Migrating from Silverlight to Wijmo 5

BERNARDO de CASTILHO

WHITE PAPER

Contents

Introduction

What attracted many developers to Silverlight starting back in 2007, was that it allowed us to write maintainable web applications quickly and easily. This was largely due to MVVM, a pattern that separates logic from presentation. The **ViewModel** portion of the app, written in C# or VB, contains all of the logic, which makes it easy to test and maintain. The **View** portion of the app is written in XAML, using a rich library of supporting classes and controls.

When Microsoft decided to stop investing in Silverlight, many developers were faced with a tough decision. How could they migrate to modern web platforms and not throw away all the benefits of MVVM and the knowledge they gained by using Silverlight?

At ComponentOne, a division of GrapeCity, we faced this challenge ourselves. We studied the alternatives and concluded that HTML5 and application frameworks such as AngularJS seemed like the best option going forward. AngularJS provides the key ingredients: a solid framework for single-page applications, a templating mechanism, and basic support for MVVM. All it was missing were the controls and a few essential classes required to create traditional MVVM apps.

Well, we know how to write controls. So we decided to add what was missing.

The result is Wijmo 5, our latest control library for HTML5 and JavaScript development. Wijmo 5 is a true HTML5 solution, based on JavaScript and CSS. It includes all of the controls commonly used in LOB (line of business) applications, as well as JavaScript implementations of the **ICollectionView** interface and a concrete **CollectionView** class.

Together with AngularJS, Wijmo 5 provides a great platform for porting existing Silverlight applications and also for creating brand new web applications faster than you ever thought possible.

The Silverlight Application

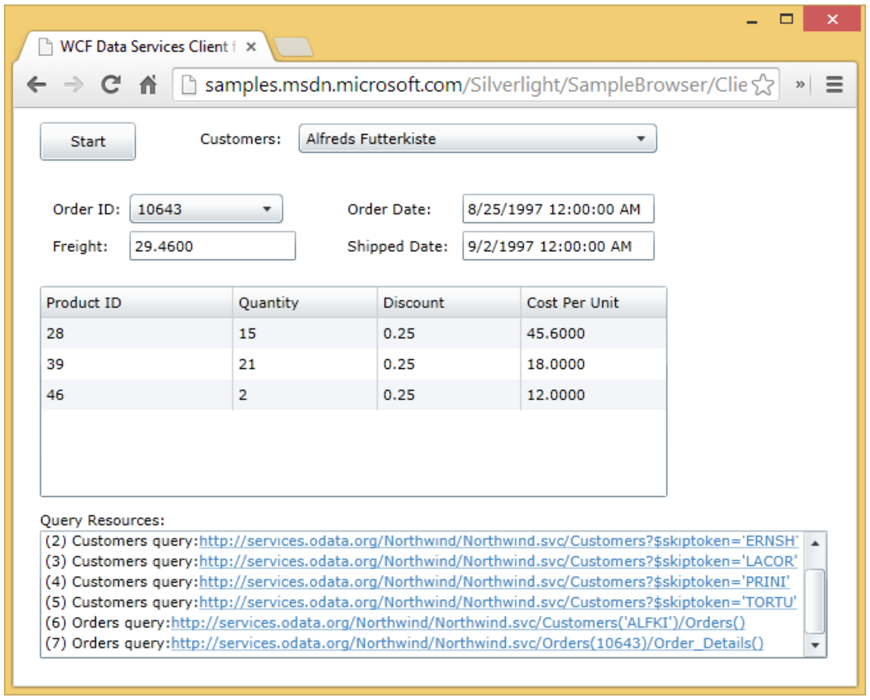
To demonstrate this, we decided to port a Silverlight application to HTML5. We chose to use an existing application, written by Microsoft before Wijmo 5 existed. The application is called "DataServicesQuickStart." It is a simple application, but it illustrates many of the key features of Silverlight and MVVM apps, including:

- Loading data into CollectionView objects that can sort, filter, group, and paginate data, as well as keep track of the currently selected item. The **ICollectionView** interface represents lists in MVVM applications.
- Managing hierarchical data to reflect the current selection. The application loads a list of customers. When the user selects a customer, the app displays the orders placed by that customer. When the user selects an order, the app displays the order details. This type of master-detail relationship is another common feature of MVVM apps.
- Binding data to controls. **ComboBox** controls provide customers and orders. **TextBox** controls provide order information, and a **DataGrid** shows order details. Most LOB apps, including MVVM, use input controls and data grids.

