Issue 05 | MAR-APR 2013

DNC Magazine

www.dotnetcurry.com

Building an HTML 5 Game Tic-Tac-ToR with SignalR & ASP.NET

Exclusive Interview with John Papa

Custom Unobtrusive jQuery Validator for ASP.NET MVC 4

Designing & Developing apps for Sharepoint 2013

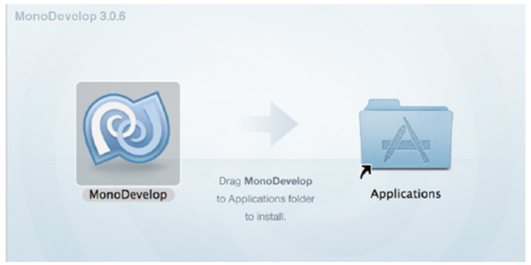
Comparison of Automated Testing Tools
Coded UI Test, Selenium and QTP

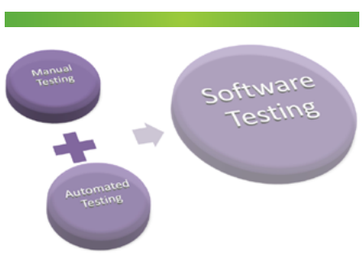
Build your own .NET view engine using Nancy & Mono on OSX

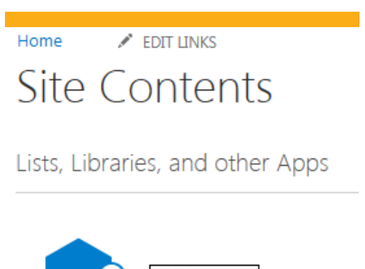
KnockoutJs Cheat Sheet

What's new in RavenDB 2.0

CONTENTS







EDITOR'S NOTE

CHANGE IS THE ONLY CONSTANT [EVERYWHERE]

It is hard to believe eight months have passed since we started DNC Magazine and the fifth installment is already here. In our opening issue (July 2012), we had Ayende (the creator of RavenDB) as our guest and ChirpyR was our featured article, which at the time used a little known hobby project called SignalR.

Things have changed a lot in the past eight months and we have these updates for you. Gregor Suttie walks us through what's new in RavenDB 2.0. SignalR, is now the newest member of the Microsoft Asp.Net team's open source stack as is now officially supported by Microsoft with multiple clients, some of which are open source contributions. Suprotim Agarwal walks us through the changes in SignalR 1.0 and builds a nice online game for us.

In the Interview hot seat we have a prolific technologist and teacher, John Papa. We talk to John about his past and current adventures with code.

We have an interesting new section where Jonathan Channon shows us how to get started with .NET development on OSX using the Mono Framework.

Plus we have jQuery Validators, SharePoint Web Applications, an awesome comparison of Unit Testing tools and a little Knockout JS cheat-sheet to pin to your soft board. Hope you have as much fun as we had putting it together and keep the feedback coming!

Sumit K. Maitra
Editor in Chief

www.dotnetcurry.com
dnc mag

Editor In Chief Sumit Maitra
sumitmaitra@a2zknowledgevisuals.com

Editorial Director Suprotim Agarwal
suprotimagarwal@a2zknowledgevisuals.com

Art & Creative Director Minal, Suprotim Agarwal
minalagarwal@a2zknowledgevisuals.com

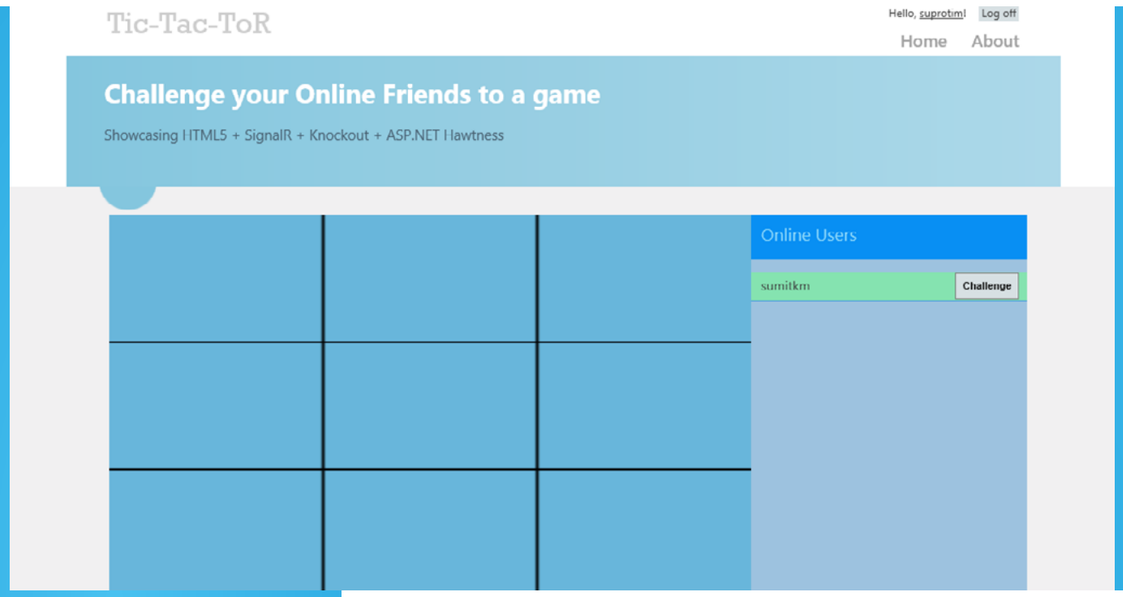
Advertising Director Suprotim Agarwal
business@dotnetcurry.com

Contributing Writers Gouri Sohoni, Gregor Suttie, Jonathan Channon, Pravinkumar Dabade, Sumit Maitra, Suprotim Agarwal

Interview Feature John Papa
Twitter @john_papa

Next Edition 1st May 2013
www.dncmagazine.com

Building an HTML 5 Game
with SignalR, KnockoutJS & ASP.NET MVC



> Let's explore SignalR and some of the changes from the early days to the present 1.0 RTW, as it is brought under the official ASP.NET MVC umbrella.

Suprotim Agarwal, ASP.NET Architecture MVP, along with Sumit Maitra explores SignalR and some of the changes from the early days to the present 1.0 RTW, as it is brought under the official ASP.NET MVC umbrella. He creates an online version of the age old Tic-Tac-Toe game powered by SignalR, HTML5, KnockoutJS and ASP.NET MVC

In the DNC Magazine launch issue (July 2012 issue), we had looked at SignalR in an article by Sumit titled ChirpyR. At that time, SignalR was largely a two person GitHub project driven by David Fowler and Damian Edwards of Microsoft.

Between then and now, SignalR has changed quite a bit. After the ASP.NET Web Tools Update 2, it is now RTW or in other words, officially supported by Microsoft and production ready. There has been some changes in the API and today we will take a sneak peek at some of the changes.

But before that, let's step back a minute and recap.

What is SignalR ?

SignalR is an asynchronous signaling framework enabling clients to get updates from the server in real time.

SignalR is also referred to as a 'persistent-connection' framework which again implies that in the stateless world of HTTP, SignalR provides abstractions that help clients maintain persistent connections with servers over HTTP.

The concept behind SignalR is not new, and ways to achieve the functionality that SignalR offers have existed in the past, however SignalR provides a fantastic abstraction over all the available options. Let's look at these briefly:

WebSocket – WebSocket is the newest technique for persistent connection. It establishes a full-duplex communication channel between the browser and the server. It is an IETF ratified protocol but the problem is it's rather new. It requires not only a browser and web server support, but also support of all proxies in the path. Hence, it may be broken due to reasons beyond our control. Excluding WebSockets, the other techniques of achieving Persistent Connections are loosely referred to as Comet.

Ajax Long Polling – In Ajax Long Polling, the client fires an AJAX request and the server doesn't return till it has data to return. Thus the connection remains open until data is returned. This however expects Ajax support in the browser (which is now common in all browsers).

Forever Frame – This is the oldest technique where an IFrame is embedded in the client HTML page and is sent to the server 'chunked' implying it's of infinite length. This keeps the connection alive and the IFrame is filled with 'script' tags as data comes in. Since scripts are rendered as available, the client is able to respond to the incoming data as it is available.

ServerSent Events – This is a more recent technique and is being formalized under the HTML5 umbrella. It's a technique first implemented by Opera in 2005. This enables web servers to broadcast events to clients once a connection has been established.

All these concepts are fine but the fact that you have to build significant scaffolding to use any of them in a web application, presented a high entry bar for applications that wanted to use persistent connections.

The SignalR framework lowers this bar significantly by completely abstracting away the guts of connection mechanism. It uses whatever is available between the server and the client and presents only a Persistent-Connection interface that lets you think on top of it rather than worrying about the connection implementation.

COMMON USE CASES

Chat Engines – The most common use-case for SignalR's persistent connections is an implementation of Chat Engines. One of the earliest implementation of SignalR was JabbR.net, an IRC clone that has now become a rather popular chat destination for discussing SignalR itself as well as other .NET related technologies.

Collaboration Apps – Collaboration apps are an area that can leverage real-time

feedback by making updates available to reviewers as they happen. You can see a sample by Sumit Maitra in action at http://funwithsignalr.apphb.com/ and read about it here (mind you the sample code is quite old and the packages have changed in v1 that we will see today).

Social Media Apps – Applications like Twitter that broadcast streams of tweets continuously can use SignalR's broadcasting abilities as well. We took a brief stab at that in ChirpyR.

Realtime Games – The game ShootR is an awesome example of how you can build real time multiplayer games with SignalR and HTML5.

Message Bus – SignalR at its core is a Message Bus and implements the Pub/Sub pattern so if you just need a lightweight message bus, feel free to leverage it.

AN HTML5 GAME WITH SIGNALR

Today we will explore SignalR by building a small game for ourselves. We'll build an online version of Tic-Tac-Toe (or Noughts and Crosses). While building the game, we will use the following SignalR features:

1. Hubs – SignalR Hubs provide basic infrastructure over Persistent connections to manage incoming connections, groups, broadcast messages and so on. Essentially they are a good starting point for simpler apps.
2. Groups – As mentioned above, Hubs help manages groups of connections. Groups loosely co-relate to Chat rooms if you think in application terms.
3. Authorization – Hubs support Authorization now via the familiar Authorize attribute. SignalR's Authorize attribute definition is in Microsoft.AspNet.SignalR namespace.

Apart from these features, we will see a technique to keep track of users as they connect/disconnect and reconnect to the server (which is a likely case over the internet). This is a custom implementation because SignalR doesn't map Users to connections. We have used an in-memory mechanism that can be easily extended to include a backing store.

Pre-Requisites

The *ASP.NET Web Tools Update 2* got released while finalizing the article, and it has been updated accordingly. So please ensure you have installed the update.

The Client Side

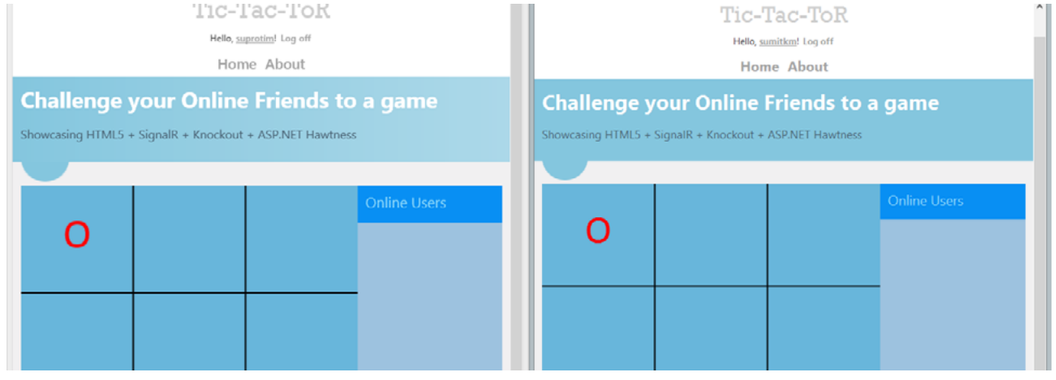
HTML5 Canvas - While we let SignalR manage persistent connections on the server, we will use a web frontend for the UI. We use an HTML5 Canvas to draw the game in the browser. Though it can be done in myriad ways without Canvas, using Canvas helps showcase the ease of building rich apps on HTML5.

KnockoutJs - Apart from Canvas manipulation using JavaScript, we will also use KnockoutJs. Knockout if you are not already aware of, is an excellent templating and two-way data-binding JavaScript library. It helps you implement client side view models.

Since SignalR will be broadcasting live data to us, having a UI that leverages data-binding to update itself as new data comes in, is a must have.

Twitter Authentication – We will leverage the OAuth security mechanism to use Twitter as our Auth provider.

BUILDING 'TIC-TAC-TOR'



The Gameplay

The Gameplay is rather simple.

- People authenticate and get Online.
- Everyone is able to see everyone else (who are) online.
- Logged in user can challenge any other Online User, if the opponent is not busy already.
- On challenge, opponent gets a popup and can either accept or refuse the game request.
- On accept, they get to play first. On play, the screen of both players get updated with the latest game status. During the game the players cannot be challenged by anyone else.
- Game continues till either a winning move is played or all nine possible positions are occupied and game ends in a draw.
- Once the game ends, both players are eligible to challenge/be challenged.
- During the game, if any of the users hit refresh, the page refreshes and gets the latest status from the server. The Game continues.

With the basics out of the way, let's roll up our sleeves and build the app. I'll skimp over the basics as in, we start with an MVC4 Internet Application Template. In the AuthConfig.cs file, we'll add the required keys from Twitter (refer to the 'Real World OAuth' article in the Jan 2013 Issue of the magazine, if you want to learn more about Twitter OAuth in ASP.NET).

Once you have added the Twitter OAuth keys, run the application and login using a twitter account to make sure OAuth is working fine.

Installing SignalR using Nuget

To install SignalR, you can run the following command from the Nuget Package Manager Console

```
install-package Microsoft.AspNet.SignalR
```

This installs the required packages and opens up a ReadMe.txt file. Don't discard it yet, follow the instructions to add the following line as the first Route in App_Start\RouteConfig.cs

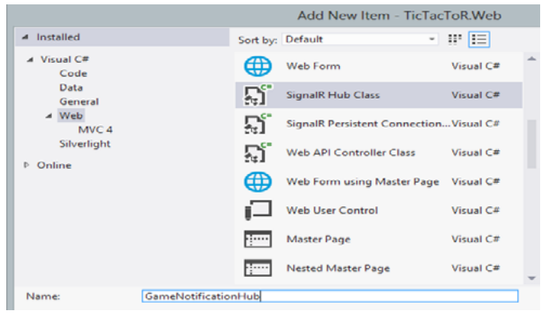
```
routes.MapHubs();
```

This ensures the dynamic paths generated by the client side Hub script work correctly.

Note: The ReadMe.txt says add the MapHubs in Application_Start but since RouteConfig is where we are configuring all other routes, adding it in RouteConfig.cs keeps things together (and it works).

The Hub

Once Authentication is setup and working, let's start on the Server Side implementation. First thing to do is add a folder, let's call it 'SignalrHubs' (do not call it SignalR or Hubs) and add a Hub class. If you have the latest Visual Studio Update, you can use the SignalR template for it or simply add a class and inherit it from Microsoft.AspNet.SignalR.Hub



The Hub Implementation

The Hub implementation is pretty simple. We have the following three methods overridden from the base Hub class.

OnConnected – gets fired by Hub when a user connects to SignalR hub via any client. Our implementation of this method broadcasts the new user to all the other connected clients.

```
this.Clients.Others.joined(
 new {
  … removed for brevity…
     },
     DateTime.Now.ToString());
```

As we can see, Hub provides the Clients property that has a dynamic property called Others. Others represents all connections except for the incoming one. The joined method is the client side method that gets invoked from the Hub on the Server for all client except the one that joined.

OnDisconnected – gets fired by Hub when a user disconnects from a SignalR hub. In case of a Web Client, hitting F5 to do a full refresh, results in a Disconnect.

```
public override System.Threading.Tasks.Task
OnDisconnected() {
 …
 return Clients.All.leave(Context.ConnectionId,
  DateTime.Now.ToString());
}
```

Here the 'All' property is used on the Clients object to reach out to all the connected clients. Again, leave is the method that gets invoked on the client.

Note: The All, Others and Caller properties are new in SignalR 1 as compared to the pre AspNet version.

OnReconnected – gets fired by Hub when a client temporarily loses connection but server is able to re-establish connection with it.

To implement these methods, earlier you had to implement two different interfaces. Now we can simply override the methods on the Hub.

Gameplay Management

We have the Gameplay related methods in the Hub. But before we go into them any further, let's quickly visit the in-memory backend in the Manager class and its related objects.

Manager.cs – The Manager is an in-memory SignalR Connection + game Manager. Since a user can connect and disconnect multiple times, it stores the logged in user with a list of all its recent connections. At the same time, it also stores each Game that's in progress.

