Remote objects

All server operations that use Remote Objects are performed asynchronously. Any code that depends on the request being completed, including handling returned results, must be placed in the callback function.

*https://developer.salesforce.com/docs/atlas.en-us.pages.meta/pages/pages\_remote\_objects.htm*

Visualforce Remote Objects are proxy objects that enable basic DML operations on sObjects directly from JavaScript.

Remote Objects remove some of the complexity from JavaScript remoting by reducing the need for @RemoteAction methods in an Apex controller or extension.

Behind the scenes, the Remote Objects controller handles sharing rules, field level security, and other data accessibility concerns. Pages that use Remote Objects are subject to all the standard Visualforce limits, but like JavaScript remoting, Remote Objects calls don’t count toward API request limits.

Using Visualforce Remote Objects consists of implementing two separate pieces of functionality on the same page.

1. Access definitions, written in Visualforce with the Remote Objects components. These components generate a set of JavaScript proxy objects that you can use in step 2.
2. Data access functions, written in JavaScript. These functions use the proxy objects that are made available by the access definitions to perform create, retrieve, update, and delete operations on your data.

Part1 code example  
  
*<!-- Remote Objects definition to set accessible sObjects and fields -->*

*<****apex:remoteObjects*** *jsNamespace="MyCorpModels">*

*<apex:****remoteObjectModel*** *name="Warehouse\_\_c"* ***jsShorthand****="Warehouse"* ***fields****="Name,Id">*

*<apex:****remoteObjectField*** *name="Phone\_\_c" jsShorthand="****Phone****"/> </apex:remoteObjectModel>*

*</apex:remoteObjects>*

создаем РО + РОМодель

jsShorthand задает псевдоним прокси обьекта в JS

jsNamespace – задает название можели по умолчанию SobjectModel но мжем задать и свое

These components generate JavaScript model classes, one per sObject in the access specification, which you use to make data access calls directly from your JavaScript code. Notice the use of the jsShorthand attribute, which maps the full Salesforce API name to a simpler, shorter name to use in your JavaScript code. If you plan to package and distribute your code, setting jsShorthand is essential because it eliminates the use of your organization’s namespace in the packaged code. Using the shorthand does all the work.

Part 2 code example  
JS

*var wh = new* ***SObjectModel****.Warehouse(); //Warehouse взят из jsShorthand*

***wh.retrieve*** *(…)*

Note that wh is a JavaScript model for the Warehouse\_\_c object, not a specific Warehouse record.

Wh represents a specific object, Warehouse, and provides a connection between your page’s JavaScript and the Salesforce service. wh can be used to perform the basic “CRUD” operations—create, read, update, and delete—on contact objects in the database.

We can set fields value as properties   
  
*var ct = new RemoteObjectModel.Contact({*

*FirstName: "Aldo",*

*LastName: "Michaels",*

*Phone: "(415) 555-1212"*

*});*

or by get() set() methods

*var ct = new RemoteObjectModel.Contact({ FirstName: "Aldo" });*

*ct.get('FirstName'); // 'Aldo'*

*ct.get('Phone'); // <undefined>*

*ct.set('FirstName', 'Benedict');*

*ct.set('Phone', '(415) 555-1212');*

**Create record**

***RemoteObjectModel.create({field\_values}, callback\_function)***

Create a record by calling create() on a Remote Objects model instance. create() accepts two arguments, both optional.

create() **doesn’t return a result directly**. The callback function enables you to handle the server response asynchronously.

***function callback(Error error, Array results, Object event) { // ... }***

**Retrieving Records**

***RemoteObjectModel.retrieve({criteria}, callback\_function)***

criteria can be a Remote Objects query object or a function that returns one. The following two calls are equivalent.

*ct.retrieve(****{where: {FirstName: {eq: 'Marc' }}}****, function() {}); // query object*

*ct.retrieve(function(){*

*return(****{where: {FirstName: {eq: 'Marc' }}}****);}, function() {});*

**Update Record**

*RemoteObjectModel.update([****record\_ids****], {****field\_values****}, callback\_function)*

**Upserting Records**

*RemoteObjectModel.upsert({field\_values}, callback\_function)*

The field\_values block enables you to set the values and save a record in one statement. Set field values as you do when you create a model, using a JSON string. For example, the following two calls to upsert() are equivalent.

