63 pages

VS ALM For High Profitability
Use Visual Studio ALM to improve software productivity

xUnit in ASP.NET MVC
Test driven development using xUnit

Issue 01 | **JULY 2012** LAUNCH EDITION

DNCMagazine

www.dotnetcurry.com

Azure Adoption
Best Practices for architects

Building ChirpyR
using ASP.NET MVC, Web API ,SignalR and Knockout.js

Plus

DI using Ninject in ASP.NET MVC

Bring Your Charts to Life with HTML5

Custom Workflows in SharePoint

Templating with JsRender

Create Next Gen Metro style Business Apps

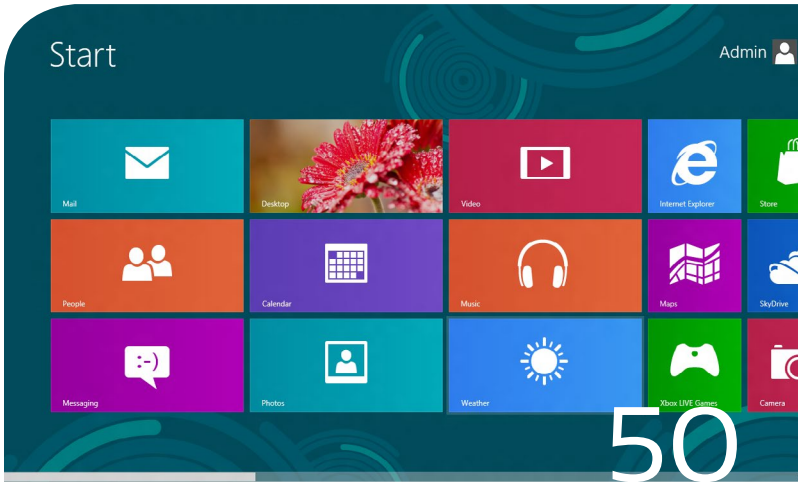
Exclusive Interview

AYENDE RAHIEN
CREATOR OF RAVENDB

JULY 2012

CONTENTS

16



50

WEB DEV

ENTERPRISE DEV

FEATURED

LETTER FROM THE EDITOR

We bring hot and toasty .NET technology tid-bits along with practical problem solution articles.

Dear Readers,

As the Chief Consulting Editor for the magazine, I am extremely excited about the launch release, as we have a big bunch of exclusive content.

We have tried to cover a broad-spectrum of technologies from the .Net stack, starting with Application Lifecycle Management (ALM) to patterns and practices in ASP.NET MVC and SharePoint to Azure Adoption Pivots. Each article in this issue is brought to you by industry veterans including Microsoft MVP's.

Our scoop is a fantastic freewheeling interview with Oren Eini aka Ayende Rahien, the creator of RavenDB the document database built entirely on .NET. We talk about everything starting from nicknames to .Net framework and how RavenDB was born.

With this July edition, we are launching into a bold venture to bring hot and toasty .NET technology tid-bits along with practical problem solution articles, once every two-months.

We hope you enjoy our selection of content. Remember we are always looking out for ideas and feedback, so feel free to email me or Tweet to us @dotnetcurry with the #dncmag hashtag. We are sure, every little feedback will help make DNC Magazine even better.

With that I will let you dive into the content. Have fun!

Sumit K. Maitra

Editor in Chief

ON THE COVER

Oren Eini aka Ayende Rahien

A freewheeling interview with Ayende Rahien, the creator of RavenDB and NHibernate

www.dotnetcurry.com

dnc mag

Editor-In-Chief • Sumit Maitra
sumitmaitra@a2zknowledgevisuals.com

Editorial Director • Suprotim Agarwal
suprotimagarwal@dotnetcurry.com

Art & Creative Director • Minal Agarwal
minalagarwal@a2zknowledgevisuals.com

Advertising Director • Satish Kumar
business@dotnetcurry.com

Writing Opportunities • Carol Nadarwalla
writeforus@dotnetcurry.com

Contributing Writers • Govind Kanshi, Mahesh Sabnis, Minal Agarwal, Raj Aththanayake , Subodh Sohoni, Suprotim Agarwal

Interview Feature• Ayende Rahien
Twitter @ayende

NEXT ISSUE - 1st September, 2012
www.dncmagazine.com

POWERED BY

a2z Knowledge Visuals

BUILDING ChirpyR

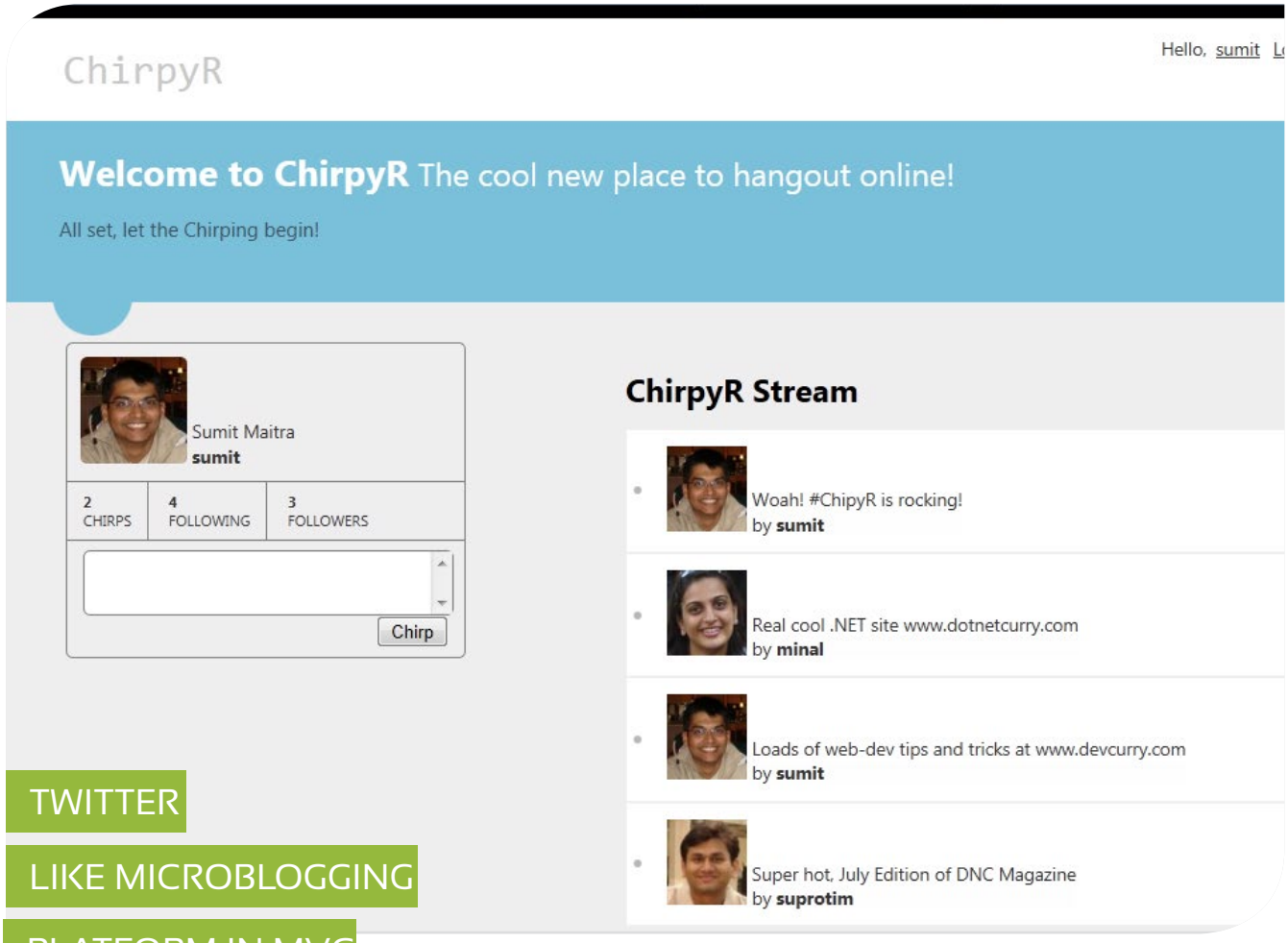
Sumit Maitra demonstrates a fun ASP.NET MVC application using WebApi + jQuery + SignalR + Knockout.js

DOWNLOAD FILES >

bit.ly/dncmag-chirpy

With ASP.NET MVC4 coming close to release, we have come to a maturity and flexibility level in the ASP.NET platform that allows us to rapidly build powerful and rich web applications quickly.

Today we will build a fun application. It is a Micro Blogging (like Twitter) platform using various out-of-the box components in the ASP.NET platform and a few Nuget packages..



TWITTER LIKE MICROBLOGGING PLATFORM IN MVC

HIGH LEVEL DESIGN

For now, we will have a simple three layer architecture with the primary goals being:

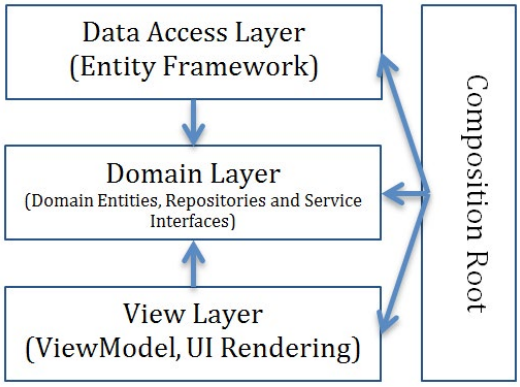
1. Create an API as we build the application so that we can release it for others to use when we are ready

2. Use Async operations as much as possible

3. Use a persistent connection framework to keep the application updated.

4. Use minimal server side rendering logic.

5. Use SQL Server backend and access it using EntityFramework



6. As seen above we have a simple layering scheme that uses the Repository pattern and Dependency Injection. For this application, we will skip an IoC container and hand-spin our Composition Root.

a. The Domain Layer is the heart of the application. It will define the following
   i. The Domain entities defined as Plain old CLR objects (POCOs)
   ii. The Repository Interfaces that will define the data access points.
   iii. A service layer to wrap business logic around the data-access

b. The Data Access Layer will have
   i. The EF Code First Entities
   ii. The Database Context to communicate with the database
   iii. Implementation of the Repository Interfaces defined in the Domain layer.

c. The View Layer will have the following
   i. The UI rendered using Knockout and Razor Views
   ii. The SignaR Hub to manage persistent connections and communicate with the web clients
   iii. Knockout helps you create observable dynamic UI that responds fluidly to user changes. It provides a templating style that is easy to use and facilitates easy rendering of repeating data.

d. The Composition Root will be responsible for mapping interfaces to their concrete implementations.

The Usage Workflow

The Usage Workflow for our application is simple.

1. Users come to our site and register. We collect minimal information and get them started.
2. Once registered they can log in.
3. After logging in they can send messages (we will refer to them as Chirps), view chirps and respond to chirps.
4. Logged in users can choose to see their own chirp stream or the public chirp stream.
5. If they chose not to register, they can watch the public chirp stream.

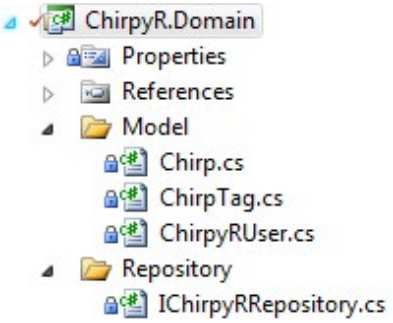
With the basics out of the way, let's dive into code straightaway and we will start with the heart of the Application - the Domain layer.

The Domain Layer

In the Domain layer, we start with the Domain Model and then explore the Interfaces and the service implementation.

ChirpyR.Domain.Model

The Domain Model is pretty simple; it has the following entities only.



The Chirp Entity is as follows

```
namespace ChirpyR.Domain.Model
{
    public class Chirp
    {
        public long Id
            { get; set; }
        public string Text
            { get; set; }
        public List<ChirpTag> Tags
            { get; set; }
        public List<Chirp> Replies
            { get; set; }
        public DateTime ChirpTime
            { get; set; }
        public string ChirpUrl
            { get; set; }
        public ChirpyRUser ChirpBy
            { get; set; }
        public Chirp InReplyTo
            { get; set; }
    }
}
```

It basically encapsulates all the information we want for the Chirp including the text, the tags (denoted by hash #), the replies to the Chirp (single level only), the date and time stamp, the direct URL of the Chirp, the user who sent out the Chirp and the Chirp to which the current Chirp is a reply to (null if it's not a reply chirp).

The ChirpTag domain entity is as follows:

```
public class ChirpTag
{
    public int Id
        { get; set; }
    public string Tag
        { get; set; }
    public int TagCount
        { get; set; }
}
```

It simply has the Id, the text of the Tag and the number of occurrences of the tag. This TagCount field is a calculated field.

The ChirpyRUser entity wraps the ASP.NET Membership User entity.

In future we can expand this to create a more detailed profile.

```
public class ChirpyRUser
{
    public long Id
        { get; set; }
    public string UserId
        { get; set; }
    public string FullName
        { get; set; }
    public string Email
        { get; set; }
    public string Gravataar
        { get; set; }
    /// <summary>
    /// Password is not loaded from
    /// the database. It is for
    /// new account registration ONLY
    /// </summary>
    public string Password
        { get; set; }
    /// <summary>
    /// Used in the ViewModel only
    /// </summary>
    public string OldPassword
        { get; set; }
    public List<ChirpyRUser> FollowerIds
        { get; set; }
    public List<ChirpyRUser> FollowingIds
        { get; set; }
}
```

The Repository

Chirp Repository has the following methods:

```
public interface IChirpyRRepository
{
    IList<Chirp> GetLatestChirps();
    IList<Chirp> GetLatestChirpsFor(string user);
    IList<ChirpyRUser> GetFollowers(ChirpyRUser user);
    IList<ChirpyRUser> GetFollowing(ChirpyRUser user);
    IList<ChirpTag> GetChirpTags(int topCount);
    long AddChirp(Chirp chirp);
    long RegisterUser(ChirpyRUser newUser);
    long UpdateUserAccount(ChirpyRUser updatedUser);
    long UnRegisterUser(ChirpyRUser removeUser);
    long FollowChirpR(ChirpyRUser currentUser,
        ChirpyRUser followeUser);
    long UnfollowChirpR(ChirpyRUser currentUser,
        ChirpyRUser unfollowUser );
    ChirpyRUser GetUserById(string userId);
    /// <summary>
    /// Testing API only. Service method should not be
    /// implemented
    /// </summary>
    /// <param name="deletedUser"></param>
    /// <returns></returns>
    long DeleteUserAccount(ChirpyRUser deletedUser);
}
```

These can be roughly grouped into the following functionalities:

Chirp Data Administration

- GetLatestChirps – Returns the top 20 chirps from the public timeline
- GetLatestChirpsFor – Returns the top 20 chirps for the given user
- GetFollowers – Returns the list of ChirpyRUsers who are following the given user
- GetFollowing – Returns the list ChirpyRUsers whom the current user if following
- GetChirpTags – Takes Top count and returns and returns that many Tags
- AddChirp – Adds a Chirp to the database
- FollowChirpR – This method adds a relationship between the current user and followed user. The information is actually stored as a relationship in the database.
- UnFollowChirpR – This method removes the relationship if it exists.

User Administration and methods

- RegisterUser: Sets up a new user.
- UnRegisterUser: Removes the user
- GetUserById: Gets a particular user details using the UserId string.
- DeleteUserAccount: Deletes are user account from the database. But we won't be exposing this as a service we will only use it for our test cleanup methods.

The Data Layer

The Domain Layer is the heart of our application. It encapsulates the complete intent and functionality of the application. The data layer is completely about serialization and de-serialization of data. It also translates the data into Domain model.

The IDataEntity<T>

Each data entity needs to be translated into a Domain entity when sending data out to the Domain layer and translated back into Data entity when the data comes in from the View/ Domain layers. The IDataEntity interface defines two methods ToDomainEntity and LoadFromDomainEntity.

```
public interface IDataEntity<S,T>
{
    T ToDomainEntity();
    S LoadFromDomainEntity(T input);
}
```

The Interface definition takes two sample types S = <The Data Entity> and T = <The Domain Entity>.

The interface is implemented on every data entity that goes across layers; for example the ChirpyRUser.

The ChirpyRRelation

The ChirpyRRelation object encapsulates a User and Follower relationship. If Joe is following Joanne, the relation is saved as Joe = Parent and Joanne = Child. This way we can query for followers and following counts for a given user.

```
public class ChirpyRRelation
{
    public long Id { get; set; }
    public ChirpyRUser Parent { get; set; }
    public ChirpyRUser Child { get; set; }
}
```

The ChirpyRSqlRepository

The ChirpyRSqlRepository has the implementation of the IChirpyRRepository.

The rest of the Repository implementation is standard DBContext operations where we serialize or deserialize data from SQL Server using the Entity Framework. We take a look at one serialization method and one deserialization method.

```
public long RegisterUser(Domain.Model.ChirpyRUser newUser)
{
    Data.Model.ChirpyRUser user =
        new Model.ChirpyRUser()
            .LoadFromDomainEntity(newUser);
    using (ChirpyRDataContext context =
        new ChirpyRDataContext(_connectionName,
            _schemaName))
    {
        context.Entry<ChirpyRUser>(user).State
            = System.Data.EntityState.Added;
        context.SaveChanges();
    }
    return user.Id;
}
```

The RegisterUser method as shown above, receives the new User's information via a Domain object. It is translated into a Data object using the LoadFromDomainEntity helper method.

Once we have the correct Data entity, we create a new DbContext and add the object to the context. We mark its state

as Added and request the Context to save the changes.

```
public Domain.Model.ChirpyRUser
    GetUserById(string userId)
{
    using (ChirpyRDataContext context = new
        ChirpyRDataContext(_connectionName,
            _schemaName))
    {
        try
        {
            ChirpyRUser user = context.ChirpyRUsers
                .Single<ChirpyRUser>
                    (c => c.UserId == userId);
            if (user != null)
            {
                return user.ToDomainEntity();
            }
        }
        catch (InvalidOperationException ex)
        {
            return null;
            // Log the exception here
        }
    }
    return null;
}
```

The GetUserById is a de-serialization method that queries the Database using the Data Context. If a User is returned, we translate into a domain entity and return it. If the user does not exist, we return a null.

The rest of the repository similarly implements the other related CRUD operations.

The WebApi Controllers

Instead of the traditional MVC Controllers, we are using WebApi controllers in our application. This results in all our queries being available for access over HTTP. We don't need to add a special web service layer.

WebApi controllers respond to the Http Get, Put, Post, Delete and other such HTTP verbs. We easily tailor our CRUD operations around these VERBS.

We have one controller per action; for example Login action has one controller and Registration action has one controller for itself.

WebApi supports rich data binding support like MVC hence the standard MVC data-binding techniques more of less work out of the box.

The View Layer

Web 2.0 applications have increasingly jostled with native application platforms with respect to ease of use, feature richness and visual appeal. This has been accelerated to by the narrowing of gap in JavaScript support across browsers and even more, because of frameworks like jQuery that help developers gloss over browser quirks and focus on customer solutions. With the maturing of HTML5 specs web developers finally have a complete toolkit that allows them to build extremely powerful web applications.

We are using jQuery 1.7.2 and jQuery UI 1.8.1 JavaScript libraries in our application. We are also using a JavaScript framework called Knockout. Knockout helps create and manage observable View Models. It provides two-way data binding support and rich template features. This helps in render repeating data like lists and spreadsheets.

Last but not least, we are using SignalR – a client-server combo framework for managing persistent connections over HTTP. What's noteworthy is that all three frameworks are community driven open source projects.

SignalR – Hubs, Clients and persistent connections

The theory of persistent connections is rather elaborate and it won't be possible to squeeze it in this article. To sum up, over a stateless protocol like HTTP, connection between the browser and web server is limited to the time the client makes the request, till the server completes serving the request. Once the request is served, there is no connection between the two. Persistent Connection solutions aim to plug this shortcoming by simulating a pipe-like behavior between a client and server so that two-way communication can happen in real-time and servers can thus push data to clients over an open pipe. SignalR is one such framework. There are various techniques to achieve persistent connections like long polling, but SignalR abstracts these away from you and decides the best technique depending on the browser support.

Why Use SignalR?

Our use case for SignalR is pretty simple. We have our web-clients who should see a stream of messages in their home page (even if they are not logged in). Essentially as soon as one user posts a message, it should get broadcasted to all connected clients.

With this basic requirement in mind, let us get a little deep with SignalR.

Hubs and Clients in SignalR

In SignalR, to establish an open connection between a server and a client, on the server side, we have a class that derives from SignalR.Hubs.Hub. This class contains all the methods that are going to be called from the client on to the server. A barebones (but functional hub) looks as follows

```
using SignalR.Hubs;
namespace ChirpyR.Mvc4.SignalR
{
    public class ChirpyRHub : Hub
    {
    }
}
```

To use the hub and broadcast a message, we need only a few lines of code on the server as seen below.

```
// POST api/chirpyr
public void Post(Chirp model)
{
    var hubContext =
        GlobalHost
            .ConnectionManager
                .GetHubContext<ChirpyRHub>();
    hubContext.Clients.NewChirp(model);
}
```

When the Chirp is posted from the UI, the SignalR ConnectionManager returns an IHubContext that is context aware of the exact type. NewChirp is a dynamic method and there should be a corresponding JavaScript method at the client end to intercept the message from server.