UserCredential.cs – This is the per user object that stores all the user's sessions (each time the user connects to SignalR hub, it's treated as one ConnectionSession). There is only one instance of UserCredential for a logged in User.

ConnectionSession.cs – This is the object that stores the SignalR ConnectionId, the connected time and disconnected time. This object acts as the bridge between a SignalR connection and the actual User using that connection.

GameDetails.cs – This class can be considered, the Game Model. It saves the two Players playing the game, the current status, as well as the logic to determine if the game is over, who won or lost or if it was a draw. Every time a challenge is accepted, a new instance of GameDetails is created and saved by the Manager. It gets a unique key that's a GUID. This GUID is also used as SignalR's group ID. Each game's players are a part of the group.

With the game entities laid out, let's look at the GamePlay related Hub Methods.

Challenge – This method is invoked when a player clicks on the Challenge button for any of the online players available. It lobs the request back to the user for whom this Challenge was targeted. So SignalR is doing a 'unicast' to the user at the other end of the Connection Id. A 'unicast' is invoked on the Clients collection as follows:

```
this.Clients.Client(connectionId).
getChallengeResponse(Context.ConnectionId,
userId);
```

Here getChallengeResponse is the method that is invoked on the client.

ChallengeAccepted – If the user who was challenged, accepts the challenge, then this method is invoked from the client. On execution, it creates a new GameDetails instance using details of the two players and also creates a SignalR group using the GameId. Once the group is created, all connections are notified via the beginGame client side method.

```
GameDetails details = Manager.Instance.
NewGame(Context.ConnectionId, connectionId);
Groups.Add(Context.ConnectionId, details.GameId.
ToString());
Groups.Add(connectionId, details.GameId.
ToString());
Clients.All.beginGame(details);
```

GameMove – The GameMove method is invoked when one of the user, makes a move on the canvas by clicking on one of the 9 cells. The starter for the game is determined and saved in the GameDetails instance that was sent on ChallengeAccepted.

Once a game move is made, the entire GameDetails is updated with the latest position and the result lobbed back to the playing group.

That outlines the Server Side implementation. Let's now look at the client side implementation

THE CLIENT

All the client logic is in the Index.cshtml. If you notice, there is no Controller code apart from presenting the View. To start off with, we need to add the KnockoutJs and SignalR script dependencies. Both don't have bundles defined, so we will update the BundleConfig.cs and then add the two new bundles in the Index.cshtml. At the bottom of the page, declare a @section called Scripts and add all the JS reference in it.

```
@section scripts{
@Scripts.Render("~/bundles/signalr")
@Scripts.Render("~/bundles/ko")
…
```

Next we add the special Hub script as follows. This script is generated at runtime and contains the client side implementation to invoke methods on the server side hub. You can pre-generate the script also.

```
<script src="~/signalr/hubs"></script>
```

The KO ViewModel

Knockout supports two way data-binding via the ViewModel that we have to define and apply. We can either create a simple Json object or we can create it as a function. Today we'll create the ViewModel as a function. The KO ViewModel is as follows

```
var gameViewModel = function ()
{
 var self = this;
 self.users = ko.observableArray([]);
 self.game = {};
 self.currentPlayer = ko.observable('Game not start-
ed');
 self.showChallenge = function (currentUser)
 {
  if (currentUser.ConnectionStatus < 3)
  {
   return "display:visible";
  }
  else
  {
   return "display:none";
  }
 }
}
```

As we can see, we've saved the scope in the 'self' variable so that we don't get into JavaScript scoping issues later on.

The ViewModel

The *gameViewModel* has an array of *Users* of type ko.ObservableArray[]; This holds the list of all online users and is bound to a list in the view. Notice it's a Knockout, Observable array. What this means is, Knockout will refresh the UI to which this array is bound when an item is added or removed from the Array.

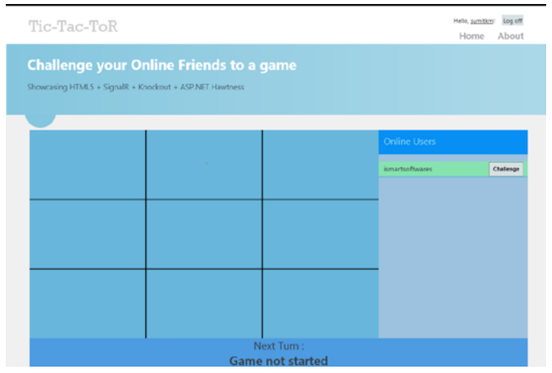
The Game property is initially an empty JavaScript object but once the game starts, it's the JSON Serialized version of GameDetails from the server side.

The CurrentPlayer property shows which player makes the next move. It shows 'Game not started' by default. Again notice we are declaring it as a Knockout Observable property and this means if a UI element is bound to it, and the value of CurrentPlayer changes, the UI element will be updated automatically.

The showChallenge function actually computes the correct display style based on the User's current status. As we will see shortly, this is used to show/hide the challenge button. If a user is already playing, then the challengeButton is hidden.

The View

Let's look at the data binding pieces of the view more closely.



Overall, the view uses server side code to check if the user is Authenticated. If not, it displays a login button, and if Authenticated, displays the game screen.
Let's look at the knockout data-binding pieces more closely.

```
<ul id="activeUsersList"
    class="game-player-list"
    data-bind="foreach: Users">
 <li class="game-list-item">
  <div style="height: 30px;">
```

```
   <div style="float: left; padding-top: 5px">
    <span data-bind="text: UserId"></span>
   </div>
   <div class="game-list-item-button">
    <div data-bind="attr: { style: $parent.show
    Challenge }">
     <button
      class="challenger game-list-button">
      Challenge
     </button>
   </div>
   </div>
   <input
    type="hidden"
    data-bind="value: ConnectionId" />
   </div>
  </li>
 </ul>
 <div style="width: 100%; text-align: center;
font-size: 20px">
  <label data-bind="text: CurrentPlayer()"></
label>
 </div>
```

Binding to a List

We are binding to the unordered list by specifying the data-bind="foreach: Users". This implies Knockout will look for an array element called Users. Knockout also treats everything inside the Ul now as a template. So you need to specify only one ListItem in the markup and then it will be repeated for all the items in the 'Users' array.

KO text binding

Knockout has binding syntax of text or value fields of an HTML element. As we can see above, the UserId is bound to a span tag using the data-bind="text: UserId" syntax. KO already uses the User object context and pulls out the UserId property to bind it to the Span tag. Similarly ConnectionId is bound to a hidden field using data-bind="value: ConnectionId". At the bottom of the play area, we have the CurrentPlayer bound to the text so whenever a player makes a move, the text get updated with the correct player's name.

KO attribute binding

Apart from text or value, if you want to bind to another property of an element; for example changing the display value on the fly, you can use what is referred to as attribute binding in KO. For example in the above markup data-bind="attr: { style: $parent.showChallenge }" calls the showChallenge method in the View Model that returns the correct style for each logged in user.

Hub Client

Now that we've looked at the bindings, let's see what the client side implementation of the SignalR Hub looks like.

```
$(function ()
{
 var viewModel = new gameViewModel();
 ko.applyBindings(viewModel);
 var canvas = document.
     getElementById("gameCanvas");
 var hSpacing = canvas.width / 3;
 var vSpacing = canvas.height / 3;
 var hub = $.connection.GameNotificationHub;

 hub.client.DrawPlay = function (rowCol, game,
 letter)
 {
  // draws the latest move and puts the 'letter'
  // updates the game object in KO ViewModel
}
hub.client.joined = function (connection,
 dateTime)
{
 // updates the user list
}
 hub.client.updateSelf = function (connections,
 connectionName)
{
 // On log in receives the list of all
 // users logged in to display in the online
 // Users' list
}
hub.client.beginGame = function (gameDetails)
{
 // Get the game details as the game starts
}
hub.client.leave = function (connectionId)
{
 // Update player list when someone leaves
};
$.connection.hub.start().done(function ()
{
 // SignalR Hub connection is complete
  // let's roll
 } … }
```

In the code above, we get started by getting the hub instance on the client first.

```
var hub = $.connection.GameNotificationHub;
```

As you can see, it uses the same name as the server-side hub we have. Once we have the hub object, we attach the function definitions that can be accessed from the server.

Once the definitions are done, we call the connection.hub.start() method on the hub. This results in the OnConnected method getting invoked on the server and SignalR registering the client instance. We also attach a callback that is called once the start() method returns.

Drawing on the Canvas

The drawing logic on canvas is rather simple. We extract the '2d' context

```
canvasContext = canvas.getContext('2d');
```

Reset the width to the bounding rectangle. This is important else the click event gets wrong co-ordinates

```
var rect = canvas.getBoundingClientRect();
canvas.height = rect.height;
canvas.width = rect.width;
```

Next we add a canvas click event listener to check on which cell the user clicked. However we have a guard clause that checks to see if the CurrentPlayer() is the same the as the current logged in user. CurrentPlayer is set as soon as the game begins and toggles every time the player makes a move.

```
canvas.addEventListener('click', function (evt) {
 if (viewModel.CurrentPlayer() == '@User.Identity.
Name')
 {
  var rowCol = getRowCol(evt);
  rowCol.Player = 'O';
  hub.server.gameMove(viewModel.Game.GameId, rowCol);
 }
}, false);
```

We have a drawGrid method that's invoked on the Document Ready event and simply renders the Tic-Tac-Toe Grid.

```
function drawGrid() {
 var hSpacing = canvas.width / 3;
 var vSpacing = canvas.height / 3;
 canvasContext.lineWidth = "2.0";
 for (var i = 1; i < 3; i++)  {
  canvasContext.beginPath();
  canvasContext.moveTo(0, vSpacing * i);
  canvasContext.lineTo(canvas.width, vSpacing * i);
  canvasContext.stroke();

  canvasContext.beginPath();
 canvasContext.moveTo(hSpacing * i, 0);
  canvasContext.lineTo(hSpacing * i, canvas.height);
  canvasContext.stroke();
 }
}
```

We have three other helper functions to get the Mouse Position, calculate the Row/Column position based on the mouse position and finally we have the method that write the X or O on the canvas surface.

That's pretty much it. We have a game at our hands. You can go play it on bit.ly/dncm5-tttah

CONCLUSION

Well, we first looked at what SignalR does under the hood and tried to gauge a sense of why it exists. Next we took a tour of SignalR with a focus on how it has changed over the past few months. Finally, we built a real-time game with it, but the underlying infrastructure that we setup to map connections to users, can be used in a multitude of applications. We can monitor on what page people are, we can provide real-time feedback to users if they are having trouble, we can build collaboration apps that multiple people can log on to and review at real time.

Essentially, it's a whole new world! ■

The entire code for this app is available on Github at **bit.ly/dncm5-ttt**

Suprotim Agarwal, ASP.NET Architecture MVP, is an author and the founder of popular .NET websites like dotnetcurry.com, devcurry.com and the DNC Magazine. You can follow him on twitter @suprotimagarwal

BUILD YOUR OWN .NET VIEW ENGINE

USING NANCY & MONO ON OSX

Jonathan Channon leads us into the world of .NET on Apple's OSX in particular and the *NIX world in general. He carefully guides us through the steps required to setup the Mono Framework and its companion IDE MonoDevelop. He develops a .NET View Engine using Nancy and highlights the caveats and gotchas along the way.

Windows and Visual Studio go hand in hand when it comes to developing .NET applications. However, more and more people are looking to develop .NET software on *NIX based operating systems where tools such as Visual Studio do not exist. In fact the .NET framework does not install natively on these operating systems. Enter two open source projects, Mono and MonoDevelop.

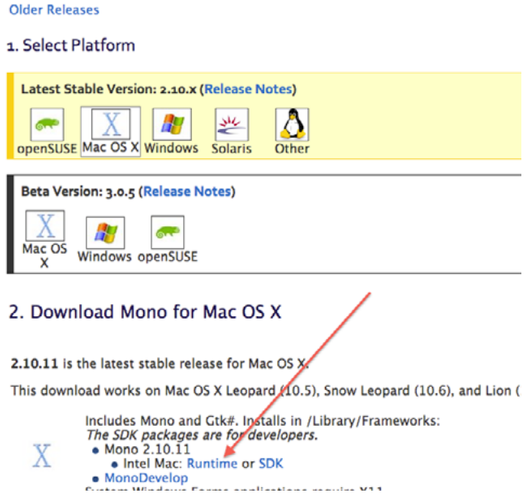
Mono is an open source, cross-platform, implementation of C# and the CLR that is binary compatible with Microsoft.NET. What this means is that software developers can now choose to use a Linux distro or OSX as their development platform and create .NET applications. This allows users to run the application on Windows and also enable the users to choose which operating system they would like to use the application on.

The integrated development environment to create these applications is MonoDevelop. It is an IDE designed to develop .NET applications but is not restricted to C# or Windows.

In this article, we will setup Mono on OSX and develop a Web Application that can be hosted on OSX or Windows.

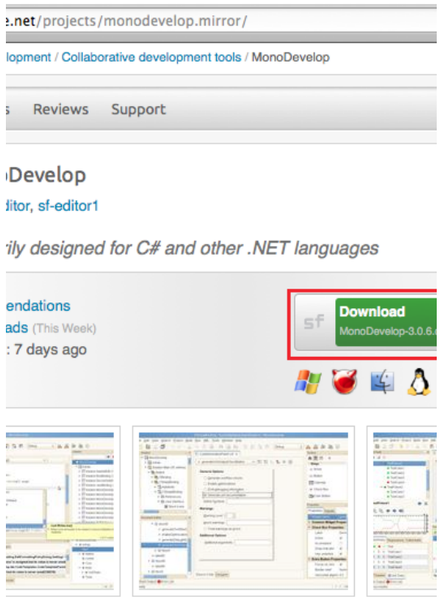
DOWNLOADING MONO/ MONODEVELOP

Editor's Note: Xamarin has recently announced Xamarin 2.0 and Xamarin Studio that supersedes MonoDevelop. We have tested the code in Xamarin Studio and it works fine, however the Nuget extension is still to be updated to support Xamarin Studio. We plan to post an update on our web-site once things settle down with Xamarin Studio.



To get going, we first need to install Mono and then MonoDevelop. Navigate to http://www.go-mono.com/mono-downloads/download.html and click on Mac OS X to get the download link for the OSX installer download. Click on the 'Runtime' link to download the runtime installer and save it.

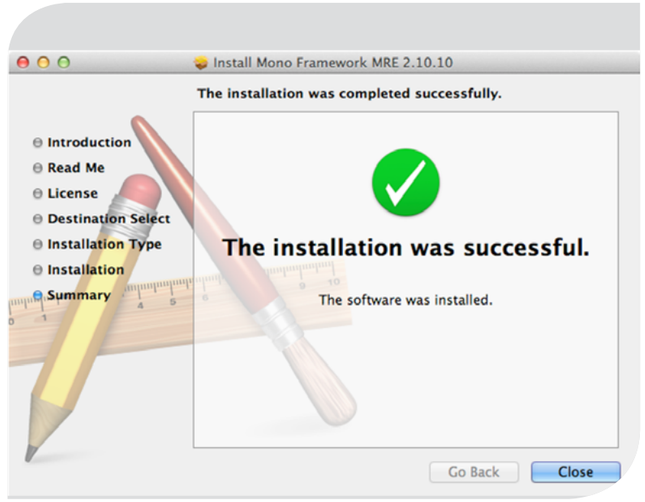
Next let's download Mono Develop the IDE. Point your browser at http://sourceforge.net/projects/monodevelop.mirror/. It will auto-detect OS and show appropriate installer. Here is it showing dmg for OSX.



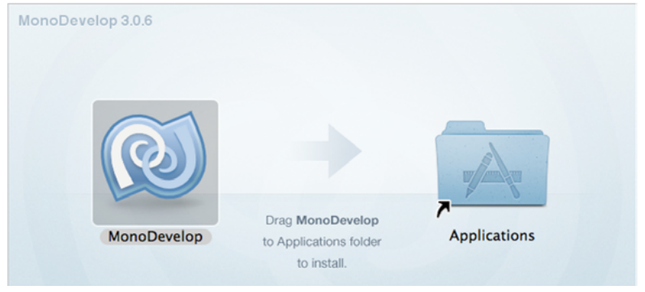
INSTALLING MONO/ MONODEVELOP

We first have to install Mono, so double click the downloaded installer MonoFramework-MRE-2.10.11.macos10.xamarin.x86.dmg to mount it and then double click the MonoFramework-MRE-2.10.11.macos10.xamarin.x86.pkg to get things going.

Simply follow the wizard and enter your password when prompted and we're all done. It's a quick install.



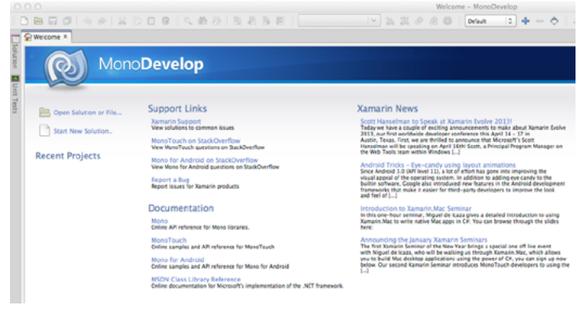
You can un-mount Mono now and double click MonoDevelop-3.0.6.dmg to mount that. You can then just drag the MonoDevelop icon to the Applications shortcut to install it.