If there’s an **Id field set on the contact**, the contact will be updated. If there’s no Id, a new contact is created.

**Deleting Records**

*RemoteObjectModel.del([record\_ids], callback\_function)*

When you delete multiple records this way, all of the records are deleted in the *same server-side transaction.*

**Format and Options for Remote Objects Query Criteria**

*ct.retrieve(*

*{* ***where:*** *{*

*FirstName: {****eq:*** *'Marc'},*

*LastName: {****eq:*** *'Benioff'}*

*},*

***orderby:*** *[ {LastName: 'ASC'}, {FirstName: 'ASC'} ],*

***limit: 1*** *},*

**Remote Objects Callback Functions**

Remote Objects sends all requests to the Salesforce service asynchronously. Your code handles responses to Remote Objects operations in a callback function that you provide. Callback functions handle updating the page with the results of the operation and errors that are returned.

**Best Practices for Using Remote Objects**

**Field Level Security**

Remote Objects respects your organization’s field level security settings. Keep this in mind when you create pages that use Remote Objects. Fields that aren’t accessible to the person viewing the page appear blank. Actions that modify field data (create(), update(), and upsert()) fail with an error if they include inaccessible fields in the request.

**Transaction Boundaries**

Remote Objects removes control of transaction boundaries from your code. Each Remote Objects operation (create(), update(), and so on) is a separate transaction. Each operation succeeds or fails on its own, which can be a problem when you need to create or modify multiple related objects as part of a business process

**Appropriate Placement and Testing of Business Logic**

* Security and consistency: Remember that users can lose their network connection in mid-transaction, or alter the way that your page’s JavaScript executes with Firebug and other tools. Remote Objects enforces your validation rules, triggers, sharing rules, field level security, and other data access restrictions, but if you put business rules in JavaScript instead of Salesforce, those can be interrupted, altered, or bypassed.
* Testability: Business logic on the server side can use the many tools that Salesforce provides for testing. For this reason, we encourage you to put complex behavior in Apex and use the Apex test framework to verify that it works as you intend.
* Performance: If your processing needs to look at many records as part of a transaction, but won’t display them in the browser, we recommend you avoid sending that data to the client, and instead process the data “locally” on the server. Think about what data your page needs to do its work, and make sure you’re not needlessly copying it over the wire.

**Handling Complexity**

Business logic in one place not in JS and APEX together.

**Remote Objects Limits**

Visualforce Although Remote Objects isn’t subject to some resource limits, it comes with limitations of its own.

Remote Objects is subject to the following limits.

* Remote Objects isn’t a way to avoid Salesforce service limits. Remote Objects calls aren’t subject to API limits, but Visualforce pages that use Remote Objects are subject to all standard Visualforce limits.
* You can retrieve a maximum of 100 rows in a single request. To display more rows, submit additional requests by using the OFFSET query parameter.
* Remote Objects doesn’t support Blob fields. You can’t retrieve or set the value of object fields of type Blob.
* Setting the rendered attribute to false on Remote Objects components disables the generation of the JavaScript for those Remote Objects. Any page functionality that depends on unrendered Remote Objects should also be disabled.

**VisualForcePage**

*<apex:page>*

*<!-- <apex:page controller="RemoteObjectAirportOverride"> -->*

*<apex:remoteObjects>*

*<apex:remoteObjectModel name="AirPort\_\_c" jsShorthand="Airport" fields="Name,Id,airport\_id\_\_c"></apex:remoteObjectModel>*

*<!-- <apex:remoteObjectModel name="AirPort\_\_c" jsShorthand="Airport" fields="Name,Id,airport\_id\_\_c" create="{!$RemoteAction.RemoteObjectAirportOverride.create}"/> -->*

*</apex:remoteObjects>*

*<script>*

*var ap = new SObjectModel.Airport();*

*var fetchAirport = function(){*

*ap.retrieve( { limit: 10}, function(err, records){*

*if(err){*

*alert(err.message);*

*}else{*

*var ul = document.getElementById("airportsList");*

*records.forEach(function(record){*

*var li = document.createElement("li");*

*li.appendChild(document.createTextNode(record.get("Id")+' // '+record.get("Name")+' // '+record.get("airport\_id\_\_c")));*

