

# SharePoint 2010 Client Object Model

Mark Nischalke

## SharePoint 2010 Client Object Model

In previous versions of SharePoint when it was necessary to access ListItems or other objects from within a SharePoint environment the only choice available was to use the server object model, perhaps from the code behind in a webpart or application page, or in a Service running on the SharePoint machine. Outside of a SharePoint environment, the only option was to use Web Services with all of the inherent limitations and inefficiencies.

Although the Web Services are still available, and can be useful for some situations, there is a better way of working with SharePoint 2010; the Client Object Model. The Client Object Model allows developers to utilize SharePoint directly from .NET managed code, in a Windows WPF application for instance, Silverlight, in browser and out of browser, or from JavaScript without the need to directly utilize Web Services. This article will discuss the inner workings of the Client OM and how to use it effectively, not just to work Lists and ListItems, but other SharePoint objects not available via Web Services. This article will focus on the details while Part 2 will focus practical usage in all supported environments.

## Supported Environments

The SharePoint Client Object Model is supported in three environments; .NET managed code, JavaScript and SilverLight.

## .NET Managed Code Assemblies

To use the Client Object Model in .NET Managed code you must add references to two assemblies in your Visual Studio project. Both of these assemblies can be found in [SharePoint Root]\ISAPI folder, which is typically C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI\.

**Microsoft.SharePoint.Client.dll (282kb)**

**Microsoft.SharePoint.Client.Runtime.dll (146kb)**

As you can see from the indicated sizes these are not large assemblies and won't bloat an application very much. Despite their relatively diminutive size however they pack a great deal of punch, as you will soon see.

## Silverlight Assemblies

Just like .NET managed code, to use the SharePoint Client OM from within a Silverlight application you must add two assembly references. These assemblies are a little trickier to find and are located in the [SharePoint Root]\Templates\Layouts\ClientBin folder.

**Microsoft.SharePoint.Client.Silverlight.dll (266kb)**

**Microsoft.SharePoint.Client.Silverlight.Runtime.dll (142kb)**

The location is understandable considering the the Layout folder is mapped to each SharePoint Web Application and the heavy use of Silverlight within SharePoint 2010. You will notice the size of these assemblies is just slightly smaller than their .NET managed code counterparts. Despite this, there is no difference in functionality.

# SharePoint 2010 Client Object Model

Mark Nischalke

## JavaScript Code

To make use of the Client OM in JavaScript you must include the SP.js file on the page. The corresponding SP.Runtime.js will be added automatically by SharePoint.

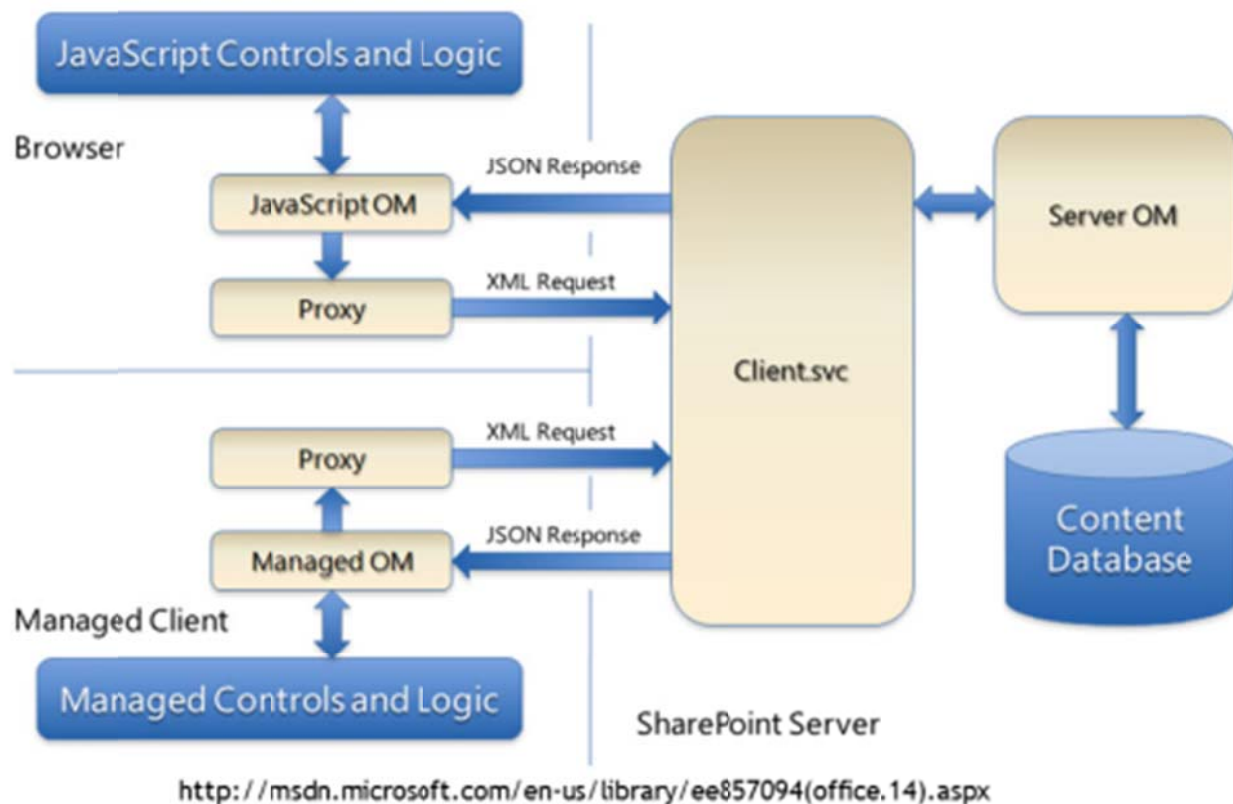
## FormDigest and Security

SharePoint uses the FormDigest control to generate a security token based on the user, site and time period and uses it to validate any action posting data and updating the content database. This control is required to be present on all pages where the Client OM is being used. However, since it is included in the SharePoint master pages, v4.master and default.master, it should not be an issue. Just keep it in mind when constructing your own master pages.

## Client OM Architecture

Before getting to actual usage we'll take a brief look at the architecture of the Client Object Model to understand how it functions.

# Client Object Model Mechanics



As can be seen in the above diagram all versions of the Client OM go through the WCF Web Service named Client.svc. This Web Service is located, along with all other SharePoint Web Services, in the folder [SharePoint Root]\ISAPI. The Client OM is responsible for packaging any requests into XML and calling the Web Server, however, when this call takes place is controlled by the developer as you will see. The response from the Client.svc Web Service is sent as JSON and the Client OM is responsible for transforming this into appropriate objects for use in the environment the call was made from.

# SharePoint 2010 Client Object Model

Mark Nischalke

The Client.svc Web Service has one method, ProcessQuery which takes a Stream object, an XML formatted stream as indicated above, and returns a Stream, JSON formatted. The implementation for this method can be found in the assembly Microsoft.SharePoint.Client.ServerRuntime.dll and the private class ClientRequestServiceImpl. Delving into this method, the first thing you find is an authentication check.

```
if (Utility.ShouldForceAuthentication(HttpContext.Current))
{
    WebOperationContext.Current.OutgoingResponse.StatusCode =
    HttpStatusCode.Unauthorized;
    WebOperationContext.Current.OutgoingResponse.SuppressEntityBody = true;
    return new MemoryStream();
}

internal static bool ShouldForceAuthentication(HttpContext httpContext)
{
    return (((httpContext != null)
        && (string.Compare(httpContext.Request.Headers["X-RequestForceAuthentication"],
            "true", StringComparison.OrdinalIgnoreCase) == 0))
        && (((httpContext.User == null) || (httpContext.User.Identity == null))
            || !httpContext.User.Identity.IsAuthenticated));
}
```

The Client OM uses Windows Authentication by default, however, later I'll show how to utilize Claims Based Authentication. As shown in this code, authentication is not performed by this Web Service, it simply returns an Unauthorized status if authentication has not been completed previously. Authentication is handled by one of the legacy Web Services, Sites.asmx. When authentication has been confirmed the response content type is set to JSON as stated previously...

```
WebOperationContext.Current.OutgoingResponse.ContentType = "application/json";
WebOperationContext.Current.OutgoingResponse.Headers.Add("X-Content-Type-Options",
"nosniff");
...then the XML from the input stream is processed.
```

```
using (ClientMethodsProcessor processor = new ClientMethodsProcessor(inputStream,
host))
{
    processor.Process();
    stream = processor.CreateResultUTF8Stream();
}
```

The XML that comes to this Web Service method is a collection of ObjectPath elements with attributes and child elements that detail the request being made. The ClientMethodsProcessor constructor method creates a Dictionary of these elements which is used in the Process method and eventually in the ProcessOne method based on the type of action the element represents.

```
private void ProcessOne(XmlElement xe)
{
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
this.ThrowIfTimeout();
if (xe.Name == "Query")
{
    this.ProcessQuery(xe);
}
else if (xe.Name == "Method")
{
    this.ProcessMethod(xe);
}
else if (xe.Name == "StaticMethod")
{
    this.ProcessStaticMethod(xe);
}
else if (xe.Name == "SetProperty")
{
    this.ProcessSetProperty(xe);
}
else if (xe.Name == "SetStaticProperty")
{
    this.ProcessSetStaticProperty(xe);
}
else if (xe.Name == "ObjectPath")
{
    this.ProcessInstantiateObjectPath(xe);
}
else if (xe.Name == "ObjectIdentityQuery")
{
    this.ProcessObjectIdentityQuery(xe);
}
else if ((xe.Name == "ExceptionHandlingScope" || xe.Name ==
"ExceptionHandlingScopeSimple"))
{
    this.ProcessExceptionHandlingScope(xe);
}
else if (xe.Name == "ConditionalScope")
{
    this.ProcessConditionalScope(xe);
}
}
```

You'll see later during actual implementation and usage how this all fits together and functions.

## Using the Client Object Model

The Client OM is very similar to the Server Object Model most SharePoint developers should be familiar with. As the table below shows the class names are essentially the same, except for ClientContext, just without the SP prefix. There are some differences and some classes specific to the Client OM, which I'll highlight as we go. All of the objects you will work with are derived from ClientObject which contains properties used for creating the ObjectPath used to form a query as described in the previous section.

# SharePoint 2010 Client Object Model

Mark Nischalke

Server OM	Client OM
SPContext	ClientContext
SPSite	Site
SPWeb	Web
SPList	List
SPListItem	ListItem

A very simple example of using the Client OM in Managed code is as follows

```
using(ClientContext ctx = new ClientContext("http://mysite"))
{
    ctx.Load(ctx.Site);
    ctx.ExecuteQuery();
}
or with JavaScript
var ctx = SP.ClientContext.get_current();
var site = ctx.get_site();
ctx.load(site);
ctx.executeQueryAsync(Function.createDelegate(this, OnSuccess),
    Function.createDelegate(this, OnFail));
```

Both examples are almost the same with the prominent exception being the asynchronous method for JavaScript. The request is, of course, out of band in the browser so requires asynchronous processing. This is the same for Silverlight. The one thing to note in either of these examples is that the Web Service is not actually called until the Execute method is called. As I'll show in the examples this is very useful when forming complex queries and optimizing the amount of data transmitted rather than making several calls to the Web Services.