The typical Client side initiation and event handling code is as follows

```
var hub = $.connection.chirpyRHub,
$msgs = $("#chirpStream");
hub.NewChirp = function (chirp) {
    $msgs.prepend("<li>" +
        chirp.Text + "</li>");
}
$.connection.hub.start();

$(document).on("click",
    "#postChirp", function () {
        var chirp = {
            "Text": $("#chirpText").val()
        };
        ajaxAdd("/Api/ChirpyR",
            ko.toJSON(chirp), function (data) {
            });
```

- The 'hub' object is initialized
- The NewChirp function is defined on the client side Hub to handle the broadcasted message. Here we see it is prepending a list item presumably into a <ul> element. Note the chirp object sent from sever is cast into a Json object by default.
- The hub connection is started. That's it.
- The subsequent click handler does a HTTP Post that the API controller handles and uses the SignalR hub to broadcast to all connected clients.

That's all we need for our basic SignalR use case. Once we have the entire Chirp data along with the user information, it will be easy to filter it at the client end itself and not touch the broadcast process.

With SignalR in place, let us take a look at the final 'Lego piece' that is Knockout (hat tip to Scott Hanselman for the analogy).

Knocking it off the park using Knockout.js

We will use Knockout's 'Observable' capabilities to update the UI as data comes in from the server asynchronously.

We will also leverage its template and data binding features to draw out a nice list of latest tweets.

Template definition using Knockout

For the Chirp Stream section of our UI, we will represent the stream as a <ul> element and dynamically add <li> elements on top as they come in. The markup is as follows

```
h2>ChirpyR Stream</h2>
ul id="chirpStream" data-bind="foreach: chirps"">
   <li style="background-color: White;
       border: 1px solid #F0F0F0">
       <div style="display: inline-block; padding: 10px">
           <div style="display: inline-block;">
               <img alt="gravatar" height="64px" width="60px
                data-bind="attr: {src: GravatarUrl}" />
           </div>
           <div style="display: inline-block;">
               <div style="vertical-align:top">
                   <span data-bind="text: Text"></span>
               </div>
               <div style="background-color: #fefefe">
                   by <span data-bind="text: By"
                   style="font-weight: bold"></span>
               </div>
               <div>
               </div>
           </div>
       </div>
   </li>
/ul>
```

The Knockout utilized data-bind attributes are highlighted. As we can see, the <ul> is bound to a collection called chirps. It iterates and adds <li> for as many items in the chirps collection. For each <li>, we have a Gravatar Image, the Text of the actual Chirp and the UserId of the Chirp author.

Each is bound using data-bind attributes. Knockout can bind common attributes like Text, however if we are to bind to attributes not directly supported, we use the attribute binding as shown for the <img> in the second highlight above. Here Knockout is using attribute binding to bind the value of the GravatarUrl field to the 'src' attribute of the <img> tag.

Finally the text of the Chirp and the UserId are bound using the 'text:' binding.

Building the ViewModel and Updating Data

The ViewModel is created and maintained in client side JavaScript.

```
var data = [
    new chirpItem(
        "Real cool .NET site www.dotnetcurry.com",
        1,
        'http://www.gravatar.com/avatar/147bacafcdb00d67d33
        'sumit'),
    new chirpItem(
        "Loads of web-dev tips and tricks at www.devcurry.c
        2,
        "http://www.gravatar.com/avatar/147bacafcdb00d67d33
        'sumit'),
    new chirpItem("Super hot, July Edition of DNC Magazine"
        3,
        "http://www.gravatar.com/avatar/147bacafcdb00d67d33
        'sumit'),
];
function chirpItem(text, id, gravatar, by) {
    return {
        Text: text,
        Id: id,
        GravatarUrl: gravatar,
        By: by
    };
}

var viewModel = {
    // data
    chirps: ko.observableArray(data)
}
ko.applyBindings(viewModel);
```

In the above image, 'data' is the JavaScript array that is going to be the container for all our chirps. The function chirpItem creates a JavaScript object with the accessors Text, Id, GravatarUrl and By. As we see, these are the same properties that we bound in to the html markup.

The viewModel object wraps our data (and can contain behavior methods too) from a JavaScript array into an observable array. In future, when we add to the 'chirps' collection, Knockout will be able to detect the change and update the data binding appropriately. The command ko.applyBindings(viewModel) actually does the data binding.

We have so far seen static binding of sample data. However, how will new chirps from other users, come up in our screen? Answer is we have to update the SignalR NewChirp method. The updated method is shown below

```
$(function () {
    var hub = $.connection.chirpyRHub,
    $msgs = $("#chirpStream");
    hub.NewChirp = function (chirp) {
        viewModel.chirps.unshift(
            new chirpItem(chirp.Text,
                          chirp.Id, chirp.ChirpBy.Gravataar,
                          chirp.ChirpBy.UserId));
    }
    $.connection.hub.start();
```

As highlighted, we are adding a new chirpItem to the viewMode.chirps observable collection. The chirpItem is populated using data sent from the server and encapsulated in the 'chirp' JSON object that is the input parameter of the NewChirp function.

That's all the change we need to do. Now if a user posts a chirp, the server will detect it and raise the SignalR event for all connected clients and the above code will add the new data into the observable collection, which Knockout will use to update the UI.

To Round off

We were able to build a rich web application using out-of-the box ASP.NET components and community supported framework like Knockout and SignalR. It is a sign of maturity of the ASP.NET platform. It's getting yummier by the day! ■

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 http://bit.ly/KZ8Zxb

Both these people are deploying tomorrow...

sql CONNECT

The Visual Studio add-in that helps develop and version control databases

Develop databases alongside application code, and track and share all changes through your source control system, all within Visual Studio.

www.red-gate.com/products/sql-development/sql-connect

redgate software

BRING YOUR CHARTS TO LIFE WITH HTML5 CANVAS & JAVASCRIPT

DOWNLOAD THE FILES >

bit.ly/dncmag-canvas

Suprotim Agarwal *shows how to create an animated Bar Chart on the HTML5 Canvas using a little bit of JavaScript and a little bit of imagination*

HTML5 is the new lingua franca for the Web. It is a promising and evolving technology under the Open Web Standard and has been written primarily to develop better, more efficient web applications in the era of cloud computing and mobile devices. HTML5 kicks off a whole new era for web developers, by moving HTML from being a relatively simple markup language to providing a host of new markup and rich API's for the construction of web applications. One such rich API is the Canvas API.

Introducing the HTML5 Canvas API

The HTML5 Canvas API is a set of client-side functions, which gives developers the ability to dynamically generate and manipulate graphics directly in the browser, without resorting to server-side libraries, Flash, or other plugins. The Canvas API gives you a fine level of control over the individual pixels on your page.

It is like a blank slate on which we use JavaScript to draw and animate objects.

The <canvas> tag is one of the most happening tags in HTML5. To draw a Canvas, simply specify how large you want the canvas area to be, and the browser creates the container accordingly. For example, here's how to draw a <canvas>, which is 600 pixels wide and 400 pixels tall.

```
<canvas id="bchart" height="400" width="600">
Your browser does not support HTML5 Canvas
</canvas>
```

As of this writing, some browsers do not support the <canvas> element. Hence we have to provide alternative content for such browsers. This alternative content is provided by placing it in between the opening and closing <canvas> tags, as shown above.

The basic HTML5 Canvas API includes a 2D context using which a programmer can draw various shapes, images and render text directly onto the Canvas. Calling getContext('2d') returns the CanvasRenderingContext2D object that you can use to draw

two-dimensional graphics into the canvas. 3D contexts are defined by passing in contexts such as webgl, experimental-webgl, and others. This is work in progress.

The Canvas tag is supported in IE9, Firefox, Chrome, Safari, iOS Safari, Opera, Opera Mobile and Android.

To use the <canvas> tag in IE 6, 7, or 8, download the opensource ExplorerCanvas project from http://code.google.com/p/explorercanvas/. To use explorercanvas, just check if the current browser is Internet Explorer and include a script tag in your page to use it.

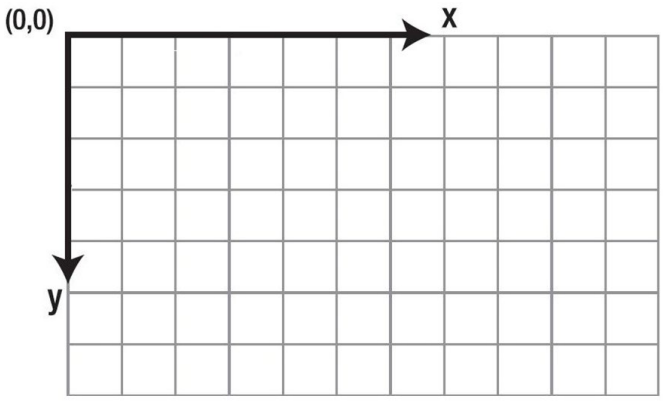
```
<head>
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
</head>
```

Alternatively also explore the Modernizr library.

Canvas 2D Context Drawing basics

The canvas's 2D context is a grid. The coordinate (0, 0) is at the upper-left corner of the canvas. When you draw on this grid, you specify the starting X and Y coordinates and the width and height. Moving to the right will increase the x value, and moving downwards will increase the y value.

Paths are used to draw lines on a canvas and fill the areas enclosed by those lines. A path can be defined as a sequence



of one or more sub-paths and you begin a new path with the beginPath() method. A sub-path can be defined as a sequence of two or more points connected by a line and you begin a new sub-path with the moveTo() method. Once you have defined the starting point of a sub-path with the moveTo() method, you can connect that point to a new point with a straight line by calling lineTo().

Similarly you can draw a rectangle using 4 different methods, one of which is the fillRect() method. If you do not want to fill the rectangle with any color, use strokeRect(). By default, the rectangle's fill color is black but you can change it using the fillStyle() function. You can fill the rectangle with either a solid color, a gradient, or even a pattern using the strokeStyle() function.

Now that we have explored some basics of the Canvas element, let's see it in action.

Animating Bar Charts using HTML5

As mentioned earlier, the HTML5 canvas is revolutionizing graphics and visualizations on the Web and we can use it to create simple or complex shapes or even create graphs and charts. In this article, I will show you how to draw a Bar Chart on the Canvas and then animate it.

Bar charts are a popular tool for visualizing data. We will create a Bar Chart using HTML5 Canvas that can automatically position and draw itself from an array of data. Since Canvas does not support Animations, we will use JavaScript to animate the chart.

Create a file named "canvaschart.html" and add the following Canvas markup in the <body> section

```
<body  onLoad="barChart();">
<canvas id="bchart" height="400" width="600">
Your browser does not support HTML5 Canvas
</canvas>
</body>
```

We have named the Canvas with an id attribute 'bchart' which will be used to link the JavaScript definition to this element.

We have declared the Canvas dimensions (height and width) right inside the markup. Although you should use CSS to control the dimensions of your HTML controls, unfortunately here you can't do that. If you do, the contents get distorted. So you are forced to decide on your canvas dimensions when you declare it.

We will draw the chart from a set of sample data defined in an array. The data represents the traffic of a site (in thousands) for a given year.

```
arrVisitors[0] = "2007, 750";
arrVisitors[1] = "2008, 425";
arrVisitors[2] = "2009, 960";
arrVisitors[3] = "2010, 700";
arrVisitors[4] = "2011, 800";
arrVisitors[5] = "2012, 975";
arrVisitors[6] = "2013, 375";
arrVisitors[7] = "2014, 775";
```

After you place the Canvas element in a document, your next step is to use JavaScript to access and draw on the element. Let's go ahead and define the barChart constructor that draws the chart.

```
// barchart constructor
function barChart() {
    canvas = document.getElementById('bchart');
    if (canvas && canvas.getContext) {
        context = canvas.getContext('2d');
    }

    chartSettings();
    drawAxisLabelMarkers();
    drawChartWithAnimation();
}
```

We retrieve the Canvas element by its ID and test if the element is available within the DOM. If it is available, create a two-dimensional rendering context. The 2d rendering context is the coolest part of the Canvas on which you can draw almost everything. The Canvas element acts as a wrapper around the 2d rendering context and provides all the methods and properties to draw on the context and manipulate it. Once you have the drawing context, we can start to draw stuff. Exciting times!

Let's first configure some settings of the chart in the chartSettings() function. Start by setting some margin and drawing area. Then calculate the total bars to be drawn on the chart and determine the width of each bar, as shown below.

```
// initialize the chart and bar values
function chartSettings() {
    // chart properties
    cMargin = 25;
    cSpace = 60;
    cHeight = canvas.height - 2 * cMargin - cSpace;
    cWidth = canvas.width - 2 * cMargin - cSpace;
    cMarginSpace = cMargin + cSpace;
    cMarginHeight = cMargin + cHeight;
    // bar properties
    bMargin = 15;
    totalBars = arrVisitors.length;
    bWidth = (cWidth / totalBars) - bMargin;
```

The following piece of code then extracts data from the array to find the maximum value to plot on the graph.

```
// find maximum value to plot on chart
maxDataValue = 0;
for (var i = 0; i < totalBars; i++) {
    var arrVal = arrVisitors[i].split(",");
    var barVal = parseInt(arrVal[1]);
    if (parseInt(barVal) > parseInt(maxDataValue))
        maxDataValue = barVal;
}
```

Define a new function drawAxisLabelMarkers() that contains call to functions drawAxis() and drawMarkers(). drawAxis() draws the X and Y axis lines depending on the parameters passed to it.

```
// draw X and Y axis
function drawAxis(x, y, X, Y) {
    context.beginPath();
    context.moveTo(x, y);
    context.lineTo(X, Y);
    context.closePath();
    context.stroke();
}
```

For the X-axis, we will draw a line from the lower left to right whereas for the Y-axis, we will draw a line from the lower left to upper left. The drawMarkers() function uses a simple loop to add labels to the Y-axis. For the X-axis, we will use the data array to mark labels, as shown below

```
// X Axis
context.textAlign = 'center';
for (var i = 0; i < totalBars; i++) {
    arrval = arrVisitors[i].split(",");
    name = arrval[0];

    markerXPos = cMarginSpace + bMargin
                + (i * (bWidth + bMargin)) + (bWidth/2);
    markerYPos = cMarginHeight + 10;
    context.fillText(name, markerXPos, markerYPos, bWidth);
}
```

Finally add the titles to the X and Y-axis. While adding a title to the Y-axis, rotate the context to add a title vertically. Once done,

```
// Add Y Axis title
context.translate(cMargin + 10, cHeight / 2);
context.rotate(Math.PI * -90 / 180);
context.fillText('Visitors in Thousands', 0, 0);

context.restore();

// Add X Axis Title
context.fillText('Year Wise', cMarginSpace +
            (cWidth / 2), cMarginHeight + 30 );
```

We are now ready with the barebones of our application. If you view the application in a browser, you will see that the drawing surface contains the X and Y-axis along with some markers.

So far, so good! Let's move ahead and plot the array data on this chart using animation.

The Canvas element does not support animations. So how are we supposed to animate the charts? You guessed it right! The answer is using JavaScript.

To achieve an animation effect with bars, we need a way to loop through the bars and set a timeout to grow the bars in steps. Once the function starts drawing the bar dimensions, it will regularly check if the step was the last step in the animation. If not, repeatedly run after a certain number of milliseconds have elapsed, till the last step in the animation has reached.

You set a timeout using the setTimeout() method, which accepts two arguments: the function to execute and the amount of time (in milliseconds) to wait before attempting to execute the code. We can use setInterval() too, but I prefer using setTimeout() over setInterval() for the reason that JavaScript dishonors call to setInterval() when the last task added to the UI queue is still in there. This leads to jerky animations.

Shown here is the code that loops through all the data elements, and draws a bar for each one in steps by calculating the bar height and the X & Y points. To change the speed of the animation, all you need to do is change the ctr and speed variables in the chartSettings() function.

With the dimensions calculated above, call the drawRectangle() helper function, which draws a rectangle around the charts.

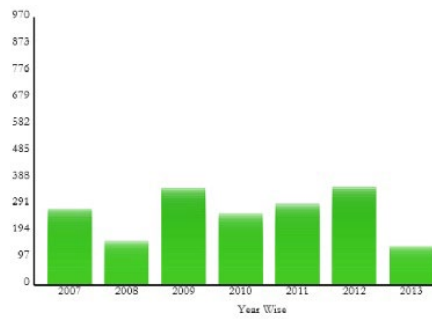
If you are dealing with heavy WebGL based graphics, explore the requestAnimationFrame method instead of setTimeout(). It works similar to the setTimeOut and you can request for an animation callback inside your function. The advantage with requestAnimationFrame is that webbrowsers can decide on the optimal fps of the animation and can reduce it for scenarios where a user moves to a different tab or minimizes the screen.

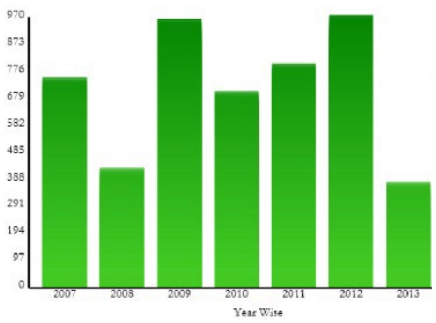
```
for (var i = 0; i < totalBars; i++) {
    var arrVal = arrVisitors[i].split(",");
    bVal = parseInt(arrVal[1]);
    bHt = (bVal * cHeight / maxDataValue) / numctr * ctr;
    bX = cMarginSpace + (i * (bWidth + bMargin)) + bMargin;
    bY = cMarginHeight - bHt - 2;
    drawRectangle(bX, bY, bWidth, bHt, true);
}

// timeout runs and checks if bars have reached
// the desired height; if not, keep growing
if (ctr < numctr) {
    ctr = ctr + 1;
    setTimeout(arguments.callee, speed);
}

function drawRectangle(x, y, w, h, fill) {
    context.beginPath();
    context.rect(x, y, w, h);
    context.closePath();
    context.stroke();

    if (fill) {
        var gradient = context.createLinearGradient(0, 0, 0, 300);
        gradient.addColorStop(0, 'green');
```

As we had discussed in the beginning of this article, paths are used to draw any shape on the canvas. A path is simply a list of points, and lines to be drawn between those points. Here the beginPath() function call starts a path and we draw a rectangle of the dimension w * h. The closePath() function call ends the path and stroke() makes the rectangle visible. We have also added a gradient to the rectangles. In order to construct this colored gradient, we have added multiple gradient.addColorStop() calls. Each of them has a unique offset - i.e. 0 and 1. Note that you do not need to use only 0 and 1; you can use any two partial values like 0.25 to 0.75.

Run the application and you will observe how the bars grow with an animation. Go ahead and try changing the values in the array and the chart updates automatically. If you plan to do some advanced graphs using the Canvas, I would suggest exploring the RGraph tool.

Working with the Canvas is a lot of fun. We just created an animated Bar Chart on the HTML5 Canvas using a little bit of JavaScript and a little bit of imagination! We have just scratched the surface of the canvas API and there's much more to come. Stay tuned!





References: http://bit.ly/KILcAg

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

USING xUNIT.NET FOR TEST DRIVEN DEVELOPMENT

Raj Aththanayake covers some of the key steps involved in developing an ASP.NET MVC application using TDD.

DOWNLOAD FILES >

bit.ly/dncmag-tdd

Traditionally TDD has gone hand in hand with the flexibility offered by the testing framework as well as the platform's 'testability' support.

The ASP.NET MVC framework was built with testability in mind and it has only improved with release of MVC4. This also means, as a TDD practitioner, you find it much easier to write Unit Tests against an ASP.NET MVC app.

In this article, I cover some of the key steps involved in developing an ASP.NET MVC application using TDD (Test Driven Development). I will use xUnit.NET (1.9v) as the Unit Testing framework and the Moq (4.0v) as the mocking/isolating framework.

Before I get into the TDD code sample, I would like to give you an overview of both xUnit.NET and VS 2012 RC from TDD point of view. In the second half of the article, I will focus on building a simple ASP.NET MVC application using TDD and xUnit.NET.

XUNIT.NET AND TDD

Brad Wilson, one of the co-authors of xUnit.NET framework comments on his blog. "We wanted xUnit.net to be a framework that was first and foremost for TDD (erm, DbE) practitioners."

As per Brad's comment, the xUnit.NET framework has been designed and developed to support TDD. This is great because xUnit.NET allows developers to move away from the QA/Test type mind set, and to purely concentrate on the design of the application. This is why xUnit.NET has attributes such as [Fact], and [Theory] etc. to support this notion. xUnit.NET also provides much cleaner test API's, which makes it a lot easier to focus on TDD itself. For example, xUnit.NET does not provide attributes such as [TestClass], [TestInitialize] or [TestCleanup], which you might have come across in frameworks such as MSTest. Instead, xUnit.NET uses the test class constructor, and Dipose() method

(In IDisposable) etc.

XUNIT.NET AND TDD TOOLING SUPPORT

Most TDD Developers who use xUnit.NET also use plugins such as TestDriven.NET and ReShaper etc. These tools in conjunction with Visual Studio makes TDD experience much better. Visual Studio 2012 (VS2012) also has a strong emphasis on TDD.