The "DataServicesQuickStart" application is available on-line. You can see it here:
http://samples.msdn.microsoft.com/Silverlight/SampleBrowser/ClientBin/DataServicesQuickStart.html

And you can find the source code here:
http://samples.msdn.microsoft.com/Silverlight/SampleBrowser/#/?sref=DataServicesQuickStart

This is what the application looks like in case you don't want to run it right now:


WCF Data Services Client f
samples.msdn.microsoft.com/Silverlight/SampleBrowser/Clie
Start

Customers: Alfreds Futterkiste

Order ID: 10643
Order Date: 8/25/1997 12:00:00 AM
Freight: 29.4600
Shipped Date: 9/2/1997 12:00:00 AM

Product ID	Quantity	Discount	Cost Per Unit
28	15	0.25	45.6000
39	21	0.25	18.0000
46	2	0.25	12.0000

Query Resources:
(2) Customers query:http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='ERNSH'
(3) Customers query:http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='LACOR'
(4) Customers query:http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='PRINI'
(5) Customers query:http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='TORTU'
(6) Orders query:http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/Orders()
(7) Orders query:http://services.odata.org/Northwind/Northwind.svc/Orders(10643)/Order_Details()

When you first run the app, it shows some empty controls. Click the **Start** button and it loads the data from an online OData source containing the traditional Northwind database. At this point, you can select a customer from the combo box on the top-right of the screen and see the data for each order. Select a different order from the combo box below the Start button to see the order details.

The application is very simple, but it does illustrate the main features necessary for many LOB apps.

Porting the ViewModel

To port the application to HTML5, we started with the ViewModel. In this example, the MainPage.xaml.cs file contains the ViewModel. The interesting part of the code looks like this (error-handling code is omitted for brevity):

```
public partial class MainPage : UserControl
{
  // Create the context and root binding collection of Customers.
  NorthwindEntities context;
  DataServiceCollection<Customer> customerBindingCollection;

  // Define the URI of the public sample Northwind data service.
  Uri serviceUri = new Uri("http://services.odata.org/Northwind/Northwind.svc/");

  void startButton_Click(object sender, RoutedEventArgs e)
  {
    // Instantiate the context based on the data service URI.
    context = new NorthwindEntities(serviceUri);

    // Instantiate the binding collection.
    customerBindingCollection = new DataServiceCollection<Customer>();

    // Register a handler for the LoadCompleted event.
    customerBindingCollection.LoadCompleted +=
                customerBindingCollection_LoadCompleted;

    // Define a LINQ query that returns all customers.
    var query = context.Customers;

    // Execute the query.
    customerBindingCollection.LoadAsync(query);
  }
  void customerBindingCollection_LoadCompleted(object sender,
                                               LoadCompletedEventArgs e)
  {
    // Get the binding collection that executed the query.
    var binding = (DataServiceCollection<Customer>)sender;

    // Consume a data feed that contains paged results.
    if (binding.Continuation != null)
    {
      // If there is a continuation token, load the next page of results.
      binding.LoadNextPartialSetAsync();
    }
    else
    {
        // Bind the collection to the view source.
        CollectionViewSource customersViewSource =
          (CollectionViewSource)this.Resources["customersViewSource"];
        customersViewSource.Source = binding;
    }
  }
}
```

The code declares a **customerBindingCollection** object that contains the list of customers. When the start button is clicked, the code loads the data asynchronously, in batches. When the last batch of data loads, it is exposed through the **customersViewSource** resource.