Now I'll take a look at the individual components of this process. I'll use a Managed Code Console application for these examples but in Part 2 of this article I'll demonstrate usage with JavaScript and Silverlight.

## Creating a ClientContext

Just as with the Server OM, when using the Client OM the first thing that must be done is to create a context to operate within, a ClientContext in this case.

```
using(ClientContext ctx = new ClientContext("http://mysite"))
{
}
}
```

The input to the constructor must be a full URL. The ClientObject is derived from ClientRunTimeContext which implements IDisposable so it should be properly disposed after use. This isn't as critical as when using the Server OM but still important for proper usage. The ClientContext object contains properties for the Site and Web for the context being used, along with an overridden ExecuteQuery method, which I'll

# SharePoint 2010 Client Object Model

Mark Nischalke

cover shortly. The ClientRunTimeContext contains methods such as Load and LoadQuery, which, again, I'll cover shortly. It should be noted that the default authentication method for the Client OM is Windows Authentication. If the site you are working with uses Forms Based Authentication there are some additional steps that must be taken. I'll get to that also.

Once you have a ClientContext it can be used to get references to the Site or to a Web within the site. As mentioned above Site and Web are properties of the ClientContext object. Just as with the Server OM you can also get Web objects by using the RootWeb property or OpenWeb method of the Site. One method not available in the Server OM is the OpenWebById method which takes the Guid ID for the Web that will be opened.

```
public Web OpenWebById(Guid gWebId)
{
    return new Web(base.Context, new ObjectPathMethod(base.Context, base.Path,
"OpenWebById", new object[] { gWebId }));
}
```

After the ClientContext has been instantiated there are few properties available, the most important of which is PendingRequest. This property contains a reference to a ClientRequest object, which, as the name implies, is the object that will make the actual query to SharePoint and process the results. It does this via a WebRequestExecutor object exposed by the RequestExecutor property. As described above all requests are processed via the Client.svc Web Service. You can see this in the WebRequest property when debugging an application.

```
ctx.PendingRequest.RequestExecutor.WebRequest.RequestUri
// http://mysite/_vti_bin/client.svc/ProcessQuery
```

## Creating a request: Load

Having a ClientContext is only the beginning of using the Client OM. As described above requests are made and processed based on a collection of ObjectPath elements. These are constructed by using the Load or LoadQuery methods.

```
public void Load<T>(T clientObject, params Expression<Func<T, object>>[] retrievals)
where T: ClientObject;
public IEnumerable<T> LoadQuery<T>(ClientObjectCollection<T> clientObjects) where T:
ClientObject;
public IEnumerable<T> LoadQuery<T>(IQueryable<T> clientObjects) where T:
ClientObject;

using(ClientContext ctx = new ClientContext(URL))
{
    ctx.Load(ctx.Site);
    ctx.ExecuteQuery();
}
```

Both of these methods (LoadQuery has two overloads) take as parameters objects that are derived from ClientObject. Remember from above this object contains the information necessary to form the

# SharePoint 2010 Client Object Model

Mark Nischalke

ObjectPath for the object that will be used in the query. One thing to keep in mind when working with the Client OM is that no matter how many Load or LoadQuery statements you have, nothing is actually done until a call to ExecuteQuery is made. This allows you to build complex queries using multiple objects but lets the Client OM code construct the ObjectPaths required without extraneous data. Taking a look at the Load method a ClientObject and params, which, of course, is optional. In the sample above the Site property is used to pass a Site object, which, as noted early, is derived from ClientObject. The retrievals parameter is used to specify what properties or objects should be in the return. I'll get to that in a moment but first I'll look inside the Load method.

```
public void Load<T>(T clientObject, params Expression<Func<T, object>>[] retrievals)
where T: ClientObject
{
    if (clientObject == null)
    {
        throw new ArgumentNullException("clientObject");
    }
    DataRetrieval.Load<T>(clientObject, retrievals);
}

internal static void Load<T>(T clientObject, params Expression<Func<T, object>>[]
retrievals) where T: ClientObject
{
    ClientObjectCollection objects = clientObject as ClientObjectCollection;
    if ((retrievals == null) || (retrievals.Length == 0))
    {
        clientObject.Query.SelectAllProperties();
        if (objects != null)
        {
            clientObject.Query.ChildItemQuery.SelectAllProperties();
        }
    }
    else
    {
        ...removed for clarity
    }
}
```

When the Load method is called the parameters are passed to the Load method of an internal object, DataRetrieval. The first thing it checks for is whether the retrievals parameter exists. If not, or it is empty, the API will assume all properties are to be retrieved. Using Fiddler you can inspect the request and results after ExecuteQuery is called and you should see something similar to this.

```
POST http://mySite/_vti_bin/client.svc/ProcessQuery HTTP/1.1
X-RequestDigest: 0x8D593CC2539B9[truncated for space]0000
Content-Type: text/xml
X-RequestForceAuthentication: true
Host: mySite
Content-Length: 554
Expect: 100-continue
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
<Request AddExpandFieldTypesuffix="true" SchemaVersion="14.0.0.0"
LibraryVersion="14.0.4762.1000"
  ApplicationName=".NET Library"
  xmlns="http://schemas.microsoft.com/sharepoint/clientquery/2009">
  <Actions>
    <ObjectPath Id="2" ObjectPathId="1" />
    <ObjectPath Id="4" ObjectPathId="3" />
    <Query Id="5" ObjectPathId="3">
      <Query SelectAllProperties="true">
        <Properties />
      </Query>
    </Query>
  </Actions>
  <ObjectPaths>
    <StaticProperty Id="1" TypeId="{3747adcd-a3c3-41b9-bfab-4a64dd2f1e0a}"
Name="Current" />
    <Property Id="3" ParentId="1" Name="Site" />
  </ObjectPaths>
</Request>
```

As you can see ObjectPath elements have been constructed from the ClientObject. Note the SelectAllProperties="true". The results are as follows:

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: application/json
Server: Microsoft-IIS/7.5
SPRequestGuid: 22e3f795-a5c9-492d-a551-3e950b8f1c10
Set-Cookie: WSS_KeepSessionAuthenticated={4e4cffb3-203e-4857-8f5a-90a4b18789a4};
path=/
X-SharePointHealthScore: 0
X-Content-Type-Options: nosniff
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
MicrosoftSharePointTeamServices: 14.0.0.6029
Content-Length: 460
```

```
[
{
"SchemaVersion":"14.0.0.0","LibraryVersion":"14.0.6106.5001","ErrorInfo":null
},2,{
"IsNull":false
},4,{
"IsNull":false
},5,{
"_ObjectType_":"SP.Site",
"Id":"\\Guid(2aa4b949-05b2-456e-8392-c694d41c95d0)\\",
"ServerRelativeUrl":"\u002f",
"Url":"http:\u002f\u002fmySite",
"UIVersionConfigurationEnabled":false,
```

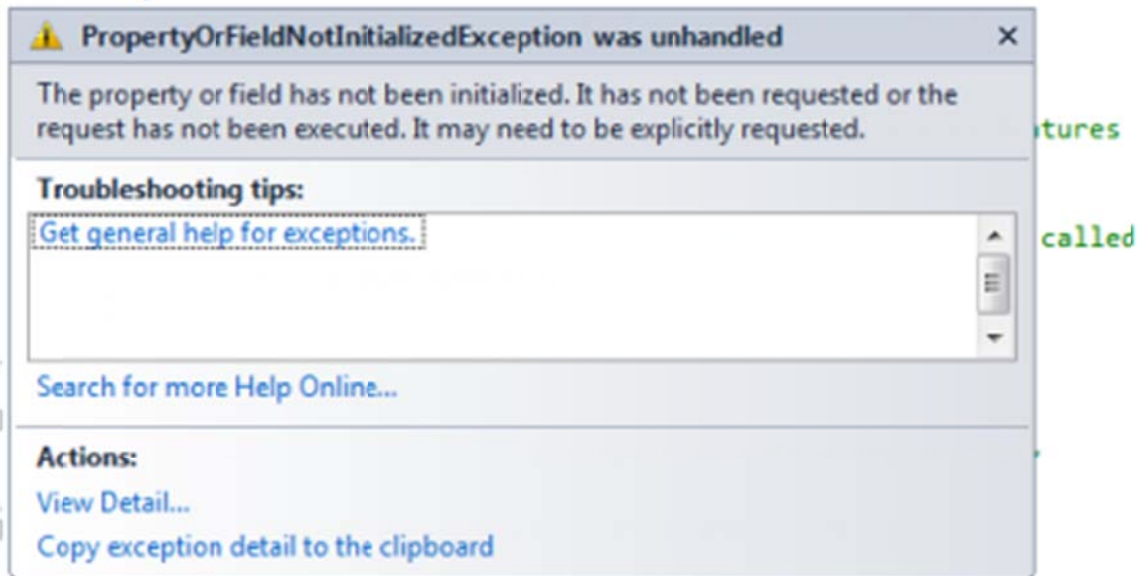
# SharePoint 2010 Client Object Model

Mark Nischalke

```
"MaxItemsPerThrottledOperation":5000,  
"AllowDesigner":true,  
"AllowRevertFromTemplate":false,  
"AllowMasterPageEditing":false,  
"ShowUrlStructure":false  
}  
]
```

Because the Site object doesn't have many properties the results are relatively small. You can see that a JSON object is returned with the values for the properties filled in. If you were to try accessing a property prior to calling ExecuteQuery, as below, a PropertyOrFieldNotInitializedException would be thrown.