The VS2012 has a new Unit Test Explorer, which makes your Unit Testing/TDD experience much more interesting than before. The Unit Test explorer has no knowledge of any of the Unit Testing

> *"We wanted xUnit.net to be a framework that was first and foremost for TDD (erm, DbE) practitioners." ~ Brad Wilson*

frameworks. In order to make your Unit Test runnable within VS, you simply have to install your favorite test adapter. For the MVC-TDD demonstrations, I will use the xUnit.NET test adapter.

One of the key points that contribute to the productivity of a TDD Developer is the ability to seamlessly switch between the production code, Unit Tests and the Test Result. The new Unit Test Explorer has been designed to make you feel more connected between your production code, Unit Tests and the Test Result.

To use xUnit, you need to install the xUnit Test Runner from the Visual Studio 2012 Extensions Manager. Once installed, you should be able to see your xUnit.NET Unit Tests within Unit Test Explorer.

ASP.NET MVC-TDD APPLICATION

What we are going to build here is a very simple ASP.NET MVC app using TDD and xUnit.NET. The application manages list of cars in a car dealership.

For this demo, I'm going to use the new VS2012 RC. First we create a new ASP.NET MVC 4 project. Note that when you first create a new ASP.NET MVC project, you get the option to select your preferred Unit Testing framework. Since there is no xUnit.NET project template for VS2012 available yet, I'm going to skip this step and add a standard C# class library for the Unit Tests project. I will also reference xunit.dll within the Unit Test project.

Let's say we have a requirement to display list of cars.

Note that this is a simple example, but the TDD concepts you apply here remain same for larger applications. I'm going to add few files to the Solution Explorer as shown below. Note that we do not have any behavior implemented in these files yet.

Within the ASP.NET MVC project:

- CarDealershipController.cs
- Car.cs
- ICarDealershipRespository

Unit Test project:

- CarDealershipControllerTests.cs

The first step of TDD is to write the Unit Test. The first behavior that I would like to verify is:

The Action method "List" should return the View named "List"

Let's go ahead and write the Unit Test to verify this behavior.

```
[Fact]
public void List_WhenActionExecute_ReturnsViewNameL
{
    //Act
    var result = sut.List() as ViewResult;

    //Assert
    Assert.Equal<string>(result.ViewName, "List");
}
```

Note: This code assumes that you have the xUnit namespace and the System.Web.Mvc namespace imported accordingly.

The **[Fact]** attribute identifies the Unit Test in xUnit.NET.

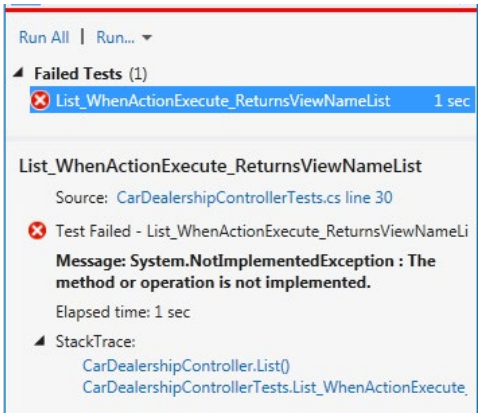
I divided my Unit Test method name into 3 sections separated by underscore '_' - the subject, scenario/condition, and the result.

This explains the exact intention of my Unit Test. Note that the List Action within the CarDealershipController does not have the implementation yet.

```
public ActionResult List()
{
    throw new NotImplementedException();
}
```

The next important step is to run the Unit Test. Some people think this is a waste of time since we don't have the implementation and it is obvious that the test is going to fail. It is important to start with a failing test because, we need to be sure that it was failing before, so we can write *enough* production code to pass the test.

Let's run this Unit Test and you can see the Test fails. This reflects the "RED" part of the RED, GREEN, and YELLOW in Test Driven Development.

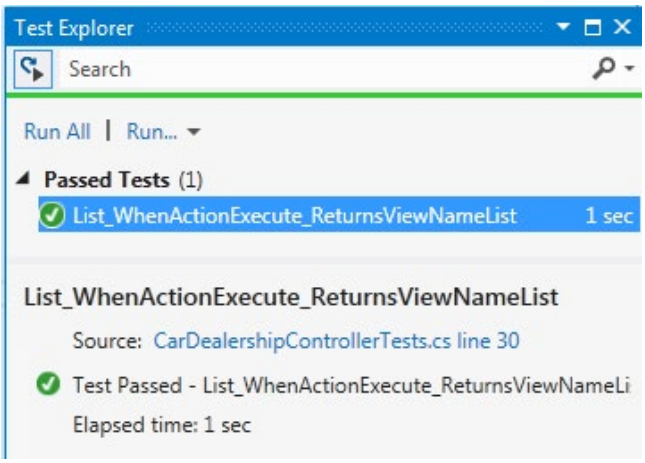


Now let's implement the minimum code that is required to pass our Unit Test.

```
public ActionResult List()
{
    return View("List");
}
```

Note that I haven't implemented the View "List". We are only concerned about the behavior of the server side code, therefore we do not need to include a View at this stage of the development.

Let's re-run the Unit Test. As you can see, this time the Unit Test passes.



This is the "GREEN" part of TDD. Now we are confident that the "List" Action executes and it returns the ViewName "List".

Refactoring for additional requirements

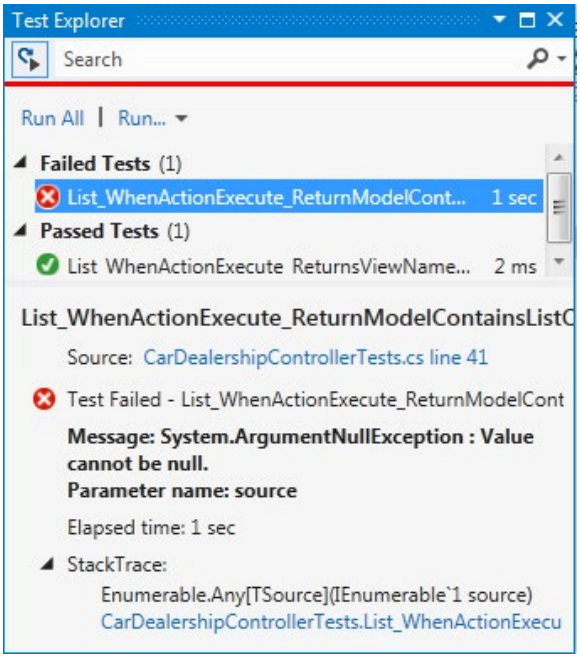
We haven't finished with the Car Dealership requirement yet.

The next step is to retrieve list of cars to the Controller's Action. Then this list can be used by the View. The behaviour that we want to verify is - When the Action Executes, the associated Model contains a list of cars.

Our Unit Test will look similar to the following:

```
[Fact]
public void List_WhenActionExecute_ReturnsViewNameList()
{
    //Act
    var result = sut.List() as ViewResult;

    //Assert
    Assert.Equal<string>(result.ViewName, "List");
}
```

Let's run this Unit Test. As we would expect, the Test fails. The exception suggests that there is no model associated (i.e. null ref exception) with the returned result.



As a TDD developer, I find myself getting into a code-test-code-test loop. As soon as I write a bit of production code, I want to make sure that I did not break any of my existing Unit Tests. Doing this task repetitively, especially with a large code base can be quite annoying. However VS 2012 can be configured to run Unit Test automatically after every successful build/compilation. To enable this, select "TEST" from the menu, and select Test Settings, and then "Run Tests After Build".

Our next step is to make the failing test pass. In order to do this, we need to refactor the Controller's Action. In TDD life cycle, this falls in to "YELLOW" which is the Refactor.

First I'll add some code to the interface ICarDealershipRepository. So we can consume the method GetCarList() within the CarDealershipController. At this stage, I do not require the implementation of this interface. Our aim is to just add enough code to pass the failing Unit Test.

```
public interface ICarDealershipRepository
{
    IEnumerable<Car> GetAllCars();
}
```

Now we change the Controller's constructor to accept the ICarDealershipRepository interface and call the method GetCarList() as shown here.

```
public class CarDealershipController : Controller
{
    private ICarDealershipRepository repository;

    public CarDealershipController(
        ICarDealershipRepository repository)
    {
        this.repository = repository;
    }

    public ActionResult List()
    {
        var cars = repository.GetAllCars();
        return View("List", cars);
    }
}
```

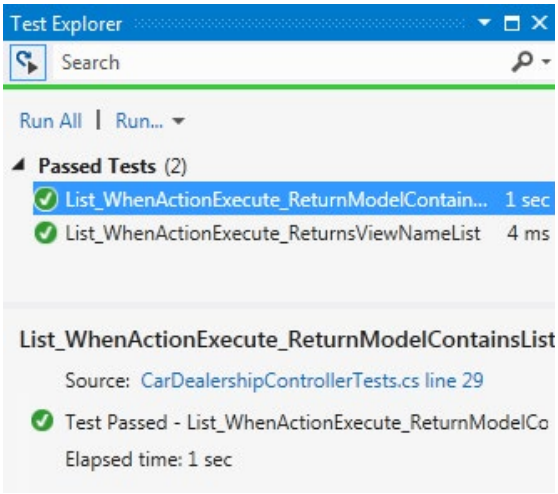
Notice that due to the new parameterized constructor, we are unable to compile our existing Unit Test. Our next step is to fix those tests. At the same time, we can refactor the existing Unit Tests for greater maintainability.

I'm going to introduce a constructor to the Unit Test class. xUnit.NET constructor can execute code that requires executing before each and every test. This is equivalent to [TestInitialize] attribute in MSTest. Some of the repeated code in each Unit Test can also be placed within the constructor.

In the beginning of the article, I mentioned about the Moq isolation framework. Using Moq, I'm going to configure/setup the CarDealershipRepository's GetCarList method to return fake data.

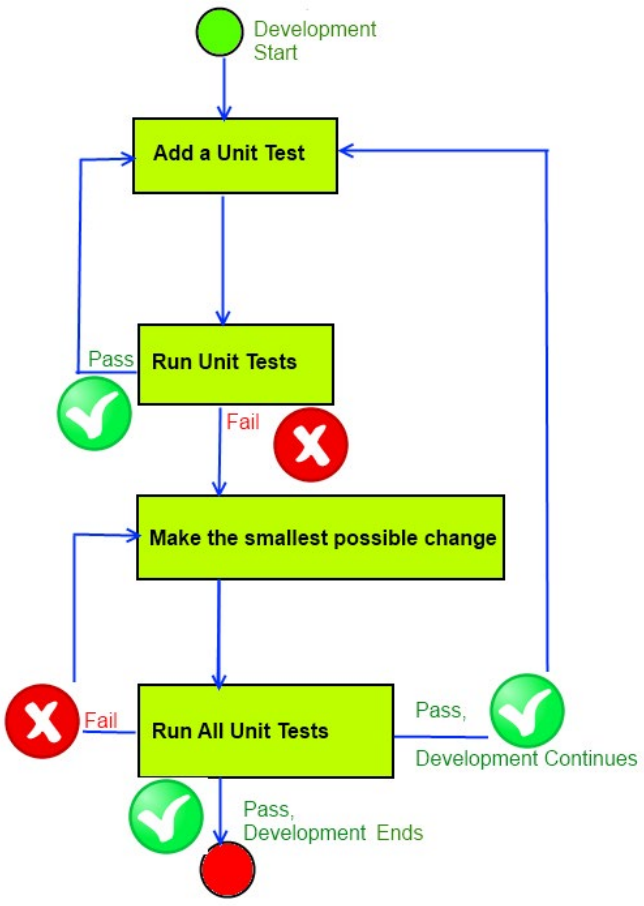
```
public class CarDealershipControllerTests
{
    private Mock<ICarDealershipRepository> repositoryStub;
    private CarDealershipController sut;

    public CarDealershipControllerTests()
    {
        repositoryStub = new Mock<ICarDealershipRepository>();
        sut = new CarDealershipController(repositoryStub.Object);
    }
    [Fact]
    public void List_WhenActionExecute_ReturnModelContainsListOfCars()
    {
        //Arrange
        repositoryStub.Setup(x => x.GetAllCars()).Returns
            (() => new List<Car> { new Car { } });
        //Act
        var result = sut.List() as ViewResult;
        //Assert
        Assert.True(((IEnumerable<Car>)result.Model).Any());
    }
```

Moq setup method on the repository is configured to return fake data to the Action List.



It is important to run all Unit Tests (at least in the same area) after every refactor. This is because we want to make sure the refactoring did not cause existing Unit Tests to fail. If any of the existing Unit Tests fails, we would refactor the production code with the smallest possible change to make the failing Test pass.

So far, we saw is a pattern where we add a Unit Test, write enough production code to pass that Unit Test, add a new Unit Test, refactor the code to pass the new Unit Test, and re-run all Unit Tests. If we can put this into a diagram, it looks like this



TDD WITH XUNIT.NET DATA THEORIES

In this example, we will look at xUnit.NET Data Theories and Unit Testing an Action Filter using TDD.

Let's assume we have a new requirement where we want to log certain messages depending on a flag returned by query string parameters. For instance, if the flag has a value "true", "1", or "yes", then we simply log a message. Sample query strings parameters would look like this.

/CarDealership/List?**log=true**
/CarDealership/List?**log=1**
/CarDealership/List?**log=yes**

To fulfil this requirement, I'm going to introduce a couple of classes.

LogFilterAttribute (An ActionFilterAttribute) - LogFilterAttribute overrides the OnActionExecutingMethod. There is no implementation for this method yet.

```
public class LogFilterAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(
        ActionExecutingContext filterContext)
    {
        throw new NotImplementedException();
    }
}
```

ILoggerService (This Interface contains a Log (message) method.) - ILoggerInterface has the Log method, which accepts the message to be logged.

```
namespace MvcTddDemo.Models
{
    public interface ILoggerService
    {
        void Log(string message);
    }
}
```

Note that we do not require the real implementation of this interface at this stage.

Unit Testing Action Filter:

Similar to our previous TDD sample, we first write the Unit Test as below. The skeleton of the Unit Test looks like this.

```
public class LogFilterTests
{
    [Theory,
    InlineData("true"),
    InlineData("1"),
    InlineData("yes")]
    public void OnActionExecuting_ForValidQueryStringVa
        (string queryStringValue)
    {
        //Arrange

        //Act

        //Assert
    }
```

This Unit Test is to verify, for a given set of query string parameters, whether the ILoggerService.Log(message) method is being called only once.

Notice that we also have a new attribute called [Theory]. If you have read Brad Wilson's, *'It is not TDD, its Design By Example'*, he explains the difference between [Theory] and [Fact] attributes.

The above Unit Test is parameterized. The parameter, 'queryStringValue', is passed into the Unit Test and contains the value provided by the InlineData attribute. Since we have three InlineData attributes, the test runner executes the Unit Test three times for each query string parameter.

Let's add some code to the Unit Test to satisfy the above expectation.

```
public class LogFilterTests
{
    [Theory,
    InlineData("true"),
    InlineData("1"),
    InlineData("yes")]
    public void OnActionExecuting_ForValidQueryStringValues_Ver
        (string queryStringValue)
    {
        //Arrange

        //Act

        //Assert
    }
```

As you can see in this Unit Test, I do not have the "stubFilterContext" and the "logServiceMock" being declared yet.

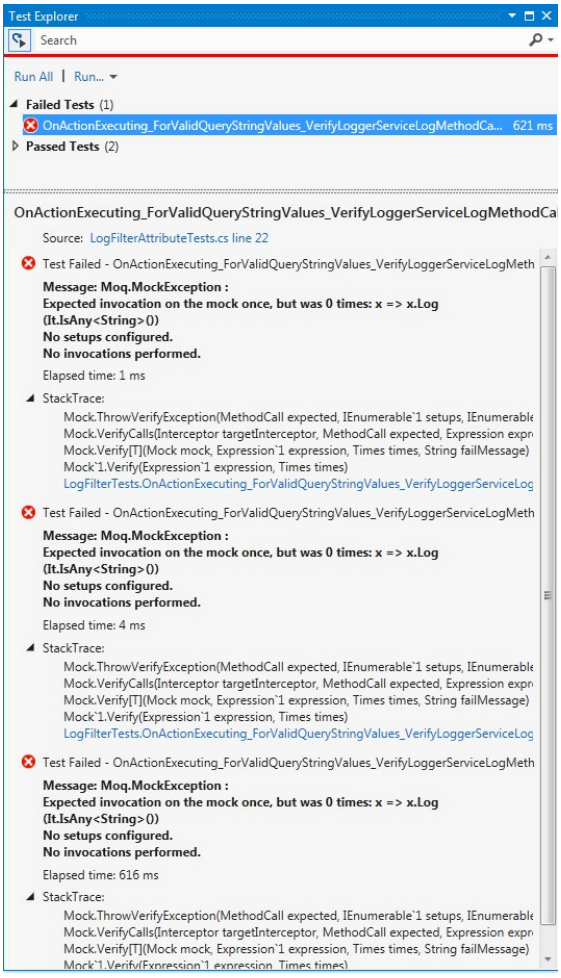
In order to stub the filterContext, which is the ActionExecutingContext, we need to stub few more instances. We also need a stub HttpContext, and a stub RequestContext so we can configure the stubbed instances to return fake query strings.

Finally we need to initialize the Mock<ILoggerService>, so we can perform the verification on the Log(message) method.

Here is the completed Unit Test.

```
var httpRequestBaseStub = new Mock<HttpRequestBase>();
httpRequestBaseStub.Setup(q =>
    q.QueryString).Returns(new NameValueCollection()
    { { queryStringKey, queryStringValue } });
httpContext.Setup(x => x.Request).Returns(httpRequestBaseStub.Object);
var actionDescriptor = new Mock<ActionDescriptor>().Object;
var controller = new FakeController();
var controllerContext = new ControllerContext(
    httpContext.Object, new RouteData(), controller);
var stubFilterContext = new ActionExecutingContext(
    controllerContext, actionDescriptor, new RouteValueDictionary());
var loggerServiceMock = new Mock<ILoggerService>();
sut.LoggerService = loggerServiceMock.Object;
//Act
sut.OnActionExecuting(stubFilterContext);
//Assert
loggerServiceMock.Verify(x => x.Log(It.IsAny<string>()), Times.Once())
```

We are still unable to compile and run this Unit Test as there is no ILoggerService property within SUT. Let's create the property in the SUT so we can run the Unit Test. The Test Results are below.



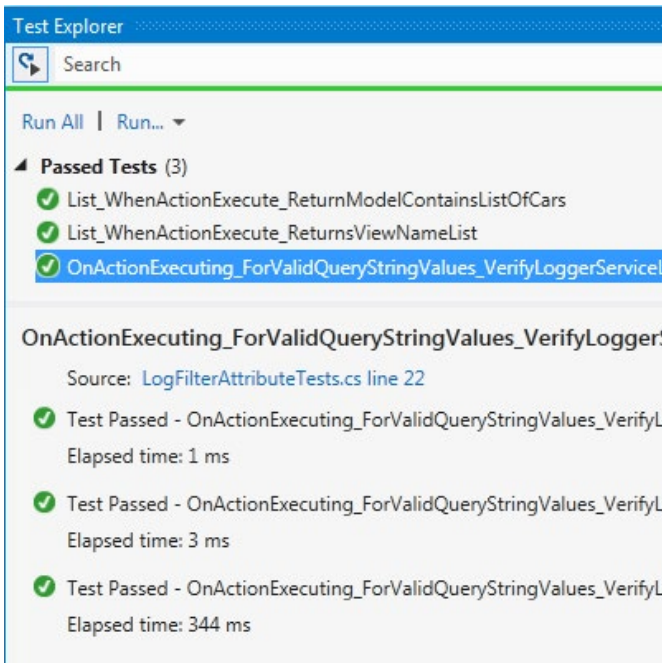
As you would expect, the Unit Tests fail due to three InlineData

attributes.

Our next task is to just add *enough* production code to make these tests pass.

```
public class LogFilterAttribute : ActionFilterAttribute
{
    public ILoggerService LoggerService { get; set; }
    public override void OnActionExecuting(
        ActionExecutingContext filterContext)
    {
        var queryString = filterContext.RequestContext
                .HttpContext.Request.QueryString;
        string value = queryString["log"];
        switch (value)
        {
            case "true":
            case "yes":
            case "1":
                LoggerService.Log("Some message to log");
                break;
            default:
                break;
        }
    }
}
```

Let's run those failing Unit Tests.



All Unit Tests are now passing including the ones I wrote before. The TDD life cycle continue as I continue to add new tests to design the behaviour of this method. For example, I can add new Unit Test to verify the design of the code that handles invalid query string arguments. Then write a Unit Test, make the test fail, and make the smallest change/refactor to make the Test pass. Finally re-run all Unit tests and ensure they all pass.

CONCLUSION

In this article, we looked at some of the key concepts behind building an ASP.NET MVC application, using TDD and xUnit.NET. We also learned some of the TDD friendly features that xUnit.Net offers. We looked at some of the enhancements that VS2012 RC provides in order to make TDD experience seamless.

We explored how to Unit Test an Action method using TDD and xUnit.NET. We also looked at xUnit.NET DataTheory/InlineData attribute, and used TDD to Unit Test an ActionFilter. ■

Raj Aththanayake is a Microsoft ASP.NET Web Developer specialized in Agile Development practices such Test Driven Development (TDD) and Unit Testing. He is also passionate about technologies such as ASP.NET MVC. He regularly presents at community user groups and conferences. Raj also writes articles in his blog http://blog.rajsoftware.com. You can follow Raj on twitter @raj_kba


Follow us on
Twitter
@dotnetcurry

CLIENT-SIDE TEMPLATING
With JsRender

Minal Agarwal explores the JavaScript library JsRender for creating HTML templates.

