and false otherwise.
   type:FieldType //an instance of FieldType.
   default:anyType //the default value for the field.
   nillable:Boolean //true if null can provided a value for the field
and false otherwise.
   editalbe:Boolean //true if field can provided with a value while
create or update operation.
}
<code></pre>
```

FieldType

```
<code>FieldType{
<pre class="_fck_mw_lspace">   name:Type //field type
   refersTo:Array //an array of CRM Object names of types of record
to which the field can refer to, its only defined for reference
FieldType.
```

```
   picklistValues:Array //an array of type PicklistValue type. its
only defined for picklist Fieldtype.
   defaultValue:String //an value of the default selected picklist
value, its only defined for picklist Fieldtype.
   format:String //date format in which date type field need to
populated with, eg, mm-dd-yyyy. its only defined for Date Fieldtype.
}
<code></pre>
```

picklist field value needs to provided explicitly.

PicklistValue

```
<code>PicklistValue{
    name:String //name of the picklist option.
    value:String //value of the picklist option.
}
</code>
```

Sample of *describeResult* Object

//$description from above example. For example, `print_r($description)` might display the result.

```
<code>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
                )
        )
)
</code>
```

CRUD Operations

The API provides operations to create, retrieve, update and delete CRM entity objects.

Create

Create a new entry on the server.

URL format

```
<code>Request Type: POST
http://your_url/webservice.php?operation=create
    &sessionName=[session id]
    &element=[object]
    &elementType=[object type]

</code>
```

Create Example

Example 1

You can create a contact using the create operation. You must consider mandatory field in the CRM marked by a red *, in this case the mandatory fields are lastname and assigned_user_id.

```
<code>//e.g.
//fill in the details of the contacts.userId is obtained from
loginResult.
$contactData  = array('lastname'=>'Valiant',
'assigned_user_id'=>$userId);
//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($contactData);
//name of the module for which the entry has to be created.
$moduleName = 'Contacts';

//sessionId is obtained from loginResult.
$params = array("sessionName"=>$sessionId, "operation"=>'create',
    "element"=>$objectJson, "elementType"=>$moduleName);
//Create must be POST Request.
$httpc->post("$endpointUrl", $params, true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('create failed:'.$jsonResponse['error']['errorMsg']);
$savedObject = $jsonResponse['result'];
$id = $savedObject['id'];
</code>
```

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

Example 2.

Create a Contact and associate with an existing Account.

```
<code>//e.g 2
//Create a Contact and associate with an existing Account.
```

```
$queryResult = doQuery("select accountname,id from accounts where
accountname='companyname';");
$accountId = $queryResult[0]['id'];
//For more details on how do a query request please refer the query
operation example.

//fill in the details of the contacts.userId is obtained from
loginResult.
$contactData  = array('lastname'=>'Valiant',
'assigned_user_id'=>$userId,'account_id'=>$accountId);
//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($contactData);
//name of the module for which the entry has to be created.
$moduleName = 'Contacts';

//sessionId is obtained from loginResult.
$params = array("sessionName"=>$sessionId, "operation"=>'create',
    "element"=>$objectJson, "elementType"=>$moduleName);
//Create must be POST Request.
$httpc->post("$endpointUrl", $params, true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('create failed:'.$jsonResponse['error']['errorMsg']);
$savedObject = $jsonResponse['result'];
$id = $savedObject['id'];
</code>
```

Example 3.

Create a Contact and associate with a new Account.

```
<code>//e.g 3
<p>//Create a Contact and associate with a new Account.
</p><p>//fill in the details of the Accounts.userId is obtained from
loginResult.
$accountData  = array('accountname'=>'Vtiger',
'assigned_user_id'=>$userId);
//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($accountData);
//name of the module for which the entry has to be created.
$moduleName = 'Accounts';
</p><p>//sessionId is obtained from loginResult.
$params = array("sessionName"=>$sessionId, "operation"=>'create',
</p>
<pre class="_fck_mw_lspace">   "element"=>$objectJson,
"elementType"=>$moduleName);

//Create must be POST Request. $httpc->post("$endpointUrl", $params,
true); $response = $httpc->currentResponse(); //decode the json
encode response from the server. $jsonResponse =
Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.

if($jsonResponse['success']==false)
```

```
   //handle the failure case.
   die('create failed:'.$jsonResponse['error']['errorMsg']);

$account = $jsonResponse['result']; $accountId = $account['id'];

$contactData = array('lastname'=>'Valiant',
'assigned_user_id'=>$userId,'account_id'=>$accountId);

//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($contactData); //name of the module
for which the entry has to be created. $moduleName = 'Contacts';



//sessionId is obtained from loginResult.
$params = array("sessionName"=>$sessionId, "operation"=>'create',

   "element"=>$objectJson, "elementType"=>$moduleName);

//Create must be POST Request. $httpc->post("$endpointUrl", $params,
true); $response = $httpc->currentResponse(); //decode the json
encode response from the server. $jsonResponse =
Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
  //handle the failure case.

   die('create failed:'.$jsonResponse['error']['errorMsg']);