```
using(ClientContext ctx = new ClientContext(URL))  
{  
    ctx.Load(ctx.Site);  
  
    // Will throw PropertyOrFieldNotInitializedException  
    Console.WriteLine("Site.URL: {0}", ctx.Site.Url);  
  
    ctx.ExecuteQuery();  
}
```

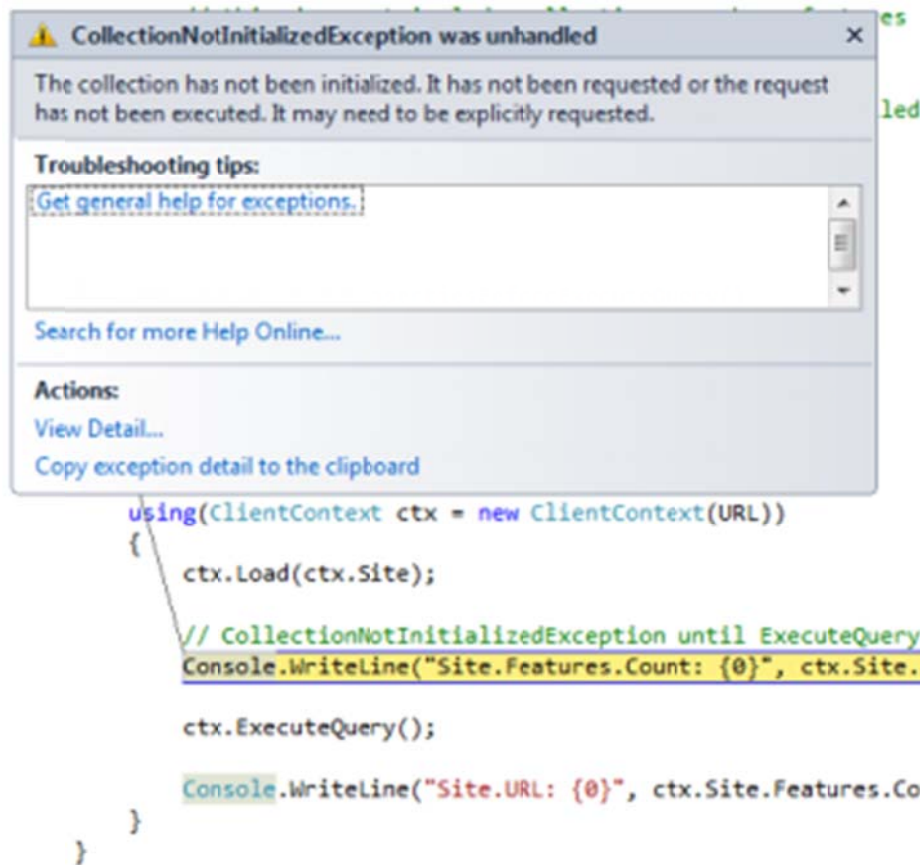


```
using(ClientContext ctx = new ClientContext(URL))  
{  
    ctx.Load(ctx.Site);  
    Console.WriteLine("Site.URL: {0}", ctx.Site.Url);  
    ctx.ExecuteQuery();  
}
```

# SharePoint 2010 Client Object Model

Mark Nischalke

When attempting to access a collection, such as, Site.Features, the exception will be CollectionNotInitializedException.



Note, however, in this example the exception will still be thrown since the only object that was loaded was Site, which does not include all of the collections within it. To include the Features collection the Load method would need to be as such

```
ctx.Load(ctx.Site.Features);
```

However, when attempting to access a property on the Site object a PropertyOrFieldNotInitializedException would be thrown because the Load method has not included it, only the Features collection. As a comparison the request and response for the above code would look like this

```
POST http://mySite/_vti_bin/client.svc/ProcessQuery HTTP/1.1
X-RequestDigest: 0xF1C00A2A5C[truncated for space]0000
Content-Type: text/xml
X-RequestForceAuthentication: true
Host: mySite
Content-Length: 714
Expect: 100-continue
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
&gl;Request AddExpandoFieldTypeSuffix="true" SchemaVersion="14.0.0.0"
LibraryVersion="14.0.4762.1000"
  ApplicationName=".NET Library"
xmlns="http://schemas.microsoft.com/sharepoint/clientquery/2009">
  &gl;Actions>
    &gl;ObjectPath Id="2" ObjectPathId="1" />
    &gl;ObjectPath Id="4" ObjectPathId="3" />
    &gl;ObjectPath Id="6" ObjectPathId="5" />
    &gl;Query Id="7" ObjectPathId="5">
      &gl;Query SelectAllProperties="true">
        &gl;Properties />
      &gl;/Query>
    &gl;ChildItemQuery SelectAllProperties="true">
      &gl;Properties />
    &gl;/ChildItemQuery>
  &gl;/Actions>
  &gl;ObjectPaths>
    &gl;StaticProperty Id="1" TypeId="{3747adcd-a3c3-41b9-bfab-4a64dd2f1e0a}"
Name="Current" />
    &gl;Property Id="3" ParentId="1" Name="Site" />
    &gl;Property Id="5" ParentId="3" Name="Features" />
  &gl;/ObjectPaths>
&gl;/Request>
```

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: application/json
Server: Microsoft-IIS/7.5
SPRequestGuid: ad372075-501e-4974-a466-2bd1004e8110
Set-Cookie: WSS_KeepSessionAuthenticated={4e4cffb3-203e-4857-8f5a-90a4b18789a4};
path=/
X-SharePointHealthScore: 0
X-Content-Type-Options: nosniff
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
MicrosoftSharePointTeamServices: 14.0.0.6029
Content-Length: 2863
```

```
[
{
"SchemaVersion":"14.0.0.0","LibraryVersion":"14.0.6106.5001","ErrorInfo":null
},2,{
"IsNull":false
},4,{
"IsNull":false
},6,{
"IsNull":false
},7,{
"_ObjectType_":"SP.FeatureCollection","_Child_Items_":[
{
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
"_ObjectType_": "SP.Feature",
"_ObjectIdentity_": "740c6a0b-85e2-48a0-a494-e0f1759d4aa7:site:2aa[truncated for
space]5d0:feature:2acf[truncated for space]18b",
"DefinitionId": "\/Guid(2acf27a5-f703-4277-9f5d-24d70110b18b)\/"
}, {
"_ObjectType_": "SP.Feature",
"_ObjectIdentity_": "740c6a0b-85e2-48a0-a494-e0f1759d4aa7:site:2aa[truncated for
space]5d0:feature:695b[truncated for space]162",
"DefinitionId": "\/Guid(695b6570-a48b-4a8e-8ea5-26ea7fc1d162)\/"
}, {
"_ObjectType_": "SP.Feature",
"_ObjectIdentity_": "740c6a0b-85e2-48a0-a494-e0f1759d4aa7:site:2aa[truncated for
space]5d0:feature:c85[truncated for space]fb5",
"DefinitionId": "\/Guid(c85e5759-f323-4efb-b548-443d2216efb5)\/"
... Removed for brevity ...
}
]
}
```

As you can see there are additional elements added to the Query and ObjectPaths elements and the returned JSON object doesn't include the properties for the Site as the previous results did. If you wanted both the properties and the collections, two Load methods would be needed.

```
ctx.Load(ctx.Site);
ctx.Load(ctx.Site.Features);
```

This, however, has a large drawback, the size of the request and responses are increased.

Method	Request Size (bytes)	Response Size (bytes)
Properties	554	460
Collections	719	2865
Properties and Collections	824	3201

The Site object is relatively small, with few properties and collections, but you can see how this could easily get out of hand. To get some control over this bloating you'll need to use the retrievals parameter to limit the results.

## Limiting the result set

As shown above the signature of the Load method takes a params collection of Expressions, which are derived from LambdaExpression.

```
public sealed class Expression<TDelegate> : LambdaExpression
```

I'll switch to the Web object for a larger set of examples and show a simple usage below.

```
Web web = ctx.Site.RootWeb;
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
ctx.Load(web, w => w.Title);
```

The first parameter is still the ClientObject derived object that will be used to form the ObjectPaths, but the second is a normal Lamda expression that you may be accustomed to when using Linq. Comparing the above to this

```
Web web = ctx.Site.RootWeb;  
ctx.Load(web);
```

you can see that although the request size increases slightly the results are reduced quite dramatically.

Method	Request Size (bytes)	Response Size (bytes)
Full	508	736
Title Only	698	297

Because retrievals is a params collection you can string expressions together, such as

```
ctx.Load(web,  
    w => w.Title,  
    w => w.Description);
```

or

```
ctx.Load(web,  
    w => w.Title,  
    w => w.Description,  
    w => w.Lists);
```

With the second example you must keep in mind that only the properties of the Lists will be loaded, not the collections, such as Fields. If you want those in the results you would use an Include statement.

```
Web web = ctx.Site.RootWeb;  
ctx.Load(web, w => w.Title,  
    w => w.Lists  
    .Include(l => l.Fields));
```

The Include method also takes Lamda expressions so you could include both properties and collections

```
Web web = ctx.Site.RootWeb;  
ctx.Load(web, w => w.Title,  
    w => w.Lists  
    .Include(l => l.Title,  
    l => l.Fields));
```

# SharePoint 2010 Client Object Model

Mark Nischalke

and since Fields is also a collection an even deeper Include could be added

```
Web web = ctx.Site.RootWeb;  
ctx.Load(web, w => w.Title,  
         w => w.Lists  
         .Include(l => l.Title,  
                 l => l.Fields  
                 .Include(f => f.InternalName)));
```

As with all of the uses of the Load method you must be cautious. The below table shows how the size of the results can grow based on the depth of the collections, and also how judicious use of the Lambda expressions can dramatically reduce it. You will rarely need all of the properties of an object or its collections so it should be planned out what is needed and form the Load based on those needs.

Method	Request Size (bytes)	Response Size (bytes)
One Include	984	1,946,393
Two Includes	1057	1,946,671
Three Includes	1193	192,660

Even if you use two Load methods it will still be more efficient than one large, deeply nested Include

```
Web web = ctx.Site.RootWeb;  
ctx.Load(web, w => w.Title,  
         w => w.Lists  
         .Include(l => l.Title));  
  
ctx.Load(web, w => w.Title,  
         w => w.Lists  
         .Include(l => l.Fields  
                 .Include(f => f.InternalName,  
                         f => f.Group)));
```

Method	Request Size (bytes)	Response Size (bytes)
Two Loads	1201	211,489

## Creating a request: LoadQuery

The Load methods shown above are used to "fill in" the object passed to it, for instance the Web object. This is fine when loading properties of the object or its collections but it is not good for access ListItems. The only thing you can get using Load is the ItemCount. To get the ListItems you'll need to use the LoadQuery method.

# SharePoint 2010 Client Object Model

Mark Nischalke

```
public IEnumerable<T> LoadQuery<T>(ClientObjectCollection<T> clientObjects) where T:
ClientObject;
```

```
public IEnumerable<T> LoadQuery<T>(IQueryable<T> clientObjects) where T:
ClientObject;
```

As you can see from the two signatures of this method it can take either a ClientObjectCollection, such as List, or an IQueryable object, both of which of course must be derived from ClientObject so the appropriate ObjectPaths can be constructed. For the latter override the IQueryable object can be passed using either method syntax or Linq syntax, which I'll explore shortly.

