# SharePoint 2013 Dev with CSOM and REST – Programming with the Client Object Model

Rob Windsor
@robwindsor

pluralsight
hardcore developer training

# Outline

- **Selecting fields to retrieve**
- **Nesting includes**
- **Retrieving list items**
  - Using CAML queries
- **Data-binding**
- **Creating a list**
- **Batch exception handling**
- **Adding and updating list items**
- **Calling external services with Web Proxy**
- **Communicating with the Host Web from an App**
- **Authentication**

# Selecting Fields to Retrieve

- Limit fields returned to reduce network traffic
- Use parameter array in Load and LoadQuery
- Use Include for collections

*Managed:*

```
var web = context.Web;
context.Load(web, w => w.Title, w => w.Description);

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

*JavaScript:*

```
var web = context.get_web();
var lists = web.get_lists();
context.load(web, "Title", "Description");
context.load(lists, "Include(Title)");
context.executeQueryAsync(success, fail);
```

# Nested Includes

- Includes can be nested to reduce service calls

Managed:

```
var web = context.Web;
var lists = web.Lists;
context.Load(lists, lc => lc.Include(
    l => l.Title,
    l => l.Fields.Include(
        f => f.Title)));
context.ExecuteQuery();
```

JavaScript:

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

# Retrieving List Items

- **Somewhat different than Server OM**

| Task | Server OM | Managed Client OM |
|---|---|---|
| Get list | web.Lists["Products"] | web.Lists.GetByTitle("Products") |
| Get items | list.Items | list.GetItems(query) |
| Get item title | item.Title | item["Title"] |
| Query type | SPQuery | CamlQuery |

- **Set of items accessed by List.GetItems method**
  - Forces use of CAML query to encourage reduced result sets
- **Selecting fields to be returned**
  - Can use ViewFields in query
  - Can use Include with Load or LoadQuery
- **CSOM does not support cross-list CAML queries**
  - Can use KeywordQuery with Search API for similar results

# Using CAML Queries

Managed:

```
var web = context.Web;
var list = web.Lists.GetByTitle("Products");
var query = new CamlQuery();
query.ViewXml = "<View>" +
                "<Query>" +
                "<Where><Eq>" +
                "<FieldRef Name='Category' " +
                    "LookupId='True' />" +
                "<Value Type='Lookup'>1</Value>" +
                "</Eq></Where>" +
                "</Query>" +
                "</View>";
var items = list.GetItems(query);
context.Load(items,
    c => c.Include(li => li["ID"], li => li["Title"]));
context.ExecuteQuery();
```

JavaScript:

```
var context = SP.ClientContext.get_current();
var web = context.get_web();
var list = web.get_lists().getByTitle("Products");
var query = new SP.CamlQuery();
query.set_viewXml("<View>" +
                  "<Query>" +
                  "<Where><Eq>" +
                  "<FieldRef Name='Category' " +
                      "LookupId='True' />" +
                  "<Value Type='Lookup'>1</Value>" +
                  "</Eq></Where>" +
                  "</Query>" +
                  "<RowLimit>5</RowLimit>" +
                  "</View>");
var items = list.getItems(query);
context.load(web, "Title");
context.load(items, "Include(ID, Title)");
context.executeQueryAsync(success, fail);
```

# Data-Binding

- **JavaScript**
  - Use LoadQuery to get results as array
  - Use one of the several JavaScript libraries that support templates
    - jQuery Templates, jsRender, Knockout…
  - Bind data to template
- **Managed**
  - ListItemCollection does not have GetDataTable helper
  - Most portable option if to copy results into data structure
    - DataTable, custom type…
  - Bind using native binding mechanism of host framework
    - Windows Forms, WPF, Web Forms, MVC…

# Data-Binding

Managed:

```
var listData = new List<DictionaryEntry>();
foreach (var item in items)
{
    var data = new DictionaryEntry(item["ID"],
     item["Title"]);
    listData.Add(data);
}

ResultsListBox.DataSource = listData;
ResultsListBox.DisplayMember = "Value";
ResultsListBox.ValueMember = "Key";
```

JavaScript:

```
<script id="products-template" type="text/x-jsrender">
    <ul>
        {{for #data}}
        <li>
            <b>{{>#data.get_item('Title')}}</b>
            <br />
            {{>#data.get_item('UnitsInStock')}}
            in stock at
            {{>#data.get_item('UnitPrice')}}
        </li>
        {{/for}}
    </ul>
</script>
```

```
var template = jQuery("#products-template");
message.append(template.render(itemsArray));
```

# Creating a List

- **Moderately different than code for Server Object Model**
- **Adding the list**
  - Web.Lists.Add(creationInformation)
  - Parameter is type ListCreationInformation

*Managed:*