$savedObject = $jsonResponse['result']; $id = $savedObject['id'];
</code></pre>
```

CreateResult

A Map representing the contents of a crmentity based object. All reference fields are represented using an *Id* type. A key called id of type *Id* represents the object's unique id. This field is present for any object fetched from the database.

Retrieve

Retrieve a existing entry from the server.

URL format

```
<code>Request Type: GET
http://your_url/webservice.php?operation=retrieve
    &session_name=[session id]
    &id=[object id
</code>
```

Retrieve Example

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

```
<code>//sessionId is obtained from loginResult.
$params = "sessionName=$sessionId&operation=retrieve&id=$id";
//Retrieve must be GET Request.
$httpc->get("$endpointUrl?$params");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);
//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('retrieve failed:'.$jsonResponse['error']['errorMsg']);

$retrievedObject = $jsonResponse['result'];
</code>
```

RetrieveResult

A Map representing the contents of a crmentity based object. All reference fields are represented using *Id* type. A key called id of type *Id* represents the object's unique id. This field is present for any object fetched from the database.

Update

To update a retrieved or newly created object, you can use the update operation.

URL format

```
<code>Request Type: POST
http://your_url/webservice.php?operation=update
    &sessionName=[session id]
    &element=[object]

</code>
```

Update Example

You can add a first name to the contact.

```
<code>//retrievedObject from previous example(Example 1).
$retrievedObject['firstname'] = 'Prince';
//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($retrievedObject);

//sessionId is obtained from the login operation result.
$params = array("sessionName"=>$sessionId, "operation"=>'update',
    "element"=>$objectJson);
//update must be a POST request.
$httpc->post("$endpointUrl", $params, true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('update failed:'.$jsonResponse['error']['errorMsg']);

//update result object.
```

```
$updatedObject = $jsonResponse['result'];
</code>
```

UpdateResult

A Map representing the contents of a crmentity based object. All reference fields are represented using *Id* type. A key called id of type *Id* represents the object's unique id. This field is present for any object fetched from the database.

Delete

To delete an object use the delete operation.

URL format

```
<code>Request Type: POST
http://your_url/webservice.php?operation=delete
    &sessionName=[session id]
    &id=[object id]
</code>
```

Delete Example

```
<code>
//delete a record created in above examples, sessionId a obtain from
the login result.
$params = array("sessionName"=>$sessionId, "operation"=>'delete',
"id"=>$id);
//delete operation request must be POST request.
$httpc->post("$endpointUrl", $params, true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('delete failed:'.$jsonResponse['error']['errorMsg']);

</code>
```

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

Queries

The CRM provides a simple query language for fetching data. This language is quite similar to select queries in sql. There are the following limitations:

- the queries work on a single module only,
- embedded queries are not supported,
- joins are not supported.

Nevertheless, the remaining options have a powerful ways for getting data from the CRM.

Query always limits its output to 100 records. Client application can use limit operator to get less records.

URL format

```
<code>Request Type: GET
http://your_url/webservice.php?operation=query
    &sessionName=[session id]
    &query=[query string]
</code>
```

Query Example

Example 1

```
<code>//query to select data from the server.
$query = "select * from Contacts where lastname='Valiant';";
//urlencode to as its sent over http.
$queryParam = urlencode($query);
//sessionId is obtained from login result.
$params = "sessionName=$sessionId&operation=query&query=$queryParam";
//query must be GET Request.
$httpc->get("$endpointUrl?$params");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('query failed:'.$jsonResponse['errorMsg']);

//Array of vtigerObjects
$retrievedObjects = $jsonResponse['result'];
</code>
```

Example 2

specify the columns to be fetch for each records.

```
<code>//query to select data from the server.
$query = "select lastname,firstname,account_id,assigned_user_id from
Contacts where lastname='Valiant';";
//urlencode to as its sent over http.
$queryParam = urlencode($query);
//sessionId is obtained from login result.
$params = "sessionName=$sessionId&operation=query&query=$queryParam";
//query must be GET Request.
$httpc->get("$endpointUrl?$params");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('query failed:'.$jsonResponse['errorMsg']);

//Array of vtigerObjects
$retrievedObjects = $jsonResponse['result'];
```

```
</code>
```

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

QueryResult

QueryResult is an Array of VtigerObjects.

VtigerObject

A Map representing the contents of a crmentity based object. All reference fields are represented using *Id* type. A key called id of type *Id* represents the object's unique id. This field is present for any object fetched from the database.

The Query format

```
<code>select * | <column_list> | <count(*)>
from <object> [where <conditionals>]
[order by <column_list>] [limit [<m>, ]<n>];

</code>
```

The column list in the order by clause can have at most two column names.

- column_list: comma separated list of field names.
- object: type name of the object.
- conditionals: condition operations or in clauses or like clauses separated by 'and' or 'or' operations these are processed from left to right. The is no grouping with bracket operators allowed.
- conditional operators: <, >, <=, >=, =, !=.
- in clauses: <column name> in (<value list>).
- like clauses: <column name> in (<value pattern>).
- value list: a comma separated list of values.
- m, n: integer values to specify the offset and limit respectively.

Sync

Sync will return a SyncResult object containing details of changes after modifiedTime.

URL format

```
<code>Request Type: GET
http://your_url/webservice.php?operation=sync
    &sessionName=[session id]
    &modifiedTime=[timestamp]
    &elementType=[elementType]
</code>
```

elementType: This is an optional parameter, if specified only the changes for that module after the given time are returned, otherwise changes to all user accessible modules after the given time are returned.

TimeStamp - A long representation of the number of seconds since unix epoch.

Example 1

Create a Account and capture it in the sync API.