Using the first override the lines below would produce the same results. The difference being that Load will "fill in" the web.Lists collection but LoadQuery will return a collection of Lists and not use the web.Lists parameter.

```
ctx.Load(web.Lists);
```

```
var lists = ctx.LoadQuery(web.Lists);
```

Another difference between these methods is the construction of the request and return. The request for the Load is shown below. For brevity I won't show the result but it is 15,657 bytes.

```
POST http://mySite/_vti_bin/client.svc/ProcessQuery HTTP/1.1
X-RequestDigest: 0x798DBB28C[truncated for space]0000
Content-Type: text/xml
X-RequestForceAuthentication: true
Host: mySite
Content-Length: 796
Expect: 100-continue
```

```
<Request AddExpandoFieldTypeSuffix="true" SchemaVersion="14.0.0.0"
LibraryVersion="14.0.4762.1000"
  ApplicationName=".NET Library"
  xmlns="http://schemas.microsoft.com/sharepoint/clientquery/2009">
  <Actions>
    <ObjectPath Id="2" ObjectPathId="1" />
    <ObjectPath Id="4" ObjectPathId="3" />
    <ObjectPath Id="6" ObjectPathId="5" />
    <ObjectPath Id="8" ObjectPathId="7" />
    <Query Id="9" ObjectPathId="7">
      <Query SelectAllProperties="true">
        <Properties />
      </Query>
      <ChildItemQuery SelectAllProperties="true">
        <Properties />
      </ChildItemQuery>
    </Query>
  </Actions>
  <ObjectPaths>
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
<StaticProperty Id="1" TypeId="{3747adcd-a3c3-41b9-bfab-4a64dd2f1e0a}"
Name="Current" />
  <Property Id="3" ParentId="1" Name="Site" />
  <Property Id="5" ParentId="3" Name="RootWeb" />
  <Property Id="7" ParentId="5" Name="Lists" />
</ObjectPaths>
</Request>
```

The LoadQuery method produces the below request and a result that is 15,574 bytes

```
POST http://mySite/_vti_bin/client.svc/ProcessQuery HTTP/1.1
X-RequestDigest: 0x798DBB28C8[truncated for space]0000
Content-Type: text/xml
X-RequestForceAuthentication: true
Host: mySite
Content-Length: 646
Expect: 100-continue
```

```
<Request AddExpandoFieldTypeSuffix="true" SchemaVersion="14.0.0.0"
LibraryVersion="14.0.4762.1000"
  ApplicationName=".NET Library"
  xmlns="http://schemas.microsoft.com/sharepoint/clientquery/2009">
  <Actions>
    <Query Id="21" ObjectPathId="7">
      <Query SelectAllProperties="false">
        <Properties />
      </Query>
      <ChildItemQuery SelectAllProperties="true">
        <Properties />
      </ChildItemQuery>
    </Query>
  </Actions>
  <ObjectPaths>
    <Property Id="7" ParentId="5" Name="Lists" />
    <Property Id="5" ParentId="3" Name="RootWeb" />
    <Property Id="3" ParentId="1" Name="Site" />
    <StaticProperty Id="1" TypeId="{3747adcd-a3c3-41b9-bfab-4a64dd2f1e0a}"
Name="Current" />
  </ObjectPaths>
</Request>
```

Although the requests are formed differently the results, aside from the size, are the same; a collection of Lists with all properties available. The size differences in this example are very modest but could become more significant with larger, more complex queries. Just keep it mind when choosing the method to use. Just as with the Load method it may be advantageous to limit the results being returned. This is where the second overload for LoadQuery is used. The below example uses method syntax to form a Lambda expression to only include the Title property in the results. Again, as the table below shows, the sizes of the request and response are significantly different when filtering is applied.

```
Web web = ctx.Site.RootWeb;
```

Revised October 13, 2011

Page 16 of 42

# SharePoint 2010 Client Object Model

Mark Nischalke

```
var lists = ctx.LoadQuery(web.Lists.Include(l => l.Title));
```

Method	Request Size (bytes)	Response Size (bytes)
Full List	797	15,657
Include Title	705	2340

As I mentioned above query syntax can also be used with the LoadQuery method.

```
Web web = ctx.Site.RootWeb;  
var query = from l in web.Lists  
            select l;  
var lists = ctx.LoadQuery(query);
```

Here I am creating an IQueryable object from the Linq expression that will be used to return an IEnumerable<List> containing the Lists for the Web. The previous examples could be replicated as this

```
Web web = ctx.Site.RootWeb;  
var query = from l in web.Lists  
            .Include(l => l.Title)  
            select l;  
var lists = ctx.LoadQuery(query);
```

## Working with ListItems

Although getting the titles for the Lists within a Web is useful, most of the time the Client OM will be used to work with ListItems; reading, inserting, updating and deleting. With all the advances SharePoint 2010 has provided, at this point you still must work with CAML via the CamlQuery class.

```
CamlQuery query = new CamlQuery();  
ListItemCollection items = list.GetItems(query);  
ctx.Load(items);
```

Although no CAML has been supplied this will return all of the ListItems, with all field values. To get only specific fields you would use a standard CAML query, such as

```
CamlQuery query = new CamlQuery();  
ListItemCollection items = list.GetItems(query);  
ctx.Load(items);  
query.ViewXml = @"<View Scope='RecursiveAll'>  
  <ViewFields>  
    <FieldRef Name='Title' />  
    <FieldRef Name='FirstName' />  
  </ViewFields>  
</View>";
```

# SharePoint 2010 Client Object Model

Mark Nischalke

If you have a List with many columns, writing the CAML can be very tedious and error prone. Luckily, CamlQuery has a static method to help with this, CreateAllItemsQuery. This method will return the following CAML

```
<View Scope="RecursiveAll">
  <Query>
  </Query>
</View>
```

There is also a static method for all folders, CreateAllFoldersQuery

```
<View Scope="RecursiveAll">
  <Query>
    <Where>
      <Eq>
        <FieldRef Name="FSObjType" />
        <Value Type="Integer">1</Value>
      </Eq>
    </Where>
  </Query>
</View>
```

As I showed above you can use standard CAML syntax to form the query, including Where and RowLimit elements. You can also use an overload of the CreateAllItemsQuery

```
public static CamlQuery CreateAllItemsQuery(int rowLimit, params string[]
viewFields);
```

which would be used like this

```
CamlQuery query = CamlQuery.CreateAllItemsQuery(10, "Title", "FirstName");
```

## ExecuteQuery

Although ExecuteQuery has been used in the previous examples for completeness I will briefly review how it works.

```
public virtual void ExecuteQuery()
{
  ScriptTypeMap.EnsureInited();
  ClientRequest pendingRequest = this.PendingRequest;
  this.m_request = null;
  pendingRequest.ExecuteQuery();
}
```

The EnsureInited method uses a lock to ensure the initialization of the object is only performed once. The lock is necessary because the Init method uses reflection to check custom attributes and create an

# SharePoint 2010 Client Object Model

Mark Nischalke

instance of classes implementing the `IScriptTypeFactory` interface. After initialization the `ClientRequest.ExecuteQuery` method will call the `BuildQuery` method.

```
private ChunkStringBuilder BuildQuery()
{
    SerializationContext serializationContext = this.SerializationContext;
    ChunkStringBuilder builder = new ChunkStringBuilder();
    XmlWriterSettings settings = new XmlWriterSettings {
        OmitXmlDeclaration = true,
        NewLineHandling = NewLineHandling.Entitize
    };

    XmlWriter writer =
    XmlWriter.Create(builder.CreateTextWriter(CultureInfo.InvariantCulture), settings);
    writer.WriteStartElement("Request",
"http://schemas.microsoft.com/sharepoint/clientquery/2009");
    writer.WriteAttributeString("AddExpandoFieldTypeSuffix", "true");
    writer.WriteAttributeString("SchemaVersion",
ClientSchemaVersions.CurrentVersion.ToString());
    writer.WriteAttributeString("LibraryVersion", "14.0.4762.1000");

    if (!string.IsNullOrEmpty(this.m_context.ApplicationName))
    {
        writer.WriteAttributeString("ApplicationName",
this.m_context.ApplicationName);
    }
    writer.WriteStartElement("Actions");

    ...removed for clarity...

    writer.WriteEndElement();
    writer.WriteStartElement("ObjectPaths");

    Dictionary<long, ObjectPath> dictionary = new Dictionary<long, ObjectPath>();
    while (true)
    {
        List<long> list = new List<long>();
        foreach (long num in serializationContext.Paths.Keys)
        {
            if (!dictionary.ContainsKey(num))
            {
                list.Add(num);
            }
        }
        if (list.Count == 0)
        {
            break;
        }
        for (int i = 0; i < list.Count; i++)
        {
            ObjectPath path = this.m_context.ObjectPaths[list[i]];

```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
        path.WriteToXml(writer, serializationContext);
        dictionary[list[i]] = path;
    }
}
writer.WriteEndElement();
writer.WriteEndElement();
writer.Flush();
return builder;
}
```

As you can this is where all the ObjectPaths created from the objects and properties used in any Load or LoadQuery methods prior to the ExecuteQuery method being called are constructed into an XML string that will be sent to the server along with setting the properties such as LibraryVersion and ApplicationName. After the query has been built the only thing left to do is send it to the server using the ExecuteQueryToServer method below.

```
private void ExecuteQueryToServer(ChunkStringBuilder sb)
{
    this.m_context.FireExecutingWebRequestEvent(new
WebRequestEventArgs(this.RequestExecutor));
    this.RequestExecutor.RequestContentType = "text/xml";
    if (this.m_context.AuthenticationMode == ClientAuthenticationMode.Default)
    {
        this.RequestExecutor.RequestHeaders["X-RequestForceAuthentication"] = "true";
    }
    Stream requestStream = this.RequestExecutor.GetRequestStream();
    sb.WriteContentAsUTF8(requestStream);
    requestStream.Flush();
    requestStream.Close();
    this.RequestExecutor.Execute();
    this.ProcessResponse();
}
```

This method will send the query, then, using the ProcessResponse method, translate the results into a JSON object.

## Authentication

As I mentioned earlier the Client OM uses Windows Authentication by default when utilizing Managed Code. With JavaScript or Silverlight the code is being executed within the context of an application already so it will use the existing security token. You can see this by using a tool such as Fiddler to monitor the web traffic being generated. When using Managed Code there will be calls to `_vti_bin/sites.asmx` requesting the FormsDigest information, which contains the security token.

```
POST http://mySite/_vti_bin/sites.asmx HTTP/1.1
Content-Type: text/xml
SOAPAction:
http://schemas.microsoft.com/sharepoint/soap/GetUpdatedFormDigestInformation
X-RequestForceAuthentication: true
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
Authorization: NTLM
TlRMTVNTUUAABAAAAt7II4gkACQAsAAAAABAAEACgAAAAGAbEdAAAAD01BUktOSVNDSEFMSOU=
Host: mySite
Content-Length: 0
```

When working with a site using Claims based authentication, such as Forms Authentication, an extra step is needed prior to calling ExecuteQuery to set the type and credentials to use. Everything else in your code remains the same.

```
ctx.AuthenticationMode = ClientAuthenticationMode.FormsAuthentication;
ctx.FormsAuthenticationLoginInfo = new FormsAuthenticationLoginInfo("loginName",
"password");
```

The FormsAuthenticationLoginInfo class is used by the ClientContext.FormsAuthenticationLogin method by first calling the EnsureLogin method to check for authentication cookies.