DOWNLOAD FILES >
bit.ly/dncmag-jsren

Developers familiar with server side technologies like WebForms on ASP.NET can easily relate to 'controls' like GridView and ListView. These 'controls' are very useful for rendering repeating data; for example a List for a single dimensional array and a Grid for two-dimensional array of data. These 'controls' have a feature wherein at design time, we can specify how a single cell or row of data should look and behave, and at runtime, with real data, the server can render the entire Grid as per the specified template. Essentially the design and behavior is replicated, for each iteration of data.

However with the popularity of JavaScript and the general movement away from server side rendering to client side rendering, we have lost the benefit of server side templating and are reduced to writing string concatenation routines in JavaScript.

Essentially without templates, we have to do a lot of inline JavaScript, looping over datasets.

JsRender, a brief history

With the rise of JavaScript, came some phenomenal JavaScript libraries viz. jQuery, Dojo and others. Templating was a problem readily recognized and towards that end came the jQuery-Template library. However development on jQuery-Template was paused after beta and eventually it was dropped from being considered as possible templating plugin due to a fundamental direction-change on templating requirements set forth by the jQuery UI team.

However, Boris Moore the chief contributor continued on a derivation of jQuery-Template and split up the implementation into JsRender (templating) and JsViews (View Model and Observables behavior). As of February 2012, JsRender is now in Beta, showing great promise of significant performance + feature when compared to other jQuery templating plugins like Handlebar, Strappend and Mustache.

Features

With the history out of the way, let's jump straight into some basic usage. Let's take an example of a User Identity object that we need to render. The Identity has the following structure

Name : <String>
Email : <String>
Gravatar: <String calculated from Email>
Address : <Object>
    Number: <String>
    Address Line 1: <String>
    Address Line 2: <String>
    City : <String>
    State : <String>
    Zip : <String>

Now let us say we have a list of such Identity objects in Json.

The data would be as follows:

```
$("#addContactPopup").hide();
var addList = [{
    "Name": "Minal Agarwal",
    "Email": "minalsagarwal@gmail.com",
    "Gravatar": "http://www.gravatar.com/a
    "Address": {
        "Line1": "1234",
        "Line2": "Obfuscated Lane",
        "City": "Nice City",
        "State": "Maharashtra",
        "Zip": "411000"
    }
},
{
    "Name": "Suprotim Agarwal",
    "Email": "suprotimagarwal@isobfuscatec
    "Gravatar": "http://www.gravatar.com/a
    "Address": {
        "Line1": "9999",
        "Line2": "Obfuscated Lane",
        "City": "Nice City",
        "State": "Maharashtra",
        "Zip": "411000"
    }
}];
```

To render this data without templating, our approach would be to have an empty container in the markup and JavaScript text injection in a separate JavaScript file. Or embed the script inside a <script> </script> section in the markup itself

Markup (Approach 1)

```
<div id="contactListContainer"
    style="list-style: none; border: 1px solid #F0F0F0;">
</div>
```

JavaScript to Render

```
function renderAddresses(dataList) {
    var el = $('#contactListContainer');
    $(dataList).each(function () {
        var data = this;
        el.append(
        '<div style="display:inline-block">' +
            '<div style="display:inline-block; ' +
                'border: 1px solid #F0F0F0; margin-bottom:5px;' +
                    ' padding:10px; width:400px;">' +
                '<div>Name: ' + data.Name + '</div>' +
                '<div>Address: ' +
                '<div style="padding:5px">' +
                    '<div>' +
                        data.Address.Line1 + '</div>' +
                    '<div>' +
                        data.Address.Line2 + '</div>' +
                    '<div>' +
                        data.Address.City + '</div>' +
                    '<div>' +
```

As we can see here, the script (that was supposed to have render logic only) now has a big set of markup as well. This kind of spaghetti development soon becomes unwieldy and large for a UI with even moderate complexity.

This is where templates come into play. In the next section we see how to render the above with JsRender templates and understand the templating syntax.

Sample – A Contact List using JsRender

Note: This sample takes off from where Sumit Maitra's Gravatar article on www.devcurry.com left. The original article takes user input for a contact sheet and looks up their Gravatar. We have modified the data being collected to highlight features of JsRender.

Before we get into the Gravatar part, let us simply render our sample data using JsRender. To do that, we update the markup as follows:

```
<div id="contactListContainer" style="display: inline-block;">
</div>
```

This div will contain the output that comes from after the data has iterated through the template below.

```
<script id="contactListTemplate"
 src="~/Scripts/jsrender.js" type="text/x-jsrender">
<div style="display:inline-block">
    <div style="display:inline-block;
        border: 1px solid #F0F0F0; margin-bottom:5px;
             padding:10px; width:400px;">
        <div>Name: {{:Name}} </div>
        <div>Email:  {{:Email}} </div>
        <div>Address:
        <div style="padding:5px">
            <div> {{:Address.Line1}} </div>
            <div> {{:Address.Line2}} </div>
            <div> {{:Address.City}} </div>
            <div> {{:Address.State}} </div>
            <div> {{:Address.Zip}} </div>
        </div>
    </div>
</div>
</script>
```

This is the syntax for specifying the JsRender template. Let us look at it in more details.

Understanding the Basic JsRender Templating Syntax

We can see in the markup above, we have some syntax that is not standard HTML. This is the special JsRender syntax. Let's take it apart line by line

- Wrapping the template in Script: We see our template for showing a contact is wrapped in a Script area with it's src set to the JsRender JavaScript and the type set to a special type of "text/x-jsrender". This tells JsRender engine to treat this element specially.

- Template tokens {{ : }} - This is the fundamental JsRender token. It indicates to the library that it should try to 'interpret' the contents inside these tokens.

- Colon (:) = No Encoding - This retrieves the data-bound value from the object and shows the text as is (it will render HTML content in the text if any). If we don't want to render HTML and spit out the raw html markup (if any), we can use the {{>}} syntax.

- Access inner objects by name: If we look at the Address object that has child elements, JsRender is accessing the child elements using the dot notation (which is common in languages as C# and Java). So binding to nested object graphs is simply a matter of traversing the graph using the dot notation. Apart from the dot notation, JsRender also supports named properties so we could use {{ : #data['name']}} to get the name also where #data gives us the current data context.

With the basics out of the way, let's take a look at what happens to our client script that earlier had the spaghetti of code and JavaScript? It looks as follows now:

```
renderAddressesTemplate();

function renderAddressesTemplate() {
    $('#contactListContainer').html(
        $('#contactListTemplate').render(addList)
    );
}
```

As we can see above, all the spaghetti of markup and JavaScript is now gone. All we have is a couple of lines of code. Thus we have separated the script from the HTML markup.

It works as follows:

- First we grab the container object in our template output. In our case, it's an empty with the Id as contactList Container (as seen above).
- Next we pick the name of the template 'script' we saw earlier and call the render method on it. The render method is what invokes JsRender.
- Finally render method is passed the data source against which the binding should happen (addList in our case).

Once this hookup is complete, our sample JsRender app is ready to go. It renders the two addresses in the Array.

Additional JsRender Tokens

Apart from the above basic tokens, JsRender supports more tokens for rich templating functionality.

- The {{for}} token – Looping through a sub-array of objects in the data source is handled using the {{for}} token. For example, if our original Identity source had the option of multiple addresses instead of one, then the Addresses accessor would give us an array of addresses and we would have to loop through all the available addresses.

```
<script id="contactListTemplate" src="~/Scripts/jsren
    type="text/x-jsrender">
<div style="display:inline-block">
    <div style="display:inline-block;
        border: 1px solid #F0F0F0; margin-bottom:5px;
             padding:10px; width:400px;">
        <div>Name: {{:Name}} </div>
        <div>Email:  {{:Email}} </div>
        <div>Address:
        {{ for addresses }}
        <div style="padding:5px">
            <div> {{:#data.Line1}} </div>
            <div> {{:.Line2}} </div>
            <div> {{:.City}} </div>
            <div> {{:.State}} </div>
            <div> {{:.Zip}} </div>
        </div>
        {{/for}}
    </div>
</div>
```

As we can see above, we are looping through the Addresses available in our each of the Identity objects in our data source using the {{for [child collection] }} ... {{/for}} tokens

#data actually is the current context. So in the above example, #data.Line1 and simply Line1 would mean the same. However if address was a collection of strings only, then we wouldn't be able to say #data.something. In such cases, only specifying

#data would retrieve the string data in current context.

- The {{if}} {{else}} Tokens : As is obvious, these tokens help evaluate conditions and thus render content conditionally based on data being bound. We add the following to our template to check if the Gravatar property is set in the data. If it is set, we add an image element. If not set, we don't do anything

```
<div style="display:inline-block">
    {{if Gravatar}}
    <img src='{{:Gravatar}}' alt='Gravatar' />
    {{/if}}
</div>
```

As we will see here, this will show the Gravatar image of new contacts as we add them.

That sums up the core tokens in JsRender. We will now complete our Contact List Application using JsRender templates instead of the inline markup + JavaScript we had earlier.

Completing the Contact List Application

- We update the 'New Contact' popup for the additional fields

```
<div style="padding: 10px">
    <div style="display: inline-block; width: 200px">
        Address
    </div>
    <div style="padding-left: 50px">
        <div>
            <span style="width: 200px">Line 1 </span>
            <input type="text" name="Line1" id="Line1" value=" " />
        </div>
        <div>
            <span style="width: 200px">Line 2 </span>
            <input type="text" name="Line2" id="Line2" value=" " />
        </div>
        <div>
            <span style="width: 200px">City </span>
            <input type="text" name="City" id="City" value=" " />
        </div>
        <div>
            <span style="width: 200px">State </span>
            <input type="text" name="state" id="State" value=" " />
        </div>
        <div style="display: inline-block;">
            <span style="width: 200px">Zip </span>
```

- Next we update the JavaScript to push the newly added data into our data source by calling the contactAdd(data) method.

```
function contactAdd(data) {
    var gravatarUrl = calculateGravatar(data.Email);
    data.Gravatar = gravatarUrl;
    addList.push(data);
    renderAddresses();
}
```

o We first calculate the GravatarUrl using the calculateGravatar call.
o We assign the Gravatar URL to the Gravatar property
o We add the data in our data source object 'addList'
o Finally we call renderAddress() as defined above to re-render the UI with the new Data.

With that we have covered the most often used JsRender tokens and their usage. Our sample application is now complete. To round off, we will look at a technique where JsRender templates can be compiled and kept for future use.

Compiling and Reusing Templates

JsRender templates can be compiled and kept away as strings and then invoked for rendering on demand. This gives a performance edge and also allows better reuse and separation of the template and the html markup.

The syntax for compilation is as follows

```
var templateString = '<div>{{:myVariable}}</div>';
var compiledTemplate = $.templates(templateString); //this compiles the template
$('#templateContainer').html(compiledTemplate.render(data));
```

Another way of compiling a template is to use named-templates

```
// templateLayout is the named template
$.templates('templateLayout','<div>{{:myVariable}}</div>');
// templateLayout can be rendered as follows
$.render.templateLayout(data);
```

We should use compiled templates wherever possible primarily because of performance gains as well as the fact that it enables better separation of concerns.

With that we conclude our introduction to JsRender and its Templating functionality. Building data intensive UIs just got a whole lot easier. ■

Minal Agarwal, Expression Web MVP, works as a freelance web designer (SaffronStroke) working on Expression Web, CSS, HTML5, UX, Photoshop and other Graphical tools. Follow her on twitter @saffronstroke

DI USING NINJECT IN ASP.NET MVC

Sumit Maitra walks you through the basics of Dependency Injection and how to use an Inversion of Control (IoC) container – Ninject in a sample ASP.NET MVC Web application

DOWNLOAD FILES >
bit.ly/dncmag-ioc

Over the last several years, as software development processes have matured, a number of best practice postulates have been realized and then formalized. One of the famous ones goes by the acronym SOLID proposed by Robert C. Martin (fondly known as Uncle Bob Martin). SOLID stands for Single responsibility, Open closed principle, Liskov substitution principle, Interface segregation principle and Dependency Inversion.

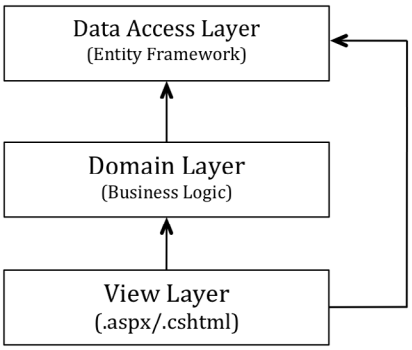
Dependency Injection is a way to achieve Dependency Inversion and Inversion of Control (IoC) containers are frameworks that help us implement Dependency Injection. Today, we will look at Ninject (an IoC container) in a sample ASP.NET MVC Web application and understand the basics of DI using IoC.

Before we jump into Ninject and Dependency Injection, let us quickly run through the basics of DI. They are:

1. All dependencies of a class should be passed (injected) into the class instead of being instantiated directly. In other words, avoid new-ing up of interdependent layers.
2. Code to Interfaces instead of concrete implementations.
3. All Class instantiation and object lifecycle management should be done at a central location referred to as the 'Composition Root'.
4. Inversion of Control (IoC) Containers refers to frameworks that help manage the object instantiation and lifecycle.
5. You can do DI without IoC containers. In such cases, the pattern is referred to as 'poor-man's-DI' and you have to hand code the object instantiation.

A SIMPLE APPLICATION

For the purposes of our discussion, we will take a simple three-layer application with one Entity on which we will be doing Create Read Update and Delete (CRUD) operations. A standard layering scheme looks like the following

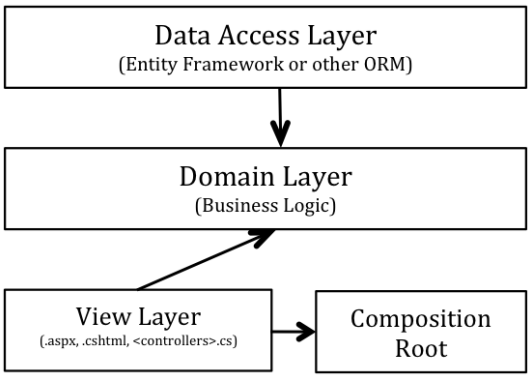


But in such a scheme, each layer instantiates the layer above it and the View layer access the Data Access Layer too. This is a classic example of a hard coupled system.

Instead of the above, if we define the Data Access Interfaces in our Domain layer and implement those interfaces in the Data layer, the dependencies would get inverted and Data Layer would then be dependent on the Domain layer. The View layer continues to refer to the Domain layer. Since View layer doesn't have access to EntityFramework or the Data Access Objects, how does it deal with the data? We define a layer of Plain Old CLR Objects (POCOs) in the Domain Layer. These POCOs are our Data Transfer Objects (DTOs) between data and Domain Layer.

Since the Domain layer only defines Interfaces to the data layer , (aka Repository Interfaces) there is nothing to new up in the View layer. The concrete instance of the Domain layer is passed in to the Domain layer. The concrete instance is created in the Composition Root.

Now our application layering looks as follows:



HOW DOES THIS TRANSLATE INTO CODE?

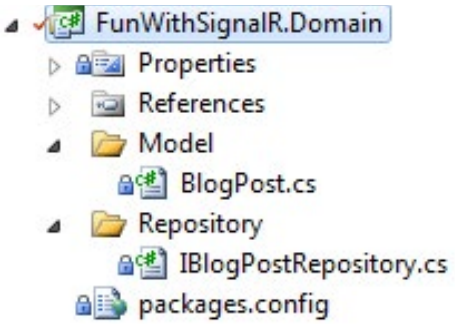
Well the sample application has the following structure.

THE DATA LAYER

This layer has

- POCOs used for serialization/deserialization of data from DB, using EntityFramework. In case of other OR/M, it will contain the Data Objects required by the OR/Ms
- The concrete implementation of the Repositories defined in the Domain Layer
- The translation layer from Domain object to Data object and vice versa.

THE DOMAIN LAYER

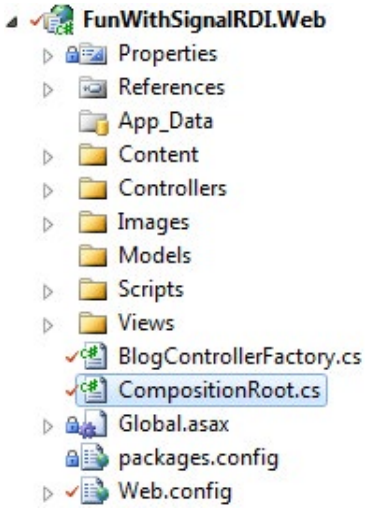


This layer has

- The Interface definitions for all data access. Only the interfaces that are defined here, is accessible to the view layer
- The Domain POCOs
- A Service Layer to manage business rules if any.

THE VIEW LAYER

- The view markup
- The rendering logic
- The Composition Root. For purposes of simplification, the Composition Root is shown in the View layer but as we will see later, this gives rise to a coupling that we will eventually have to break.



A CLOSER LOOK AT THE COMPOSITION ROOT

Before we jump into an IoC container, lets take a closer look at the Composition Root.

```
public class CompositionRoot
{
    private readonly IControllerFactory
        _controllerFactory;

    public CompositionRoot()
    {
        IBlogPostRepository repository =
            CompositionRoot.InitializeRepository();
        _controllerFactory =
            new BlogControllerFactory(repository);
    }
```

The Composition Root is essentially determining the concrete instance (SqlBlogPostRepository) for an interface (IBlogPostRepository), instantiating the concrete instance.

```
string blogRepositoryTypeName =
    ConfigurationManager.AppSettings["BlogRepository
string connectionName =
    ConfigurationManager.ConnectionStrings[0].Name;
string schemaName =
    ConfigurationManager.AppSettings["schemaName"];
var blogRepositoryType =
    Type.GetType(blogRepositoryTypeName, true);
var repository =
    (IBlogPostRepository)Activator.CreateInstance(
        blogRepositoryType, new object[]
        {
            connectionName,
            schemaName
        });
return repository;
```

Next the code is fetching an instance of IControllerFactory that is responsible for generating all the controllers required in the MVC application. The Composition Root is also injecting the repository into the Controller Factory instance, thus making the Repository available for use in the View layer.

For a simple application as this sample, this discovery instantiating and injection looks trivial enough, but imagine a project with multiple controllers and repositories and other related dependencies, trying to track all of those manually and wiring them in the composition root will soon become a nightmare and devs in the team will start trying to hack their way around. This is where Inversion of Control Containers (or IoC Containers) comes into picture. IoC containers when configured to map an interface to a concrete type, can generate a new concrete instance whenever required. This automates the process of dependency wiring. As long as everyone is referring to Interfaces and the IoC container knows how to get a concrete class for that Interface, we are good. Ninject is once of the newer and popular IoC containers. It's an open source project that is actively worked upon and has a vibrant community.

GETTING STARTED WITH NINJECT

Now that we have seen the code factored to use Poor Man's DI, we will see how we can nicely transition to use Ninject and solve the challenges on the way.

INSTALLING NINJECT

- Select the web project
- Open the (Nuget) Package Manager Console and use the following commands

```
Install-package Ninject
Install-package Ninject.MVC3
```

This installs the required dependencies for Ninject in an MVC application.

'CONFIGURING' NINJECT TO BE YOUR IoC CONTAINER

Step 1: Open Global.asax and change the subclass from System.Web.HttpApplication to Ninject.Web.Common.NinjectHttpApplication

Step 2: Override the 'CreateKernel' method from the NinjectHttpApplication class in Global.asax and add the following

```
protected override IKernel CreateKernel()
{
    var kernel = new StandardKernel();
    kernel.Load(Assembly.GetExecutingAssembly(),
        Assembly
            .Load("FunWithSignalR.Domain"),
        Assembly
            .Load("FunWithSignalR.Data")
        );
    return kernel;
}
```

Step 3: Mapping the Dependencies.

- Add reference to FunWithSignalR.Data to your composition root (currently this is the web project. Don't worry about the seemingly backwards dependency coupling, we will decouple it later).
- Create a new Class called DependencyMapper and inherit it from NinjectModule
- Override the Load() method and map the IBlogPostRepository to SqlBlogPostRepository as follows:

```
public class DependencyMapper : NinjectModule
{
    public override void Load()
    {
        this.Bind<IBlogPostRepository>()
            .To<SqlBlogPostRepository>()
            .WithConstructorArgument("connectionName",
                ConfigurationManager
                    .ConnectionStrings["FunWithSignalR"]
                        .ConnectionString)
            .WithConstructorArgument("schemaName",
                ConfigurationManager
                    .AppSettings["schemaName"]);
    }
}
```

This code tells Ninject that whenever IBlogPostRepository is requested for, instantiate SqlBlogPostRepository with the given arguments in its constructor.

This method is where we will map other dependencies when our project grows in size.