When copied across, you can then unmount MonoDevelop-3.0.6.dmg.

RUNNING

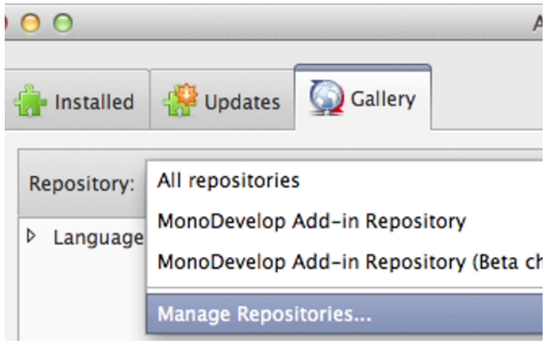
We can now fire up MonoDevelop by going to our Applications folder and clicking the MonoDevelop icon. You may observe it executes a shell script in the background but that's a one-time thing. You should now have something like this:



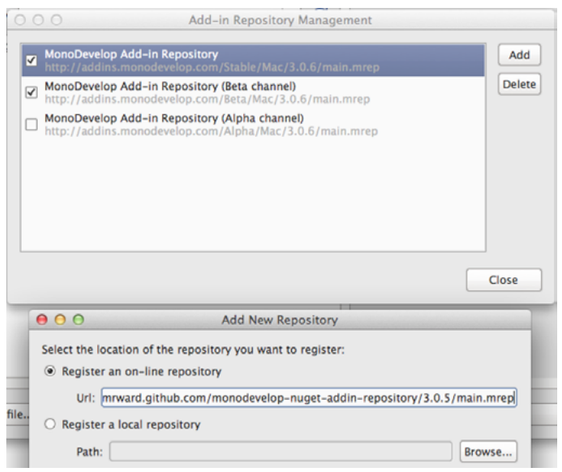
NUGET

Next thing on our checklist is to install the NuGet add-in. This is a new development system compared to the Windows version we've had for a long time. You can find out more about the MonoDevelop NuGet add-in and its origins here. Before the NuGet add-in, often developers had to clone library source code to their machine and compile it for Mono and then add a reference to the built DLL. We all love NuGet, for the most part, so this add-in makes things a lot easier for us.

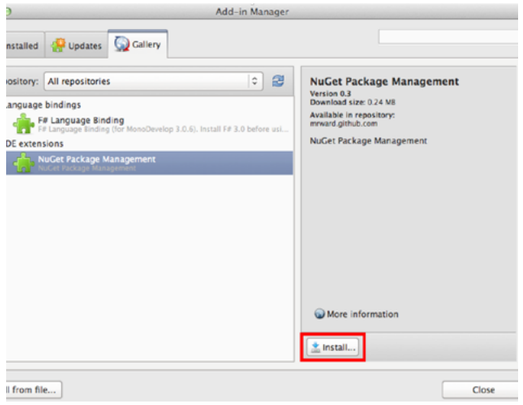
Click MonoDevelop in the menu bar and then select Add-in Manager. Click the Gallery tab and then the drop down box titled All Repositories and choose Manage Repositories.



Click the Add option and enter in http://mrward.github.com/monodevelop-nuget-addin-repository/3.0.5/main.mrep



This will add the repository to the items in the drop down box. Select the one titled mrward.github.com, expand the IDE extensions, select NuGet Package Management and then click the Install button on the right side pane.



It will ask you to confirm installation and a progress bar will appear and once complete, we're ready to rock!

NANCY TIME

To test that our development environment is all setup and ready to develop that amazing app idea you have, we should probably get cracking and write a simple "Hello World" app. However, that's boring and we like to have fun, don't we? So we are going to write a Markdown View Engine for Nancy http://nancyfx.org, then an ASP.NET hosted application that uses Nancy and our new view engine.

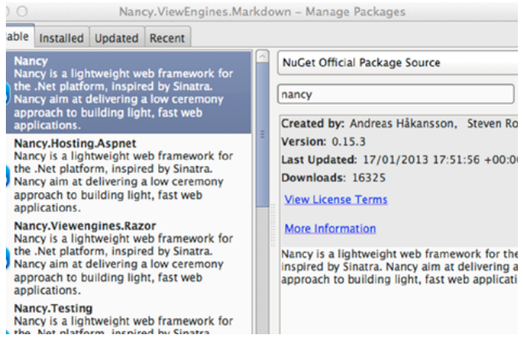
If you're new to Nancy then here is a snippet taken from their documentation: *Nancy is a lightweight, low-ceremony, framework for building HTTP based services on .Net and Mono. The goal of the framework is to stay out of the way as much as possible and provide a super-duper-happy-path to all interactions.* If you would like to learn more about Nancy, you can read the numerous community blog articles about it and use the following references

1. The official documentation,
2. A blog post I did "Why use NancyFX"
3. The Herding Code's recent podcast with the main developers of Nancy

SETTING UP NANCY

To get started, we need to create a new class library. In MonoDevelop, click File-New-Solution. In the window that pops up, select C# from the left pane and Class Library on the right pane. Give your project a name and choose where you would like your project to be saved and click the Forward button, then the OK button. You should now see something familiar, i.e. a class' source code. We need to rename our class to MarkdownViewEngine.cs and we are ready to import Nancy.

Right click your project and you'll see Manage NuGet Packages (thanks to our earlier setup). On the 'Available' tab, type Nancy into your search box and hit Enter. All the Nancy packages should appear, click the one titled Nancy and then the Add button.



We now need to search for another package called MarkdownSharp and add that to our project. Once added, we can click the Close button. Don't we just love having Nuget around?

IMPLEMENTING THE VIEW ENGINE

In our class, we now need to reference MarkdownSharp in our using statements and for our class to implement IViewEngine (As per Nancy code guidelines, place using MarkdownSharp within the namespace). We can use MonoDevelop to right click on IViewEngine and for it to implement the members for us. This exposes Initialize, RenderView and Extensions. The initialize method is called when the view engine is created and all views that the engine can render, will be passed in, along with the view cache. The RenderView method is called when one of the Nancy modules makes a call to render a View and the ViewLocationResult of the view is passed in, along with a model object and a RenderContext. The Extensions property is an IEnumerable<string> which exposes the file extension that our view engine supports.

The idea behind our ViewEngine is that we can write our views in Markdown and when the View is called within our Modules e.g. slash (/) returns View["Home"]; the ViewEngine will find the relevant Markdown file, convert its content to HTML and issue that in a response.

Our ViewEngine class therefore, needs to make our Extensions property to return a IEnumerable<string> with the "md" file extension, read the contents of the file called by View["Home"], convert it to HTML, cache that operation and return the HTML in a response. Once that is completed, you should have something like this:

```
namespace Nancy.ViewEngines.Markdown
{
 using System;
 using Nancy.ViewEngines;
 using System.Collections.Generic;
 using Nancy.Responses;
 using System.IO;
 using MarkdownSharp;

public class MarkDownEngine : IViewEngine
{
  public IEnumerable<string> Extensions
  {
   get { return new[] { "md" }; }
  }

  public MarkDownEngine()
  {

  }

  public void Initialize(ViewEngineStartupContext
  viewEngineStartupContext)
  {
  }

  public Response RenderView(ViewLocationResult
  viewLocationResult, dynamic model, IRenderContext
  renderContext)
  {
   var response = new HtmlResponse();
   string HTML = renderContext.ViewCache.
   GetOrAdd(viewLocationResult, result =>
    {
     string markDown = viewLocationResult.Contents().
                       ReadToEnd();
     var parser = new MarkdownSharp.Markdown();
     return parser.Transform(markDown);
    });

    response.Contents = stream =>
    {
     var writer = new StreamWriter(stream);
     writer.Write(HTML);
     writer.Flush();
    };
    return response;
   }
  }
}
```

The renderContext has a ViewCache property that we can use to cache the operation of reading our files and converting it so we don't have to do that expensive operation each time the view is called. We find our Markdown by reading all the content in the file by using the Contents property of the ViewLocationResult. This is a Func<TextReader> which we call and then use ReadToEnd to get all the information out. We convert our Markdown by calling the Transform method from MarkdownSharp. This returns a HTML string for us. To get that content into our response, we can use the Contents property of HtmlResponse. This is an Action<Stream> property that is invoked at the last minute within Nancy to render the content. Here we can create a new StreamWriter with the Stream argument passed in and then write and flush our HTML content to it. Then finally return our response.

That's it! You have written a view engine for Nancy.

USING THE VIEWENGINE IN A WEB APPLICATION

I guess we should make sure this works. In MonoDevelop, right click on the solution and select Add > Add New Project. Select C#-ASP.NET on the left pane and choose Empty Web Application on the right pane. Again choose where and what to call your project, in this case we'll call it Nancy.ViewEngines.Markdown.Demo.

Right click the demo project and select Manage NuGet Packages, search for Nancy and select Nancy.Hosting.Aspnet, click Add, then Close. This will also pull the Nancy dependency for us. Unfortunately on Mono, web.config transforms do not work so we have to create a web.config file manually and add in the request handlers for Nancy. The source for this can be found at bit.ly/dncm5-nancymono

We also need to add a reference to our Nancy.ViewEngines.Markdown project so right click on the demo project's References folder and select Edit References. On the Projects tab, select our viewengine project and click OK.

Create a folder called Modules and Views in our demo project. Under the Views folder, put a Markdown file called home.md and populate it with Markdown. Under the Modules folder, create a class called HomeModule. This should inherit off NancyModule. If you are from an ASP.NET MVC background, Modules can be likened to Controllers. The constructor of Nancy Modules define our routes using HTTP verbs. In this case, we just want to handle a GET request. Shown below is what you should end up with:

```
namespace Nancy.ViewEngines.Markdown.Demo {
 using System;
 using Nancy;
 public class HomeModule : NancyModule {
  public HomeModule ()
  {
   Get["/"] = _ => View["Home"];
  }
 }
}
```

When a request comes in for "/" we tell it to return a View called Home. This is where our ViewEngine comes into play. One of the great things about Nancy is that it has an IoC container built into it and auto scans for dependencies. Therefore apart from the reference to our ViewEngine project, we do not have to do anything as the IoC container will find an implementation of IViewEngine and our code will be executed when the View["Home"] call is made.

We are now ready to fire up our demo project. Instead of F5, press the CMD+Enter keys and MonoDevelop will start a browser and begin to debug. If everything has gone according to plan, you should now be seeing HTML in your browser. Obviously to make this a fully functional ViewEngine, we need to implement layout/master pages, some sort of template processing to do the equivalent of @section{} and possibly model/dynamic viewbag processing, but this short introduction should show you how easy Nancy is to extend.

CAVEATS

Setting up Mono, MonoDevelop, Developing and being able to debug proved to be really straightforward however it's not perfect. If you were following the steps above, everything will work fine. However, if you cloned the code repository or decided to switch between VisualStudio and MonoDevelop, there are some special mentions.

Tinkering and tweaking to get Nuget Package Restore going

During development of the ViewEngine project, I moved over to Windows at one point and enabled NuGet Restore in Visual Studio and then came back to MonoDevelop to find this feature did not automatically work. I had to make some changes to MonoDevelop to make sure the solution built correctly and used Nuget Package Restore automatically. Specifically we have to do these changes:

1. In MonoDevelop, change the build mechanism to XBuild/MSBuild experimental. To do this
   a. Go the MonoDevelop menu and bring up the Preferences dialog.
   b. Select Build and check the 'Compile Projects using MSBuild / XBuild' option.

If you are not working with Web Development, no futher changes are required. But since we are building a Web Application, we got to make some more changes.

2. Copy the v9.0 MSBuild targets as v10.0 build targets.
   a. Open Finder and navigate to /Library/Frameworks/Mono.framework/Libraries/mono/xbuild/Microsoft/VisualStudio/
   b. Copy the folder v9.0 paste it at the same location. It will create a folder called 'v9.0 copy'
   c. Rename 'v9.0 copy' to 'v10.0'. Ultimately you will have a folder structure as follows /Library/Frameworks/Mono.framework/Libraries/mono/xbuild/Microsoft/VisualStudio/v10.0/WebApplications/

3. Updating Microsoft.WebApplication.targets in Nancy.ViewEngines.Markdown.Demo.csproj
   a. Open the *.csproj in a text editor and look for Microsoft.WebApplication.targets
   b. The Import should have Project reference pointing to the Microsoft.WebApplication.targets file in the above folder e.g.

```
<Import Project = "/Library/Frameworks/Mono.framework/Libraries/mono/xbuild/Microsoft/VisualStudio/Microsoft.WebApplication.targets" Condition="'$(VSToolsPath)' != ''" />
```

   c. Again, in our case we have already made this change in the code repository so you can skip Step 3 altogether.

Luckily Matt Ward (developer of the NuGet Package Manager for MonoDevelop) was able to help me out with these issues but it does highlight some issues with the cross platform portability that Mono/MonoDevelop has.

SOME OTHER ODDITIES

I also encountered an odd error when developing the ViewEngine. When the GetRootPath method was called, it threw a silent exception that just stopped executing the code. Putting a watch on that method showed me a System.InvalidProgramException: Missing or Incorrect header.

Remember I earlier said to use Path.DirectorySeparatorChar, well this is the reason. In my code, I had the path separator as "\\" which is not correct on a UNIX platform but fine on Windows. It was obviously not right but MonoDevelop gave me a red herring when looking into the issue. I've also spoken to people that had issues with the debugger not working correctly and a complete reinstall of Mono & MonoDevelop fixed the issue. I'm sure there are oddities out there but I guess this is part of the problem when porting something across to another platform.

Overall it was a good experience and using Mono friendly libraries like Nancy certainly helps the process of developing applications on Mono. I hope my demonstration shows the ease of building applications on Mono and I certainly hope more people contribute and iron out the issues that exist within it as it has great potential. As a personal favorite, it also adds weight to my belief that OSX is a great platform to develop applications on giving you all the UNIX and .NET based platforms/frameworks to develop with.

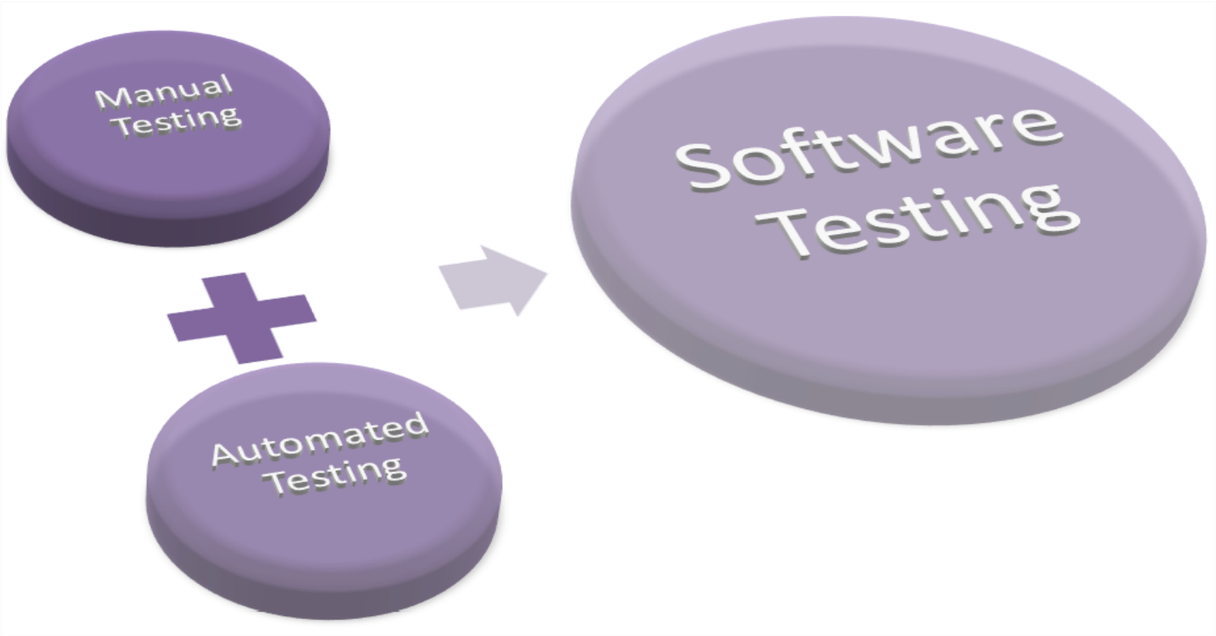
I encourage you to install Mono/MonoDevelop and begin writing applications and explore the possibilities out there for writing software on OSX, and most importantly have fun whilst doing it.