```
internal CookieCollection EnsureLogin(Uri contextUri)
{
    if ((this.m_authCookies == null) || !this.m_cookieValid)
    {
        this.m_authCookies = new Dictionary<Uri, CookieInfo>();
        this.m_cookieValid = true;
    }
    contextUri = new Uri(contextUri, "/");
    CookieInfo info = null;
    if (!this.m_authCookies.TryGetValue(contextUri, out info) || (info.Expires <=
DateTime.UtcNow))
    {
        info = this.Login(contextUri);
        this.m_authCookies[contextUri] = info;
    }
    return info.Cookies;
}
```

If the cookie is not present, or has expired, a call to Login which will call the authentication service to complete the Login.

```
Uri authenticationServiceUrl;
if (this.AuthenticationServiceUrl == null)
{
    authenticationServiceUrl = new Uri(contextUri, "/_vti_bin/authentication.asmx");
}
else
{
    authenticationServiceUrl = this.AuthenticationServiceUrl;
}
Authentication authentication = new Authentication(authenticationServiceUrl) {
    CookieContainer = this.CookieContainer
};
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
LoginResult result = authentication.Login(this.LoginName, this.Password);
```

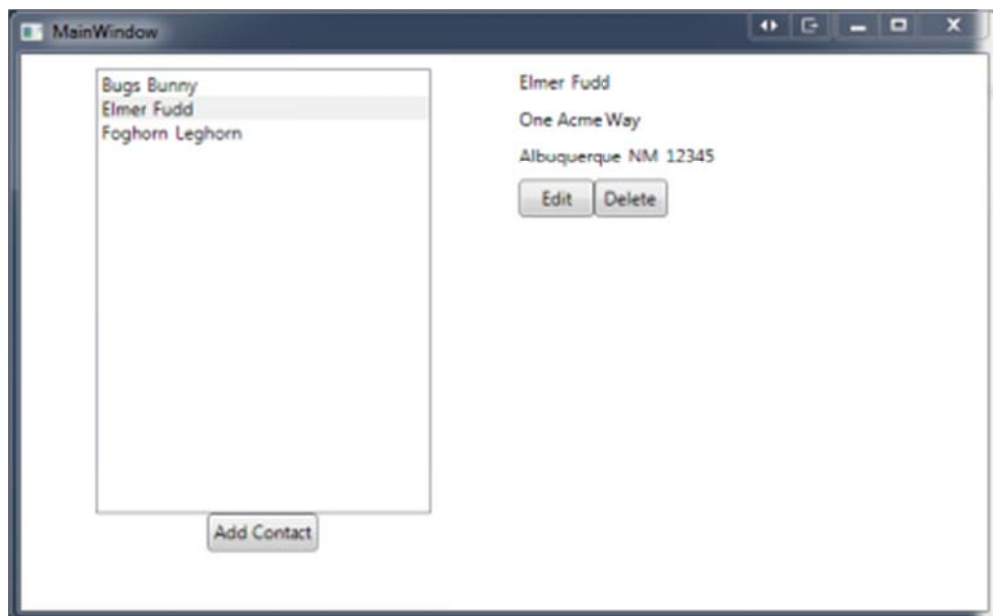
## SharePoint 2010 Client Object Model

In Part 1 of this article I took an in-depth look at the SharePoint Client Object Model; how it works and how to form queries and work with objects. Now I'll take a practical look at using the Client OM with examples using .NET Managed code, using WPF, JavaScript used in a Application page, and finally with Silverlight in a SharePoint webpart.

## Setup

To facilitate the examples the first thing was to create a SharePoint project that creates a simple Contact list and populates it with a few Contacts to begin with. This project is included in the downloads for this article. For the Silverlight example I created a Silverlight project and added the output to a Document Library on the site.

The demo apps may not be the best designed, but they are functional and effectively demonstrate using the Client Object Model in a somewhat practical manner.



This application first gets a ListItemCollection containing all the Contacts in the demo list.

```
public static ListItemCollection GetList()
{
    using(ClientContext ctx = new ClientContext(SITE))
    {
        Web web = ctx.Web;
        List list = web.Lists.GetByTitle(LIST_NAME);
        CamlQuery query = new CamlQuery();
        query.ViewXml = "<View>" +
            "<Query>" +
            "<OrderBy>" +
            "<FieldRef Name='Title' />" +
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
        "<FieldRef Name='FirstName'/>" +
        "</OrderBy>" +
    "</Query>" +
    "<ViewFields>" +
        "<FieldRef Name='ID'/>" +
        "<FieldRef Name='Title'/>" +
        "<FieldRef Name='FirstName'/>" +
        "<FieldRef Name='WorkAddress'/>" +
        "<FieldRef Name='WorkCity'/>" +
        "<FieldRef Name='WorkState'/>" +
        "<FieldRef Name='WorkZip'/>" +
    "</ViewFields>" +
    "</View>";

    ListItemCollection listItems = list.GetItems(query);
    ctx.Load(listItems);
    ctx.ExecuteQuery();

    return listItems;
}
}
```

Since this article is more about usage of the Client Object Model I won't delve into the XAML too much. Just to show, however, the ListBox is bound to the ListItemCollection returned from the above code and the TextBlocks in the DataTemplate are bound to the indexer property of the ListItem for the specified FieldValue

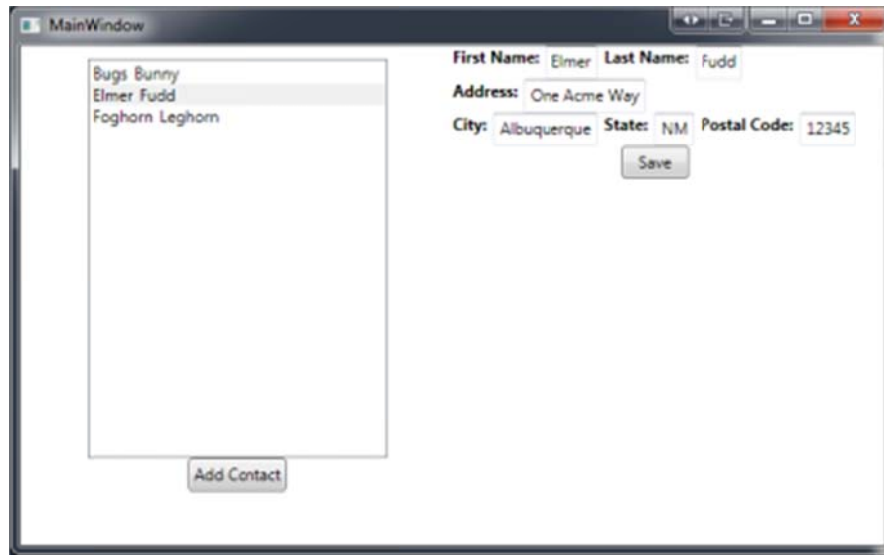
```
<ListBox x:Name="Contacts" ItemsSource="{Binding}" Width="225" Height="300"
SelectionChanged="Contacts_SelectionChanged">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Path=[FirstName]}" Margin="0,0,5,0"/>
                <TextBlock Text="{Binding Path=[Title]}" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Part 1 of this article covers the details for retrieving ListItems, ClientContext, CamlQuery and other objects and methods being used so I won't go into them here.

# SharePoint 2010 Client Object Model

Mark Nischalke

## Updating ListItems



Clicking on the Edit button in the application causes the edit fields to be displayed, once again I'm not focusing on the WPF, you can explore the download files for more details on that aspect. After clicking the Save button it's simply a matter of finding the ListItem to be updated, update the FieldValues, then commit the changes.

```
public static void Update(int id, string firstName, string lastName, string address,
    string city, string state, string postalCode)
{
    using(ClientContext ctx = new ClientContext(SITE))
    {
        Web web = ctx.Web;
        List list = web.Lists.GetByTitle(LIST_NAME);

        // Get the item being updated
        ListItem item = list.GetItemById(id);

        // Update the FieldValues
        item["Title"] = lastName;
        item["FirstName"] = firstName;
        item["WorkAddress"] = address;
        item["WorkCity"] = city;
        item["WorkState"] = state;
        item["WorkZip"] = postalCode;

        // Must make sure to call this
        item.Update();

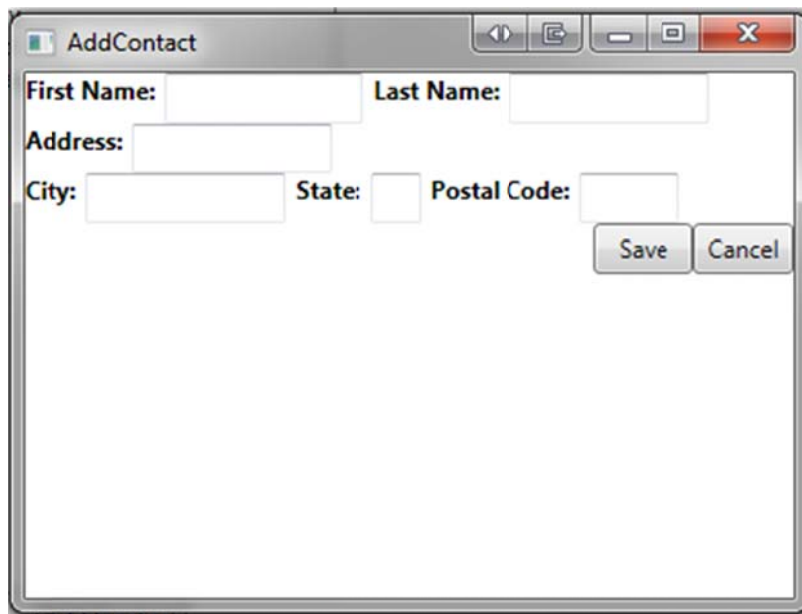
        // Commit the change
        ctx.ExecuteQuery();
    }
}
```

# SharePoint 2010 Client Object Model

Mark Nischalke

One thing to note here the lack of any calls to Load or LoadQuery as in the examples in Part 1. Since nothing is being returned in this case there is no need to use these methods. Think of it as the equivalent of the ADO.NET ExecuteNonQuery method. Another thing to note is although it may seem tempting to cache the ListItem and update it directly, without the GetItemById call, this won't work. The item must be retrieved within the context of the ClientContext object, similar to how the Entity Framework functions for those familiar with it. It is not recommended to cache the ClientContext since it does hold resources and may cause degradation of your application. Best to follow the database connection model; open late, use and close.

## Adding ListItems

A screenshot of a Windows-style dialog box titled "AddContact". The dialog contains several text input fields: "First Name:" and "Last Name:" are on the top line; "Address:" is on the second line; "City:", "State:", and "Postal Code:" are on the third line. At the bottom right of the dialog are two buttons labeled "Save" and "Cancel". The dialog has a standard Windows window border with minimize, maximize, and close buttons in the top right corner.