- Note how we pass constructor arguments to Ninject so that Ninject can use them at the time of type initialization.
- Setup the Application Start. Comment out the Application_Start() event's code and add an override for the OnApplicationStarted method in the Global.asax

```
protected override void OnApplicationStarted()
{
    base.OnApplicationStarted();
    BundleTable.Bundles.RegisterTemplateBundles();
    AreaRegistration.RegisterAllAreas();
    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
    RegisterDependencies();
}
```

If you run your application now, it will be up and running fine using Ninject IoC Container! W00t!

That covers up our migration from Poor Man's DI to using Ninject as your IoC Container in MVC.

For a sample project like ours, the benefits seem negligible but for larger projects having an IoC Container, the benefits goes a long way in helping manage object instantiation, Interface based development and therefore loosely coupled apps.

BUT... YOU JUST BUNCHED ALL THE REFERENCES INTO THE VIEW LAYER!

Yes, you are right, and it's a valid objection. If using IoC Containers and DI is the holy grail of decoupled application development, we just shattered that concept. The answer is rather simple.

> *"...If IoC Containers and DI is the holy grail of decoupled application development..."*

The 'Composition Root' of your application HAS to have access to all possible dependencies. Come to think of it, it's a reasonable expectation. If it is going to be responsible for wiring up Interfaces to Concrete instances. It SHOULD know where to find the concrete instances.

IoC Containers support various 'discovery modes' for concrete instances. They are roughly bunched into

- XML Configuration
- Configuration via code conventions
- Configuration via code

What we saw was Configuration via code. Ninject does not support an XML Configuration directly. The Extensions project help in that regard. We are not looking at these extensions today.

To solve our problem of having all references in the View layer, we will create a 'Bootstrap' project and add all references to that project. Then call the bootstrap from our Application start up. This way we can avoid the referencing issues.

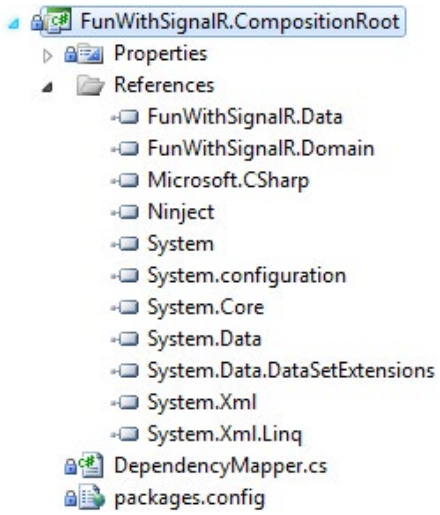
MOVING 'COMPOSITION ROOT' OUT OF THE WEB PROJECT

To solve our architectural issue of View layer referring to Data layer directly, we split the code up as follows:

1. Add a new ClassLibrary project and call it FunWithSignalR.CompositionRoot

2. Remove the default Class1.

3. Remove EntityFramework, FunWithSignalR.Data dependencies from the FunWithSignalRDI.Web project and add FunWithSignalR.Data and FunwithSignalRDI.Domain to the FunWithSignalR.CompositionRoot.



4. Add Ninject dependencies to the CompositionRoot project by using the following command

```
install-package Ninject
```

Note you don't need Ninject.MVC3 package here.

5. Move DependencyMapper.cs to the CompositionRoot project and update the Namespaces are required

6. Add references to FunWithSignalR.Data, FunWithSignalR. Domain to the CompositionRoot

```
protected override IKernel CreateKernel()
{
    var kernel = new StandardKernel();
    kernel.Load(Assembly.GetExecutingAssembly(),
        Assembly
            .Load("FunWithSignalR.Domain"),
        Assembly
            .Load("FunWithSignalR.Data"),
        Assembly
            .Load("FunWithSignalR.CompositionRoot"));
    return kernel;
}
```

7. Add reference to System.Configuration.dll to the CompositionRoot.

8. Back in the Global.asax of the Web project, update the CreateKernel() method to include the CompositionRoot.

9. Remove the old BlogControllerFactory and CompositionRoot classes from the Web project.

10. Run the application. Voila! We have Ninject + ASP.NET MVC going like a charm.

CONCLUSION

We just scraped the surface of IoC Containers, specifically Ninject. We just resolved our repository types using the container. We could potentially extract interfaces for our Domain Entities and get the container to resolve them for us too.

Utility of the IoC Containers becomes quickly evident in very large projects.

For our well-experienced readers hope this was a sufficient 'quick-summary'. For our readers getting into DI and IoC, we hope to have provided you with enough nudge to first adopt DI and next use IoC containers as a regular development practice.

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 http://bit.ly/KZ8Zxb

Both these people are deploying tomorrow...

sql CONNECT

The Visual Studio add-in that helps develop and version control databases

Develop databases alongside application code, and track and share all changes through your source control system, all within Visual Studio.

www.red-gate.com/products/sql-development/sql-connect

redgate software

Quick Bytes

Name	Oren Eini (a.k.a Ayende Rahien)
Born	December 20, 1981
Occupation	Founder - Hibernating Rhinos
Last Book Read	Read six of them the past weekend (Percy Jackson mostly)
Favorite Sites	Mostly news sites, through rss
Twitter	@ayende
Online	www.ayende.com

Interview with Ayende Rahien

We don't often come across alpha developers who work tirelessly slinging awesome code; help the community almost 24x7 and find time to blog about their coding adventures. My fellow geeks, the DNC magazine is extremely happy to bring to you, in it's launch edition, a freewheeling interview with Oren Eini (aka Ayende Rahien), the man behind 'Hibernating Rhinos' and 'RavenHQ'. HR has produced several very successful products like the NHibernate Profiler, the EF Profiler, the LINQ to SQL Profiler and RavenDB.

DNC: So Oren... err... or Ayende?

OE: My name is Oren Eini, while I was in the army, I wrote a technical blog under the nickname Ayende Rahien. That lasted for a couple of years, and I was really excited about being out of the army and being able to tell everyone who I am. Unfortunately, the general response was: "Yawn, Ayende, now get back to coding".

DNC: Ah, so that's the mystery! Thanks for clearing that up for us. Let's get started with an easy one, how did you start writing software?

OE: I remember playing with the Logo turtle at around 10 years old or so, and being suitably impressed when looking at 8" floppies (not a typo).

At high school, I started playing around with code, building Windows applications, and getting to know Web Development through ASP. My pride and joy at the time was a single page forum system that was implemented through a 15,000 lines of VBScript code with the "nested switch statements" design pattern.

At some point, I decided that if I like this so much, I might as well get good at this, and I took a course in C++ development. I credit this course with giving me the low level understanding of how machines really work, and that let me pick up .NET when it came out fairly early.

Since then, I have been practicing coding on a daily basis, it is fun.

DNC: '...practicing coding...' is a very modest way of putting all the awesome stuff you are doing, and we will come back to that. Lets digress for a moment and imagine what would you be doing if you were not writing software?

OE: Writing, probably. I am not too good at that, but I wrote several novella size stories in the past, and I routinely play with plot lines in my head for a bunch of different stories. If I weren't into software, I would probably try writing SciFi or Fantasy books.

DNC: We sure hope someday we have a SciFi series from you...Coming back to development; use of the .NET platform, is it something that 'just happened' or was there something you found really interesting that drew you to .NET and C# as your platform of choice?

OE: As I mentioned, I did a lot of C++ work. And when .NET alpha came out, I decided to try this out.

I wrote the following application:

```
using System.Windows.Forms;

public class MyForm : Form
{
}

public class Program
{
    static MyForm form;
    public static void Main()
    {
        form.Show();
    }
}
```

When I tried to run it, I got an error. I decided that .NET is alphaware crap, and that it doesn't deserve any attention from me.

A few months later, I actually tried it again, wrote the same code, and got the same error. Then I actually looked at what this code was doing. I was getting a NullReferenceException because I didn't create an instance of MyForm.

Coming from C++, I expected this to work, since it was a direct reference, not a pointer. When I realized what was wrong, I looked at the error a lot more closely. I then wrote the exact same code (including the same error) in C++ and looked at the error that gave me. Comparing "Unhandled exception at 0x77a015de" from C++ to the NullReference Exception in C#, which included stack trace and line numbers, I knew what platform I wanted to work on.

Since then, I have done just about everything that you *can* do on the platform, from building toy applications to enterprise wide system. I had to face some challenges, but overall, I am very happy with the choice.

DNC: Well, we are glad you took a second look at .NET and things turned out as they have. Moving on, tell us a little bit about how Hibernating Rhinos (HR) came about?

OE: In 2005, I started working for a consulting group in Israel. The job was interesting, the people were great and I had a lot of fun working there. I learned a lot, both technical stuff and product stuff.

The problem started towards the second half of 2007. I was put on a Microsoft CRM project. That was an extremely painful experience, and I decided that I would rather not subject myself to such agony again. The company I was working for at the time had a lot more MS CRM projects, so we parted ways in the beginning of 2008.

I started doing consulting on my own, and for a while it was really great, but I got tired of being in every project for just a few weeks or months at a time. That wasn't a good way to actually do a project from start to finish, and I found that I was missing that.

At the same time, going to all those different customers gave me a good perspective about the kind of issues that they were facing, and from this experience, the idea for creating a Profiler for Hibernate came out.

Eventually, that became the NHibernate Profiler, and Hibernating Rhinos was founded as a product company, rather than a consulting company.

Since then, we have created 4 more profilers (for Linq to SQL, Entity Framework, Hibernate – java, and LLBLGen Pro) and, of course, we have created RavenDB.

> *"I decided that .NET is alphaware crap, and that it doesn't deserve any attention from me, until a few months later..."*

DNC: Nice, would you be able to walk us through a rough timeline of how each product in the HR portfolio came into existence? For example we know Raven DB came around from your desire to learn more about document databases and not finding a good .NET solution, the other products like EF Profiler or RhinoServiceBus, how did these come about?

OE: There is some distinction between commercial products such as the profilers suites and even RavenDB and our open source projects like the Rhino Service Bus.

All of the Rhino projects (Rhino Mocks, Rhino Service Bus, Rhino ETL, Rhino Security, etc) are actually fully OSS projects. They were written because they were needed for the customer projects that I was working at the time, and they allowed us to rapidly share code and approaches between different projects.

They aren't products, in the sense that they aren't supported / paid projects.

In contrast, the suite of profilers came about because I got frustrated with the complexities of explaining NHibernate's operations to customers, and because I thought that I could actually codify a lot of my understanding about OR/M usage and best practices.

Six months after the NHibernate Profiler came out; we started working on additional profilers, first to target Java's Hibernate - since that was a natural progression, and then Linq to SQL and Entity Framework.

Building the profilers was a lot of fun and I had a blast doing so, but all the same, I kept looking at other things to do. NoSQL already caught my imagination very early on.

I had written a distributed key/value store (based on the Dynamo model) already – Rhino DHT and I played around with some ideas for a while, based on improvements that I wanted to make to the CouchDB model.

For a while, I resisted the urge to create the project. Instead, I vented out my creative ideas on a series of blog posts, detailing how I would approach building a document database. But in the end, I realized that I couldn't really get anything done.

RavenDB was constantly in my head, and I had to get it out. So we sat down and created a business plan. It was obvious that we were talking about a significant time & money investment, not a hobby project. Several years later, I can say that I am both proud and happy with the way RavenDB has taken off and the shape it is in.

DNC: Yup, we can't feel enough good about RavenDB ourselves. It's super cool. From a Platform+Market point of view, what are the gaps in the .NET space that you would like to see filled (if any)?

OE: It is hard to paint the entire .NET space in a single stroke. From the point of view of the framework itself, I think that it is a masterpiece of engineering. I have spent a lot of time digging in, and I am continuously impressed with the framework, how it is put together and what we can do with it.

That isn't to say that I don't have my pet peeves, of course. For building servers (for example, a database server like RavenDB), it turns out that you want greater control over the way the system behaves, and it isn't easy or possible to modify some of the basic behavior of the system. Being able to allocate .NET types on a custom heap is high on my lists of things that would be very useful to have, for example. There are ways you can work around that, but it would be nice if we had a real solution.

"I am both proud and happy with the way RavenDB has taken off and the shape it is in."

In case you are wondering, this feature would be really useful in implementing memory quotas for internal operations inside RavenDB. But as I said, there are other options, and overall I am very happy with the Framework.

What I am *not* happy about is the number of frameworks that we have. At a current count, we have:

- .NET Framework
- .NET Framework Client Profile
- .NET Compact Framework
- Silverlight
- WinRT

Each of which has different API and capabilities. It gets some what ridiculous, and creating libraries for the .NET framework can become a game of "Whack the Mole" because of that.

".NET is a masterpiece of engineering and I have spent a lot of time digging in."

DNC: Those are very good points and I am sure the .NET Framework team has noticed some of them, if not, well they just heard them now. Coming back to RavenDB, after version 1.0.960 what are the highest priority things HR is working on to make RavenDB even more awesome?

OE: Hibernating Rhinos is currently active working on two parallel tracks

- Major bug fixes only on the 1.0 RavenDB branch
- Major enhancements for the 1.2 version, including enterprise operations toolset, full database encryption, Windows clustering support, dynamic index prioritization and much more.

DNC: Would you say RavenDB has gone web scale (ready for high volume, high traffic production websites) since the MSNBC project or you believe it had proven itself much before that?

OE: Web scale is a really funny term. It is very hard to explain what this means, and in many applications, the actual features required to scale to high number of users and requests are quite different.

For example, in one application you might have a data set small enough to fit into a single machine, and you scaling needs are simply an aspect of the number of requests that you need to do. In that case, web scale means being able to do read striping so you can spread your load among multiple machines.

In another scenario, you have to deal with a data set that can't easily fit into a single machine, so you need to share the data among multiple servers.

In yet another, you need to make sure that even when failure happens, your application stays up and running.

Note that RavenDB has specific features to support all three scenarios, but that each of those represent a drastically different way to increase your system capacity, with different tradeoffs and considerations that you need to make.

I think that the MSNBC project success (and their move to introduce RavenDB in even more projects) is an important validation of RavenDB's abilities to handle very high load. In the same vein, any NServiceBus 3.0 project is also making use of RavenDB for managing state, subscriptions and traffic.

I certainly think that RavenDB's 'web-scaleness' (however you want to define it) has been proven repeatedly in the only place where it matters - actual production environments.

DNC: I am sure that is music to the ears of people, on the fence, regarding adoption of RavenDB. On the same thread , can you share with us more about RavenDB production deployments? Maybe share with us the largest RavenDB deployment in terms of (non-replicated) data-volume and some key takeaways you/your team at HR can recall from the production deployments?

OE: In terms of the actual database size, we have a customer that is rapidly approaching the 1 TB size on a single RavenDB database. Even more impressive, that number was dropped casually during a discussion on a completely separate topic.

There *weren't* any take-away from that customer, because everything just worked, as it should.

Another customer ran into performance issues running map/reduce indexes over a large data set (over hundred million documents). There were actual inefficiencies in how we handled that scenario, which was fixed and a new build was issued to the customer.

Very early on, we had made the mistake of doing an O(N) operation when counting the number of documents that existed in the database (which is something that is reported often, in stats). That got... expensive pretty quickly, but that never got to customers, it was caught in our own internal tests.

Nevertheless, it is quite memorable for me, because I felt utterly ridiculous when I found that I was the one who introduced the inefficient counting.

DNC: How did RavenHQ come about? Was it a market pull (as in users requested for it) or was it a natural progression to offer RavenDB as a cloud-based service, or was it something completely different?

OE: One of the most commented aspects in RavenDB circles almost from the day of the launch was the request for a hosted solution. Database as a Service is a really nice feature that saves you from all the management hassles and lets you rest easy knowing that someone else is out there making sure that your data is protected, secured and backed up.

RavenHQ was a very natural progression from RavenDB. It allows you to have more choices with regards to how you use RavenDB and it makes it much easier to work with

RavenDB in cloud scenarios (where just running RavenDB is a non-trivial process).

DNC: How you do compare RavenHQ to other similar services like MongoHQ, SimpleDB or even SQL Azure? What do you see as the key differentiator for RavenHQ?
OE: Obviously, RavenHQ uses RavenDB and is thus superior :)

> *"More seriously, RavenHQ is the only hosted solution that I am aware of that actually has the PRODUCT dev team involved in supporting and building the hosted solution."*

That means that we can do a lot more to optimize how we are running on the cloud, and that we have deep insights into exactly what is needed to make sure that we can offer our users the best performance and features that are out there.

For example, for customers who want to store sensitive information on the cloud, we offer encrypted storage (for both data and indexes) so that even on the cloud, all the data is fully secured and cannot be accessed.

Another aspect in which we are different from other services is with the notion of commoditizing replication. With RavenHQ, you can replicate your data to multiple destinations. If you are working on a system that is being used from all over the world, you can have a RavenHQ instance that runs in USA and another one in Europe. Both of them replicating to one another and your customers always access the nearest server. For that matter, you aren't limited to just two server replicating to one another.

The MSBNC RavenDB production topology includes multiple RavenDB nodes in multiple (geographically distributed) data centers. We aim to make such high available and scalable scenario easy to use and mange using RavenHQ.

DNC: What are your tips for Developers on how they should approach building a product and/or a framework?
OE: Don't write frameworks.

Seriously, don't do that. Building a framework is a really cool technical challenge and most developers would enjoy that immensely. But there are really very few cases where this is a good thing to do in terms of return on investment. And I am saying that as someone who has written many frameworks, I should know.

When building a product, having a clear vision is **important.** If you don't know what you want to do, there is little chance that you can make something that users will really love.

Beyond that, I found that focusing on removing friction at all levels is something that is tremendously important. From having an automated build that results in a customer deliverable (that they can actually access) seems like a very trivial technical decision, but it has major implications on the product itself.

"Thanks for the bug report, the issue has been fixed and in 5 minutes you can download an updated version" is something that we do frequently across all our products. Customers have repeatedly told us that this generates a very positive experience, and that in turn generates more customers.

Another thing that you want to take into account is friction from the user points of view as well. If you got someone to actually try your product, you have about five minutes or so before they get bored or frustrated and move to something else.

It is crucial to make sure that the user is facing a smooth sail for that time, and try to get a "Wow!" moment during that time.

Reducing friction during that phase is important, and is likely to mean that a potential user is going to convert to a customer in the future.

DNC: That is very good advice. Developers and Entrepreneurs should really think hard every time an urge to build a framework overcomes the 'need' to finish a compelling product and support package that adds real value to end users. To round off what's next from Oren and HR?
OE: We recently completed our newest RavenDB 1.0 release. We are hard at work towards a beta for the 1.2 version. We are also expecting a totally new version of all the profilers with a major feature - production profiling, which we are all very excited about. ■

A WORKFLOW BASED SOLUTION FOR SHAREPOINT 2010

Mahesh Sabnis demonstrates how SharePoint 2010 can be used to create workflow's for automating business processes.

Several organizations are using SharePoint based solutions for internal process automation where multiple users collaborate together as a part of the process. One of the natural techniques of any automation process is use of Workflow.

SharePoint has a builtin Workflow Engine. When any SharePoint object like the List, or the Document library is created; these objects can be managed as per the rules specified in a Workflow.

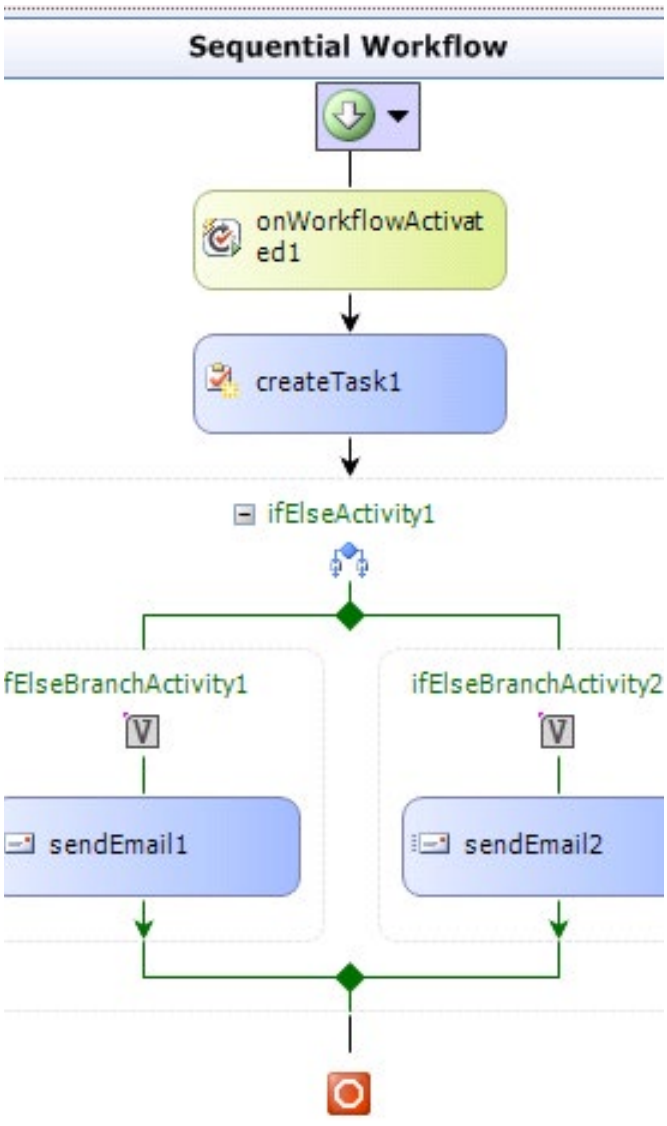
Workflows associated with a SharePoint site can be created using SharePoint 2010 designer. There are two types of Workflows:

- A Sequential Workflow, this has one start point and then it can be completed successfully or terminated based upon the defined conditions. Sequential Workflow is mostly used for automated processes.