CONCLUSION

As .NET developers, we are all used to developing applications on Windows and Visual Studio but more and more people are looking to use their Linux or OSX machines to develop .NET software. The guys behind Mono and MonoDevelop have to be applauded in their efforts to bring us .NET and Visual Studio to a UNIX platform. This cannot be an easy task. Hopefully if more and more people begin to use Mono, then they will contribute back to the project and make it more stable and feature rich. From my research, it certainly appears as if more people are using it, so watch this space. ■

Source for the Sample application is available on Github at bit.ly/dncm5-nancymono

Jonathan has been a .Net developer for 9 years who has a passion for OSS and using the best tool for the job. He works with the full MS stack, web and desktop, believes in SOLID and Unit Testing and is also a NancyFX MVM. You can follow him on Twitter @jchannon and read his blog bit.ly/WnCkV4v



COMPARISON OF AUTOMATED TESTING TOOLS - CODED UI TEST, SELENIUM AND QTP

ALM veteran *Gouri Sohoni* evaluates multiple Test Automation tools viz. Selenium, QTP and CUIT. She undertakes a side by side comparison of the features to give us a high level picture of the capabilities of these suites.

Software Testing is a way to validate and verify the working of a particular product or application. It can be incorporated at various points of time in the development process depending upon the methodology and tools used. Testing usually starts after the crystallization of requirements. At a unit level, it starts concurrently with coding whereas at an integration level, when coding is completed. Testing is used for finding out the bugs in the application. It also helps identify potential points of failure before the application crashes in production. Finally, it helps validate stakeholders requirement of features and quality.

MANUAL AND AUTOMATED TESTING

There are mainly two broad types of software testing - Manual Testing and Automated Testing.

Manual Testing

Manual Testing involves testing the software without any automation script or tool. Testers check the application or software by taking up the role of an end user. They try to find out if there is any unexpected behavior or failure in the application. Test Management can be taken care of by using test plans and test cases.

Automation testing

Automation testing process involves testing with the help of automation scripts and executing the scripts to run the application with the help of some automation tool. Once the script is ready, then these tests can run quickly and efficiently.

Since the cost of automated testing is in the form of efforts and time required to create the scripts, not all tests can be automated. There should be valid reason to pay that cost.

Reasons for Automation

1. Regression testing to confirm that new changes have not affected the application adversely. It considers already existing test cases for execution. This is an efficient process when we need to provide feedback to the developer immediately.
2. The test cases need to be iterated multiple number of times often with varying datasets to cover multiple workflow paths.
3. When we require support for agile methodologies.
4. Customized reports are required for monitoring.

GETTING STARTED WITH AUTOMATED TESTING

Once the need for automated testing has been established, it involves creation of relevant test scripts. Test script creation can be done only by a skilled tester having knowledge of testing tools, as well functionality under development. Such resources are costly and their time is premium. Considering this fact, it is often not possible to budget the automation of all tests. Some of the major decision points while identifying cases for testing automation are:

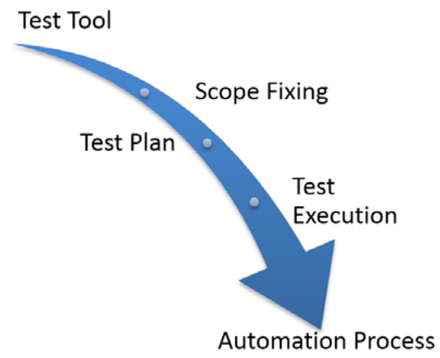
1. System modules where requirements do not change frequently
2. Ample time is at hand to describe a test via scripts
3. The application/software module is critical enough to justify the upfront cost of automation.
4. After functional testing, we want to do performance testing with multiple virtual users using the same test script.

With the scope of automation decided, next step is to pick the testing tool. The following checklist can help with the selection.

1. The tool should be easy to work with. It should execute test cases in an unattended manner, should provide an interface to write scripts, have an efficient IDE and ease of test execution.

2. The tool should provide support to various technologies. It should support testing using different browsers, languages, and types of applications.

3. It should integrate with a software that does Application Lifecycle Management (ALM) so that it can be used for running automated Build Verification Tests as well as the reports can be integrated with other reports created by ALM software.



SOME AUTOMATED TESTING FRAMEWORKS

Today we will compare three automated testing tools namely Selenium, QTP (Quality Test Professional) and Coded UI Test (CUIT) with Visual Studio 2012.

We will consider the above mentioned aspects of automation and see how these tools provide support for each category.

General information about these tools

Selenium was created by Jason Huggins. It is an open source testing tool. Later Simon Stewart started WebDriver (to overcome some limitations of Selenium). Both the tools are now

merged to get one awesome testing tool. HP QTP (Quick Test Professional) was originally written by Mercury Interactive. It is a part of HP Quality Centre Suite (QC). Coded UI Test (CUIT) was introduced by Microsoft along with Visual Studio 2010. It integrates with Team Foundation Server.

Ease of Use

• Recording and Playback Functionality
Each of the testing tools has the ability of recording the actions and playback the recorded actions. Selenium provides a plug-in named Selenium IDE with Mozilla Firefox with which actions can be recorded. QTP provides record button to record a new test. Recording for CUIT can be done using two different tools. Microsoft Test Manager can be used to record actions that can be then converted to CUIT. A fast forward playback is available to run the test case in a semi-automatic mode even before converting to script. With Visual Studio, CUIT provides Coded UI Test Builder to record the actions.

In my opinion, all the three mentioned tools are very easy for record and playback.

• IDE and tools with which the tester can write scripts
With Selenium IDE, there is no special tool and specific technology to write the script. We can insert commands with 'Table or Source View' when required.

QTP provides Keyword View to display test steps graphically or Expert View which shows VB Script lines.

For CUIT we can easily use Visual Studio IDE to write scripts.

Selenium IDE comes as a plug-in with Mozilla Firefox. With this, we can create a test suite which comprise of various test cases. With Selenium IDE, you can convert recorded Selenium IDE scripts into different languages and after conversion you can run it in Selenium RC. Selenium RC has two components, one is "Selenium Server" and another is "Selenium Client".

With QTP IDE, for the first time, 3 add-ins are provided - ActiveX, Visual Basic and Web. Various links to best practices, new features for the current version are available on the start page. We can either open an existing test case or create a new one.

For CUIT, we have a very elaborate IDE as the recording can be done using Visual Studio. All the features of Visual Studio are applicable. The script writing support with Visual Studio is an excellent way of writing and debugging.

In my opinion, CUIT scores more points in this area.

• Ease of Test Case execution
With Selenium IDE, we have the option of executing the entire test suite already recorded or a test case at a time.
Depending upon the add-ins loaded in QTP IDE, the record and run window shows tabs. Windows Application tab is always available. The tests can be executed with the Run button which in turn opens the Run dialog box. We can specify the location for Run specific results and provide parameters if any.

CUIT can either be executed with Visual Studio or by using Microsoft Test Manager (MTM). With MTM we can provide various settings for test case execution so as to gather a lot of information while executing the test case behind the scene. Test Execution can be done in more or less a very straight manner with the 3 tools. CUIT provides various test settings to execute test cases so as to capture different data when we need to create a bug (commonly called as rich bug).

All the 3 tools can execute test cases without human intervention.

Platform Support

• Language Support
Selenium uses Selenese, a high-level, cross platform language to write Selenium commands which is a domain specific language. There are 3 basic categories for the commands - named actions, accessors and assertions. To write tests, there are a lot of programming languages like C#, Java, Perl, PHP, Python or Ruby.

QTP scripts can be written with VBScript which is a high-level language with support to everything except polymorphism and inheritance.

For CUIT, we can write a script with Visual Studio using object programming concepts if required. Writing VBScript code is very easy from tester's perspective. Even though CUIT supports object oriented programming, the testers may not prefer as it involves a lot of skill for writing or customizing the created script.

• Support for various application types
Selenium supports only Web application.

QTP supports almost any kind of application.

CUIT supports Windows Applications, Web Application, WPF applications, SharePoint, Office Client Applications Dynamics CRM Web Client applications.

Selenium scores fewer points in this regard as it supports Web applications only. QTP supports almost all kinds of applications as against CUIT.

• Support for various browsers
Selenium supports all versions of IE, Firefox, Safari and Opera and a few more browsers.

QTP supports IE & Firefox. But both do not provide full cross browser support.

CUIT supports only IE8, IE 9 and IE 10 (IE 10 supported only on desktop). There is no support to IE6, IE7, Chrome, Opera or Safari.

Selenium is a clear winner in this respect

• Support for Data Driven Testing
Selenium IDE supports XML data source using user extensions.

Data Driven testing is implemented as Excel workbook that can be accessed by QTP. There are 2 types of data sheet - global and local. Global sheet is a single one which can be accessed from every action in a test. There can even be a local data sheet associated with every action.

Coded UI Test supports any data source supported by .NET framework which can come in the form of a .CSV file, XML file or any other data source like SQL Server table, Access table etc.

In my opinion, CUIT provides better ways of data driven testing.

• Exception Handling
Selenium IDE does not support error handling particularly unexpected errors (as it supports only HTML language). Selenium RC will provide support for it (it supports languages with .NET, Java, Perl, Python, PHP, Ruby).

QTP provides VBScript with the help of which we can use On Error statements.

As CUIT supports high level languages like C# or VB.Net we can use try catch construct here. With CUIT it is suggested to capture the base exception and write code accordingly.

In my opinion all the 3 tools have their limitations here.

• Validations or Assertions
Selenium assertions can be used in 3 modes - assert, verify and waitFor. When an "assert" fails, the test is aborted. When a "verify" (Soft Assertions) fails, the test will continue execution, logging the failure. This facility can be used with TestNg framework. The "waitFor" commands wait for some condition to become true. They will succeed immediately if the condition is already true. However, they will fail and halt the test if the condition does not become true within the current timeout period.

For QTP there are checkpoints: to verify application under test. These are of 10 types – Standard, Table, Image, Bitmap, Database, Text, Text Area, Page, Accessibility, and XML. A checkpoint is a verification point that compares the current value with the expected value. If the current and expected value match, it generates a PASS status otherwise FAIL status.

We can use Coded UI Test Builder to add assertions for your UI controls. We need to edit the assertion condition as required (equal to, in between, contains etc.), provide expected value and generate code for it. Mouse hover events can be recorded manually if required.

In my opinion QTP and CUIT are better tools to be used in this aspect.

• Support for Objects
Object properties are not supported by Selenium. Selenium objects can be managed by using UI element user extensions.

QTP comes with in-built object repository. QTP objects have user friendly names.

Coded UI Test code is written with 3 main parts for UI controls - IMap.designer.cs, UIMap.cs and UIMap.uitest. The first two are different physical files for the same partial class while the third is a XML equivalent of all the actions recorded with CUIT Builder. Any changes required can be incorporated with the help of the partial class file. We can also edit the UIMap with the help of Coded UI Editor and find out object's properties. CUIT can be completely hand coded if required. Coded UI Test includes a rich API library to code against and a resilient record and playback tool. Coded UI Test can be extended to support custom controls.

CUIT is the clear winner in this regard.

Integration with Application Lifecycle Management and going beyond

• ALM Integration
Selenium being an Open Source software can be integrated with other Open Source products for Application Lifecycle Management like QMetry. This in turn can provide platform for software development lifecycle platform in the form of Atlassian

Jira (project tracking tool), FogBugz or Bugzilla (bug tracking tool).

QTP being a part of Quality Centre supports requirement traceability matrix. QTP integrates seamlessly with QC. Test management and mapping the manual testing process with automation becomes a lot easier with this integration

For CUIT and MTM we can provide all the ALM support Team Foundation Server provides. It supports work item tracking, source control or version control, build automation, various reports. The support is in-built; we do not have to do anything extra.

QC is still not a complete life cycle management tool. It does not provide support for efforts management, build management or support to different process templates. It supports only test management, bug management and requirement management.

CUIT is a winner here as it seamlessly integrates with Team Foundation Server (TFS). TFS in turn supports work item tracking, source or version control, requirements management, project management, build automation and various reports for monitoring.

• Monitor with Customized Reports

Test results can be made available with each tool. Coded UI Test supports all the reports supported by Team foundation Server as well as has the option of creating any custom report. The custom report can be created in any of the 3 ways, by using Report Project with Business Intelligence Development studio (BIDs), by using Microsoft Excel or by using Report Builder facility to create reports on the fly.

• Going beyond

Selenium being Open Source, has a lot of plugins. Selenium IDE has plug-ins for customization, for adding new functionality to API, changing existing functionality and so on.

QTP provides plug-ins for ActiveX controls, web application and VB objects. Other than these, plug-ins for objects like Microsoft .NET, multimedia plug-ins and Windows Mobile are also available. These QTP plugins are available at an additional cost.

Apart from hand coding complete CUIT, there is another feature available. CUITe Coded UI Test enhanced is a thin layer developed on top of Microsoft's Coded UI engine which helps reduce the code. It also increases readability and maintainability. It is very easy to install and will be referred with CUITe.dll in the project. CUITe provides simple object repository.

Each of the tools keeps on adding features and evolves as needed.

I have evaluated these tools from different angles and each has got its strength and weaknesses. You may choose a tool depending upon your need and the support the tool provides.

THE FINAL MATCHUP

The following table provides a bird's eye view for the categories and the tools' support for it.

Category	Selenium	QTP	Coded UI Test (CUIT)
Record and Playback	●	●	●
Ease of IDE and features with tool	◐	◐	●
Ease of execution	●	●	●
Languages Support	C#, Java, Perl, PHP, Python or Ruby	VB Script	C# and VB.NET
Support for Objects	○	○	●
Browser Support	Almost All	IE and Firefox	Some IE Versions
Support to various applications	Only Web	Almost All	Almost All
Integration with ALM	◐	◐	●

●	Completely Supported	◐	Partially Supported
○	Not Supported	◐	Feature available but not at par

CONCLUSION

To conclude, we did a quick overview of what is automation testing and when is it a good time is to start thinking about test automation in a software development cycle.

We looked at three popular automation tools -Selenium, CUIT and QTP and gauged their strengths and weaknesses. Final selection of tool is almost always based on budgeting and team strengths (tool familiarity), however for Web Application testing all three have compelling strengths. For desktop application testing, the choice gets reduced to two with Selenium dropping out. ■

Gouri Sohoni, is a Visual Studio ALM MVP and a Microsoft Certified Trainer since 2005. Check out her articles on TFS and VS ALM at bit.ly/dncm-auth-gsoh

INTERVIEW WITH JOHN PAPA

Dear Readers, in this episode of DNC Magazine, we talk to another industry stalwart, John Papa. John has worked with Microsoft technologies for more than 15 years, both on the outside as well as inside Microsoft and currently is sharing his wealth of knowledge by crafting courses for Pluralsight.

Hello John, welcome and thank you for taking some time out for the interview. We have been following your SPA and TypeScript exploits with great interest and it's awesome to have you in our Interview hot seat this month.

DNC: To start off with - a lot about your area of work can be traced by following your MSDN posts from way back in 1998 (almost 15 years of blogging, awesome!), but we want to hear it from you. Give us a bit of history, how did you get interested in software development? Walk us through initial days of John Papa as a developer?

JP: Like most software developers, I've changed the technologies and areas I've focused on, several times over the years. C++ was the first language that I really enjoyed. I was drawn to OOP and went looking for a job in that area. I did C++ till the job market shifted to a newer upstart: VB. At that point I had learned several languages and had fallen in mentoring roles at the companies I worked at.

I enjoyed teaching and mentoring, so I shifted to a role where I could be a tech-lead half time and trainer the other half. This was a scary jump for me since I was petrified of speaking in public. This is the same kid who was too chicken to try out for his High School plays! I took a few train-the-trainer courses and observed how others trained. This gave me confidence to give it a shot and I am glad I did. I became addicted to sharing technology and the rush of excitement when you see people "getting it" for the first time. It's a great feeling. At that point I knew I wanted to share technology with people and help inspire them to do more. So I started writing books, magazine articles and speaking at conferences.

DNC: Well, we sure are glad you made the jump and started sharing your enthusiasm. In our interview with Jon Galloway in the November issue, we got a glimpse into what an Evangelist role is like at Microsoft. As a former Sr. Technical Evangelist for Silverlight, tell us what it was like to evangelize a technology like Silverlight?

JP: Working at Microsoft was a lot of fun. I really enjoyed my 2 years there and if they'd move operations to Florida I would be there in a heartbeat. The Evangelist role was a perfect fit for me. What won me over was when my boss told me during the interview that I had the freedom to perform the role however I wanted, as long as it met with our goals. That freed me to try a bunch of news things in an effort to connect with the worldwide Silverlight developer community. We hosted training events, created demos to show of the features, worked with partners to help them create great software, and provided a great circle of communication between the product engineers, the customers and the community of developers. The great fun of it all was that I got to connect those groups which I hope helped make the product better.

DNC: Tell us more about your current role as a Regional Director. How is different from the role you held at Microsoft? Is it just being about inside MS and outside, or does it cover a different scope altogether?

JP: A Microsoft Regional Director is a funny title. I don't work for Microsoft. I don't represent my region. And I certainly am not directing anyone. Seriously though, my experience with the RD program has been great. I'm one of the newer RD's, who are a group of community influencers in various technology areas. While the exact number escapes me, I believe there are less than 150 of us worldwide. We chat with each other and provide feedback to Microsoft on what we see outside of Redmond.