Adding a new ListItem to the List is almost as easy as updating. One of the main differences though is the ListItemCreationInformation object.

```
public static void AddContact(string firstName, string lastName, string address,
    string city, string state, string postalCode)
{
    using(ClientContext ctx = new ClientContext(SITE))
    {
        Web web = ctx.Web;
        List list = web.Lists.GetByTitle(LIST_NAME);

        // Create the Listitem
        // ListItemCreationInformation can be null for root folder
        ListItemCreationInformation createInfo = null;

        // Or for adding item to a folder
        //ListItemCreationInformation createInfo = new ListItemCreationInformation();
        //createInfo.FolderUrl = "site/lists/listname/folder";
    }
}
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
ListItem item = list.AddItem(createInfo);

// Set the FieldValues
item["Title"] = lastName;
item["FirstName"] = firstName;
item["WorkAddress"] = address;
item["WorkCity"] = city;
item["WorkState"] = state;
item["WorkZip"] = postalCode;

// Save changes
item.Update();

// Commit
ctx.ExecuteQuery();
}
}
```

Just as with the methods demonstrated in Part 1 when the ExecuteQuery method is invoked the Client OM API will create an XML string and POST it to the client.svc Web Service. As you can see below this is very similar to the "get" methods but you'll notice the Method element specifying which method is to be called and the Parameter elements passing the type and value.

```
POST http://mySite/_vti_bin/client.svc/ProcessQuery HTTP/1.1
X-RequestDigest: 0xE8312251E4B[truncated for space]0000
Content-Type: text/xml
X-RequestForceAuthentication: true
Host: mySite
Content-Length: 2461
Expect: 100-continue
```

```
<Request AddExpandoFieldTypeSuffix="true" SchemaVersion="14.0.0.0"
LibraryVersion="14.0.4762.1000"
  ApplicationName=".NET Library"
xmlns="http://schemas.microsoft.com/sharepoint/clientquery/2009">
  <Actions>
    <ObjectPath Id="25" ObjectPathId="24" />
    <ObjectPath Id="27" ObjectPathId="26" />
    <ObjectPath Id="29" ObjectPathId="28" />
    <ObjectPath Id="31" ObjectPathId="30" />
    <ObjectIdentityQuery Id="32" ObjectPathId="30" />
    <ObjectPath Id="34" ObjectPathId="33" />
    <Method Name="SetFieldValue" Id="35" ObjectPathId="33">
      <Parameters>
        <Parameter Type="String">Title</Parameter>
        <Parameter Type="String">Betic</Parameter>
      </Parameters>
    </Method>
    <Method Name="SetFieldValue" Id="36" ObjectPathId="33">
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
<Parameters>
  <Parameter Type="String">FirstName</Parameter>
  <Parameter Type="String">Alpha</Parameter>
</Parameters>
</Method>
<Method Name="SetFieldValue" Id="37" ObjectPathId="33">
  <Parameters>
    <Parameter Type="String">WorkAddress</Parameter>
    <Parameter Type="String">Main street </Parameter>
  </Parameters>
</Method>
<Method Name="SetFieldValue" Id="38" ObjectPathId="33">
  <Parameters>
    <Parameter Type="String">WorkCity</Parameter>
    <Parameter Type="String">Anytown</Parameter>
  </Parameters>
</Method>
<Method Name="SetFieldValue" Id="39" ObjectPathId="33">
  <Parameters>
    <Parameter Type="String">WorkState</Parameter>
    <Parameter Type="String">PA</Parameter>
  </Parameters>
</Method>
<Method Name="SetFieldValue" Id="40" ObjectPathId="33">
  <Parameters>
    <Parameter Type="String">WorkZip</Parameter>
    <Parameter Type="String">12345</Parameter>
  </Parameters>
</Method>
<Method Name="Update" Id="41" ObjectPathId="33" />
<Query Id="42" ObjectPathId="33">
  <Query SelectAllProperties="false">
    <Properties>
      <Property Name="Title" ScalarProperty="true" />
      <Property Name="FirstName" ScalarProperty="true" />
      <Property Name="WorkAddress" ScalarProperty="true" />
      <Property Name="WorkCity" ScalarProperty="true" />
      <Property Name="WorkState" ScalarProperty="true" />
      <Property Name="WorkZip" ScalarProperty="true" />
    </Properties>
  </Query>
</Query>
</Actions>
<ObjectPaths>
  <StaticProperty Id="24" TypeId="{3747adcd-a3c3-41b9-bfab-4a64dd2f1e0a}"
Name="Current" />
  <Property Id="26" ParentId="24" Name="Web" />
  <Property Id="28" ParentId="26" Name="Lists" />
  <Method Id="30" ParentId="28" Name="GetByTitle">
    <Parameters>
      <Parameter Type="String">SPUG Demo Contacts</Parameter>
    </Parameters>
  </Method>
</ObjectPaths>
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
</Parameters>
</Method>
<Method Id="33" ParentId="30" Name="GetItemById">
  <Parameters>
    <Parameter Type="Int32">4</Parameter>
  </Parameters>
</Method>
</ObjectPaths>
</Request>
```

Although nothing is returned, such as with Load or LoadQuery, there is a response from the Client.svc Web Service call. As you see below the response simply contains the object that was updated, however not the `_ObjectVersion_` property which incremented with the version of the object returned.

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: application/json
Server: Microsoft-IIS/7.5
SPRequestGuid: 472b1d44-cc1e-4295-a7b0-c38cf3b68a62
Set-Cookie: WSS_KeepSessionAuthenticated={4e4cffb3-203e-4857-8f5a-90a4b18789a4};
path=/
X-SharePointHealthScore: 0
X-Content-Type-Options: nosniff
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
MicrosoftSharePointTeamServices: 14.0.0.6029
Content-Length: 672
```

```
[
{
  "SchemaVersion":"14.0.0.0","LibraryVersion":"14.0.6106.5001","ErrorInfo":null
},25,{
  "IsNull":false
},27,{
  "IsNull":false
},29,{
  "IsNull":false
},31,{
  "IsNull":false
},32,{
  "_ObjectIdentity_":"740c6a0b-85e2-48a0-a494-e0f1759d4aa7:web:6bb[truncated for
space]3964:list:9bf3[truncated for space]d045"
},34,{
  "IsNull":false
},42,{
  "_ObjectType_":"SP.ListItem",
  "_ObjectIdentity_":"740c6a0b-85e2-48a0-a494-e0f1759d4aa7:web:6bb[truncated for
space]964:list:9bf[truncated for space]d045:item:4,1",
  "_ObjectVersion_":"2",
  "Title":"Betic",
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
"FirstName":"Alpha",
"WorkAddress":"Main street ",
"WorkCity":"Anytown",
"WorkState":"PA",
"WorkZip":"12345"
}
]
```

## Deleting ListItems

Deleting a ListItem is the simplest of the operations. After obtaining the ListItem, call its DeleteObject method and commit the changes with an ExecuteQuery.

```
public static void Delete(int id)
{
    using(ClientContext ctx = new ClientContext(SITE))
    {
        Web web = ctx.Web;
        List list = web.Lists.GetByTitle(LIST_NAME);

        // Get the item being deleted
        ListItem item = list.GetItemById(id);
        item.DeleteObject();

        // Commit
        ctx.ExecuteQuery();
    }
}
```

```
POST http://mySite/_vti_bin/client.svc/ProcessQuery HTTP/1.1
X-RequestDigest: 0x2BD0E4[truncated for space]0000
Content-Type: text/xml
X-RequestForceAuthentication: true
Host: mySite
Content-Length: 994
Expect: 100-continue
```

```
<Request AddExpandoFieldTypeSuffix="true" SchemaVersion="14.0.0.0"
LibraryVersion="14.0.4762.1000"
    ApplicationName=".NET Library"
xmlns="http://schemas.microsoft.com/sharepoint/clientquery/2009">
  <Actions>
    <ObjectPath Id="54" ObjectPathId="53" />
    <ObjectPath Id="56" ObjectPathId="55" />
    <ObjectPath Id="58" ObjectPathId="57" />
    <ObjectPath Id="60" ObjectPathId="59" />
    <ObjectIdentityQuery Id="61" ObjectPathId="59" />
    <ObjectPath Id="63" ObjectPathId="62" />
    <Method Name="DeleteObject" Id="64" ObjectPathId="62" />
  </Actions>
</ObjectPaths>
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
<StaticProperty Id="53" TypeId="{3747adcd-a3c3-41b9-bfab-4a64dd2f1e0a}"
Name="Current" />
<Property Id="55" ParentId="53" Name="Web" />
<Property Id="57" ParentId="55" Name="Lists" />
<Method Id="59" ParentId="57" Name="GetByTitle">
  <Parameters>
    <Parameter Type="String">SPUG Demo Contacts</Parameter>
  </Parameters>
</Method>
<Method Id="62" ParentId="59" Name="GetItemById">
  <Parameters>
    <Parameter Type="Int32">6</Parameter>
  </Parameters>
</Method>
</ObjectPaths>
</Request>
```

Once again, although nothing is returned a response is sent from the Web Service. This time though there is no object since it has been deleted.

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: application/json
Server: Microsoft-IIS/7.5
SPRequestGuid: 28594025-6d96-4798-ba72-06a6b81f0c3e
Set-Cookie: WSS_KeepSessionAuthenticated={4e4cffb3-203e-4857-8f5a-90a4b18789a4};
path=/
X-SharePointHealthScore: 0
X-Content-Type-Options: nosniff
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
MicrosoftSharePointTeamServices: 14.0.0.6029
Content-Length: 343
```

```
[
{
"SchemaVersion":"14.0.0.0","LibraryVersion":"14.0.6106.5001","ErrorInfo":null
},54,{
"IsNull":false
},56,{
"IsNull":false
},58,{
"IsNull":false
},60,{
"IsNull":false
},61,{
"_ObjectIdentity_":"740c6a0b-85e2-48a0-a494-e0f1759d4aa7:web:6bb[truncated for
space]964:list:9bf[truncated for space]d045"
},63,{
"IsNull":false
}
]
```