This populates the combo box with customers. The application uses the **SelectionChanged** event of the combo box to load the orders for the selected customer:

```
void customersComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
  // Get the selected Customer.
  Customer selectedCustomer = ((ComboBox)sender).SelectedItem as Customer;

  if (selectedCustomer != null)
  {
    // Load the orders for the selected customer
    selectedCustomer.Orders.LoadCompleted += Orders_LoadCompleted;

    if (selectedCustomer.Orders.Count == 0)
    {
      // Load the related Orders for the selected Customer,
      // if they are not already loaded.
      selectedCustomer.Orders.LoadAsync();
    }
}
}
```

The code is similar to the previous block. If the selected customer's orders have not yet been loaded, it calls the **LoadAsync** method to load them.

The pattern repeats to load the order details.

Porting this ViewModel to JavaScript would be difficult without a CollectionView class to hold the data and keep track of selection. Here is the JavaScript version of the code:

```
// URL to Northwind service
var svcUrl = 'http://services.odata.org/Northwind/Northwind.svc';

// create customers, orders, details view
$scope.customers = new wijmo.collections.CollectionView();
$scope.orders = new wijmo.collections.CollectionView();
$scope.details = new wijmo.collections.CollectionView();

// when the current customer changes, get his orders
$scope.customers.currentChanged.addHandler(function () {
  $scope.orders.sourceCollection = [];
  $scope.details.sourceCollection = [];
  var customer = $scope.customers.currentItem;
  if (customer) {
    loadData(svcUrl, $scope.orders,
             'Customers(\'' + customer.CustomerID + '\')/Orders', {
      OrderDate: wijmo.DataType.Date,
      RequiredDate: wijmo.DataType.Date,
      ShippedDate: wijmo.DataType.Date,
      Freight: wijmo.DataType.Number
    });
  }
});

// when the current order changes, get the order details
$scope.orders.currentChanged.addHandler(function () {
  $scope.details.sourceCollection = [];
  var order = $scope.orders.currentItem;
  if (order) {
    loadData(svcUrl, $scope.details,
             'Orders(' + order.OrderID + ')/Order_Details', {
      UnitPrice: wijmo.DataType.Number
    });
  }
});
```

The **$scope** object is a typical feature of AngularJS applications. It defines the part of the ViewModel that is accessible to the View. In this case, the code adds three CollectionView objects to the scope: customers, orders, and order details.

Rather than handling the **SelectionChanged** event of the combo box (which it could do), the code attaches handlers to the **CurrentChanged** event. This way, any time the current customer changes, whether it was selected from a combo box or by any other means, the code loads the orders for that customer. When the current order changes, the code loads the details for the selected order.

Very simple. Very MVVM.

As before, the process starts when the user clicks the start button:

```
// load the customers when the user clicks the Start button
$scope.startButton_Click = function () {
  loadData(svcUrl, $scope.customers, 'Customers');
}
```

The **loadData** function is the only part missing now. We implement it using the jQuery.ajax method, which you have probably used many times if you are a JavaScript developer:

```
// utility to load OData into a CollectionView
function loadData(baseUrl, view, table, types) {

  // build url
  var url = baseUrl + '/' + table;
  url += (url.indexOf('?') < 0) ? '?' : '&';
  url += '$format=json';

  // go get the data
  $.ajax({
    dataType: 'json',
    url: url,
    success: function (data) {
```

```
// append new items
      for (var i = 0; i < data.value.length; i++) {

        // convert data types (JSON doesn’t do dates...)
        var item = data.value[i];
        if (types) {
          for (var key in types) {
            if (item[key]) {
              item[key] = wijmo.changeType(item[key], types[key]);
            }
          }
        }

        // add item to collection
        view.sourceCollection.push(item);
      }

      // continue loading more data or refresh and apply to update scope
      if (data['odata.nextLink']) {
        loadData(baseUrl, view, data['odata.nextLink']);
      } else {
        view.refresh();
        view.moveCurrentToFirst();
        $scope.$apply();
      }
    }
  });
}
```

The interesting parts of this method are:

- The data returns as JSON, rather than XML.
- The data types convert when necessary (since JSON does not support dates).
- A recursive call continues loading data until the last item is retrieved.
- The call to $scope.apply when the data finishes loading signals AngularJS to update the View.