> COULD WE SEE A VISUAL STUDIO FOR WINDOWS 8 RT?

DNC: Thanks for that insight about RDs. Coming to software development, do you think the often claimed distinction between 'Enterprise Development' and 'Non-Enterprise Development' (for a lack of better term) actually exists? If so, what do you think are the key differentiators?

JP: We hear that a lot. I think to a certain degree it's a myth. Software is software. However I think what developers think of when we hear enterprise development is often lots of screens, lots of data, big monolithic apps that run a part of a business. But these days the barriers between the two are a little fuzzier. We are seeing an increase in the number of enterprises who are building smaller apps that work together. Some of these are in app stores for the various platforms, while others are built for internal business app stores. The enterprise still needs the large apps, but the sands have begun to shift in a new and exciting direction where apps that do everything are now being replaced for modular and smaller apps that can be developed faster and integrate well with other apps. I think there are a lot of opportunities in that area.

DNC: Jumping on to Windows 8 Store apps, Microsoft did an excellent job of creating a wide palette of development options in - HTML5 and JS, C# or VB.NET and XAML, C++ and XAML or C++ and DirectX. Question is, do you think the Windows 8 Store (aka Metro Apps) application model is deep enough and ready for all types of application development? Especially with respect to guidance for complex apps like for example a Modern UI style MS Word or Visual Studio or even an Outlook/Mail app for that matter?

JP: I have been impressed with the Windows 8 application model and what it can do, especially on ARM devices. It's great for the types of apps that we have seen become very popular the past few years. I think it has the potential to become very good for the companies to use internally, too. Most companies are slower to adopt than consumers though, so that may take time and they have to do it right. Large companies don't like to swing and miss too many times before they start looking at other options.

The Office products are some of the best apps on Windows 8. OneNote in particular is fantastic. The mail client leaves a lot to be desired, in my opinion. I'm still hoping for a much better version of it. Even if it is half of what Outlook is, that would be good. You hit on an interesting point though about Visual Studio. Could we see a Visual Studio for Windows 8 RT? It would be a huge shift to think I could develop on that platform, but yes I do think it's possible. But wow, that would be quite an achievement.

DNC: Agree on the One Note comment and again when we compare the desktop version vs. the Store version we see some missing features in the store version, which are more to do with deciding how to do it best in Modern UI. Coming back to Web Development, the transition from Silverlight to HTML 5+JavaScript, is it a radical shift, assuming one is reasonably well versed with JavaScript as a language? And how much

of a learning curve is potentially involved if you have not used JavaScript before?

JP: Quite often JavaScript doesn't get the due it deserves. Developers tend to read a lot of books and get formal training on C# and other languages, but their JavaScript training has often been much smaller in comparison. Why is that? There are big differences in static vs. dynamic languages. Some are great and some not so fun. But it goes both ways. Some would argue that the dynamic nature of JavaScript makes it easier to do some things than in a static language. I don't buy into picking a winner. It benefits the developer if s/he can learn multiple languages and in today's age, knowing C# and JavaScript is very marketable.

JavaScript is not too difficult to learn. I recommend starting with a book that covers the basics, then move on to Douglas Crockford's "JavaScript: The Good Parts". His book covers a lot of things that are great about the language, and he also points out places he suggests you avoid. It really gets you thinking about how to code with JavaScript. Two other books I highly recommend are JavaScript Patterns by Stoyan Stefanov and High Performance JavaScript by Nicholas Zakas. The patterns book really opened my eyes up to how I could apply all of the guidance and patterns that I knew in other languages to JavaScript to create better and more efficient code. The performance book is a must read for those wondering why some conventions are used and for learning how to write efficient code in the browser.

Then there is HTML5 and CSS. HTML5 is pretty easy to pick up from a number of books. I have yet to find a good book on CSS, though. Most of my best experiences have come from reading web articles on CSS and from sites like caniuse.com

Of course, if you are not into books and you prefer a visual more interactive learning technique, you can watch the JavaScript courses at www.Pluralsight.com . There are some great JavaScript, CSS, and HTML courses in there on each of those topics. The next step would be to learn how to put them together to build robust apps, which you can see in my recent Single Page Apps course jpapa.me/spaps. It combines HTML, CSS and JavaScript along with a series of powerful JavaScript libraries to create an app.

DNC: Thanks for the references John, these will definitely serve as a good starting point for devs who are on the fence regarding JavaScript. What do you perceive from the attempts to 'update' JavaScript via wrappers like TypeScript? While TypeScript is keeping up with ECMAScript v6 proposal it also 'compiles down' to ' current' JavaScript. Do you think TypeScript is going to be the next JavaScript or is the 'compile down to the lowest common denominator', always going to stay?

JP: If you type JavaScript and copy it to the TypeScript editor, it works. TypeScript takes JavaScript and adds ECMAScript 6 features to it, plus a few other things like static typing. This is how it differs from Dart and CoffeeScript, which are not JavaScript. They compile to JavaScript. All 3 of these are great, but I still recommend learning JavaScript and then using TypeScript. It's always good to understand what the base language is doing for you.

DNC: There you go Devs, you heard John, and it's time to drop all inhibitions about JavaScript. What are some of the current or upcoming projects you are working on? Is there a scoop we can claim for our readers?

JP: I just wrapped up a TypeScript course that went live at Pluralsight. My next venture is create a SPA JumpStart course which walks the developer step by step through in creating a SPA. It is due out in March 2013. My first SPA course was intentionally a little more advanced, more of an intermediate level course that put a lot of pieces together to build a SPA. I wanted to first bust through the barrier that people have that makes them think you can't do powerful apps with JavaScript. The next logical step for me was to take a step back and now create a beginner's course that shows how to get started and how to do it one step at a time. I'm very excited about this one.

DNC: I am sure quite a few of our readers will look forward to its release. To wrap up, apart from Family and Software Development is there a hidden aspect of John Papa that we don't know about? How do you unwind - music, reading, and sports et al?

JP: I spend a lot of time with my family. As much as I can, in fact. I love playing sports and acoustic guitar, but it comes down to picking the few things I can do that really matter most to me, so we try to do as much as possible with the kids and each other outside of work time. We live close to Walt Disney World so it is very nice to be able to jump in the car and head over for an afternoon or evening as a family to unwind. ■

A KNOCKOUTJS **CHEAT SHEET**

We used Knockout in our Tic-Tac-Tor article earlier and while doing the article, we thought it would be good to share our notes regarding KO. The outcome is this cheat-sheet by *Sumit Maitra* that gives you a glance at what is Knockout, why to use it and how to use it. If you are a newcomer to KO or if you are juggling between multiple JS libraries, this cheat-sheet is a handy guide to get your KO karma flowing.

01 Knockout is a JavaScript Library (as opposed to Backbone.js which is a framework) that helps you improve the User Experience of your web application.

02 Knockout provides two way data-binding using a ViewModel and provides DOM Templating, it doesn't deal with sending data over to server or routing.

03 You use Knockout as a drop-in enhancement library to improve usability and user experience, whereas Framework like Backbone is used ground up in new applications (Single Page Apps is a good example).

04 You can create a KO ViewModel as a JavaScript object or as a Function (known as prototype). It's outside the jQuery document ready.

05 **ViewModel as a function**

```
var myViewModel = function() {
this.Email = "myemail@email.com";
this.Name = "Sumit";
this.LastName = "Maitra";
this.WebSite = "http://www.dotnetcurry.com"; }
```

06 **ViewModel as an object:** you can follow the JavaScript Object Notation –

```
var myViewModel = {
 Email: "myemail@email.com",
 Name: "Sumit",
 LastName: "Maitra"
}
```

07 **Observable Properties are functions:** When dealing with observable values you have to end the property with parenthesis like a function call, because computed values are actually functions that need to be evaluated on bind.

08 **Applying a ViewModel**

```
// In case the View Model is defined as a function
ko.applyBindings(new orderViewModel());
// In case the View Model is defined as a JSON Object
ko.applyBindings(orderViewModel);
```

Here ko is the global reference to Knockout that you get once you add Knockout Script reference in your page.

09 **Observable Properties:** KO has the concept of Observable properties. If you define a property as an observable, then DOM elements bound to it, will be updated as soon as the property changes.

e.g.

```
var myViewMode = {
 Email: ko.observable("myemail@gmail.com")
}
```

10 **Observable Arrays:** When you define an array of Json objects as Observable, KO refreshes DOM elements bound to it when items are added or removed from the Array.

```
var orderViewModels = function() {
 …
 // Binds to an empty array
 this.Address = ko.observableArray([]);
}
```

Note: When properties of an Item in the array changes then KO does not raise 'modified' events.

11 **Assign value to observables:** We saw how to declare observable Properties above. But when we have to assign a value to an observable in JavaScript we assign it as if we are calling a function, for example:

```
myViewModel.Email("myNewEmail@email.com");
```

12 **Simple Data Binding** to properties

```
<input data-bind="value : Email" />
```

Value of input element to the Email in the ViewModel.

13 **Binding to Computed Values:** Binding KO to a function gives you added flexibility of defining methods that do computation and return a computed value. KO can actually bind DOM elements to these methods as well.

For example if we wanted to bind First Name and Last Name and show it in a single DOM element we could do something like this:

On the View Model

```
var myViewModel = function() {
 this.Email = "myemail@email.com";
 this.Name = "Sumit";
 this.LastName = "Maitra";
 this.FullName = ko.computed(function () {
  return this.LastName() + ", " + this.Name();
 }, this);
}
```

Binding to DOM Element

```
<label data-bind= "text: FullName" />
```

14 Binding to an array of objects: KO can bind the DOM to an array in the view Model. We use KO's foreach syntax as follows

```
<table>
 <tbody data-bind= "foreach: Address">
  …
 </tbody>
</table>
```

You can do the same for an Ordered <ol> List or an Unordered List <ul> too. Once you have done the foreach binding, KO treats anything DOM element inside as a part of a template that's repeated as many times as the number of elements items in the list to which it is bound.

15 Binding elements of an array: Once you have bound an Array to a DOM element KO gives you each element in the array to bind against. So an Address object may be bound as follows:

```
<table>
 <tbody data-bind= "foreach: Address">
  <tr> <td>
     Street: <label data-bind: "text: Street" />
     #: <label data-bind: "text: Number" />
     City: <label data-bind: "text: City" />
     State: <label data-bind: "text: State" />
   </td> </tr>
 </tbody>
</table>
```

16 Binding to properties other than text and value: Now let's say we want to bind the WebSite element of the ViewModel an anchor tag

```
<a data-bind="attr : {href : WebSite}">
 <span data-bind= "text: Name"</a>
```

What we are doing here is using Knockout's attribute binding technique to bind the 'href' attribute to the URL in the WebSite property of the View Model. Using attribute binding we can bind to any HTML attribute we want to.

17 **Getting KO Context in jQuery:** Now let's say we want to have a Delete button for each address. The following markup will add a button

```
<table>
 <tbody class="addressList" data-bind= "foreach:
   Address">
  <tr><td>
     Address: <label data-bind: "text: $parent.
     AddressString(Street, Number, City)" />
   </td>  <td>
    <button class= "addressDeleter" ></button>
   </td> </tr>
 </tbody></table>
```

Now to handle the click even using jQuery. Since the button is generated on KO binding we cannot use the normal click handler assignment of jQuery. Instead we use the parent container and assign a delegate as follows

```
$("#addressList").delegate(".noteDeleter", "click",
function() {
 var address = ko.dataFor(this);
 // send the address to the server and delete it
});
```

As we can see above the ko.dataFor(this) helper method in KO returns the object that was bound to that particular row of data. So it returns an Address object. If you need the entire ViewModel you can use ko.contextFor(this). ■

DESIGNING & DEVELOPING APPS FOR SHAREPOINT 2013

Pravinkumar Dabade introduces the new SharePoint 2013's App Model and highlights how this is a new opportunity for App Developers to build custom SharePoint and Office solutions.

A SharePoint App is a solution or a web application built using web technologies like HTML 5, JavaScript, OAuth 2.0 etc. but not necessarily installed on SharePoint Host servers. This enables full trust apps that do not have a big footprint on SharePoint server itself. This way, the App Model works around the issues with lack of 'full trust' in Sandboxed Apps of SharePoint 2010 and the server footprint issue for custom, 'full trust' SharePoint apps built using SharePoint APIs.

SharePoint 2013 Apps support the following hosting models.

1. Provider-Hosted Model.
2. Autohosted Model.
3. SharePoint Hosted Model.

We'll learn more about them in a bit.

WHY DO WE NEED APPS FOR SHAREPOINT

From a user's perspective, apps for SharePoint extend, customize and often simplify the default functionalities provided by SharePoint in a trusted, low deployment-footprint environment.

From a developer's perspective, the App framework helps by providing a rich set tools to be used in conjunction with familiar web technologies (not limited to the Microsoft Stack) to build apps, that can be used with SharePoint. If you are a web developer and know how to build web apps, then you can easily build Apps for SharePoint by using your set of technologies like HTML 5, JavaScript, .NET, PHP etc. In essence, the App model lowers the bar of entry for a vast majority of web developers.

THE SHAREPOINT APP HOSTING MODELS

Microsoft SharePoint 2013 Preview has introduced two new hosting categories in which you can host your apps.

Cloud Hosted Models

Under Cloud hosted category, there are two options which you can use for hosting your apps –

1. Provider-Hosted Model – In Provider-Hosted Model, developer is free to deploy the Non-SharePoint components in their own hardware or in a cloud account. But in some cases, it can be hosted by your SharePoint site – On Premise or SharePoint Online.

2. Autohosted – In this model, Apps are hosted on your host web or SharePoint online with components automatically installed in Windows Azure Web Site account.

SharePoint-Hosted Model

This model is used only for the apps which includes SharePoint components and hosted on your own SharePoint Farm or SharePoint Online tenancy.

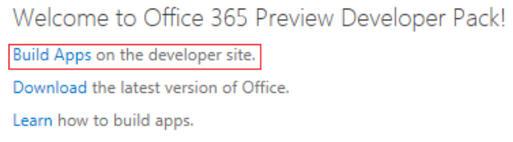
BUILDING AN AUTOHOSTED SHAREPOINT APP

Today, we are going to see how we can build and deploy an Auto Hosted SharePoint App. Microsoft via Office365 has made it very easy.

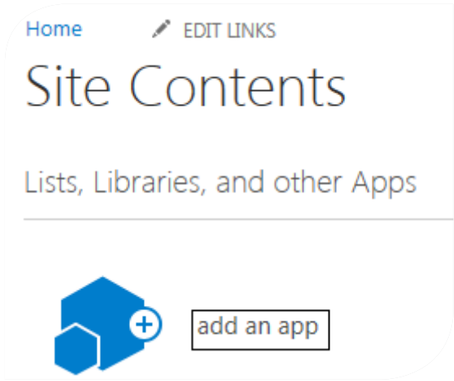
The first step is to register/signup for an "Office 365 Developer Site". To register, click on the following link which will guide you to Sign up for an Office 365 Developer Site.

http://msdn.microsoft.com/en-us/library/fp179924.aspx

Once your site is ready, click on 'Site Contents' > 'Build Apps on the developer Site' as shown here



The wizard will take you to your SharePoint site. Once you are at the SharePoint site, click on 'Site Contents' from the left navigation pane. Then click on 'add an app' :

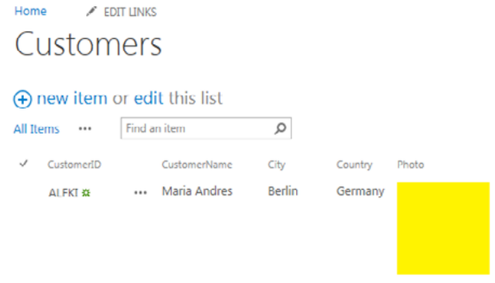


Now we will add a Custom List and a Picture Library. To create a custom list, click on "Custom List" and give a name to our list as "Customers".

Your custom list is now ready. Add the following columns in the list

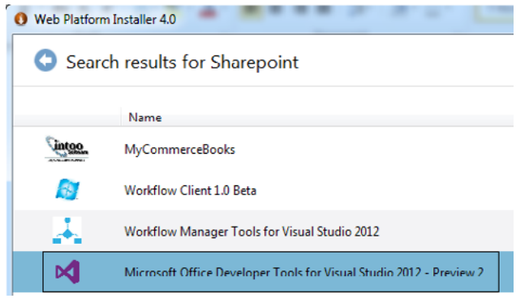
Column Name	Type
CustomerID	Rename the "Title" column of the custom list.
CustomerName	Single line of text.
City	Single line of text.
Country	Single line of text.
Photo	Hyperlink or Picture.