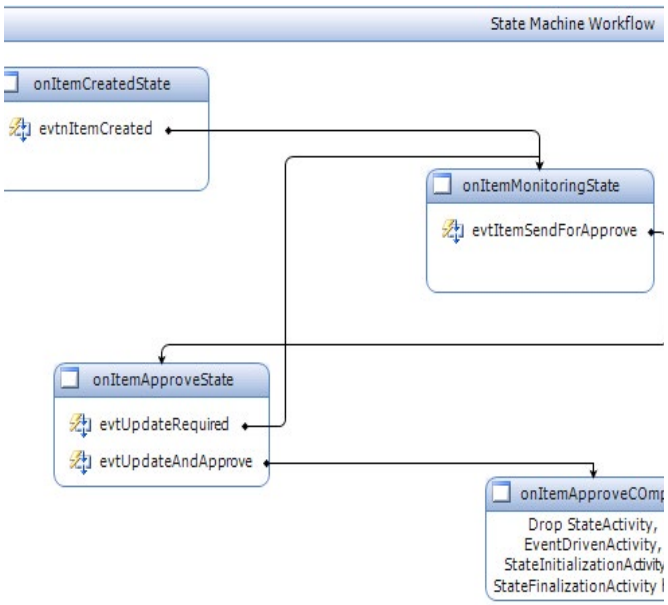
- The second one is the State Machine Workflow, which is purely based upon the state transition methodology. This is an ideal model for more complex business processes where these processes require human intervention at various states. In this workflow, an Initial State and Completion State exists. The execution path from Initial State to End State is completely based upon transition between states and events triggered. In this model, every state can represent a sub-process in the big complex process.

The following two diagrams show simple Models for Sequential and State Machine Workflows:

Sequential Workflow



State Machine Workflow:



These two diagrams clearly show the Workflow model. The Sequential Workflows are like flowcharts where there is a defined Start and End. Whereas the State Machine starts with an Initial State and there are various transitions based upon triggered events, which determine the path of the workflow.

Typically for SharePoint applications, Workflow instances can be started on the following actions:

- When document or List Item is created, user can instantiate a workflow manually from the option on the ribbon control.
- Automatically when the document or List Item is created or changed.

Develop Workflow for SharePoint 2010 Applications

Developing Workflow for SharePoint apps is now possible with SharePoint Designer 2010 (SPD) and also with Visual Studio (VS) 2010. The workflow designed using SPD can be imported in VS and be changed. In this article, I am going to explain mechanism of developing a Workflow using SPD.

SPD provides workflow editor and workflow settings to create and configure workflow. The Workflow editor works within the scope of the SharePoint site. To create a workflow, you must have enough access rights on the particular site. Following are the categories offered by SPD for workflow creations:

Category	Description
List Workflow	The workflow created is attached to a single List in the Site.
Reusable Workflow	These types of workflow allow creation of a template that can be attached to a content type. If we want to export the Workflow to Visual Studio then it should be created as a Reusable Workflow.
Site Workflow	The name itself indicates that it is used at site level context.
Import From Visio Workflow	The Workflow created in SPD can be exported to the Visio Premium or can be imported from Visio Premium to SPD.

Creating List in SharePoint 2010

For this article, we will be using a simple scenario of Patient Admission process in a hospital having a Diseases department. The Workflow generates the doctor and registration fees based upon the admission category e.g. Cancer, Cardio, and Orthopedic etc.

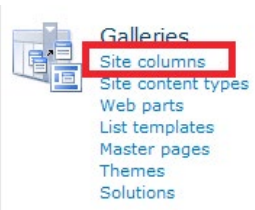
We already have the Site created at the URL http://intranet.MyHospital.com/Sites/MISPortal. We will start creating a List and then we will attach the workflow to it.

Task 1: Visit the URL in the browser and click on the Lists on the Left side panel

Task 2: A new Page with 'All Site Contents', shows up. Click on Create

Task 3: Select the 'Filter By' as 'List' and then select 'Custom List'. Set the name as 'Patient Category Admit'. This List will contain information of Patients and based upon the Category of the Admit (e.g. Cancer, Cardio), the associated workflow will return Doctor Fees and Registration Fees information.

Task 4: The Workflow which is to be associated with the List, may be imported in Visual Studio for further modification in future. To do this, we need to create 'Reusable Workflow'. A Reusable workflow requires 'Site Columns' for defining rules. So we will define site columns next. Go to Site Settings and select 'Site Columns' from Galleries as shown below:



The reason we are using 'Site columns' is in Hospital Automation, we will need the Admission category for the patient across multiple functions, e.g. various fees etc

To add Site Columns, use the 'Site Columns' shortcut on the ribbonl. Click on 'Create' to get to the 'New Site Column' page where we can enter information like, Column Name, its Type

- o Set 'Column Name' as 'Patient Disease Dept.'
- o Select 'Type' as 'Choice'
- o Make it mandatory by selecting 'Yes' for the 'Require that this column contains information' setting.
- o Set it to non-Unique by selecting 'No' for the 'Enforce unique value' field.
- o Add choice values of the following (one in each line)
  - Cancer
  - Cardio
  - Orthopedic
  - Eye

Now create two more Custom Columns - 'Doctor Fees' and 'Registration Fees'. Set their type to Numeric. You can see the site columns created on the 'Site Column' page as below:

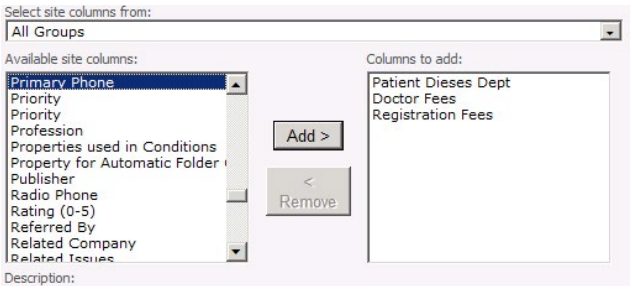
Custom Columns

AdmitType	Choice
Doctor Fees	Number
Patient Disease Dept	Choice
Registration Fees	Number

Task 5: To create rest of the columns for the List, click on the 'Patient Admin Category' List on the Left Panel, you will get the List Details, select 'List Settings':

Task 6: Create columns using 'Create Column' Link

Task 7: Since we have already created site columns, to add them in this list, click on 'Add from existing site columns'. You will get the Page where all site columns are specified, you can select site columns from the available site columns. Add the 'Patient Disease Dept.', 'Doctor Frees' and 'Registration Fees' from the Available site Columns.

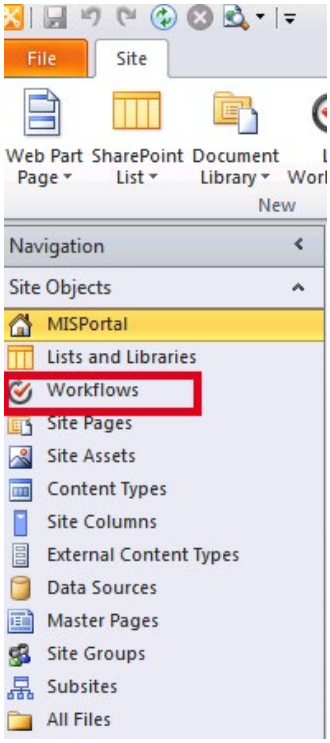


Now the complete List will have the following columns:

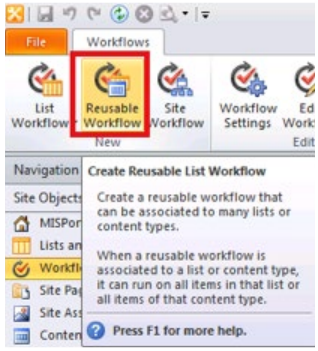
- Title
- Patient ID
- Patient Home
- Patient Disease Dept.
- Doctor Fees
- Registration Fees
- Created By
- Modified By

Creating Workflow using SharePoint Designer 2010

Task 1: Open SPD and open the SharePoint portal where you have the List. You will find all the Portal objects. Select the Workflow from the Panel as shown here:



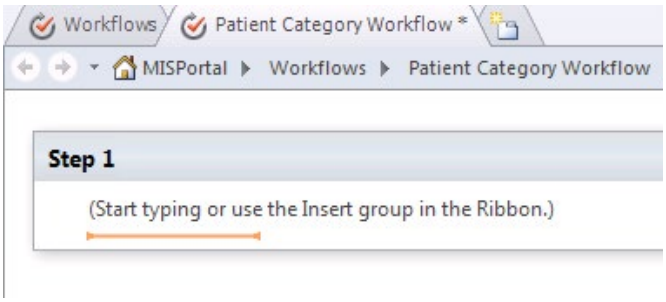
Task 2: We need to create a Reusable workflow, so click on the 'Reusable Workflow' from the Ribbon as shown below:



Set the workflow name and its description as follows:

Name: Patient Category Workflow
Description: The Workflow is used to define the Rules for Admitting Patient in Hospital in a specific Department.

You will see the Workflow Designer as shown here:

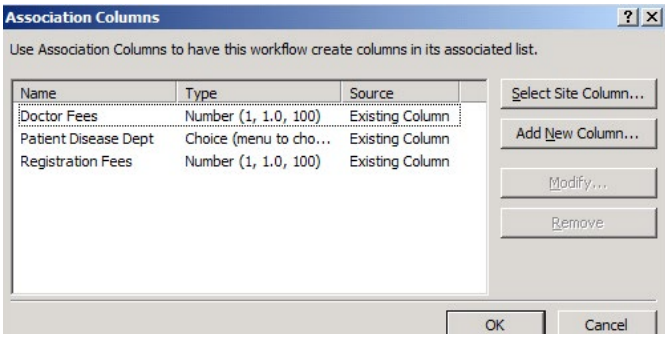


The workflow designer shows 'Workflow' ribbon which provides various features listed here:

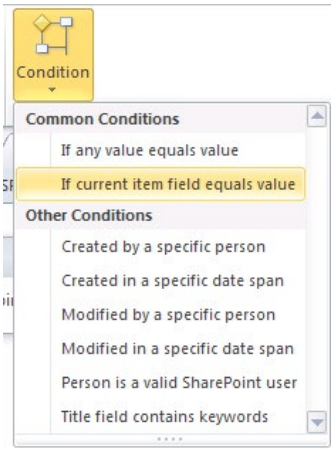
- Save: The workflow is saved in the XOML file, this allows us to export file to Visual Studio.
- Publish: This publishes workflow which is then available in the SPS portal and you can associate with the SharePoint objects.
- Check for errors: This checks any problem with workflow. It is strongly recommended that you should check for errors before publishing.
- Condition: This provides various conditions to be put in workflow e.g. if-else.
- Action: Provides the mechanism of steps to be taken when the condition is evaluated.
- Publish Globally: This allows workflow to be published on the Global Workflow Catalog so that it can be available to all SharePoint Sites.
- Export to Visio: Export the workflow to the Vision 2010 premium.
- Workflow Settings: Define the workflow settings for specifying the behavior the Workflow e.g. Auto start or manual start of the Workflow.

Task 3: In the reusable workflow, we require the Site columns to be associated for defining conditions. To do this, select 'Association Columns'

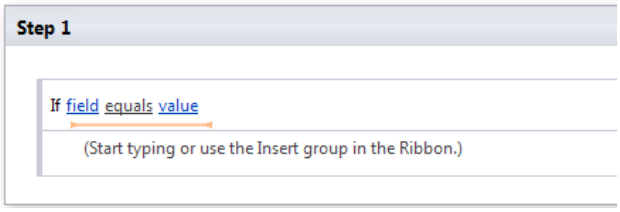
Now add the Association site columns as shown below:



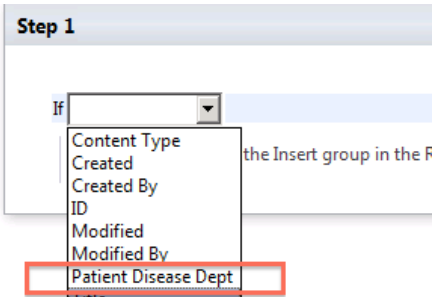
Task 4: Drag-Drop the 'if' condition from the Action Pane of the Ribbon as below:



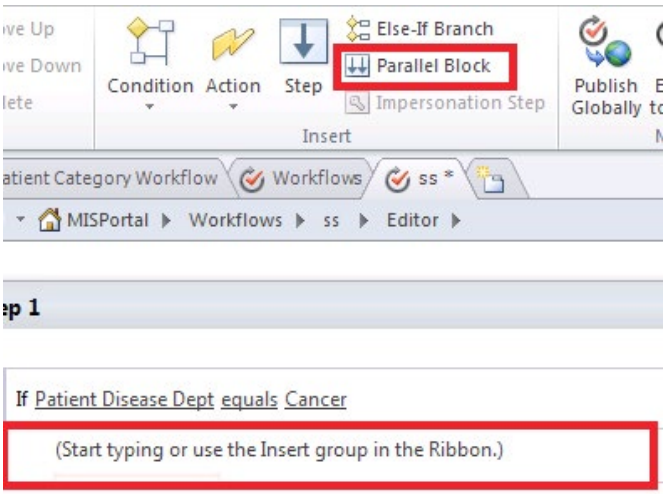
You will get a condition:



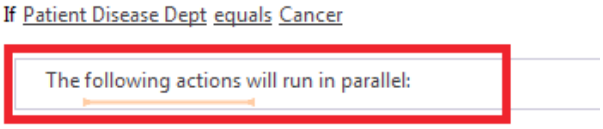
Here the 'field' represents the Column from the List (or site column) on which the condition is defined and the 'value' represents the condition evaluation expression. Since this workflow defines rules for 'Patient Diseases Dept.', if you click on the field, you will get the List of fields for a given condition



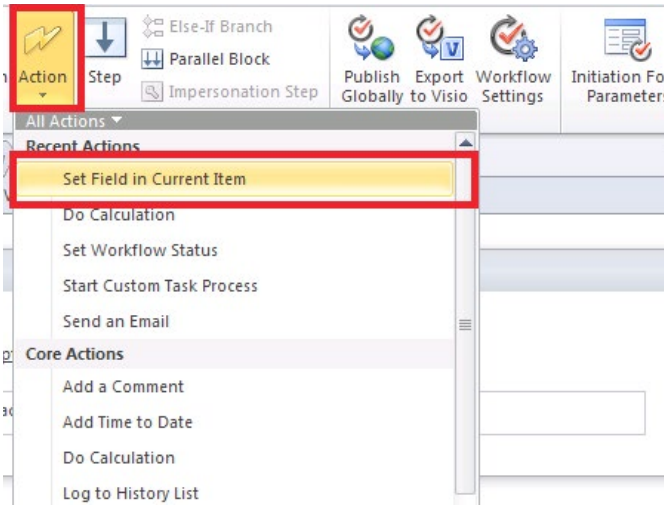
If you click on 'value', you will see the values which we have set for the 'Patient Diseases Dept.' site column in previous List Creation steps e.g. Cancer, Cardio etc. Since we have 'and' condition for getting Doctor Fees and Registration Fees, we need to have both statement to run in Parallel so below the 'If' drag drop, the parallel block is as below:



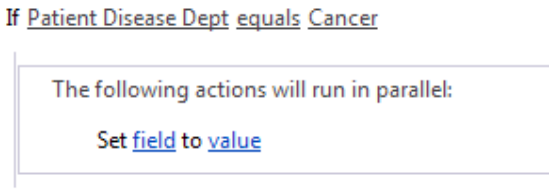
The 'if' condition now gets displayed:



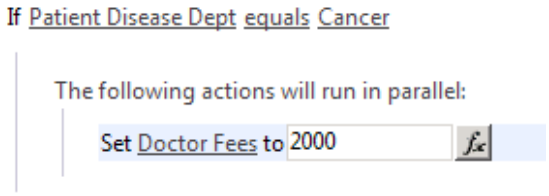
Put the set statement inside the 'Parallel' block as below:



You will get the 'Set' statement as below:

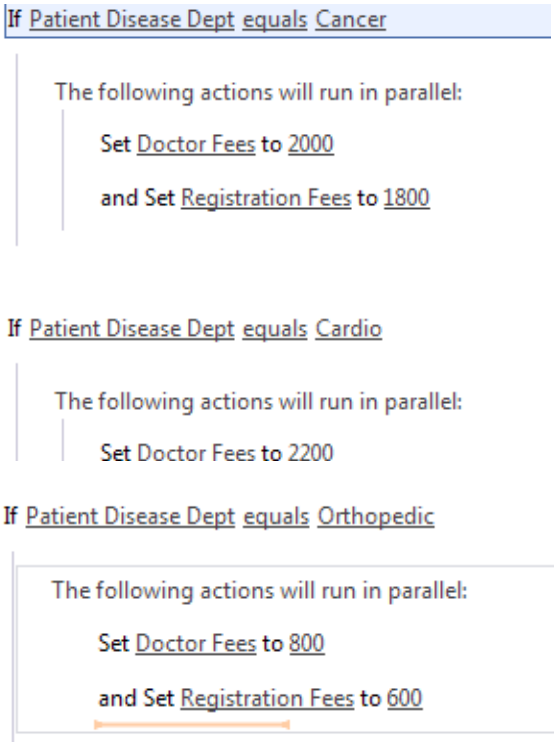


Set rule for the set statement as shown below:



Drag-Drop one more 'Set Field in Current Item' from the Action of the Ribbon inside the Parallel Block, and set value for Registration Fees.

Task 5: Repeat Task 4 for all condition like Cardio, Orthopedic, etc. The workflow design will be as shown here:

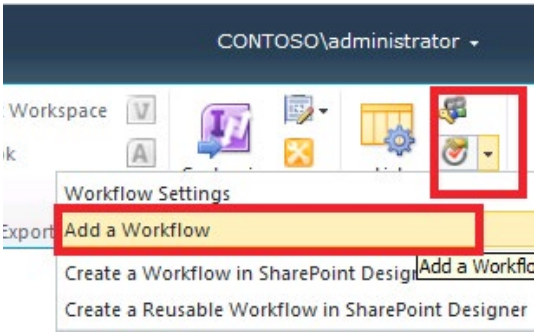


Task 6: Save and publish the workflow.

Associating the Workflow with the List in the SharePoint Portal

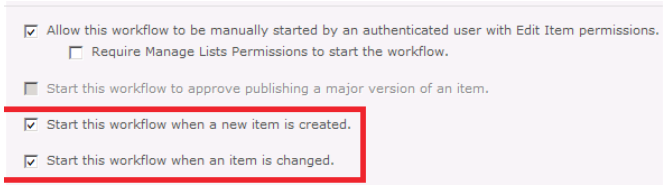
Now it's time for us to associate the workflow with the List created in previous steps.

Task 1: Go back to the portal, click on the 'Patient Admit Category' List, from the 'List tools', select 'List', and from the ribbon, select 'Workflow Settings' as shown below:



Select 'Add a Workflow' from the options, and you will get an 'Add Workflow' Page. Select the 'Patient Category Workflow' from the Workflow List and set its unique name to – 'Patient Category Workflow Status'

Set the Workflow Start Options as shown below:



Here the start options are set to 'Start this workflow when a new item is created' and 'Start this workflow when an item is changed'. This means that the moment any new entry is created or updated from the List. Once you click on 'Ok' you will get the Workflow Settings page

Task 2: Now add a new entry in the List, the Workflow status against the entry will show 'In Progress'.

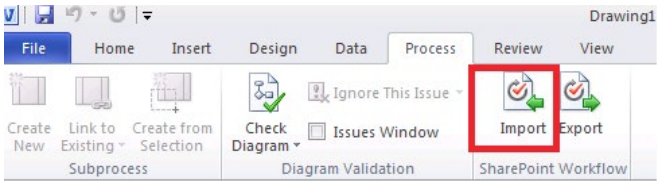
Task 3: Refresh the Page and the Workflow status will be shown as 'Completed'.

Exporting the Workflow to Visio Premium 2010

We can see the diagrammatic representation of the above created Workflow in SPD by exporting it into Visio Premium.

Task 1: Start SPD and open the Workflow created above. Select Export to Visio from the 'ribbon'. It will ask you to provide the export name; the file will be saved as 'Visio Workflow Interchanged File'.

Task 2: Open Visio and create a new 'Microsoft SharePoint Workflow'. You will get Shapes. Now click on the 'Process' menu and select 'Import' as below:

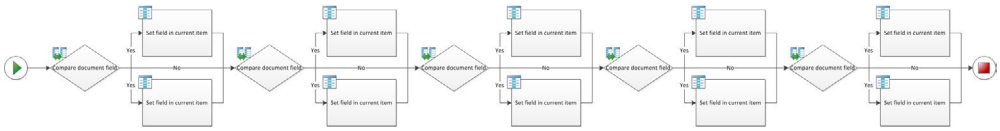


Select the file which is exported in Task 1, the workflow will be as below:

Conclusion

In SharePoint, workflow provides a powerful mechanism for automation of business processes. SharePoint Designer is a handy tool provided to developers for designing and publishing the Workflow. Apart from automated calculations as seen in this article, we can use Workflows for document approval in various departments where various stake holders can be setup as approvers as a part of a list to which if a document is attached, all users get a chance to approve and add comments.

