Using OData in a Windows Store App to access online databases (JavaScript)

<ALREADY POSTED>

If you don’t know anything about the Open Data Protocol—known as OData—it’s really quite simple. OData is a standard for providing a REST interface to a database. On the service side, OData is a layer that you create in top of a database (SQL Server, Azure, and MySQL) to provide those REST entry points using .NET, Java, PHP, and node.js. On the client side, you can then make HTTP requests to those endpoints directly, or use one of many OData client libraries.

Those libraries, along with more information about OData, can be found on <http://www.odata.org>. here’s the definition of the technology from that page:

The **Open Data Protocol (OData)** is a Web protocol for querying and updating data that provides a way to unlock your data and free it from silos that exist in applications today. OData does this by applying and building upon Web technologies such as [HTTP](http://www.w3.org/Protocols/), [Atom Publishing Protocol](http://www.ietf.org/rfc/rfc4287.txt) (AtomPub) and [JSON](http://json.org/) to provide access to information from a variety of applications, services, and stores. The protocol emerged from experiences implementing AtomPub clients and servers in a variety of products over the past several years. OData is being used to expose and access information from a variety of sources including, but not limited to, relational databases, file systems, content management systems and traditional web sites.

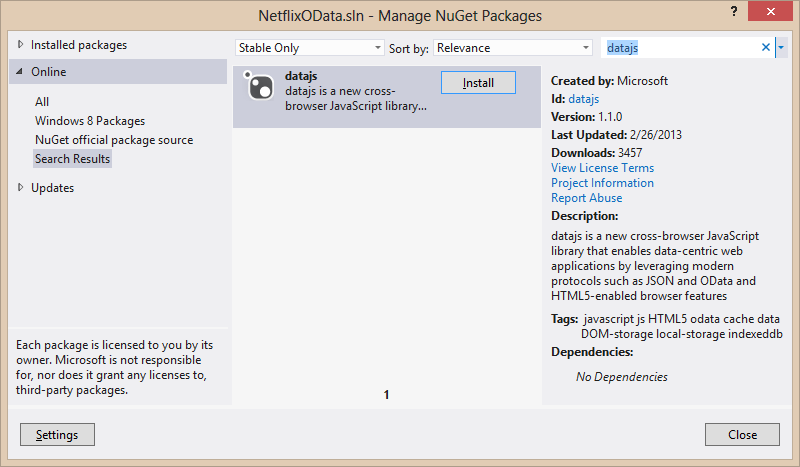
If you go to the [OData Ecosystem page](http://www.odata.org/ecosystem), you’ll see listings of consumers (apps and libraries for both clients and servers), serious applications, significant services that support OData (such as Netflix, eBay, and StackOverflow), and some other bits.

<END>

For the purposes of this post, we’ll be using the [JavaScript client library](http://www.odata.org/libraries#JavaScript) which [can be installed via NuGet in Visual Studio](http://nuget.org/List/Packages/datajs). We’ll use this first to retrieve data from the Netflix catalog (a read-only source), then use it to manage data (with updates and deletions) from a database of our own.

Note that the documentation for the JavaScript client library—which I’ll refer to as datajs—is found on <http://datajs.codeplex.com/documentation>.

Let’s start with a Grid App project in Visual Studio Express 2012, so we can see how to pull in the library. Having created a new project from the Grid App template, calling it NetflixOData, I right click on the solution, choose Manage NuGet Packages…, and in that dialog search for **odatajs**, which is supplied by Microsoft.



After following the install procedure, I have datajs-1.1.0.js in the Scripts folder (along with a minified version, but we can just use the readable one). To pull it into the project, let’s go ahead and add this line to default.js:

<script href="/scripts/datajs-1.1.0.js"></script>

Now we’re ready to do some queries and get some data.

Let’s do this as part of an extended splash screen, so we can load up some data while showing a progress indicator.

<TODO: not going to do this part yet>

For loading the data, we’ll do that initially in data.js, which comes with the Grid App template. The template’s file comes with a function *generateSampleData* (using an array), which we’ll delete and replace with *acquireInitialData* using Netflix as a source.

With the OData JavaScript library, which I’ll just refer to as datajs from now on for convenience, data retrieval is as simple as calling [*OData.read*](http://datajs.codeplex.com/wikipage?title=datajs%20OData%20API&referringTitle=Documentation#OData.read) against the appropriate URI. In this case, we check the [Netflix OData API reference](http://developer.netflix.com/docs/oData_Catalog) and see that we should query this URI:

*http://odata.netflix.com/v2/Catalog/Titles*

Because we don’t want to retrieve the *entire* catalog, let’s limit this query to the first 200 titles for the moment:

*http://odata.netflix.com/v2/Catalog/Titles?$top=200*

The OData.read function, for its part, takes this URI as the first argument, or, alternately, an object that describes the HTTP request if you need to specify other headers and such. For most queries, query parameters on the URI suffice.

The second argument to *OData.read* is a function that will receive the results when they’re ready. So here’s a starter version of *acquireInitialData* that just outputs the first result to the console:

function acquireInitialData() {

//Make a query to Netflix for the first 200 titles.

OData.read("http://odata.netflix.com/v2/Catalog/Titles?top=200",

function (data, request) {

console.log(JSON.stringify(data.results[0]));

});

}

What you’ll get from this is a fairly long string with a bunch of information in it. You can inspect that string if you want, but it’s probably easier just to look at it in the debugger.

The important part, though, is that each item from the query in *data.results* is an object with many properties to which you can then data-bind. So let’s just populate our *WinJS.Binding.List* with those items:

function acquireInitialData() {

//Make a query to Netflix for the first 200 titles.

OData.read("http://odata.netflix.com/v2/Catalog/Titles?top=200",

function (data, request) {

for (var i = 0; i < data.results.length; i++) {

list.push(data.results[i]);

}

});

}

The documentation for what you get back from Netflix is found on <TODO>. For our purposes, we want to show each title’s name and image in our list. The name is found in the *Name* property on the item (or *ShortName*). Cover art is found in *BoxArt.SmallUrl* (65x89), *BoxArt.MediumUrl* (88x120), and *BoxArt.LargeUrl* (110x151). So we’ll change the item template in groupedItems.html (the home page in the Grid App template) to reflect this:

<div class="itemtemplate" data-win-control="WinJS.Binding.Template">

<div class="item">

<img class="item-image" src="#"  
 data-win-bind="src: BoxArt.LargeUrl; alt: ShortName" />

<div class="item-overlay">

<h4 class="item-title" data-win-bind="textContent: ShortName"></h4>

<h6 class="item-subtitle win-type-ellipsis"  
 data-win-bind="textContent: Name"></h6>

</div>

</div>

</div>

We also need to change the grouping function in data.js to group by the first letter of the name (being sure to use [*Windows.Globalization.Collation.CharacterGroupings*](http://msdn.microsoft.com/en-us/library/windows/apps/windows.globalization.collation.charactergroupings.aspx) instead of making language assumptions):

var cg = Windows.Globalization.Collation.CharacterGroupings();

function getGroupKey(dataItem) {

return cg.lookup(dataItem.Name); //Name is a property as defined by the data source

}

function getGroupData(dataItem) {

return { groupTitle: getGroupKey(dataItem.ReleaseYear) };

}

var groupedItems = list.createGrouped(getGroupKey, getGroupSelector);

With a few other tweaks to the CSS, we now get a home page that looks like this <TODO: fix groupings>