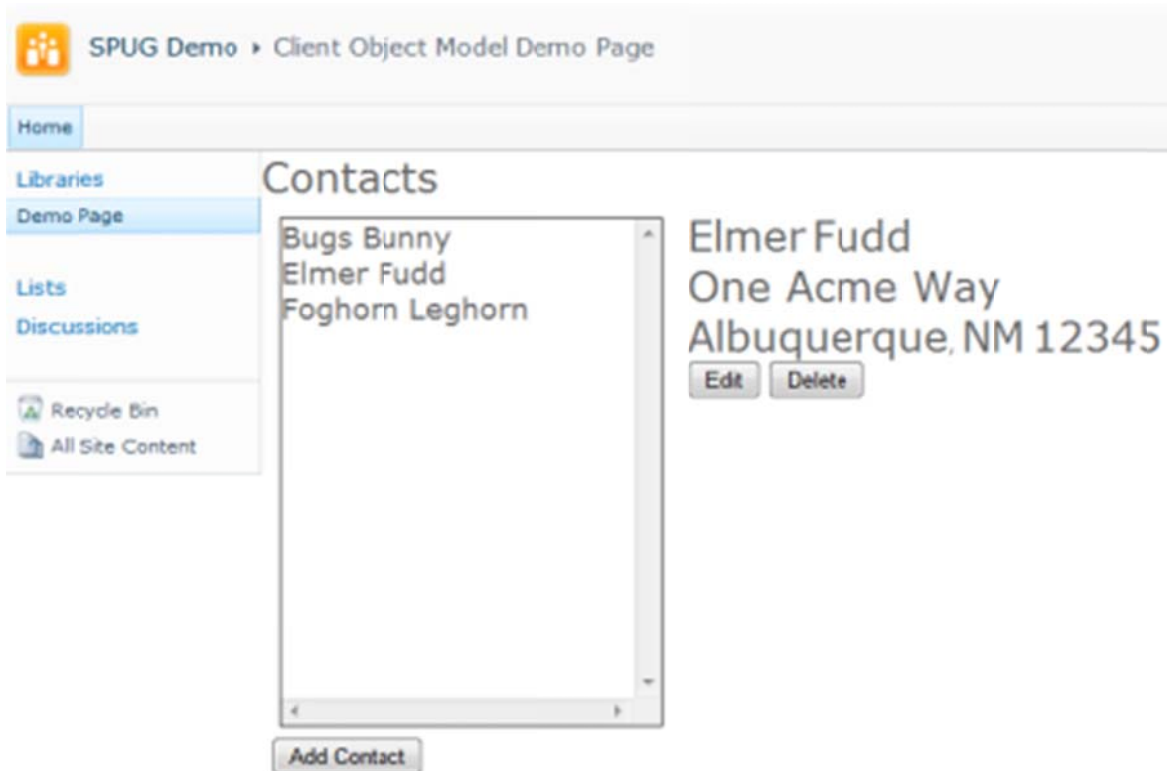
# SharePoint 2010 Client Object Model

Mark Nischalke

1

## Client OM with JavaScript

Working with the Client OM in JavaScript is quite similar to Managed Code. The objects are essentially the same, except the method names conform to JavaScript coding standards.



One major difference, however, is the Client OM implementation in JavaScript uses asynchronous methods where the Managed Code implementation uses synchronous methods. In Managed Code though this doesn't prevent you from using Asynchronous methods when implementing your code as the Silverlight example will show.

```
/// Retrieve all contacts in the list
function loadContactsList()
{
    var ctx = SP.ClientContext.get_current();
    var web = ctx.get_web();
    var list = web.get_lists().getByTitle(LIST_NAME);

    var camlQuery = new SP.CamlQuery();
    var queryXml = "<View>" +
        "<Query>" +
            "<OrderBy>" +
                "<FieldRef Name='Title' />" +
                "<FieldRef Name='FirstName' />" +
            "</OrderBy>" +
        "</Query>" +
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
// Can specify fields here
//          "<ViewFields>" +
//          "<FieldRef Name='ID'/">" +
//          "<FieldRef Name='Title'/">" +
//          "<FieldRef Name='FirstName'/">" +
//          "</ViewFields>" +
//          "</View>";

camlQuery.set_viewXml(queryXml);
this.listItems = list.getItems(camlQuery);
// Can use this or LoadQuery
//ctx.load(this.listItems);
// Fields can be included here rather than in CAML
ctx.load(this.listItems, 'Include(ID, Title, FirstName)');

ctx.executeQueryAsync(
    Function.createDelegate(this, onLoadContactsSuccess),
    Function.createDelegate(this, onFail));
}
```

As you can see in the above code querying for the ListItems is the same as in the WPF example; get a ClientContext, get the objects necessary, form a CamlQuery and call executeQueryAsync. As you'll notice, getting the ClientContext with JavaScript is slightly different. Just as with the Managed Code implementation you can construct a ClientContext object by passing a URL. However, in JavaScript this is a relative URL, rather than a full URL, since the code is executing with a SharePoint site already.

```
var ctx = SP.ClientContext("serverRelativeUrl");
```

The get\_current method is particular to the JavaScript Client OM implementation and, as you can see below, constructs the ClientContext object from the get\_webServerRelativeUrl method which has been filled in from other SharePoint JavaScript code.

```
SP.ClientContext.get_current = function() {ULS5V1;;
    if (!SP.ClientContext.$1S_1) {
        SP.ClientContext.$1S_1 = new
SP.ClientContext(SP.PageContextInfo.get_webServerRelativeUrl());
    }
    return SP.ClientContext.$1S_1;
}
```

After executeQueryAsync has been called successfully the delegate specified in the first parameter, onLoadContactsSuccess in this case, will be called.

```
/// Function to be called after executeQueryAsync
/// in loadContactsList has been successful
function onLoadContactsSuccess(sender, args)
{
    $("#contactList").empty();
}
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
if(this.listItems.get_count() == 0)
{
    $("#contactList").append("<span>No contacts found</span>");
}
else
{
    // Get Enumerator and iterate through it
    var listEnumerator = this.listItems.getEnumerator();
    while (listEnumerator.moveNext())
    {
        var item = listEnumerator.get_current();

        // Properties on FieldValues will match
        // Fields returned by query
        var title = item.get_fieldValues().Title;
        var firstName = item.get_fieldValues().FirstName;
        var id = item.get_fieldValues().ID;

        // Create the html element
        var contact = "<span onclick='displayContactDetails(" + id + ")'>" +
        firstName + " " + title + "</span>";

        // Append to "list"
        $("#contactList").append(contact + "<br/>");
    }
}
}
```

Getting an individual ListItem is again almost the same as previous examples. The load method, however, is a little different. Since JavaScript doesn't support Lambda expression you can pass an array of strings that specify the columns to be returned in the object.

```
function displayContactDetails(id)
{
    selectedItemId = id;
    var ctx = SP.ClientContext.get_current();
    var web = ctx.get_web();
    var list = web.get_lists().getByTitle(LIST_NAME);

    listItem = list.getItemById(id);

    // Returns all fields
    //ctx.load(this.listItem);

    // Returns only fields specified
    ctx.load(this.listItem, "Title", "FirstName", "WorkAddress", "WorkCity",
    "WorkState", "WorkZip");

    ctx.executeQueryAsync(
        Function.createDelegate(this, onContactDetailsSuccess),
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
Function.createDelegate(this, onFail));  
}
```

## Updating ListItem

No real surprises here. The difference with JavaScript is using the `set_item` method rather than using the indexer in Managed Code.

```
function saveContact()  
{  
    $(".field").each(function ()  
    {  
        var id = $(this).attr("ID");  
        var value = $(this).next("input").val();  
  
        listItem.set_item(id,value);  
    });  
  
    listItem.update();  
  
    var ctx = SP.ClientContext.get_current();  
  
    ctx.executeQueryAsync(  
        Function.createDelegate(this, onSaveContactSuccess),  
        Function.createDelegate(this, onFail));  
}
```

## Adding ListItems

By now you should be getting the hang of this.

```
function onAdd()  
{  
    var ctx = SP.ClientContext.get_current();  
    var web = ctx.get_web();  
    var list = web.get_lists().getByTitle(LIST_NAME);  
  
    // can be null when adding item to root folder  
    var createInfo = null;  
    //var createInfo = new SP.ListItemCreationInformation();  
    var item = list.addItem(createInfo);  
  
    $(".field").each(function ()  
    {  
        var id = $(this).attr("ID");  
        var value = $(this).val();  
  
        item.set_item(id, value);  
    });  
  
    item.update();  
}
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
ctx.executeQueryAsync(  
    Function.createDelegate(this, onAddContactSuccess),  
    Function.createDelegate(this, onFail));  
}
```

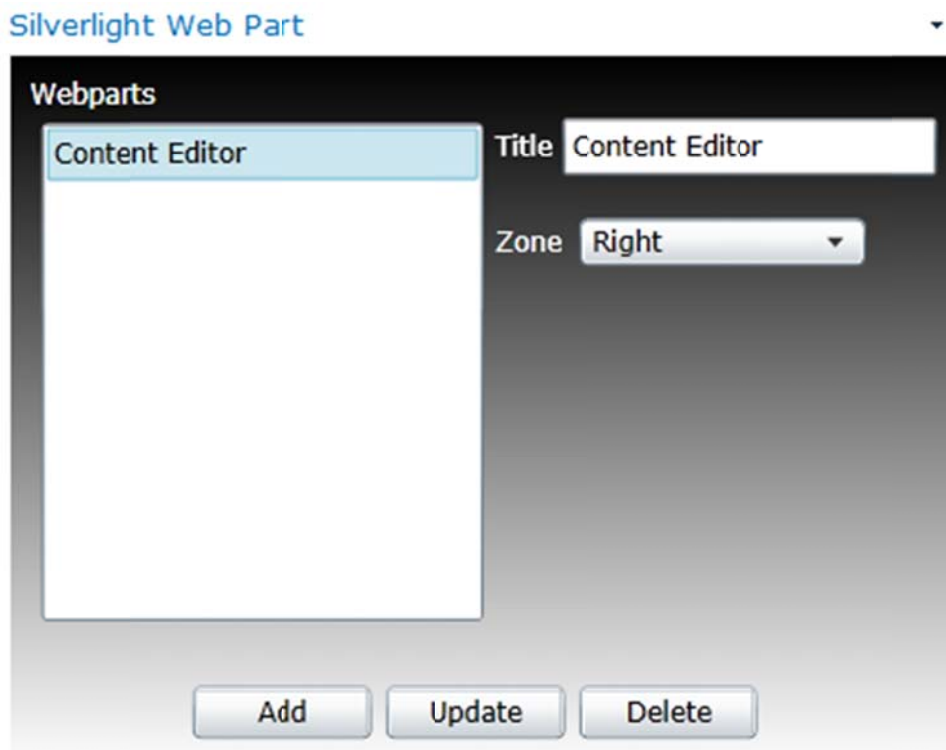
## Deleting ListItem

Just to complete the example

```
function onDelete()  
{  
    var ctx = SP.ClientContext.get_current();  
    var web = ctx.get_web();  
    var list = web.get_lists().getByTitle(LIST_NAME);  
  
    var listItem = list.getItemById(selectedItemId);  
    listItem.deleteObject();  
  
    ctx.executeQueryAsync(  
        Function.createDelegate(this, onDeleteSuccess),  
        Function.createDelegate(this, onFail));  
}
```

## Client OM with Silverlight

By now you are probably bored working with ListItems using the Client OM. Since there isn't much difference with the previous methods I'll use Silverlight to focus on other things that can be done with the Client OM; in Managed code, JavaScript or Silverlight. Specifically, with Silverlight I'll demonstrate working with WebParts on a page.