A flexible workflow engine like the one in SharePoint 2010 gives an added dimension to the other collaboration features it has and makes it a very powerful intranet platform. ■



Mahesh is a Microsoft Certified Trainer (MCT) since 2005 and has conducted various Corporate Training programs for .NET Technologies (all versions). Follow him on twitter @maheshdotnet.

VISUAL STUDIO ALM FOR HIGH PROFITABILITY

Subodh Sohoni, VS ALM MVP, shares his views on how the Visual Studio ALM suite can be used for improvement in productivity of software development and in turn lead to high profits.

During the last couple of years, business of software development has not remained as profitable as it used to be about a decade ago. To retain high profitability, it has become necessary for organizations to look critically at their processes and optimize those for the highest productivity of the teams and the best quality of the product.

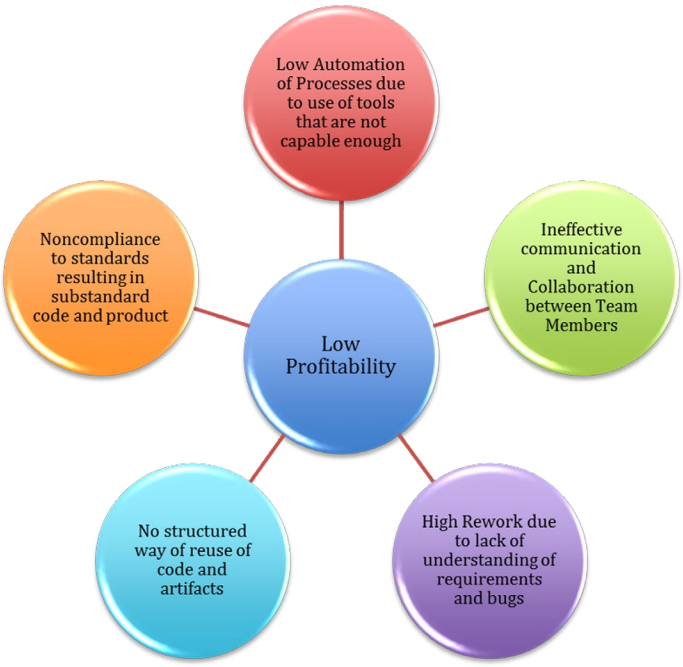
Using Microsoft Visual Studio ALM is one of the most effective ways to improve productivity of the team and the quality of product.

Issues That Affect Software Development Productivity and Quality

Let us first look at the issues that can adversely affect the software productivity and quality and in turn profitability.

1. Not using the tools that are capable of supporting automation
2. Ineffective communication and collaboration between team members
3. High Rework due to lack of understanding of requirements and bugs
4. Duplication of efforts due to unstructured coding pracitces and artifacts reuse
5. Noncompliance to standards resulting in substandard code and product

Although this is not an exhaustive list of all the issues faced by a team developing a non-trivial software, it is a list of most significant issues that affect the profitability of the organization.



How Visual Studio ALM can help

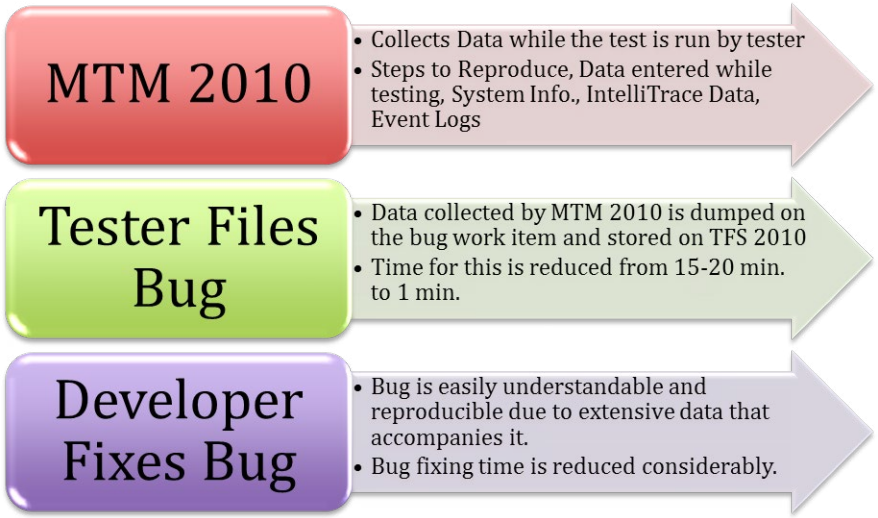
Let us now delve deeper in each of these issues and understand how Microsoft Visual Studio ALM can help us to resolve these issues.

Not using capable tools for automation of processes

Most of the processes have some activities that need to be executed manually and some activities that can be automated. For example, if testers need to file a bug, they need to provide some general information about the bug, like the title of the bug, iteration and module in which it was found etc. At the same time, testers also need to provide more specific data about the bug; like what were the steps that lead to the bug and environmental information under which that bug was found etc.

This data can be collected by the tools while the test is running and then embedded in the bug so that it becomes useful when the developer fixes that bug.

"Microsoft Visual Studio ALM has abundant tools that can automate the activities and improve the productivity of the team."

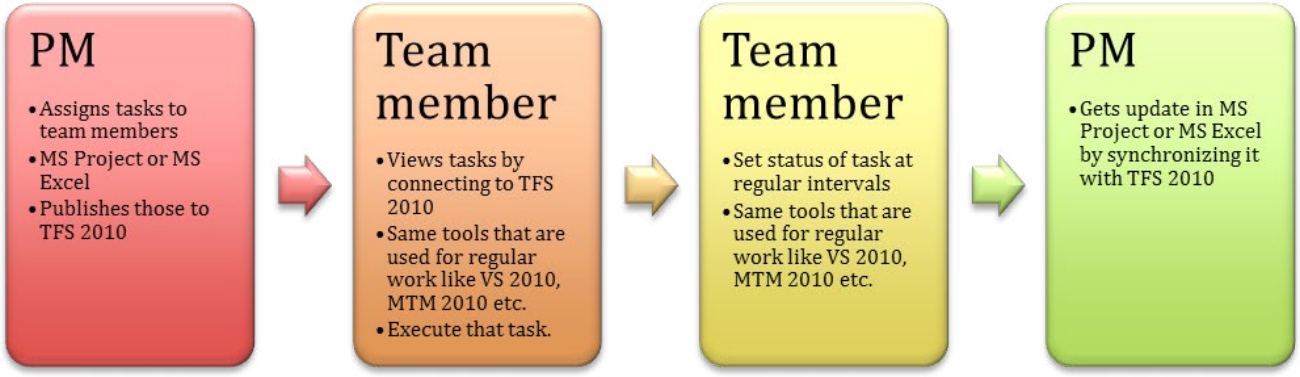


Monitoring the tasks that are assigned to team members

Usually project managers may use out of band communication media like emails, telephone calls or stand up meetings to get to know the status of tasks that are assigned to various team members. After collecting that data, they enter it in tools like Microsoft Project or Microsoft Excel.

Team Foundation Server (TFS 2010) is an integral hub like component of Microsoft Visual Studio ALM that provides editable access to the tasks for all the team members, to the tasks that are assigned to them. They enter the status of the tasks using their regular tools like Visual Studio directly in TFS 2010 and that status is then made available to the project manager. This eliminates the inefficient out of band communication.

TFS not only provides this centralized access to tasks, but it also stores, analyzes and reports the data related to tasks. These features of Microsoft Visual Studio ALM improves the productivity of project managers and allows them to focus on activities of planning and control that need more attention and value addition



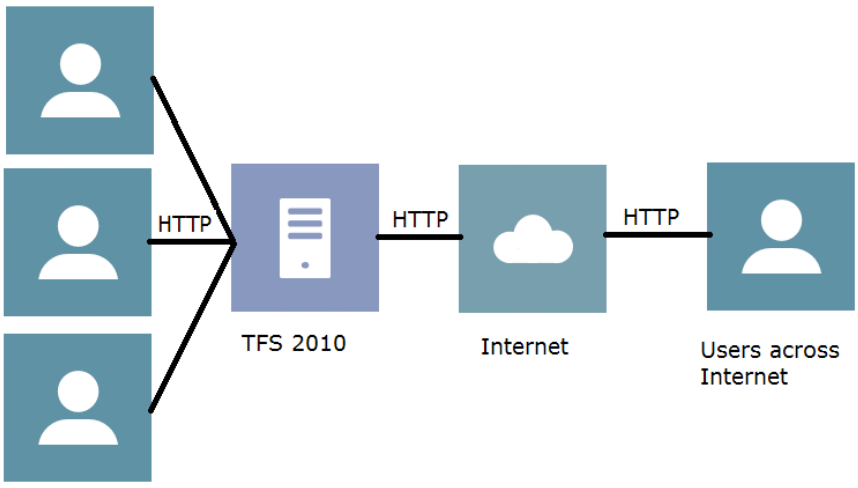
from the project manager.

Microsoft Visual Studio ALM has all the required tools to automate project and improve productivity of the team.

Ineffective communication and collaboration between team members

Most of us have had experience of working on projects where team does not communicate well and team members do not collaborate with each other. Fate of such projects is known even before the deadline approaches. These projects are doomed not because the team members are incapable, but because they do not work as a team, they are just collection of persons.

Reasons for such lack of communication can be as trivial as not having a good communication media or as complex as having a multi locational team working on a project. Microsoft Team Foundation Server facilitates communication and collaboration between team members regardless of their location. It



communicates using standard http / https protocols so that communication across the Internet is possible. It also provides a strong platform like Microsoft SharePoint to collaborate within a team.

Effort wastage in rework due to lack of understanding of customer requirements and bugs

Sometimes tools are not capable of automatic tracing from requirements to tasks that implement those requirements and test cases, that ensure the correctness of the implementation. Due to such tools, some requirements may not get implemented

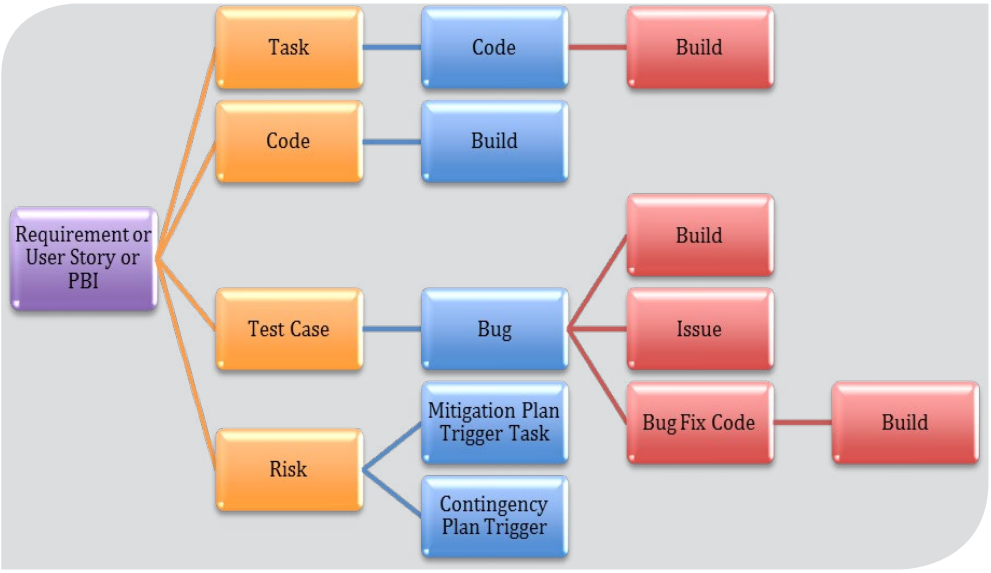
"MANY A TIMES GREAT SOFTWARE IS CREATED BUT FOR WRONG REQUIREMENTS"

at all or other requirements may not be correctly implemented. TFS 2010 provides the service of hierarchical work items that can link the related work items and then automatic traceability is made available. For example, a requirement is linked to tasks and test cases as they are created, so that a requirement can be traced forward and the tasks or test cases can be traced back to specific requirement. Code can be linked at the time of check-in, to tasks, requirements or bugs, so that traceability to and from code is also facilitated. Reports related to this traceability are also made available. (see figure 1)

Another cause of reduction in productivity is developer not able to understand the bug that is filed by the tester. As mentioned earlier in (1) Microsoft Test Manager 2010 collects a lot of data that is useful to the developers for doing bug fixing.

Data like - Steps to reproduce, test data entered by the tester, system information where the application is running, IntelliTrace data, screen snapshots, video of the screen while the test is running etc. is collected while the test is running. This same data is then attached to the bug when the tester files that bug. It is saved in TFS 2010 when the bug is saved. Such a bug is also called a 'Rich Bug' since it is rich with enough information required by the developer to fix that bug.

A developer can open the same bug in Visual Studio 2010 and then have a look at the attached data to get more understanding of the bug. Developer can also start a debugger session with the attached IntelliTrace data and trace through the entire code to find the bug in that code. Time required to understand the bug and then fix it correctly is reduced due to the 'Rich Bug' which results in improvement of productivity of the developer directly and the team indirectly.



(figure 1)

Non-existence of a structured way for code and artifacts reuse

One of the causes of low productivity is duplication of code and work items. Innovative code created by team members within a team and also outside the team but within the organization, is not well published. This may lead to similar code being recreated by other team member at a cost of time required to create such code.

TFS integrates with Microsoft SharePoint, on which it creates a portal for every project that it is managing. Organization can leverage this automation to internally publicize availability of reusable components.

Once available in SharePoint, anyone with proper access can search these libraries to find and reuse the same solution and not waste their time and efforts in reinventing the solution.

This feature will improve the productivity and also the quality since the published code is well tested and has less chance of having a bug in it.

Mediocre product and code quality due to non-compliance to the standards

Best quality product stands a chance of commanding high price vis-à-vis the cost of production and such a high price does result in higher profits to the organization. Although high quality code is not the only factor that influences quality of the product, it does have a large impact. Maintaining quality of coding as per the policies of the organization results in consistent and better quality products.

TFS facilitates compulsory quality checks like static code analysis at the time of code check-in. This ensures that code that is being checked-in, has passed the necessary quality checks. Checked-in code is ensured to be of the quality that is as per standards set by the organization. It also results in less rework and bugs that are major cause of reduction of productivity. This is done with a feature of the TFS called 'Check-in Policy'. Check-in policy does not allow the check-in to take place as long as its condition is not met by the code. There are a few built in check-in policies available and custom check-in policies can also be built. Organization can decide the policy that governs the quality checks that are made at the time of check-in of the code. Only necessary check-in policies can be enabled to maintain desirable quality of the code. Maintaining quality as part of standard workflow improves the quality of the product and without any adverse effect on overall productivity of the team.

Conclusion

Profitability is the most important goal of any business including that of software development. Profitability is adversely affected due to low productivity of the team that is developing software and low prices due to mediocre quality of the product. Productivity and quality are usually low because of not using the best available tools and services. Microsoft Visual Studio Application Lifecycle Management Suite of Products is a set of tools and services that enhances the productivity of the team and ensures high quality of product which directly and indirectly improves the profitability of the organization. ■

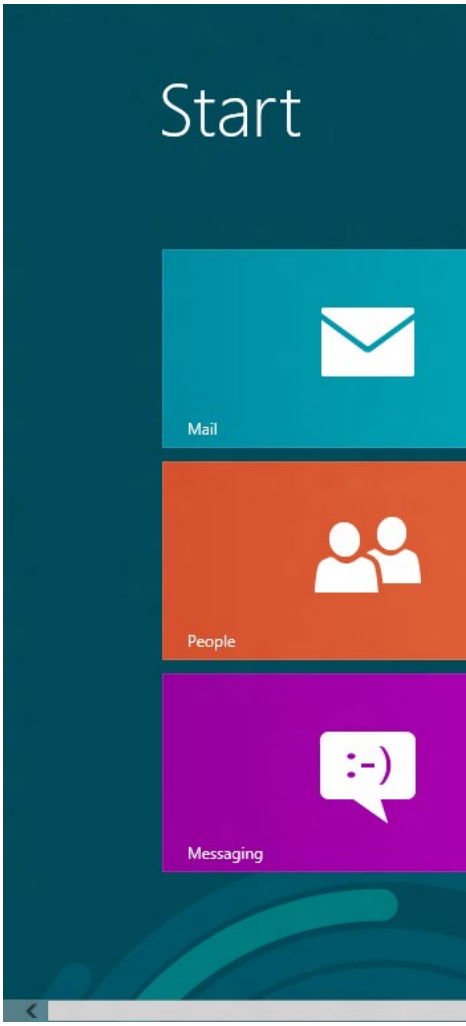
Subodh Sohoni, is a VS ALM MVP and a Microsoft Certified Trainer since 2004. Follow him on twitter @subodhsohoni and check out his articles on TFS and VS ALM at http://bit.ly/Ns9TNU

Metro Style Business Applications in Visual Studio 2012

Mahesh Sabnis develops a Metro Style Business Application on Windows 8 and Visual Studio 2012 and demonstrates how simple and flexible it is to create such apps.

DOWNLOAD FILES >

bit.ly/dncmag-metro



The Metro Design language became popular with the maturing of the Windows Phone 7 platform. Due to its popularity, it got adopted on the Windows 8 platforms too. Now that Windows 8 is at a Release Preview stage, we find the Metro Design to be rather elegant on the Desktop as well. WinRT is a new runtime on Windows 8 and it provides a rich toolset for development of consumer and business applications.

DEVELOPERS AND METRO APPLICATIONS

Apps developed using the new WinRT Runtime are referred to as Metro Apps. An important fact with these applications is that the devloper community can easily get started with developing Metro Apps

- .NET developers with the knowledge of C# and XAML etc. can start developing applications using VS2012 RC, because XAML is a familiar technology from Silverlight and WPF. Metro Style apps uses similar UI elements and design patterns.
- Web developers having expertise with JavaScript, jQuery, HTML 5 and CSS can also design these applications. Using JavaScript, the applications can make external call to WCF services or any external services to fetch data. Alongside, .NET 4.5 has come up with Web-API, using which Data Bound operations can be performed over plain HTTP (instead of the earlier WS-* layer).

Metro style apps can be considered as the future of Windows app development, be it x86 or ARM processors.

For end users, the shift from desktop computers to the more portable Laptops is now more or less complete. Most consumers now prefer a laptop to a desktop. However, technology is moving towards even more portability in terms of smaller than laptop devices; like tablets and big-screen smartphones. These devices are much smaller, their main user interface is touch-centric as opposed to pointer centric on the desktop and they have certain restrictions on CPU power. These devices are also geared towards single purpose, intuitive applications. These devices are not geared to be multipurpose fully functional computing devices. They allow users to consume and interact with the content. In

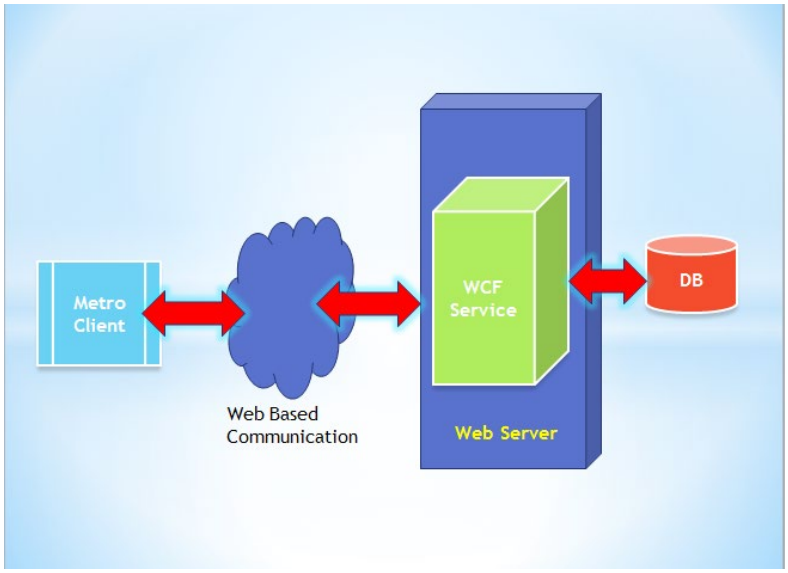
developer terms, these devices are targeted for using an app rather than developing the app. It is this segment of application usage that needs apps with a touch-centric design philosophy like *Metro*.

If we take a tablet or a large screen mobile device using Windows 8, a typical business case could be for users from the Marketing field. These users could use applications on this device to record orders and generate invoices.

Similarly doctors, can make use of applications to view their appointments, record their observations and so on. These kinds of applications and devices were earlier a niche, but with the popularity of tablet devices, they are fast becoming mainstream.