*ul.appendChild(li);*

*});*

*}*

*}*

*);*

*}*

*var createAirport = function(){*

*var airportName = document.getElementById("airname").value;*

*var airportInnerId = document.getElementById("airid").value;*

*ap.create(*

*{"Name": airportName, "airport\_id\_\_c": airportInnerId},*

*function(err, record, event){*

*if(err){*

*alert(err.message);*

*}else{*

*var sfid = document.getElementById("airportsfid");*

*sfid.appendChild(document.createTextNode(ap.get('Id')));*

*//sfid.appendChild(document.createTextNode(ap.get('Id') + ' Custom='+event.result.custom));*

*}*

*});*

*}*

*var updateAirport = function(){*

*var newAirportName = document.getElementById("updairname").value;*

*var airportSFId = document.getElementById("airportsfid").textContent;*

*ap.update(*

*[airportSFId],*

*{"Name":newAirportName},*

*function(err){*

*if(err){*

*console.log(err);*

*alert(err.message);*

*}else{*

*document.getElementById("updateresult").appendChild(document.createTextNode('AirPort Updated'));*

*}*

*});*

*}*

*function getAirpotByName(name){*

*}*

*var deleteAirport = function(recordSFId){*

*var deleteAirportName = document.getElementById("deletename").value;*

*ap.retrieve(*

*{'where': {'Name': {'eq': deleteAirportName}},*

*'limit': 1},*

*function(err, records) {*

*if(err){*

*alert(err.message);*

*}else{*

*if(records.length == 0) {*

*document.getElementById("deleteresult").appendChild(document.createTextNode('No such AirPort'));*

*}else{*

*ap.del(*

*[records[0].\_props.Id],*

*function(err, ids){*

*if(err){*

*alert(err.message);*

*}else{*

*document.getElementById("deleteresult").appendChild(document.createTextNode('AirPort Deleted'));*

*}*

*}*

*);*

*}*

*}*

*}*

*);*

*}*

*</script>*

*<apex:pageBlock title="List of 10 AirPorts (retrieve example)">*

*<ul id="airportsList"></ul>*

*<apex:pageBlockButtons>*

*<button onclick="fetchAirport()">Retrieve 10 Airports</button>*

*</apex:pageBlockButtons>*

*</apex:pageBlock>*

*<br/>*

*<apex:pageBlock title="Create new Airport (create example)">*

*<div>*

*AirportName: <input id="airname" name="airportname" type="text" size="30"/>*

*</div>*

*<div>*

*AirportInnerId: <input id="airid" name="airportid" type="text" size="15"/>*

*</div>*

*<p id="airportsfid"></p>*

*<apex:pageBlockButtons>*

*<button onclick="createAirport()">Create Airport</button>*

*</apex:pageBlockButtons>*

*</apex:pageBlock>*

*<br/>*

*<apex:pageBlock title="Update created Airport (update example)">*

*<div>*

*AirportName: <input id="updairname" name="updairportname" type="text" size="30"/>*

*</div>*

*<div id="updateresult"></div>*

*<apex:pageBlockButtons>*

*<button onclick="updateAirport()">Update Airport</button>*

*</apex:pageBlockButtons>*

*</apex:pageBlock>*

*<apex:pageBlock title="Retrieve/Delete Airport (retrieve/delete example)">*

*<div>*

*AirportName: <input id="deletename" name="deletaportname" type="text" size="30"/>*

*</div>*

*<div id="deleteresult"></div>*

*<apex:pageBlockButtons>*

*<button onclick="deleteAirport()">Delete Airport</button>*

*</apex:pageBlockButtons>*

*</apex:pageBlock>*

*</apex:page>*

**Controller**

*public with sharing class RemoteObjectAirportOverride {*

*@RemoteAction*

*public static Map<String, Object> create(String type, Map<String, Object> fields) {*

*Map<String, Object> result = RemoteObjectController.create(type, fields);*

*Map<String, Object> customResult =*

*new Map<String, Object> {'custom' => 'my custom data' };*

*customResult.putAll(result);*

*return customResult;*

*}*

*}*