```
<code>//time after which all the changes on the server are needed.
$stime = time();
//Create some data now so it is captured by the sync API response.
//Create Account.
//fill in the details of the Accounts.userId is obtained from
loginResult.
$accountData  = array('accountname'=>'Vtiger',
'assigned_user_id'=>$userId);
//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($accountData);
//name of the module for which the entry has to be created.
$moduleName = 'Accounts';

//sessionId is obtained from loginResult.
$params = array("sessionName"=>$sessionId, "operation"=>'create',
    "element"=>$objectJson, "elementType"=>$moduleName);
//Create must be POST Request.
$httpc->post("$endpointUrl", $params, true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('create failed:'.$jsonResponse['error']['errorMsg']);
$createResult = $jsonResponse['result'];

$params =
"operation=sync&modifiedTime=$stime&sessionName=$sessionId";

//sync must be GET Request.
$httpc->get("$endpointUrl?$params");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('query failed:'.$jsonResponse['errorMsg']);

//Array of vtigerObjects
$retrievedObjects = $jsonResponse['result'];
</code>
```

Response contains the Account which was created.

Example 2

Create an Account and a Contact, use sync API only for Accounts module only changes to Accounts module are returned.

```
<code>//time after which all the changes on the server are needed.
$stime = time();
//Create some data now so it is captured by the sync API response.

//fill in the details of the Accounts.userId is obtained from
loginResult.
$accountData  = array('accountname'=>'Vtiger',
'assigned_user_id'=>$userId);
//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($accountData);
//name of the module for which the entry has to be created.
$moduleName = 'Accounts';

//sessionId is obtained from loginResult.
$params = array("sessionName"=>$sessionId, "operation"=>'create',
    "element"=>$objectJson, "elementType"=>$moduleName);
//Create must be POST Request.
$httpc->post("$endpointUrl", $params, true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('create failed:'.$jsonResponse['error']['errorMsg']);
$createResult = $jsonResponse['result'];

//Create a Contact.
//fill in the details of the contacts.userId is obtained from
loginResult.
$contactData  = array('lastname'=>'Valiant',
'assigned_user_id'=>$userId);
//encode the object in JSON format to communicate with the server.
$objectJson = Zend_JSON::encode($contactData);
//name of the module for which the entry has to be created.
$moduleName = 'Contacts';

//sessionId is obtained from loginResult.
$params = array("sessionName"=>$sessionId, "operation"=>'create',
    "element"=>$objectJson, "elementType"=>$moduleName);
//Create must be POST Request.
$httpc->post("$endpointUrl", $params, true);
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('create failed:'.$jsonResponse['error']['errorMsg']);
$savedObject = $jsonResponse['result'];
$id = $savedObject['id'];

$params =
"operation=sync&modifiedTime=$stime&sessionName=$sessionId";

//sync must be GET Request.
$httpc->get("$endpointUrl?$params");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);
```

```
//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('query failed:'.$jsonResponse['errorMsg']);

//Array of vtigerObjects
$retrievedObjects = $jsonResponse['result'];
</code>
```

SyncResult

An object representing the response of a sync operation,

```
<code>SyncResult{
    updated:[Object] //List of Objects created or modified.
    deleted:[Id] //List of *Id* of objects deleted.
    lastModifiedTime:Timstamp //time of the latest change. which can
used in the next call to the Sync API to get all the latest changes
that the client hasn't obtained.
}
</code>
```

Logging Out

Logout

Logout from the webservices session, this leaves the webservice session invalid for furthur use.

URL format

```
<code>Request Type: GET
http://your_url/webservice.php?operation=logout
    &sessionName=[session id]
</code>
```

Example

```
<code>//SessionId is the session which is to be terminated.
$params = "operation=logout&sessionName=$sessionId";

//logout must be GET Request.
$httpc->get("$endpointUrl?$params");
$response = $httpc->currentResponse();
//decode the json encode response from the server.
$jsonResponse = Zend_JSON::decode($response['body']);

//operation was successful get the token from the reponse.
if($jsonResponse['success']==false)
    //handle the failure case.
    die('query failed:'.$jsonResponse['errorMsg']);
//logout successful session terminated.
</code>
```

Acknowledgement

This tutorial has been retrieved from "http://wiki.vtiger.com/index.php/vtiger510:WebServices_tutorials" with the friendly permission of vtiger. It has been modified to meet the specific requirements of the CRM offer from crm-now.