# SharePoint 2013 Dev with CSOM and REST – Introduction

Rob Windsor

@robwindsor

pluralsight
hardcore developer training

## Course Outline

- **Introduction to the CSOM and REST API**
- **Programming with the Client Object Model**
  - Incudes, Working with Lists, Data-binding, Exception Handling, Authentication
- **Programming with the REST API**
  - Queries, Working with Lists, Data-binding, Calling External Services
- **Performing Common Tasks**
  - Taxonomy, Custom Lists, Uploading Documents, User Profiles, Search
- **Additional Topics**
  - JavaScript in Farm and Sandbox Solutions, Custom WCF Services, SOAP Web Services, ListData.svc

## Module Outline

- **Client Object Model**
  - Implementations
  - Communication with SharePoint
  - Load and LoadQuery
- **REST API**
  - REST API history
  - Using the REST API

## Client(-Side) Object Model (CSOM)

- **API used when building remote applications**
  - Designed to be similar to the Server Object Model
  - Introduced in SharePoint 2010, expanded in SharePoint 2013
- **Three implementations**
  - .NET Managed, Silverlight (plus Mobile), JavaScript
  - Façades on top of /_vti_bin/Client.svc
- **Communication with SharePoint done in batches**

## Three Implementations

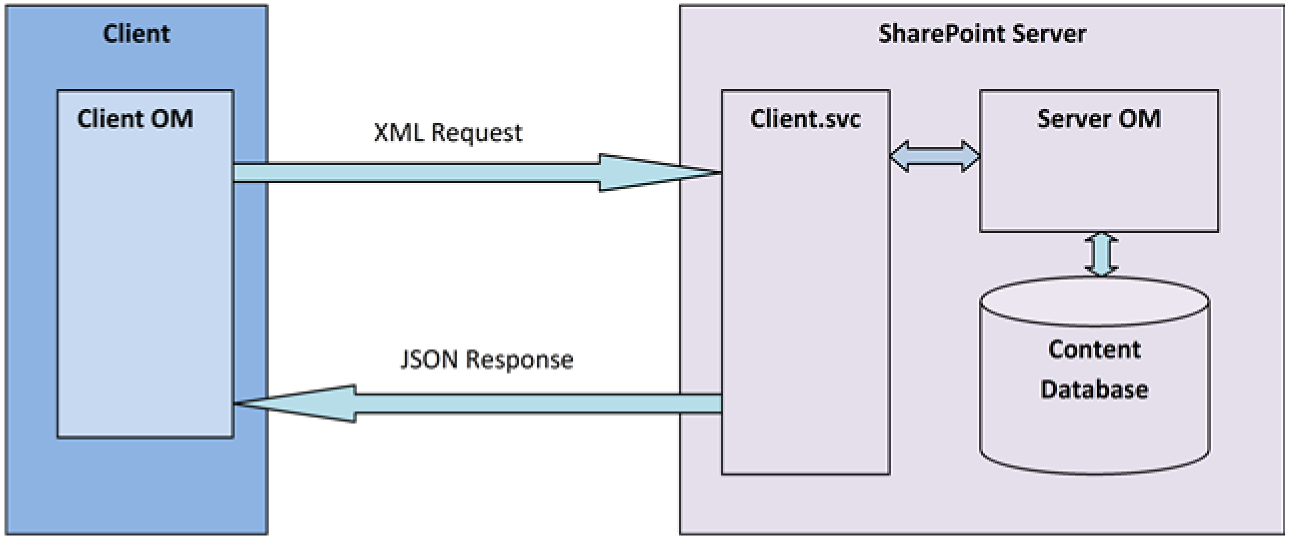
- **.NET Managed**
  - Located in <System Root>\ISAPI
  - Microsoft.SharePoint.Client.*.dll
- **Silverlight**
  - Located in <System Root>\TEMPLATE\LAYOUTS\ClientBin
  - Microsoft.SharePoint.Client.*.Silverlight.dll
  - Microsoft.SharePoint.Client.*.Phone.dll
- **JavaScript**
  - Located in <System Root>\TEMPLATE\LAYOUTS
  - SP.*.js

## Client Object Model Coverage

- Sites, Webs, Features, Event Receivers, Site Collections
- Lists, List Items, Fields, Content Types, Views, Forms
- Files, Folders
- Users, Roles, Groups, User Profiles, Feeds
- Web Parts
- Search
- Taxonomy
- Workflow
- IRM
- E-Discovery
- Analytics
- Business Data

## Communicating with SharePoint

- **All CRUD operations are automatically batched**
- **Requests for resources batched using Load and LoadQuery methods**
- **Batches are executed using ExecuteQuery or ExecuteQueryAsync**
    - XML document with batched request information sent to server
    - JSON response with requested resources returned



## Retrieving Resources Using Load

- Indicates object data should be included in next batch retrieval
- Not all property values are retrieved
    - Example: collections of associated objects