Now let's create a 'Picture Library' and upload some images for our customers. After that, you can add some customers with their photos. The customers list should look like the following (we are using colored boxes as placeholders for the images)



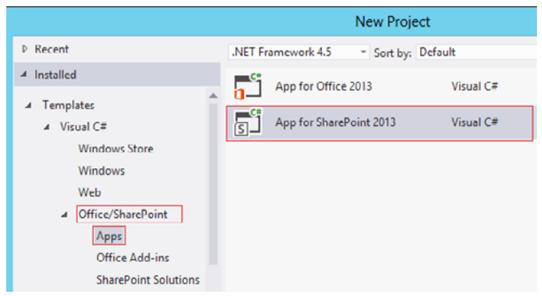
For this demonstration, I am using "Windows 7" and Visual Studio 2012. If you want to develop Apps for Office or SharePoint, you will have to download the tools for "Microsoft Office Developer Tools For Visual Studio 2012 – Preview 2". From the start menu, click on "Microsoft Web Platform Installer". In a Search box type SharePoint and search. You will find "Microsoft Office Developer Tools for Visual Studio 2012 – Preview 2". Click on "Add" button and then the "Install" button as

shown here



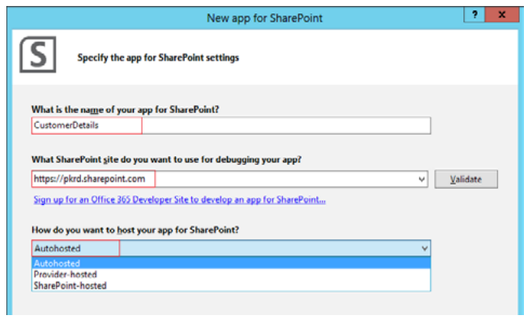
Another alternative to Visual Studio is the new Web Tools available under the product code 'Napa'. However Napa can be used only for SharePoint Hosted Apps. Since we are building Auto Hosted apps, we will continue with the traditional Visual Studio tooling.

Start Visual Studio 2012 and create an "App for SharePoint 2013" project as shown here –

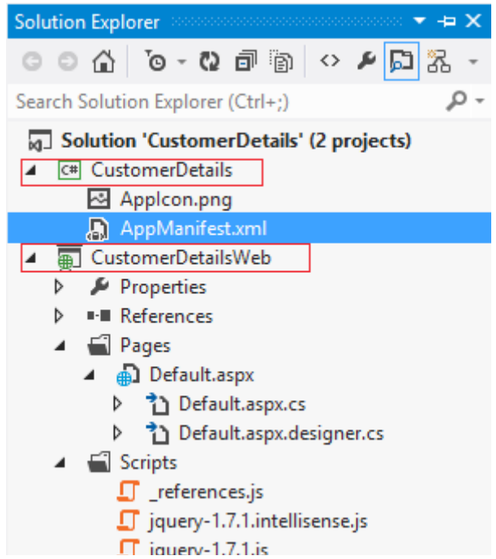


Name the project as 'CustomerDetailsSPSApp'. Click on the "OK" button to create a project. In the next window, the wizard will ask you for the following information:

1. Name of the SharePoint App. (Keep it default).
2. Provide the Office 365 SharePoint site URL to be used for debugging.
3. Hosting Model – Choose "Autohosted" model.



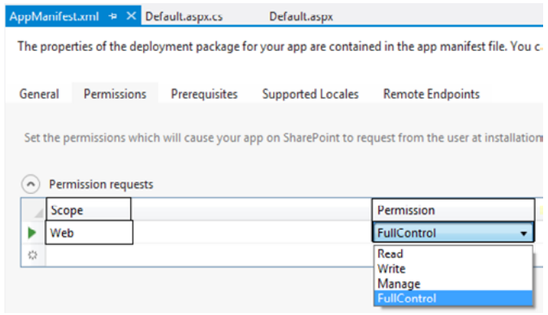
Once your SharePoint App is ready, you will find that our Solution Explorer has two projects
1. SharePoint Project.
2. ASP.NET web application.



Now let's open the "AppManifest.xml" file. Right click the "AppManifest.xml" file and click on "View Code" context menu. As you can see, it's a simple xml describing the Auto-Hosting configuration.

The start page URL has been set to the remote URL. It also has the Title of our App.

Close the file and double click on "AppManifest.xml" file in Solution Explorer to open a Design view of our AppManifest.xml file. Click on "Permissions" tab and then set the "Scope" to "Web" and Permission to Read/Write/Full Control as shown here



When you work with Autohosted SharePoint Apps, you can interact with SharePoint Lists and Libraries using REST or SharePoint Client APIs. In this demonstration, we will use *SharePoint Client Object Model*.

We will first layout our Default.aspx page to display the data which we will be fetching from our Custom List "Customers". The page for displaying customer information is rather simple for the demo. We have a DataList with an ItemTemplate that has an image control and three labels bound to the Photo, CustomerID, CustomerName, City, Country fields. These fields, as we will see shortly, are coming from the SharePoint list we created earlier.

Once the Default.aspx page is laid out, now it's time to write the code and fetch the data from our "Customers" list and display the same in our Repeater control.

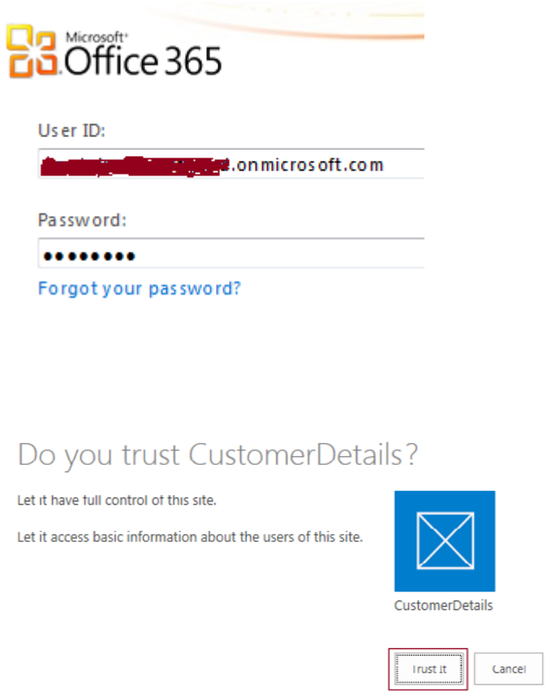
In the code behind file of Default.aspx, we update the Page_Load event as follows:

```
protected void Page_Load(object sender, EventArgs e)
{
 TokenHelper.TrustAllCertificates();
 var contextToken =
 TokenHelper.GetContextTokenFromRequest(Page.Request);

 var hostWeb = Page.Request["SPHostUrl"];

 using (var clientContext =
  TokenHelper.GetClientContextWithContextToken(
   hostWeb, contextToken,
   Request.Url.Authority))
   {
    List customersList =
     clientContext.Web.Lists.GetByTitle("Customers");

CamlQuery query = new CamlQuery();
ListItemCollection items =
 customersList.GetItems(query);

clientContext.Load(items);
clientContext.ExecuteQuery();
DataTable customerDataTable = new DataTable();
customerDataTable.Columns.Add("CustomerID");
customerDataTable.Columns.Add("CustomerName");
customerDataTable.Columns.Add("City");
customerDataTable.Columns.Add("Country");
customerDataTable.Columns.Add("Photo");
foreach (ListItem item in items)
{
 DataRow newRow = customerDataTable.NewRow();
 newRow[0] = item.FieldValues["Title"].ToString();
 newRow[1] = item.FieldValues["CustomerName"].
  ToString();
 newRow[2] = item.FieldValues["City"].ToString();
 newRow[3] = item.FieldValues["Country"].ToString();
 newRow[4] = ((FieldUrlValue)
              (item["Photo"])).Url.ToString();
 customerDataTable.Rows.Add(newRow);
}
DataList1.DataSource = customerDataTable;
DataList1.DataBind();
 }
}
```

The code is rather simple. We are creating a Client Context for the SharePoint Host using the SPHostUrl. Once we have the context, we get an instance of the SharePoint List in which the Customer data is stored and then load it into a DataTable. Once all the data is loaded, we bind it to the DataRepeater we used earlier.

Point to note, the client objects list ListItemCollection and ListItem come from the Microsoft.SharePoint.Client Namespace. That's all is there for this basic App demo. Let's deploy it in our Office 365 SharePoint site. To deploy the app, right click on the solution and click on "Deploy" context menu.

Once your deployment is successful, it will show you your SharePoint site. In case you have not signed in, it will take you to Sign In page. Login using the Office 365 Developer site login which you created when you signed up for Office 365 earlier.



Once you have signed in, the wizard takes you to the page where it will ask you "Do You trust [App Name]". Click on "Trust It"

Now you can see your application on the SharePoint site.

WHAT DO APPS FOR OFFICE AND SHAREPOINT MEAN FOR DEVELOPERS?

Apps for SharePoint have existed in form of plugins, custom site templates and custom applications for a while now. But they always had a high bar for entry and acceptance in the enterprise due to various factors that can be lumped as 'Server footprint' or how much they affected a SharePoint server farm. With the new App model, developers building productivity apps using Office 365 or SharePoint as a platform will find it easier to convince end users to adopt these apps, thanks to the various deployment models. Moreover, with the introduction of the App Store, there is an exciting opportunity to make apps reach a wider audience which in turn is likely to result in better revenue. Also an App's availability on the App Store will imply basic amount of compliance as per store app validation policies http://msdn.microsoft.com/en-us/library/office/apps/jj220035(v=office.15)

The App Store model provides developers with a Seller's Dashboard (https://sellerdashboard.microsoft.com) where they can register, maintain a developer profile, submit their apps and sell them at the price they desire.

The Dashboard also shows you how your app is doing by viewing the page views, downloads or purchases. In case you want to rollout the updates, the updates will be automatically notified to the existing customers.

SUMMARY

In this article, we have seen how to build a very simple SharePoint App and deploy it to the Office 365 SharePoint Site. This deployment model is referred to as the Auto Hosted model where the application we built gets deployed in Azure behind the scenes. The beauty of apps is that we are using SharePoint Client Object model and as a result are not depending on Application Server's resources, except when we are querying for data.

This article showcases only a small part of the App Development gamut that has been introduced with SharePoint 2013 and Office 365. So watch out for a lot more action in this space. ■

The entire source code for this article can be downloaded from GitHub at **bit.ly/dncm5-sp13**

Pravinkumar works as a freelance corporate trainer and consultant on Microsoft Technologies. He is also passionate about technologies like SharePoint, ASP.NET, WCF. Pravin enjoys reading and writing technical articles. You can follow Pravinkumar on Twitter @pravindotnet. and read his articles at **bit.ly/pravindnc**

UNDER THE HOOD - WHAT'S NEW IN RAVENDB 2.0

Gregor Suttie walks us through this magical Document Database written in .NET by the folks at Hibernating Rhinos. He quickly dives into the gems of the 2.0 release and highlights the new features and functionality that makes RavenDB better than ever before.

RavenDB is an open-source, second generation, NoSQL, document database which can be used to build high-performing, scalable applications, quickly and easily. It's available under AGPL license that permits free use for personal and Open Source projects. For commercial/ production use, a Commercial license needs to be acquired. RavenHQ is the cloud hosted solution for RavenDB.

Editor's Note: To learn more about RavenDB you can refer to some of our older tutorials here and here.

This article discusses the new features in RavenDB 2.0 which was released in the beginning of January 2013. This latest release contains a number of performance improvements and some breaking changes to the previous versions.

WHERE CAN I GET RAVENDB?

There are multiple ways in which you can get started with RavenDB.

Download

The latest stable version which was released on February 8, 2013 was build number 2261. You can download this version from http://ravendb.net/download. It's a 40Mb download (zip archive) and the easiest way to get the build at a central location. For the client, it's suggested you use Nuget, but if you are unable to use Nuget, add references from the above archive extract.

Install via Nuget

In Visual Studio, you can add RavenDB to your projects using the following Nuget command to install the client

```
PM> Install-Package RavenDB.Client -Version 2.0.2261
```

Next install the Server using the following command

```
PM> Install-Package RavenDB.Server -Version 2.0.2261
```

Once the server is installed (you will notice it doesn't get added to the package manifest because it's not supposed to be used from inside Visual Studio), open explorer and navigate to the packages\RavenDB.Server.2.0.2230\tools folder of your project and run Raven.Server.exe. The first time, you will receive a UAC command prompt asking for permission. Provide permissions as required.

The Source

RavenDB being open source, you can browse the entire codebase including unit tests here https://github.com/ravendb/ravendb

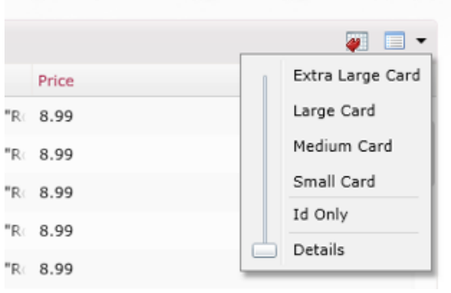
What's New in RavenDB 2.0

STUDIO IMPROVEMENTS

RavenDB has a Silverlight frontend for managing operational aspects of the database. It is referred to as the 'RavenDB Management Studio', and it underwent significant improvements for the 2.0 release. It got a host of new features as well. We start off by looking at the improvements to the Studio which are very important from the operational aspects of RavenDB, traditionally considered to be its weakness when compared to other RDBMS.

Data Management Improvements

Layout of the studio has been improved making it easier to select documents. You can control the view of what you see, with more options like in the screen shot here.



Toggle indexing

This feature allows you to actually turn off indexing on the database you're working with. If an index is currently being updated, it will wait for the current batch to complete.

CSV Import/Export

RavenDB has better support for importing and exporting your data to and from a csv file, importing a csv will insert your data into a collection and each column will be treated as a property.

Other Settings

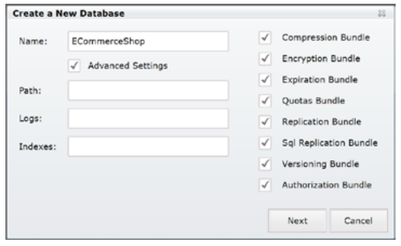
You can also edit the settings for the database you're working with and do things like disable the database, as well as change the settings for periodic backups to Amazon Cloud, and change the settings on the bundles you selected.

Database Setup

Bundle Management

Bundles in Raven are additional features on top of the core product and are used to extend Raven. There are a host of changes with respect to bundles in RavenDB 2.0.

- To start off with, you can pick bundles while creating the Database itself.



-

STUDIO IMPROVEMENTS

Silverlight Frontend: 'RavenDB Management Studio' has undergone significant improvements with host of new features.

PATCHING

Quick Patches: Now we can easily patch a document, collection of documents or even an index.

DOCUMENT EDITING

Easing Day to Day Operations: You can do much more, much quicker with new features like Document Outlining and Document Intellisense.

CORE DB CHANGES

Engine improvements: The Changes API, Bulk Insert API & Index Related Documents enhance the Engine features.

- The 'Create New Database' wizard is a nice addition that you can use to create a brand new database. The wizard gives you options to specify which bundles you would like to add to the new database.

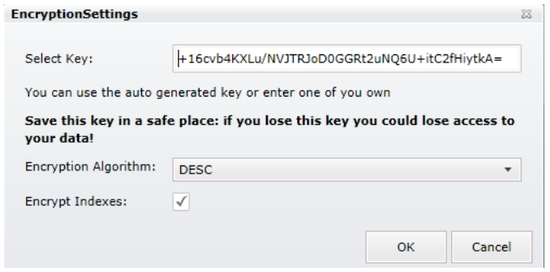
New Bundles

The following bundles are new for version 2.0.

- Encryption bundle – this allows us to encrypt index and all of the data which is a welcome addition.
- Periodic Backup bundle – this allows you to periodically back your Raven database to Amazon Vault/Amazon S3 every x minutes which is nice.

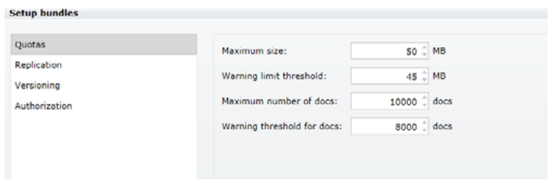
Other Setup Configurations

We can also specify the path to the database, the location of the logs and the location where the indexes are stored. The options you see in the wizard depend on the bundles you choose for your new database. Selecting the Encryption bundle for example will bring up the following screen



Here we can change the encryption algorithm used and select whether or not we wish to encrypt the indexes which will be created later on.

Next we see the setup for bundles where we can change various details such as the maximum number of documents, the max size of the database, as well as on the other tabs where we can configure other included bundles.



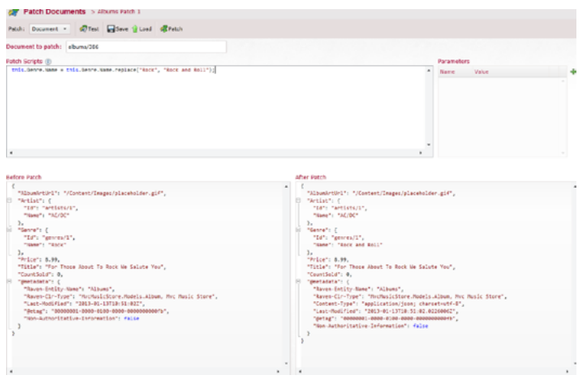
PATCHING

This feature allows us to patch documents quickly, so if you need to make a change to a number of documents on the server, you can write a patch, test it out on the server then apply it.