That is the whole ViewModel.

Porting the View

In the original application, the MainPage.xaml file defines the View.

They implemented the interesting parts of the View as follows:

```
<!-- Start button -->
<Button Content="Start" Click="startButton_Click" />

<!-- ComboBox to pick customers -->
<ComboBox
ItemsSource="{Binding Source={StaticResource customersViewSource}}"
DisplayMemberPath="CompanyName"
SelectionChanged="customersComboBox_SelectionChanged">
</ComboBox>

<!-- ComboBox to pick orders -->
<ComboBox
  ItemsSource="{Binding}"
  DisplayMemberPath="OrderID"
  SelectionChanged="orderIDComboBox_SelectionChanged">
</ComboBox>

<!-- DataGrid to show order details -->
<sdk:DataGrid
  AutoGenerateColumns="False"
  ItemsSource="{Binding Source={StaticResource detailsViewSource}}">
  <sdk:DataGrid.Columns>
    <sdk:DataGridTextColumn
      x:Name="productIDColumn" Binding="{Binding Path=ProductID}"
      Header="Product ID" Width="80*" />
    <sdk:DataGridTextColumn
      x:Name="quantityColumn" Binding="{Binding Path=Quantity}"
      Header="Quantity" Width="60*" />
    <sdk:DataGridTextColumn
      x:Name="discountColumn" Binding="{Binding Path=Discount}"
      Header="Discount" Width="60*" />
    <sdk:DataGridTextColumn
      x:Name="unitPriceColumn" Binding="{Binding Path=UnitPrice}"
      Header="Cost Per Unit" Width="60*" />
  </sdk:DataGrid.Columns>
</sdk:DataGrid>
```

There is a ComboBox control bound to the customer list. Its **DisplayMemberPath** property is set to **CompanyName**, so the combo lists the customer's company name.

Below that there is another ComboBox control bound to the customer's Orders collection. The DisplayMemberPath property is set to **OrderID**, so the combo lists the order numbers.

Finally, there is a DataGrid bound to the selected order's details. The grid contains a list of column definitions that specify what parts of the details the grid displays and how to format the data.

Porting this view to HTML would be very difficult if you did not have AngularJS, the required controls, and the AngularJS directives that allow you to add the controls to the application markup.

This is what the HTML5 version of the View looks like. Notice that although this is HTML, it looks a lot like the XAML above:

```
<!-- Start button -->
<button ng-click="startButton_Click()">
  Start
</button>

<!-- ComboBox to pick customers -->
<wj-combo-box
  is-editable="false"
  items-source="customers"
  display-member-path="CompanyName">
</wj-combo-box>

<!-- ComboBox to pick orders -->
<wj-combo-box
  is-editable="false"
  items-source="orders"
  display-member-path="OrderID">
</wj-combo-box>

<!-- FlexGrid to show order details -->
<wj-flex-grid
  items-source="details">
<wj-flex-grid-column
  binding="ProductID" header="Product ID" width="80*">
  </wj-flex-grid-column>
<wj-flex-grid-column
```

```
  binding="Quantity" header="Quantity" width="60*">
</wj-flex-grid-column>
<wj-flex-grid-column
  binding="Discount" header="Discount" width="60*" format="p0">
</wj-flex-grid-column>
<wj-flex-grid-column
  binding="UnitPrice" header="Cost Per Unit" width="60*" format="c2">
</wj-flex-grid-column>
</wj-flex-grid>
```

The Wijmo 5 ComboBox control has **itemsSource** and **displayMemberPath** properties that facilitate porting the view.