*Managed:*

```
var web = context.Web;
context.Load(web);
context.Load(web.Lists);
context.ExecuteQuery();
ResultsListBox.Items.Add(web.Title);
ResultsListBox.Items.Add(web.Lists.Count);
```

*JavaScript:*

```
var context = SP.ClientContext.get_current();
var web = context.get_web();
var lists = web.get_lists();
context.load(web);
context.load(lists);
context.executeQueryAsync(success, fail);

function success() {
    var div = jQuery("#message");
    div.text(web.get_title());
    div.append("<br />");
    div.append(lists.get_count());
}
```

## Retrieving Resources Using LoadQuery (Managed Code)

- **Indicates result of query should be included in next batch retrieval**
- **Query executed on server**
- **Result returned from call**
  - Not loaded in-place as with Load

```
var web = context.Web;

var query = from list in web.Lists
            where list.Hidden == false &&
                    list.ItemCount > 0
            select list;
var lists = context.LoadQuery(query);
context.ExecuteQuery();

Console.WriteLine(lists.Count());
```

# Retrieving Resources Using loadQuery (JavaScript)

- No LINQ in JavaScript
- loadQuery very similar to load
  - Returns new object
  - Returns array for collections

load:

```
var context = SP.ClientContext.get_current();
var lists = context.get_web().get_lists();
context.load(lists);
context.executeQueryAsync(success, fail);

function success() {
    var div = jQuery("#message");
    div.text(lists.get_count());
}
```

loadQuery:

```
var context = SP.ClientContext.get_current();
var lists = context.get_web().get_lists();
var myLists = context.loadQuery(lists);
context.executeQueryAsync(success, fail);

function success() {
    var div = jQuery("#message");
    div.text(myLists.length);
}
```

## REST API

- **Another API used when building remote applications**
- **What is the REST API in SharePoint**
  - Data-centric web services based on the Open Data Protocol (OData)
    - More on OData later
  - Each resource or set of resources is addressable
    - http://<site url>/_api/web
    - http://<site url>/_api/web/lists
    - http://<site url>/_api/web/lists/getByTitle('Customers')
  - Operations on resources map to HTTP Verbs
    - GET, PUT, POST, DELETE, …
  - Results from service returned in AtomPub (XML) or JavaScript Object Notation (JSON) format

## REST API History

- **SharePoint 2010**
  - Initial REST API added
  - /_vti_bin/ListData.svc
  - Exposed CRUD operations on list data
- **SharePoint 2013**
  - REST API expands and evolves
  - ListData.svc deprecated
    - Still available for backwards compatibility
  - RESTful operations added to /_vti_bin/Client.svc
  - /_api added as an alias for /_vti_bin/Client.svc

## REST API Coverage

- Sites, Webs, Features, Event Receivers, Site Collections
- Lists, List Items, Fields, Content Types, Views, Forms, IRM
- Files, Folders
- Users, Roles, Groups, User Profiles, Feeds
- Search

# Retrieving Data using REST API (Managed Code)

- /_api does not expose metadata
  - You cannot add a Service Reference in Visual Studio
- Two options
  - Get data in XML format and use LINQ to XML
  - Get data in JSON format and use built-in or third-party serializer
    - JavaScriptSerializer, JSON.NET, …

XML:

```
var url = "http://localhost/sites/dev/_api/Web/";
var client = new WebClient();
client.UseDefaultCredentials = true;
var xml = client.DownloadString(url);

var doc = XDocument.Parse(xml);
XNamespace ds = "http://schemas.microsoft.com/" +
    "ado/2007/08/dataservices";
var titles = doc.Descendants(ds + "Title");
var title = titles.First().Value;

Console.WriteLine(title);
```

JSON:

```
var url = "http://localhost/sites/dev/_api/Web/";
var client = new WebClient();
client.UseDefaultCredentials = true;
client.Headers[HttpRequestHeader.Accept] =
    "application/json;odata=verbose";
var json = client.DownloadString(url);

var ser = new JavaScriptSerializer();
dynamic item = ser.Deserialize<object>(json);

Console.WriteLine(item["d"]["Title"]);
```

# Retrieving Data using REST API (JavaScript)

- Use jQuery or SP.RequestExecutor to make service call
- Use _spPageContextInfo to get site URL

```
var call = jQuery.ajax({
    url: _spPageContextInfo.webAbsoluteUrl + "/_api/Web/",
    type: "GET",
    dataType: "json",
    headers: {
        Accept: "application/json;odata=verbose"
    }
});
call.done(function (data, textStatus, jqXHR) {
    var div = jQuery("#message");
    div.text(data.d.Title);
});
call.fail(function (jqXHR, textStatus, errorThrown) {
    var response = JSON.parse(jqXHR.responseText);
    var message = response ? response.error.message.value : textStatus;
    alert("Call failed. Error: " + message);
});
```

## Summary

- **Client Object Model**
  - Implementations
  - Communication with SharePoint
  - Load and LoadQuery
- **REST API**
  - REST API history
  - Using the REST API

- **More detail on each API coming up**