Now we can easily patch a document, collection of documents or even an index. Previously we had to write code to do this in say a console application, but now we can modify documents in a collection almost instantly by using the Raven Studio. Here's an example.

• Create a brand new blank database, once you've created a test database click on it and then locate the Task menu option.
• Click on Create Sample Data, this will generate some test data we can use to test patching documents, collections and Index with.
• If we navigate to Collections and take a look at Albums, we can see that some Albums have the genre 'Rock'. Let's change that to 'Rock and Roll'

Click on the Patch option and then path a collection from the drop down and select Albums. Within Patch Script, add the following



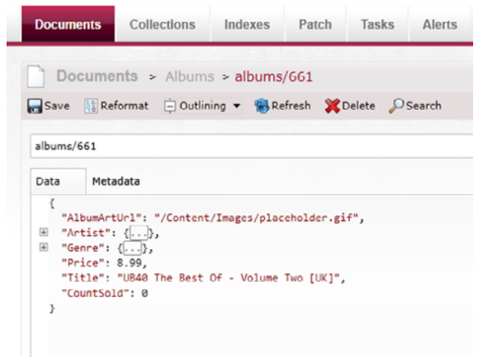
• Here within the studio, we have the ability to test a new patch on a single document. In the screen shot above, I changed the album genre 'Rock' to 'Rock and Roll'.
• Before you make the change, you can test it out where you can see the Before Patch and After Patch sections in the screen shot above.
• The patch script is using JavaScript so we have access to JavaScript methods like .replace() that I am using. We also get Intellisense which is a very nice and handy feature in my opinion.
• Once happy with the changes, you can choose to run the patch against the entire collection and all documents will be updated.
• You can also save and then re-load the Patches you have created at a later date. Patches are actually saved as documents and you can see them listed in the studio like other collections.
• All in all, patching has been improved immensely and makes multiple document changes far easier and quicker.

DOCUMENT EDITING

Day to day operations and general use is greatly improved; you can do much more, much quicker. For example, deleting a number of documents at a time is so much easier, navigating through documents within a collection has been added, as well as searching through your document too.

Document Outlining

A nice little addition is outlining your document as seen here:-



Document Intellisense/AutoComplete

While initiating a search the Studio provides an auto-complete list that helps you quickly search through documents:

These simple usability improvements make the management studio slicker to use, which is always what a developer desires.

CORE DATABASE CHANGES

After a review of the changes in RavenDB Management Studio, let's look at the changes to the core Database Engine.

The Changes API

The idea behind the new Changes API is that you now have the ability to subscribe to events from the server, on the client. This means that we can get Push notifications from the Raven server without having to do any kind of polling.

This opens up a whole new world of possibilities for us as developers using Raven – which means we can subscribe to changes in documents, or collections, and even Indexes, which in turn means we can write code to inform users of these changes.

So if you have a need to monitor a document being changed on the server, you can write some code :-

```
/// <summary>
/// Write to the console if an Album document changes.
/// </summary>
/// <param name="documentStore"></param>
private static void CheckForAlbumChanges(DocumentStore documentStore)
{
 documentStore.Changes().ForDocumentsStartingWith("Albums/")
  .Subscribe(change =>
  {
   if (change.Type == DocumentChangeTypes.Put)
   {
    Console.WriteLine("An Album has changed on the server!!");
   }
  });
  Console.WriteLine("Running :)");
  Console.Read();
}
```

There is no doubt that this feature will be exceptionally useful and I am sure users will come up with a whole number of ways to use it within their own applications.

Bulk Insert API

In the previous version of Raven, we had to use the FOR loop and send a set of updates to Raven. The new version now adds parallelism to the operation and is now fully streamed between the client and the server.

With the code examples we have, we can see just how easy it to create 1000 new Albums in our test database:

```
private static void DoBulkInsert(DocumentStore
documentStore)
{
 using (BulkInsertOperation bulkInsert = documentStore.
BulkInsert())
 {
  for (int i = 0; i < 1000; i++)
  {
   bulkInsert.Store(new Album
   {
    Title = "Title #" + i,
    Price = 5.99
   });
  }
 }
}
```

This is a very fast, simple way to create a large number of documents, all done in only one request to the server. Essentially, in the above loop, the BulkInsertOperation instance queues up the new Albums being thrown at it, and then posts to the server in bulk. This results in better performance as compared to doing inserts in a loop. Given that speed is the key to this operation, there are a set of limitations with the API

- We have to provide the id at the client side.
- It can't take part in DTC transactions.
- Put triggers get executed, but the AfterCommit are not executed.
- It bypasses the indexing memory pre fetching layer.
- Changes() will not be raised for documents inserted using bulk-insert.
- There isn't a single transaction for the entire operation, rather, this is done in batches and each batch is transactional on its own.

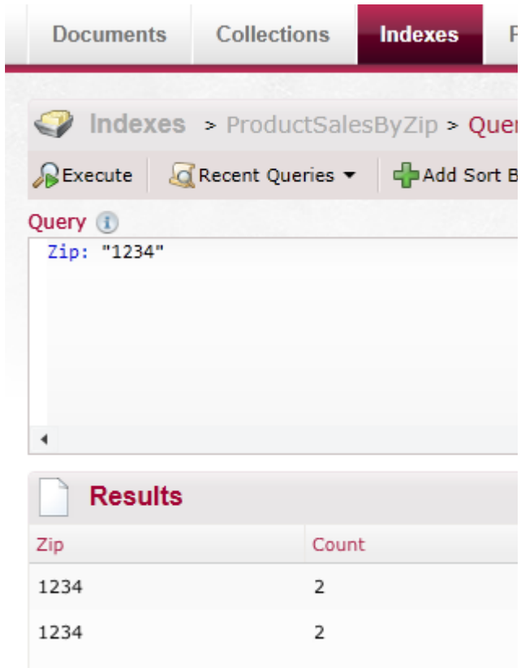
For more information refer to Ayende's post here.

Index Related Documents

This is another really nice new feature added to RavenDB and I will show you an example of it in code to give you an idea of what it does, but first let me try to explain it.

We have a collection of Products, Customers and Orders and we want to be able to query these related documents to work out let's say how many products has a customer bought by their zip code.

```
public class ProductSalesByZip :
 AbstractIndexCreationTask<Order, ProductSalesByZip.
Result>
{
 public class Result
 {
  public string Zip { get; set; }
  public string ProductId { get; set; }
  public int Count { get; set; }
 }

 public ProductSalesByZip()
 {
  Map = orders =>
   from order in orders
   let zip = LoadDocument<Customer>(order.CustomerId).
             ZipCode
     from p in order.ProductIds
      select new
      {
       Zip = zip,
       ProductId = p,
       Count = 1
      };
  Reduce = results =>
   from result in results
   group result by new { result.Zip, result.ProductId
}
    into g
     select new
     {
      g.Key.Zip,
      g.Key.ProductId,
      Count = g.Sum(x => x.Count)
     };
 }
}
```



Once we have created the Index using the code shown, we can query on Zip, ProductId and Count like in the screen shot above (you could do this before in RavenDB but it would mean a far more complex Map/Reduce).

OTHER IMPROVEMENTS

A larger number of other improvements have been made over the previous release and these include the following:

• **Better IN Query Support**– previously if you had say a list of product ids and you wanted to query products using the list of id's using the IN query it would complain if the list of ids was too large. The new version of Raven has dedicated support for this built in now which makes it more efficient and easy to use.

• **Indexing Performance** – Improvements have been made to indexing so that more cores are used to do the job and therefore indexing is achieved much faster and even with large amounts of data stale indexes are stale for a lot shorter period and so the indexing latency is far lower.

• **Debugging** – Raven now exposes more hooks for the developer to use to debug how indexes are built, inspect the steps of a map/reduce as well as looking at index times and how the IO process for loading a document works.

• **Improved RavenDB Replication** – The replication bundle within RavenDB has been improved to show conflict resolution in the RavenDB studio, conflicts now appear as separate documents making it easier to work out the conflict and resolve them easily and quickly.

SUMMARY

RavenDB is a superb open source product which before the latest release was an already pretty good. The latest release has improved RavenDB to make it faster, more reliable, with more support options, better indexing, a very much improved management studio and a lot more. You can take a look at Ayende's post – What is up with RavenDB 2.0 for some more details. There is no better time to go take RavenDB for a spin – you will be impressed just how easy it is to use and get up to speed with.

DEMO APPLICATION

There is a simple console application which demonstrates the Changes API, the Bulk Insert API and the Indexing Related Documents feature which helps with searching. ■

The code can be downloaded from the Git repo **bit.ly/dncm5-ravendb2**

Gregor Suttie is a developer who has been working on Microsoft technologies for the past 14 years, he is 35 and from near Glasgow, Scotland. You can Follow him on twitter @gsuttie and read his articles at bit.ly/z8oUjM

Creating a Custom Unobtrusive jQuery Validator for ASP.NET MVC 4

> Client side validation should always be backed by server side validation

Sumit Maitra gets down and dirty with the internals of building a custom jQuery validator that helps compare length of a field to a value that is obtained from the form itself instead of being a fixed value defined in a data annotation.

The ASP.NET MVC Framework comes with a set of validators that can cover almost all simple 1:1 validation requirements. By 1:1 I mean, if you have an input element that needs validation, apply the appropriate validator (length, required, max, min etc.), and you are done.

But often life is not 1:1, what then? How would you do custom validation? I had to look at multiple places to build a custom validator, most places had what to do, but didn't explain the 'why' properly. Today I will try to explain what we are doing and why we are doing it.

THE USE CASE

Let's take a scenario where I am maintaining addresses in my system. These can be international addresses with each having different length of Postal Code. For example in United States (Postal Code referred to as ZIP) is 5 characters in length whereas in India it's 6 characters (both are Numeric but that's not a topic of discussion). So if a user selects a country, we want the postal code validation to kick in and validate the length provided correctly based on the country selected. This is a reasonable validation that you would normally do at the server once the data is posted to you. But imagine this, you have already sent the list of countries to the client, you can potentially have the length for each country also at the client. So you can do this at the client itself and save some server and database round trips. Not to mention a vastly improved user experience.

With the use case laid out, let's do it the MVC way and build a custom validator that will hook into the existing validation framework and work with other validators on screen.

Before I go any further, let me warn you, in custom validator terms this is a fairly complex one. In fact anything that requires you to check or compare against another field in the View, is not going to be pretty when it comes to creating validators. However, the benefit of having the entire validation process consistent by using validators instead of hacking up temporary JavaScript solutions, is reason enough to build a validator, apart from the fact that it gives you a much more complete and well-rounded solution.

COMPONENTS OF A JQUERY CUSTOM VALIDATOR

Let me start by saying the jQuery validator framework is independent of MVC and if you are not using MVC, only the client side script for the validator will do. But you have to initiate the validation process yourself. Today we will use custom validators the way they are used in MVC framework which is to say, they have a server-side fallback also.

With that out of the way, here are the major components of the validator:

1. A class that inherits from ValidationAttribute.
2. A class that inherits from ModelClientValidationRule.
3. Snippet of JavaScript to register with the validation framework and do the validation.
4. Other changes in markup to ensure you have the required data to validate against.
5. It is suggested you have a ViewModel or at least you don't deal with the Database entity directly.

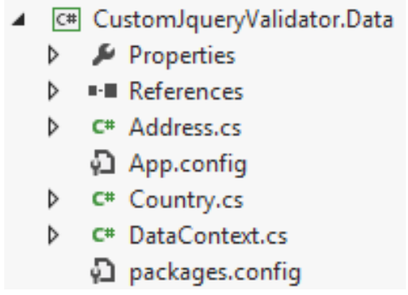
At first look this looks daunting. It is not with respect to the amount of code that you have to write (mostly 10-20 lines on the server per class and the JavaScript to do the validation).

THE PROJECT SETUP

I'll quickly walk through the project setup. This may vary slightly in your case. Bottom line is you should have a Domain model or a View Model handy to stuff in a few extra properties that we'll need.

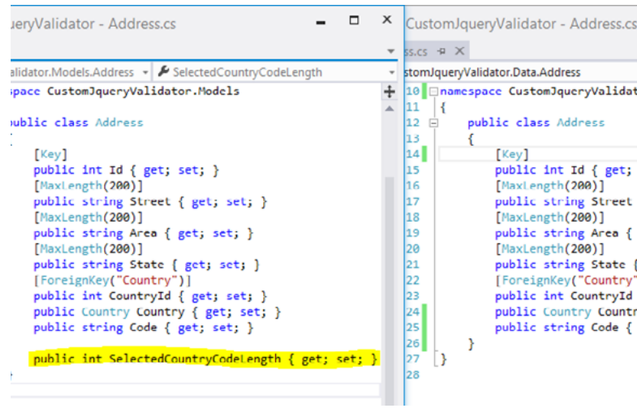
For sake of simplicity I have split up the code into two layers View and Data and I am not using IoC today.

There are two projects in my Solution CustomJqueryValidator and CustomJqueryValidator.Data. The CustomJqueryValidator.Data project has the data entities as well as the DBContext (in the DataContext class).



The CustomJqueryValidator project has the view model for the Account Class.

The ViewModel entity has an additional property called SelectedCountryCodeLength. We will shortly see how we use it



The Controller and Related Views

I've created two controllers, one for Countries and one for Addresses. To quickly generate the views, I use the classes in the Data layer. Once done, the Country controller stays the same. However I have modified the Create.cshtml and the AddressController's Create Action method to use the ViewModel Address instead of the Data.Address.

Address\Create.cshtml

Before we start making the changes, the Create.cshtml is as follows

```
@model CustomJqueryValidator.Models.Address
@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>Address</legend>

… Removed for brevity …
```

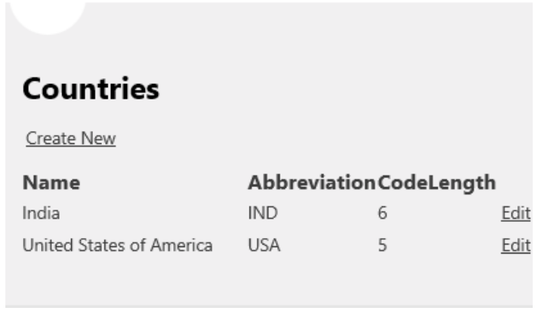
```
<div class="editor-label">
   @Html.LabelFor(model => model.CountryId, "Country")
        </div>
        <div class="editor-field">
            @Html.DropDownList("CountryId", String.
                                                Empty)
            @Html.ValidationMessageFor(model =>
                                model.CountryId)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Code)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Code)
            @Html.ValidationMessageFor(model =>
                                        model.Code)
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

AddressController

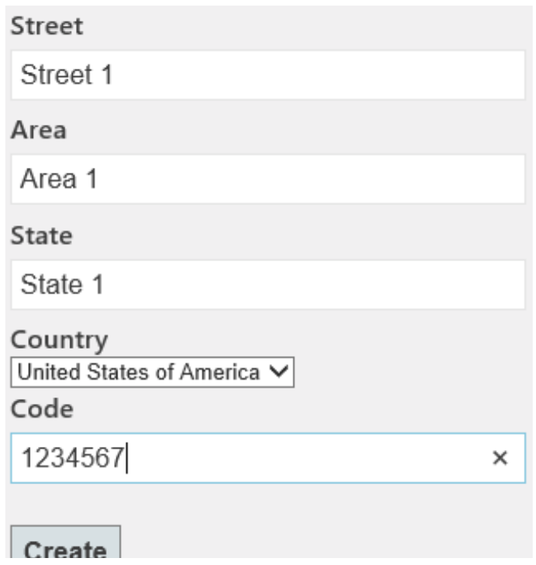
The controller is slightly modified so that it converts the data entity to a view entity before passing it back to the view. Also the view sends back a view entity that is mapped to the correct data entity before create is called on the Database.