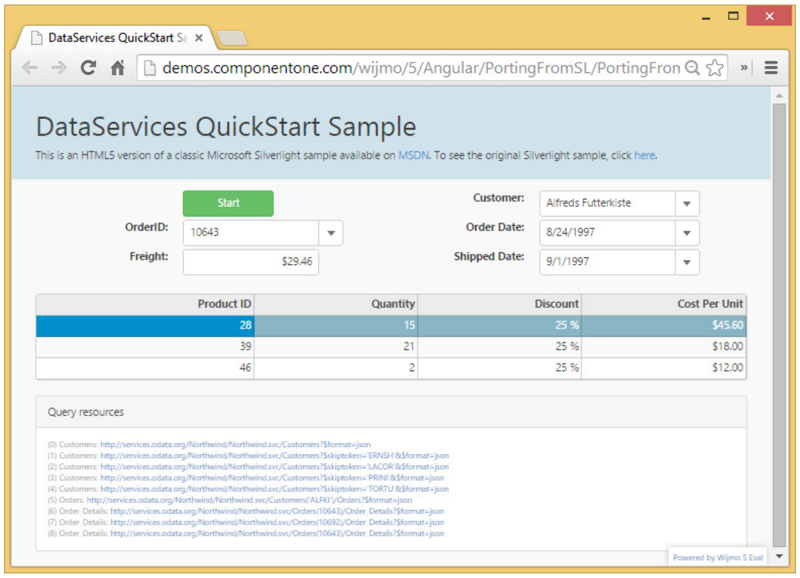
The FlexGrid control also has an **itemsSource** property, and the columns have the same **binding**, **header**, and **width** properties that are common in Silverlight, but missing from HTML. The **width** property even supports star sizing.

This is the core of the View. The actual implementation has a few extra details. We used the Bootstrap layout library to create an adaptive layout, so the application looks good on small devices such as tablets and phones. Bootstrap is easy to use; in most cases you can add some div elements to the page and set their class attributes to values that describe the layout that you want. And it's free…

The Result

The porting effort took only a couple of hours, and some of that was spent studying the original application. You can see the result online:
http://demos.componentone.com/wijmo/5/Angular/PortingFromSL/PortingFromSL/

This is what the application looks like in case you don't want to run it right now:


DataServices QuickStart Sa
demos.componentone.com/wijmo/5/Angular/PortingFromSL/PortingFron
Start

DataServices QuickStart Sample

This is an HTML5 version of a classic Microsoft Silverlight sample available on MSDN. To see the original Silverlight sample, click here.

OrderID:	10643
Freight:	$29.46
Customer:	Alfreds Futterkiste
Order Date:	8/24/1997
Shipped Date:	9/1/1997

Product ID	Quantity	Discount	Cost Per Unit
28	15	25 %	$45.60
39	21	25 %	$18.00
46	2	25 %	$12.00

Query resources

```
(0) Customers: http://services.odata.org/Northwind/Northwind.svc/Customers?$format=json
(1) Customers: http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='ERNSH'&$format=json
(2) Customers: http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='LACOR'&$format=json
(3) Customers: http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='PRINI'&$format=json
(4) Customers: http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='TORTU'&$format=json
(5) Orders: http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')/Orders?$format=json
(6) Order_Details: http://services.odata.org/Northwind/Northwind.svc/Orders(10643)/Order_Details?$format=json
(7) Order_Details: http://services.odata.org/Northwind/Northwind.svc/Orders(10692)/Order_Details?$format=json
(8) Order_Details: http://services.odata.org/Northwind/Northwind.svc/Orders(10643)/Order_Details?$format=json
```

Powered by Wijmo 5 Eval

The application looks a lot like the original, only a little prettier.

But it has the following important advantages:

- It runs on desktops, tablets, and phones.
- It has an adaptive layout, so if you resize the browser the content re-flows automatically and the application remains usable.
- It is only 15% of the size of the original app, so it loads much faster.

We hope this example effectively illustrates how Wijmo 5 can make migrating Silverlight applications to HTML a relatively painless task.

Remember, Wijmo 5 was not designed only for porting Silverlight applications. It was designed to bring the best MVVM features to HTML5 development, so you can be more productive than ever, and deliver solid, powerful, pure HTML5 applications in record time.