```
var web = context.Web;
var lci = new ListCreationInformation();
lci.Title = "Tasks";
lci.QuickLaunchOption = QuickLaunchOptions.On;
lci.TemplateType = (int)ListTemplateType.Tasks;
var list = web.Lists.Add(lci);
```

*JavaScript:*

```
var web = context.get_web();
var lci = new SP.ListCreationInformation();
lci.set_title("Tasks");
lci.set_quickLaunchOption(SP.QuickLaunchOptions.on);
lci.set_templateType(SP.ListTemplateType.tasks);
var list = web.get_lists().add(lci);
```

# Batch Exception Handling

- Traditional exception handling may require multiple service calls
- Client Object Model includes batch exception handling
  - Exception handling information included with batch request

```
var scope = new ExceptionHandlingScope(context);

using (scope.StartScope()) {
    using (scope.StartTry()) {
    }

    using (scope.StartCatch()) {
    }

    using (scope.StartFinally()) {
    }
}
```

# Adding and Editing List Items

- **Virtually the same as code for Server Object Model**
- **Adding a list item**
  - List.AddItem(creationInformation)
  - Parameter is type ListItemCreationInformation
- **Updating field values**
  - Exactly the same as Server Object Model code

*Managed:*

```
var web = context.Web;
var list = web.Lists.GetByTitle("Tasks");

var ici = new ListItemCreationInformation();
var item = list.AddItem(ici);
item["Title"] = "Sample Task";
item["AssignedTo"] = web.CurrentUser;
item["DueDate"] = DateTime.Now.AddDays(7);
item.Update();
```

*JavaScript:*

```
var web = context.get_web();
var list = web.get_lists().getByTitle("Tasks");

var ici = new SP.ListItemCreationInformation();
var item = list.addItem(ici);
item.set_item("Title", "Sample Task");
item.set_item("AssignedTo", web.get_currentUser());
var due = new Date();
due.setDate(due.getDate() + 7);
item.set_item("DueDate", due);
item.update();
```

# Web Proxy

- Use SP.WebProxy to make requests to external services
- Request made by SharePoint
- Response forwarded to calling code
- Apps must register target site as remote endpoint in app manifest
- Response size must not exceed 200 Kb

```
var context = SP.ClientContext.get_current();
var request = new SP.WebRequestInfo();
request.set_url("<external service url>");
request.set_method("GET");
var response = SP.WebProxy.invoke(context, request);
context.executeQueryAsync(success, fail);

function success() {
    if (response.get_statusCode() == 200) {
        var data = JSON.parse(response.get_body());
    } else {
        var errorMessage = response.get_body();
    }
}
```

# Connecting to the Host Web

- Get host web URL from query string
- Use SP.AppContextSite to get context for host web
- App must request permissions to read/write resources

```
var hostUrl = decodeURIComponent(getQueryStringParameter("SPHostUrl"));

var context = SP.ClientContext.get_current();
var hostContext = new SP.AppContextSite(context, hostUrl);
var web = hostContext.get_web();

var list = web.get_lists().getByTitle("Categories");
var camlQuery = new SP.CamlQuery();
camlQuery.set_viewXml("<View />");
var qitems = list.getItems(camlQuery);

var items = context.loadQuery(qitems, "Include(Title)");
context.executeQueryAsync(success, fail);
```

# Authentication

- **.NET Managed**
  - Windows credentials passed by default
  - ClientContext.AuthenticationMode
    - Default
    - Anonymous
    - FormsAuthentication
  - ClientContext.Credentials
    - Expects System.Net.ICredentials
    - NetworkCredential, SharePointOnlineCredentials, …
  - ClientContext.FormsAuthenticationLoginInfo
- **Silverlight and JavaScript**
  - Credentials of the hosting Web application are always used

# Authentication

```
var siteUrl = "https://vandelay-industries.sharepoint.com";
var loginName = "george@vandelay-industries.onmicrosoft.com";
var password = "bosco";

var securePassword = new SecureString();
password.ToCharArray().ToList().ForEach(c => securePassword.AppendChar(c));

using (var context = new ClientContext(siteUrl))
{
    context.Credentials = new SharePointOnlineCredentials(
        loginName, securePassword);

    var web = context.Web;
    var list = web.Lists.GetByTitle("Site Pages");

    var query = new CamlQuery();
    query.ViewXml = "<View />";
    var items = list.GetItems(query);
    context.Load(items, c => c.Include(li => li.File.Name));
    context.ExecuteQuery();

    foreach (var item in items)
    {
        Console.WriteLine(item.File.Name);
    }
}
```

# Summary

- **Selecting fields to retrieve**
- **Nesting includes**
- **Retrieving list items**
  - Using CAML queries
- **Data-binding**
- **Creating a list**
- **Batch exception handling**
- **Adding and updating list items**
- **Calling external services with Web Proxy**
- **Communicating with the Host Web from an App**
- **Authentication**