# SharePoint 2010 Client Object Model

Mark Nischalke

Upon loading, this Silverlight WebPart will read the WebParts on the designated page, default.aspx in this example, and display the titles in the ListBox.

```
private void LoadWebParts()
{
    SelectedItem = null;
    wpList.Items.Clear();
    using(ClientContext ctx = new ClientContext(SITE))
    {
        try
        {
            // Get the default page for the site
            File file = ctx.Web.GetFileByServerRelativeUrl(PAGE);

            // Get the WebPart manager to locate all the WebParts
            LimitedWebPartManager wpm =
file.GetLimitedWebPartManager(PersonalizationScope.Shared);

            // Load all WebPart definitions found on the page
            WPDefinitions = ctx.LoadQuery(wpm.WebParts.Include(w => w.Id, w =>
w.WebPart));

            ctx.ExecuteQueryAsync(OnLoadSucceeded, OnFail);
        }
        catch(System.Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}
```

As you can see, like all of the other examples, you first get a ClientContext to work with, using a server relative URL since the page is operating within a SharePoint site already. Next is to get the File to work with by using the well named method GetFileByServerRelativeUrl. After this is getting the LimitedWebPartManager to obtain a collection of WebPartDefinitions from via the LoadQuery method. As with other examples I'm using a Lambda expression to limit the results to only what is necessary, the ID and WebPart properties in this case. At this point using the Client OM with Silverlight will diverge from the other environments. Since this code is being run from the main thread of the application you'll need to use the asynchronous method, ExecuteQueryAsync to make the query to prevent blocking the UI thread. This method takes delegates to ClientRequestSucceededEventHandler and ClientRequestFailedEventHandler that will be called when the method succeeds or fails.

```
private void OnLoadSucceeded(object sender, ClientRequestSucceededEventArgs e)
{
    // Create delegate to update ListBox in UI thread
    Action updateList = () =>
    {
        if(WPDefinitions.Count() != 0)

```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
{
    foreach(WebPartDefinition def in WPDefinitions)
    {
        wpList.Items.Add(new ListItemHelper() { ID = def.Id, Title =
def.WebPart.Title });
    }
};

this.Dispatcher.BeginInvoke(updateList);
}
```

In this method I create an anonymous delegate to pass to the BeginInvoke method that will iterate through the WebPartDefinition collection populated by the LoadQuery call.

## Adding WebPart

Using the Client OM to add WebParts to a page is relatively straight forward.

```
private void OnUpdate(object sender, System.Windows.RoutedEventArgs e)
{
    // Get the title and zone that may have been updated
    SelectedItem.Title = wpTitle.Text;
    SelectedItem.Zone = ((ComboBoxItem)wpZone.SelectedItem).Content.ToString();

    // Start a worker thread to do the processing
    ThreadPool.QueueUserWorkItem(new WaitCallback(CSOMHelper.UpdateTitle),
SelectedItem);

    // Update the controls
    wpList.Items[SelectedItem.ItemIndex] = wpTitle.Text;
    wpTitle.Text = "";
}
```

In this example I'll use a slightly different method to call the Client OM methods. Once again since this code is being executed in the main UI thread you can't use any blocking methods. In this case I use a worker thread to do the processing since there isn't anything that needs to be updated on the UI from the Client OM code.

```
public static void UpdateTitle(Object state)
{
    ListItemHelper item = (ListItemHelper)state;

    using(ClientContext ctx = new ClientContext(SITE))
    {
        // Get the default page for the site
        File file = ctx.Web.GetFileByServerRelativeUrl(PAGE);

        // Get the WebPart manager to locate all the WebParts
    }
}
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
LimitedWebPartManager wpm =
file.GetLimitedWebPartManager(PersonalizationScope.Shared);

// Load all WebPart definitions found on the page
IEnumerable<WebPartDefinition> definitions =
ctx.LoadQuery(wpm.WebParts.Include(w => w.Id, w => w.WebPart));

// Use the synchronous method here since the
// method is being called in separate thread
ctx.ExecuteQuery();

// Find the definition for the webpart to be updated
WebPartDefinition def = definitions.FirstOrDefault(d => d.Id == item.ID);
if(def != null)
{
    def.WebPart.Title = item.Title;
    def.MoveWebPartTo(item.Zone.ToLower(), 0);
    // Save the changes
    def.SaveWebPartChanges();
    // Commit
    ctx.ExecuteQuery();
}
}
```

This method is just like the load method, get a ClientContext and collection of WebPartDefinitions. After finding the WebPart being updated you set the Title property and call the handy and well named method MoveWebPartTo to move the WebPart to a new WebPartZone. Again, the well named method SaveWebPartChanges saves the updates that have been made and the changes are committed by the call to ExecuteQuery. Since this is being executed within the worker thread you can use the synchronous method. This shows that both synchronous and asynchronous methods are available in the Silverlight implementation of the Client OM, unlike Managed Code where only synchronous methods are available and JavaScript where only asynchronous methods are available.

## Deleting WebParts

Deleting WebParts is just as easy. Follow the same steps as above and call the DeleteWebPart method.

```
// Find the definition for the webpart to be deleted
WebPartDefinition def = definitions.FirstOrDefault(d => d.Id == item.ID);
if(def != null)
{
    def.DeleteWebPart();
    // Save the changes
    def.SaveWebPartChanges();
    // Commit
    ctx.ExecuteQuery();
}
```

# SharePoint 2010 Client Object Model

Mark Nischalke

## UserAction Menus

For the final example I'll demonstrate working with the UserActions menu to add items to the EditControlBlock menu of a list and the Site Actions menu.

```
public static void AddUserActions()
{
    using(ClientContext ctx = new ClientContext(URL))
    {
        Web web = ctx.Web;
        List list = web.Lists.GetByTitle(LIST_NAME);
        UserCustomActionCollection actions = list.UserCustomActions;

        UserCustomAction action = actions.Add();
        action.Location = "EditControlBlock";
        action.Sequence = 100;
        action.Title = "SPUG Custom Action";
        action.Url = URL + @"/default.aspx";
        action.Update();

        ctx.Load(list, l => l.UserCustomActions);

        ctx.ExecuteQuery();
    }
}
```

Adding a menu item to the Site Actions menu is same as the above example, only the Location and Group properties need to be changed.

```
UserCustomAction action = actions.Add();
action.Location = "Microsoft.SharePoint.StandardMenu";
action.Group = "SiteActions";
action.Sequence = 101;
action.Title = "SPUG Custom Action";
action.Url = URL + @"/default.aspx";
action.Update();
```

Removing the menu item just a matter of iterating through the UserCustomActions, finding the correct one and calling the DeleteObject. This code will remove menu items from both the EditControlBlock and Site Actions menus since it is only comparing the title. Obviously, you'll want to check the Location property to be more specific.

```
public static void RemoveUserActions()
{
    using(ClientContext ctx = new ClientContext(URL))
    {
        Web web = ctx.Web;
        List list = web.Lists.GetByTitle(LIST_NAME);
        UserCustomActionCollection actions = list.UserCustomActions;
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
ctx.Load(list, l => l.UserCustomActions.Include(a => a.Title));

ctx.ExecuteQuery();

for(int x = 0; x < actions.Count; x++)
{
    if(actions[x].Title == "SPUG Custom Action")
    {
        actions[x].DeleteObject();
        ctx.ExecuteQuery();
        break;
    }
}
}
```

## Creating a List

So far all of the examples have used previously created lists. Now I'll show how to create one and add Fields to it.

Before creating a List you should, of course, check if it already exists. One somewhat awkward part of this is the Client OM will throw a `ServerException` when the List can't be found rather than just returning a null object.

```
List list = web.Lists.GetByTitle("CSOM List");
try
{
    ctx.ExecuteQuery();
    if(list != null)
    {
        list.DeleteObject();
        ctx.ExecuteQuery();
    }
}
catch
{
    // ServerException is thrown if list does not exist
}
```

After the check has been made and the List deleted if necessary, you can create the List using the `ListCreationInformation`. Just as with the `ListItemCreationInformation` shown earlier, this objects applies information about the List that is to be created.

```
// Set info about this list
ListCreationInformation createInfo = new ListCreationInformation();
createInfo.Title = "CSOM List";
createInfo.QuickLaunchOption = QuickLaunchOptions.On;
createInfo.TemplateType = (int)ListTemplateType.GenericList;
```

# SharePoint 2010 Client Object Model

Mark Nischalke

```
// Add list to the web
list = web.Lists.Add(createInfo);
list.EnableFolderCreation = true;
// Make sure to call Update to apply settings
list.Update();
```

After the List has been created you add the Fields. The XML supplied to the AddFieldAsXml method is the same as you would use in an Elements.xml to define Fields being added to your by a SharePoint project.

```
// Add fields to the list
Field field = list.Fields.AddFieldAsXml("<Field Type='Text'
DisplayName='SomeField'></Field>",
    true, AddFieldOptions.DefaultValue);

field.DefaultValue = "Hello, World";
// Don't forget to Update after setting values
field.Update();

field = list.Fields.AddFieldAsXml(@"<Field Type='Choice' DisplayName='IsThisFun'
Format='RadioButtons'>
                                <Default>Yes<</Default>
                                <CHOICES>
                                    <CHOICE>Yes</CHOICE>
                                    <CHOICE>No</CHOICE>
                                <</CHOICES>
                                </Field>",
    true, AddFieldOptions.DefaultValue);
```

Now that the List and Fields have been created you add ListItems as in previous examples.

```
ListItemCreationInformation itemCreateInfo = new ListItemCreationInformation();
ListItem item = list.AddItem(itemCreateInfo);
item["Title"] = "One";
item.Update();
```

## Adding ListItem with Folder

In the previous examples all ListItems have been added to the root folder of the list. However, there are cases when the ListItem should be placed in folder. In this example I've left out checking of the folder exists since the List is being created here also but you should be able to understand from the previous examples how to make that check.

```
// Create the folder first
itemCreateInfo = new ListItemCreationInformation();
itemCreateInfo.LeafName = "New Folder";
itemCreateInfo.UnderlyingObjectType = FileSystemObjectType.Folder;

item = list.AddItem(itemCreateInfo);
item.Update();
```

# SharePoint 2010 Client Object Model

Mark Nischalke

Here I set the UnderlyingObjectType property to FileSystemObjectType.Folder and the LeafName property to the name of the folder. Afterward, I use the FolderUrl property to the relative address of the folder that was created in this list before adding the ListItem

```
// Now add item to folder
itemCreateInfo = new ListItemCreationInformation();
itemCreateInfo.FolderUrl = "/Lists/CSOM List/New Folder";

item = list.AddItem(itemCreateInfo);
item["Title"] = "Three";
item.Update();
```

## Conclusion

The SharePoint 2010 Client Object Model is a great improvement over previous editions of SharePoint and opens SharePoint to wide array of applications and possibilities. Just like with SharePoint itself the Client OM is vast with many more features then I have time to cover here. Hopefully both of these articles have given you a good background on what the SharePoint Client Object Model is, how it works and a hint of the possibilities it presents.