```
//
// GET: /Address/Create

public ActionResult Create()
{
    ViewBag.CountryId = new SelectList(db.Countries, "Id", "Name");
    return View();
}

//
// POST: /Address/Create

[HttpPost]
public ActionResult Create(CustomJqueryValidator.Models.Address address)
{
    if (ModelState.IsValid)
    {
        Address addr = new Address
        {
            Area = address.Area,
            Code = address.Code,
            CountryId = address.CountryId,
            State = address.State,
            Street = address.Street
        };
        db.Addresses.Add(addr);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    ViewBag.CountryId = new SelectList(db.Countries, "Id", "Name", address.CountryId);
    return View(address);
}
```

With these changes in place, our application works fine.
We have created two Countries and as we can see below, we've defined the length of Postal Code for each (6 for India and 5 for United States).



When we try to create a new Address, it works fine with respect to showing the Countries appropriately, however it has no way to use the 'CodeLength' property to validate the length of the Postal Code.



> "US is the only country in the world as far as I know that has dual lengths for their Postal Code/Zip - 5 and 9

So it saves any value as long as it fits in the backing store. As we can see here, the above code has been saved in the database successfully. From a database operation perspective, it's valid because the Postal Code needs to accommodate code for all countries and hence will have a length of 15-16. But we have to enforce the correct validation here.

Updating View to load Validation Data

Before we continue building the validator, we have to ensure that we have all the data required to do the validation on the client.

If we do a view source of the Create.cshtml, we'll see that the list of countries have been transformed into Option elements. The option element has a Value attribute that corresponds to the Id of the Country entity and as a label it shows the name of the country. We don't have the length of the postal code anywhere in our HTML. If we don't have it in the DOM, how do we verify against it?

Using HTML5 data- attributes

HTML5 now allows you to have attributes starting with data- that are treated as markup related meta information. If you look closely at the source of Address > Create view, you'll see existing Validators already uses a few such elements like data-val-msg-for etc.

Well if jQuery validators can use them, why can't we!

So in our cshtml, we will replace the existing 'DropDownListFor' helper with the following code

```
<div class="editor-field">
@{
  if (ViewBag.CountryId != null)
  {
   List<Country> countries = (List<Country>)
      ViewBag.CountryId;
   <select id="CountryId" name="CountryId">
   @foreach (Country country in countries)
   {
    <option value="@country.Id"
     data-length="@country.CodeLength">@country.
     Name
    </option>
   }
   </select>
  }
 }
 @Html.ValidationMessageFor(model => model.CountryId)
</div>
```

The custom code is simple. We have mixed server-side code with Razor syntax to pick the list of countries from the ViewBag. CountryId. Next we are creating a Select with the id='CountryId' and then looping through the list of countries and generating option elements with the required 'data-length' attribute assigned the value of the permitted length of the Postal Code.

We make a slight change in the controller to assign the List of Country object instead of a SelectList to the ViewBag.CountryId.

```
public ActionResult Create()
{
 ViewBag.CountryId = db.Countries.ToList<Country>();
           //new SelectList(db.Countries, "Id", "Name");
 return View();
}
```

If we run the application and view source of the Create page, we'll see the option elements have the new data-length attribute associated with the correct values for each country.

```
<div class="editor-field">
 <select id="CountryId" name="CountryId">
  <option value="1" data-length="6">India</option>
   <option value="2" data-length="5">United States of
America</option>
 </select>
  <span class="field-validation-valid" data-valmsg-
for="CountryId" data-valmsg-replace="true"></span>
</div>
```

Now that we have the data on the client side, let's start the process to build a validator to use it.

CUSTOM VALIDATION ATTRIBUTE – CLIENT SIDE VALIDATION

In MVC, as we know validators work off convention based on DataAnnotation attributes. For example if you have a DataAnnotation indicating the length of the field, the length validator kicks in, similarly required field validators and type constraints all are based on the DataAnnotations.

We will have to create a validation attribute that will be applied to required fields in our data element. Let's see what it takes to build such an attribute.

The DynamicLengthValidationAttribute

Step 1: We add a folder called Validators in our View project and add a class DynamicLengthValidationAttribute. This class inherits from the ValidationAttribute base class from System. ComponentModel.DataAnnotations

```
namespace CustomJqueryValidator.Validators
{
 public class DynamicLengthValidationAttribute :
 ValidationAttribute
 {
  public override bool IsValid(object value)
  {
   return base.IsValid(value);
  }
  protected override ValidationResult IsValid(object
   value, ValidationContext validationContext)
  {
   return base.IsValid(value, validationContext);
  }
 }
}
```

As we can see we have overridden two overloads of the IsValid method in it.

Step 2: Next we add a constructor that takes two parameters

```
string _validationField, _dataLengthField;
public DynamicLengthValidationAttribute
    (string validationField, string dataLengthField)
{
    _validationField = validationField;
    _dataLengthField = dataLengthField;
}
```

We will see their use shortly.

Step 3: To let MVC know that this attribute should result in a client-side validation as well, we need to implement the IClientValidatable interface from the System.Web.Mvc namespace. The interface has one method that we need to implement

```
public IEnumerable<ModelClientValidationRule>
GetClientValidationRules
    (ModelMetadata metadata, ControllerContext con-
text)
{
    …
}
```

NOTE: Purpose of the GetClientValidationRules method is to pass attribute parameters to the jQuery validator. It acts as the glue between the attribute parameters set at design time and their usage on the client side at runtime.

To pass the two parameters we got in the constructor to our custom jQuery validator we need to create another class that inherits from ModelClientValidationRule class in the System. Web.Mvc namespace.

The DynamicLengthValidationRule

In the Validators folder created earlier add a new class DynamicLengthValidationRule that inherits from ModelClientValidationRule. This class turns out to be rather simple in the sense we only have to implement the constructor and pass the parameters that we would want to access on the client side jQuery, to it.

```
public DynamicLengthClientValidationRule(string
errorMessage, string validationField, string
dataLengthField)
{
  ErrorMessage = !string.IsNullOrEmpty(errorMessage) ?
        errorMessage : “Dynamic length validation
        failed”;
  ValidationType = “dynamicmaxlength”;
  ValidationParameters.Add(“validationfield”,
                          validationField);
    ValidationParameters.Add(“datalengthfield”,
                             dataLengthField);
}
```

We have passed three parameters, the errorMessage that we'll show when the validator fails, and the validationField and dataLengthField that are getting passed in from the DynamicLengthValidationAttribute.

What are these assignments for?

Some explanation is due for the ValidationType and ValidationParameters collection. Well, the values we are assigning here are used to create data-* attributes for the Field to which our custom validation attribute is going to be assigned. In our case, we'll use the validator for the “Code” property in Address Entity. So it will end up with additional attributes that go –

```
data-dynamicmaxlength-validationfield=<value in
validationField>
data-dynamicmaxlength-datalengthfield=<value in
dataLengthField>
```

So the ValidationType is appended to each attribute which is in turn used by the validator framework later.

Once our Rule is ready, we head back to the Attribute and update the GetClientValidationRules method to return an instance of our custom rule as follows

```
public IEnumerable<ModelClientValidationRule>
GetClientValidationRules
    (ModelMetadata metadata, ControllerContext context)
{
    var rule = new DynamicLengthClientValidationRule
                (ErrorMessage, _validationField,
                                _dataLengthField);
    yield return rule;
}
```

That sets up MVC so that it can hook the custom attribute to jQuery Validator. However jQuery Validator by default doesn't know what to do with a 'dynamicmaxlength' validator. Time to hook in some jQuery.

Custom Adapter and Validation Method

We add a new JavaScript file in the Scripts\Validators folder and name it jquery.unobstrusive-dynamiclength.js

First thing we do is add a function to the unobtrusive adapter

```
$(function ()
{
    jQuery.validator.unobtrusive.adapters.add
        (‘dynamicmaxlength’, [‘validationfield’,
         ‘datalengthfield’], function (options)
    {
      var attribs = {
       validationfield: options.params.validationfield,
       datalengthfield: options.params.datalengthfield
      };

      options.rules[‘dynamicmaxlength’] = attribs;

      if (options.message)
      {
         options.messages[‘dynamicmaxlength’] = options.
message;
      }
 });
}(jQuery));
```

This chunk of JavaScript looks rather strange, so let's take it line by line.

When adding to the adapters, we pass three things:

1. Name of the adapter. This has to be the same as the 'ValidationType' that we setup earlier in the rule. In our case it is 'dynamicmaxlength'.

2. Array of parameter names: These are parameters that we added to the ValidationParameters collection in the Rule we setup earlier.

3. Function that initializes the adapter with a single input parameter (options). This 'options' object has fields corresponding to the ValidationParameters that we added in the ValidationRule above.
   a. In this function, we are creating a JSON object of the parameters we got from our validation rule and putting it in the rules object, mapped by our validator name.
   b. In the options.message property, we have the error message that was assigned in the ValidationRule again.

Now, the parameters passed from the DynamicLengthValidationRule have been used to create a custom unobtrusive adapter.

Finally, what we need is the JavaScript function to do the actual validation.

```
jQuery.validator.addMethod(‘dynamicmaxlength’, function
(value, element, param)
{
    var validationFieldSelector = ‘#’ +
        param.validationfield + ‘ option:selected’
    var requiredLength = $(validationFieldSelector).
        attr(param.datalengthfield);
    if (value.length != requiredLength)
    {
        return false;
    }
    else
    {
        return true;
    }
});
```

This function registers the actual method to be called when validation needs to happen. The registration happens by keying the function to the validator name 'dynamicmaxlength'.

The function has three input parameters

1. value – The actual value in the html element to which the validator is applied. In our case it will be the string that we pass for the Postal Code.
2. element – The actual HTML Dom element on which the validation is happening. So it includes all the additional attributes that were set on it.
3. param – This is the object named attribs that we created while registering the adapter earlier.

With this much meta-information, we are now in a position to determine if the provided string is of correct length or not. How? Let's step through the function we passed to addMethod.

1. It uses a value passed from the attribute called validation field and creates a jQuery selector for the Option. So validationfield must contain the Id of the dropdown that shows the list of countries. The selector finds out which option is currently selected.
2. Next it uses the above option element and in its attributes, looks for an attribute that is passed as datalengthfield. If we rewind, you'll remember we had stored the length of the postal code in an attribute called data-length, so the value that we have to pass from the Attribute is a string value – "data-length".
3. Finally we check the length of the value string is the same as the value with 'data-length' attribute. If yes, we say validation passed (return true) else failed (return false).

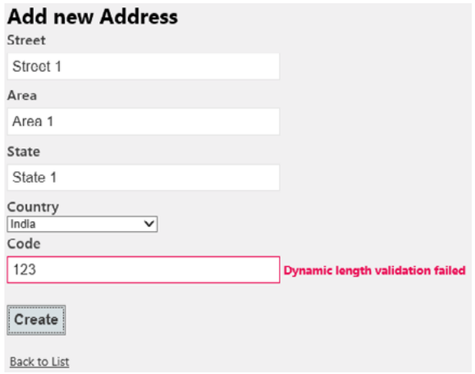
Using the Attribute

We were able to deduct the desired values that should be passed into the Validation attribute from above. So let's actually set it up before we run the application.

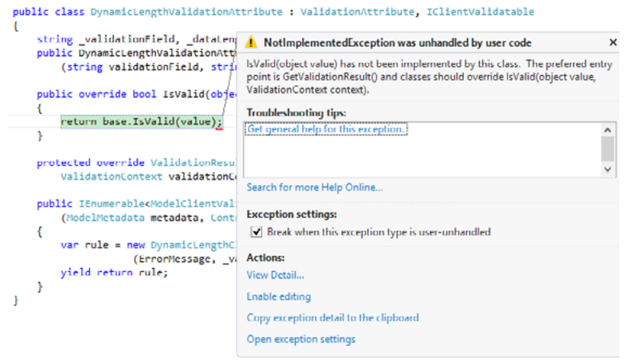
In the ViewModel's Address class, we decorate the 'Code' property with our new Validation Attribute and we pass 'CountryId' (name of the dropdown) and 'data-length' (the attribute that holds the length in the option element.

```
public class Address
{
    [Key]
    public int Id { get; set; }
    [MaxLength(200)]
    public string Street { get; set; }
    [MaxLength(200)]
    public string Area { get; set; }
    [MaxLength(200)]
    public string State { get; set; }
    [ForeignKey("Country")]
    public int CountryId { get; set; }
    public Country Country { get; set; }
    [DynamicLengthValidation("CountryId", "data-length")]
    public string Code { get; set; }

    public int SelectedCountryCodeLength { get; set; }
}
```

We now run the application and try to pass a string length != permitted length for the Postal Code. This is what we will get.



Great! This is now working so let's provide the correct length and create a new one. Wham!



Exception on the server-side! Well, the ValidationAttribute class does not implement the IsValid method. You have to provide an implementation of your own. Knowing safely that jQuery validation has done its job, you can safely return true here by default. That however is not such a great idea because as a best practice, client side validation should always be backed by server side validation. We have two options here, call the database and validate against the database or enhance our Validator. In the next section, we'll see how to enhance the validator.

CUSTOM VALIDATIONATTRIBUTE - SERVER SIDE VALIDATION

We have seen how our custom validator does client side validation without any postback. However it's not really doing any validation on the server side.

We have two overloads of IsValid method that can be utilized to do the server side validation. The first method is as follows

```
protected override ValidationResult IsValid(object
value, ValidationContext validationContext)
{
    return base.IsValid(value, validationContext);
}
```

This method receives the value of the field as well as the entire ViewModel in the validationContext's ObjectInstance property.

This makes is pretty easy to call the database here and verify the length. However that will hard bind this validator to a particular type of Object. Instead, we'll do something outside the validator to use it generically.

Using a ViewModel property to double-check Input

We added a property called SelectedCountryCodeLength, in the Address view entity, but have not used it so far. We will use it now. We will pass this as a string to the Validator and assign the 'data-length' to it when we do the validation in jQuery. That way, the value against which the length was validated will be returned to the server.

To do this, we need to update the constructor of DynamicLengthValidationAttribute and DynamicLengthClientValidationRule to add a new parameter in the constructor.

```
string _validationField, _dataLengthField, _returnField;
public DynamicLengthValidationAttribute
(string validationField, string dataLengthField,
                         string returnfield)
{
    _validationField = validationField;
    _dataLengthField = dataLengthField;
    _returnField = returnfield;
}
```

We pass the new value when instantiating our custom rule as well

```
public IEnumerable<ModelClientValidationRule>
GetClientValidationRules
     (ModelMetadata metadata, ControllerContext context)
{
     var rule = new DynamicLengthClientValidationRule
                (ErrorMessage, _validationField, _
dataLengthField, _returnField);
     yield return rule;
```

In the Rule we add it to the ValidationParameters collection as well.

```
public class DynamicLengthClientValidationRule:
ModelClientValidationRule
{
  public DynamicLengthClientValidationRule(string
    errorMessage, string validationField, string
    dataLengthField, string returnfield)
    {
        ErrorMessage = !string.
            IsNullOrEmpty(errorMessage)  ?
            errorMessage : "Dynamic length validation
            failed";
    ValidationType = "dynamicmaxlength";
    ValidationParameters.Add("validationfield",
      validationField);
    ValidationParameters.Add("datalengthfield",
       dataLengthField);
    ValidationParameters.Add("returnfield",
       returnfield);
    }
}
```

To have this value in the DOM, we declare a hidden variable in the Create.cshtml

```
<div class="editor-field">
    @Html.HiddenFor(model => Model.
SelectedCountryCodeLength, new { id =
"SelectedCountryCodeLength" })
</div>
```

Next we add it to the adapter's input parameter collection and save it in the attribs object.

Finally in the function passed to addMethod (in the .js file), we set the value from the selected Country's data-length property to the above hidden field as follows

```
jQuery.validator.addMethod('dynamicmaxlength', function
(value, element, param)
{
    …
    if ($(element).val().length != requiredLength)
    {
        return false;
    }
    else
    {
```

```
        $('#' + param.returnfield).
        val(requiredLength);
        return true;
    }
});
```

So now when a user selects a country in the dropdown on Post, the data is sent all the way back to the Server side via the custom validation attribute.

What do we do with it?

We update the IsValid methods to do the same check as we did on the client end as follows:

```
public override bool IsValid(object value)
{
    return value.Length == _requiredLength;
}

protected override ValidationResult IsValid(object
value, ValidationContext validationContext)
{
    PropertyInfo propInfo = validationContext.
ObjectType.GetProperty(_returnField);
    _requiredLength = (int)propInfo.
GetValue(validationContext.ObjectInstance);
    return base.IsValid(value, validationContext);
}
```

We are extracting the property from the validationContext. Then using reflection and the property Info on the ObjectInstance, we are determining the value that was set in the SelectedCountryCodeLength property. The bool IsValid method compares with _requiredLength passed in the hidden field against the actual field length and returns true or false.

Last but not least in we have to update the Attribute declaration in Address.cs to pass the Property Name.

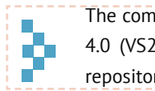
```
[DynamicLengthValidation("CountryId", "data-length",
"SelectedCountryCodeLength")]
public string Code { get; set; }

[NotMapped]
public int SelectedCountryCodeLength { get; set; }
```

Now if we run the application and try to save a new Address, assuming it passed all validation on the client side, it will sail through the server side validation and save the entry.

CONCLUSION

That was a rather long process to build a custom validator for ASP.NET MVC that uses the unobtrusive jQuery Validation framework. Even though it seems long drawn, the actual amount of code is rather small, it's more about knowing how to couple the various pieces together. Once set up, they are generic and reusable anywhere in your application.

Hope I have been able to clarify enough for you to write other validators now. Remember, if in your use-case it is okay to make a Database/Repository call from the validator go for it, be pragmatic about it and don't break your system's architecture for it. In the above example I've taken the extreme approach of keeping all kinds of Business inputs declarative and as a result, we have nice generic solution. ■



The complete code is available for both .NET 4.0 (VS2010) and .NET 4.5 (VS2012) on our GitHub repository at **bit.ly/dncm5-jqva**

Sumit is a .NET consultant and has been working on Microsoft Technologies for the past 12 years. He edits, he codes and he manages content when at work. C# is his first love, but he is often seen flirting with Java and Objective C. You can Follow him on twitter @sumitkm and read his articles at bit.ly/KZ8Zxb