DEVELOPING A METRO APP

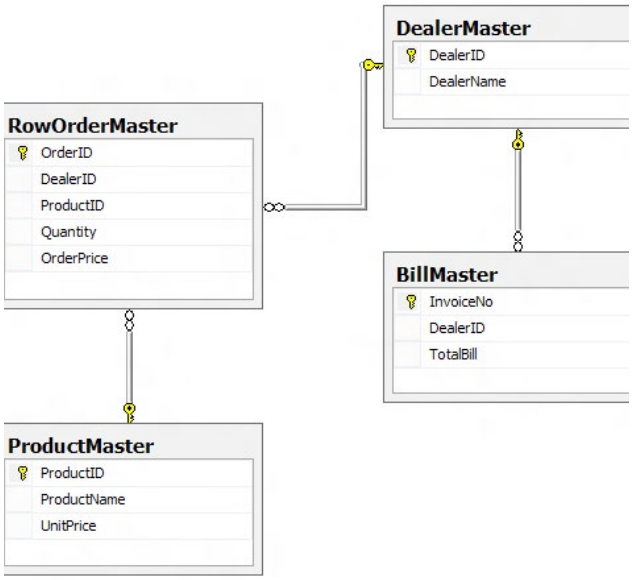
Let us develop a Metro styled application for marketing professionals that generates Invoices for medical orders. The architecture of the application is as below:



The Metro client application makes use of WCF Service to communicate with database server. The WCF service is hosted on IIS and is accessed by the client over HTTP.

CREATING THE APPLICATION

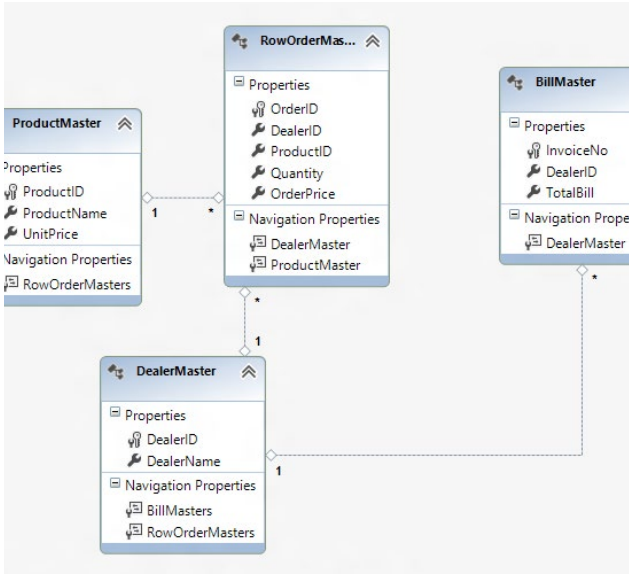
We use SQL Server 2008 as the backend. Our table structure is defined here.



DealerMaster and ProductMaster are master tables used to store information about Dealers and Products respectively. RowOrderMaster table is used to store Product wise row order bill and BillMaster is used store total bill.

Step 1: Open VS2012 RC and create a blank solution, name it as 'SLN_MetroApps'. In this solution add a new WCF service, name it as 'WCF_Sales_Service'. Rename 'IService1.cs' to 'IService.cs', 'Service1.svc' to 'Service.svc'.

Step 2: We will use Entity Framework for data access. Add a new ADO.NET Entity data model in the WCF service and name it as 'SalesInfoEDMX'. Complete the wizard and select the above tables. The corresponding entity diagram will be as follows.



Step 3: Add the following contracts in the interface IService. These operations will return collection for Products, Dealers and Invoices. We also have a contract that is called for generating the Invoice.

```
[ServiceContract]
public interface IService
{
    [OperationContract]
    ProductMaster[] GetProducts();
    [OperationContract]
    DealerMaster[] GetDealers();
    [OperationContract]
    int GenerateInvoice(BillMaster objBill,
        RowOrderMaster [] orders);
    [OperationContract]
    BillMaster[] GetInvoices();
}
```

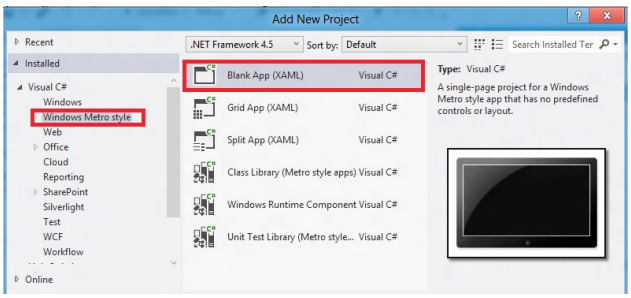
Step 4: Implement this interface in the Service class. The implementation makes the DB calls via the EntityFramework ObjectContext. For example, the implementation below generates the invoice and Saves it to the database. Similarly we implement rest of the interfaces, using EF ObjectContext to save data.

```
public int GenerateInvoice(BillMaster objBill,
    RowOrderMaster[] orders)
{
    foreach (var Order in orders)
    {
        objContext.AddToRowOrderMasters(Order);
    }
    objContext.AddToBillMasters(objBill);
    int Res = objContext.SaveChanges();
    if (Res > 0)
    {
        Res = 1;
    }
    return Res;
}
```

Step 5: At this point our service layer is complete. We build the Service and publish it on IIS.

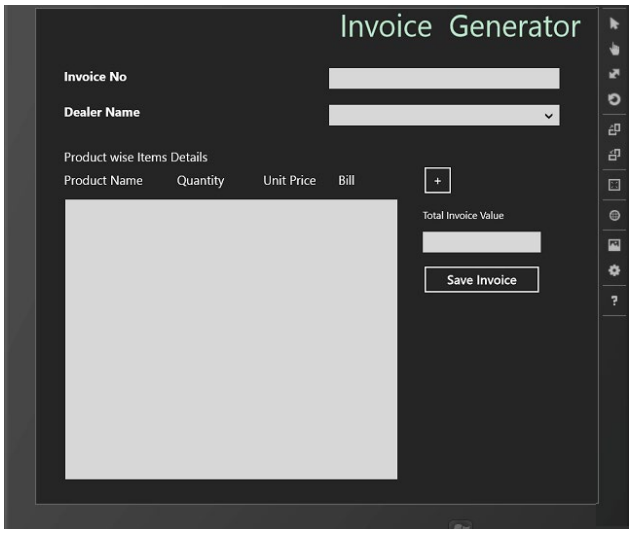
CREATING METRO STYLE CLIENT APPLICATION

Step 1: To create a Metro app, Visual Studio provides multiple templates to get started with. These templates dictate how the applications must be laid out and bound to data. For our single page app scenario, we will pick the Blank App from the Windows Metro Style projects. We call the project 'SalesInformation' as shown below:



Note: You will get the template for Metro Style Application in Visual Studio 2012 installed on Windows 8 operating system.

Step 2: Open MainPage.Xaml and set it up the UI to look as follows.



The TextBox for 'Invoice No' will show the generated Invoice Id after successfully storing Invoice details. The ComboBox for Dealer Name will display list of all dealers. The big gray box is a ListBox where the Productwise bill will be generated. The button with "+" will add a new row in ListBox for Productwise order details. The TextBox above the button 'Save Invoice' will show the Total Invoice value.

Step 3: Write helper methods to Read all dealers and Products using WCF Service Proxy as shown below:

```
/// Helper Method to Read All Dealers
/// </summary>
void  LoadDealers()
{
   Task<ObservableCollection<MyRef.DealerMaster>> taskDealers =
       Proxy.GetDealersAsync();
   cmbDealerName.ItemsSource = taskDealers.Result;
   cmbDealerName.DisplayMemberPath = "DealerName";
   cmbDealerName.SelectedValuePath = "DealerID";
}

/// <summary>
/// Helper Method to Load all Products
/// </summary>
void LoadProducts()
{
    Task<ObservableCollection<MyRef.ProductMaster>> taskProduct
        Proxy.GetProductsAsync();
    Products = taskProduct.Result;
}
```

One important observation in the above code is that, the call to the service is asynchronous, but there is no completed event. The reason is that the return type of the asynchronous method is set to **Task<T>**. This class takes care of the call completion return data using **'Result'** property.

Step 4: Write the helper method, to generate ListBox elements. This method will be called when the "+" button is clicked:

```
/// <summary>
/// Helper Method to Create UI inside the ListBox for adding
/// Productwise order details
/// </summary>
void CreateBillUI()
{
    stkPnlContainer = new StackPanel();
    stkPnlContainer.Orientation = Orientation.Horizontal;
    stkPnlContainer.Width = lstContainer.Width;
    stkPnlContainer.Height = 50;
    cmbProducts = new ComboBox();
    cmbProducts.ItemsSource = Products;
    cmbProducts.Width = 150;
    cmbProducts.Height = 50;
    cmbProducts.DisplayMemberPath = "ProductName";
    cmbProducts.SelectedValuePath = "ProductID";
    cmbProducts.SelectionChanged += cmbProducts_SelectionChanged;
    stkPnlContainer.Children.Add(cmbProducts);

    txtUnitPrice = new TextBox();
    txtUnitPrice.Width = 150;
    txtUnitPrice.Height = 50;
    txtUnitPrice.IsEnabled = false;
    txtUnitPrice.Text = "000";
    stkPnlContainer.Children.Add(txtUnitPrice);
```

The code shown above instantiates a control hierarchy for displaying the Medicine List, Price and Quantity controls. The SelectionChanged event of the ComboBox fetches the Unit Price from the database. The Lost Focus event of the Quantity triggers the calculation of the item-wise total and the total bill.

Step 5: We instantiate the Products and Orders collections in the Constructor. In the MainPage_Loaded event handler, we instantiate the Proxy and call the helper methods to load the data.

Step 6: Call the 'CreateBillUI' method on click event of the '+' button

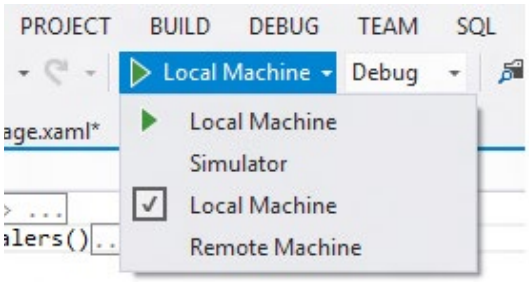
Step 7: Finally on the click event of the 'Save Invoice' button, write the code for saving the invoice in the database.

```
private void btnSaveInvoice_Click(object sender, RoutedEventArgs
{
    try
    {
        MyRef.BillMaster objBill = new MyRef.BillMaster();
        objBill.DealerID = Convert.ToInt32(cmbDealerName.Selected
        objBill.TotalBill = Convert.ToInt32(txtTotalBill.Text);
        Task<int> Res =  Proxy.GenerateInvoiceAsync(objBill, Orde
        if (Res.Result == 1)
        {
            txtError.Text = "Invoice Genetared Successfully";
        }
        var Invoices = Proxy.GetInvoicesAsync();
        var InvoiceData = Invoices.Result;
        //Get the Newly generated Invoice from Table
        int LatestInvoice = (from inv in InvoiceData
                             select inv.InvoiceNo).Max();
        txtInvoiceno.Text = LatestInvoice.ToString();
    }
    catch (Exception ex)
    {
        txtError.Text = ex.Message;
    }
}
```

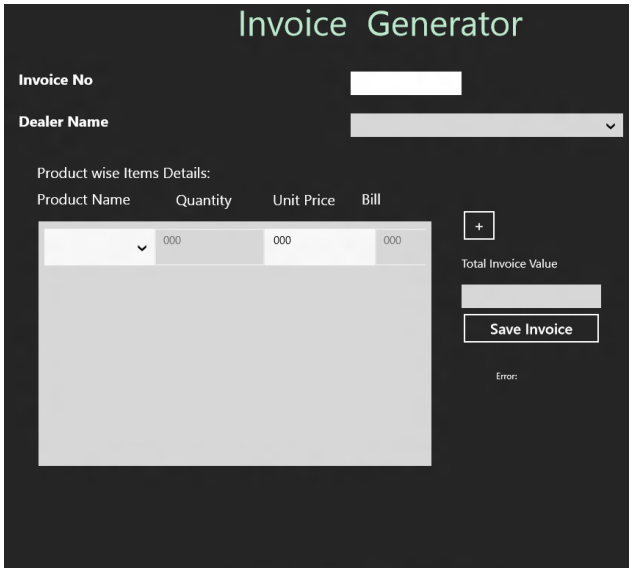
This code makes a call to GenerateInvoiceAsync method to save details in the database. We are now all set for our test run.

TEST RUN A METRO STYLE APP

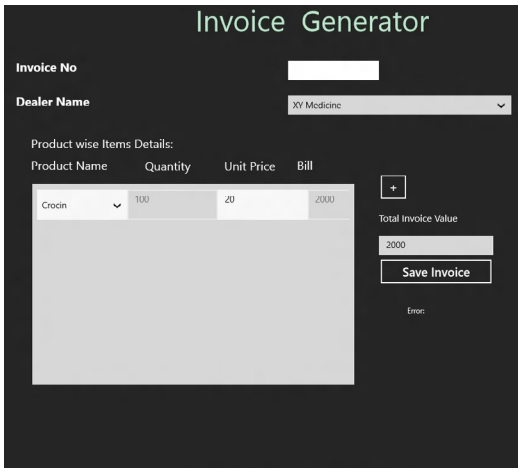
In the VS2012, we can test metro style apps in multiple ways.



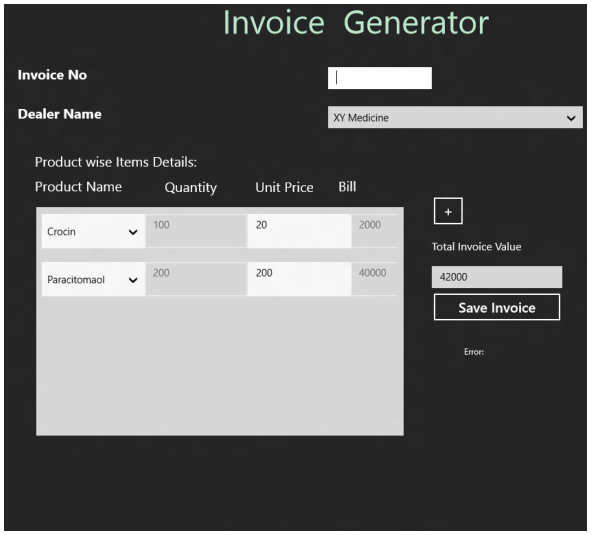
We initially select the Local Machine (default) and press F5. Win 8 goes into Metro Mode and shows us the application



Select the Dealer Name and Product Name from the respective dropdowns. Put in the Quantity and Unit Price. The Row bill will be calculated and the Total Invoice Value will be updated once user navigates away from the Product Row.

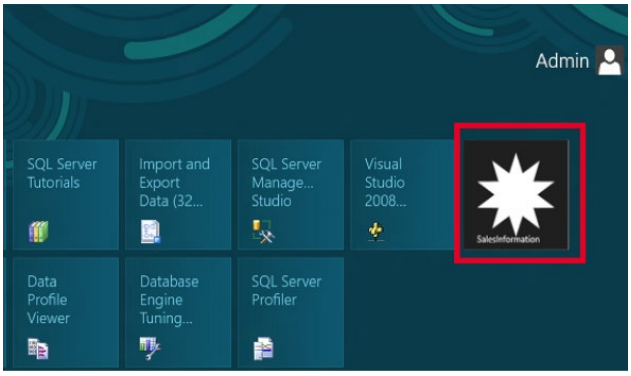


Now click on the '+' button to add a new row and add another row of data



After clicking 'Save Invoice', the generated invoice number will come up in the 'Invoice No' field.

To test it in a Tablet Simulator, we can change the Run option to the Simulator. On pressing F5, the simulator fires up and shows our application's tile before launching the application itself.



WRAP UP

We developed a Metro Styled Business Application. We could definitely reflect the Metro Design Aesthetic better but as a quick sample, we focused on demonstrating the power of the platform. Metro Styled Apps with their rich design aesthetic are aimed at non-business Consumers. But as we saw it provides enough flexibility to build functional Business Applications with minimum effort and using existing C# and XAML skills.

Mahesh is a Microsoft Certified Trainer (MCT) since 2005 and has conducted various Corporate Training programs for .NET Technologies (all versions). Follow him on twitter @maheshdotnet. Mahesh blogs regularly on Azure, SharePoint, Metro UI, MVC and other .NET Technologies at http://bit.ly/HsS2on


Join us on
Facebook
www.facebook.com/
dotnetcurry

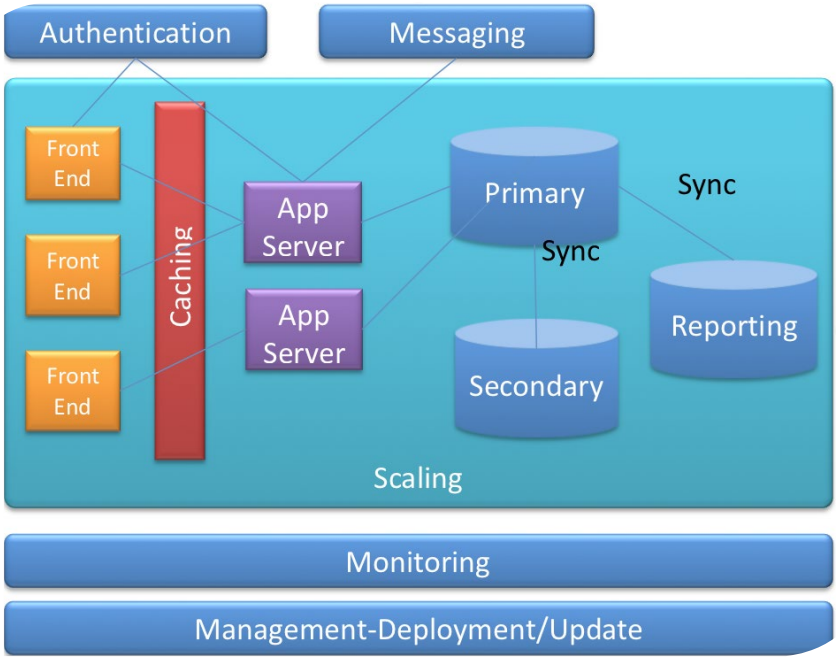
AZURE ADOPTION BEST PRACTICES

Govind Kanshi discusses some very useful tips and clarifies the Azure stack offering for those planning to move to Azure platform

Every once in a while, in a technology cycle comes a new wave, where the industry faces challenges in terms of how best to tackle it. Azure and related Cloud technologies are both an opportunity and a challenge. In this article, we will focus on a set of pivots that will help the 'Azure Adoption' decision-making process. We will try to address questions like - What does it mean to have reasonable monitoring and scaling methodology? What are the issues surrounding "development and deployment" of the application?

Let us consider an application which has its front end served by a web server (IIS for our discussion), caching, inter-app messaging and a database. Although the focus is on the Microsoft stack, other stacks like node.js, php, java etc. are supported on Windows, so Azure has no issues in supporting them. The biggest challenge lies in the support for proprietary databases, but with latest Infrastructure as a Service (IaaS) offering, we should see some better solutions in that direction.

If you are new to Cloud Computing, this article will serve as handy feature-availability list and help clarify the Azure offerings bouquet.



SCALABILITY

Areas Discussed:

Computing Scalability
Use of Caching
Messaging
Multitenancy Pivots,
Storage Scalability

ALM

Areas Discussed:

Change of Code
Create/Update/Delete Metadata
Updating Schema
Updating Blob Storage

OPERATIONS

Areas Discussed:

Capacity Planning
Monitoring
Backup Restore
High Availability and SLAs

NON MS PLATFORM

Areas Discussed:

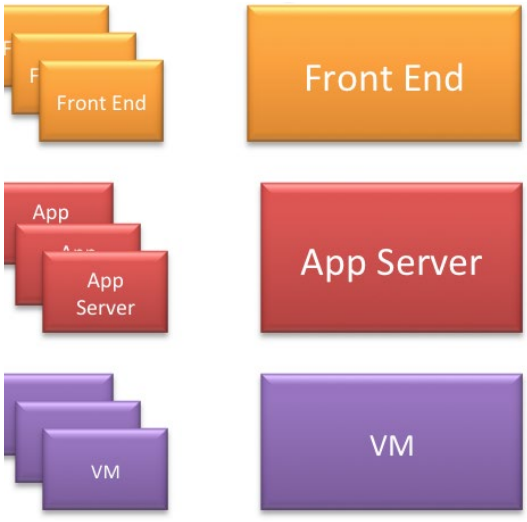
Frameworks
OS support
Database

1 SCALABILITY – FEATURES AND PRACTICES

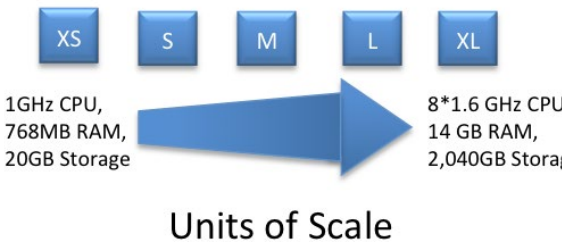
Scale is one of the first things that comes to mind when thinking of cloud-based architecture. One needs to think through two important pivots of scaling - Throughput and Response Times. On the cloud, resources such as Storage and Compute Power allows the option to scale vertically or horizontally.

At the planning stage, we need to think through whether creating modular application pieces will help. Sometimes a "chatty" – application will not be of a great help. At times – this might make sense if "last n time-period" freshness is important. For example, Twitter will lose its appeal if it does not deliver tweets in near-real time. But that may not be the case for "indicating" shipment has reached "in between location" for retailer. So every application has different needs and these needs will guide Scaling approaches to be adopted.

Compute Scalability



Many small/medium versus eXtra large instance

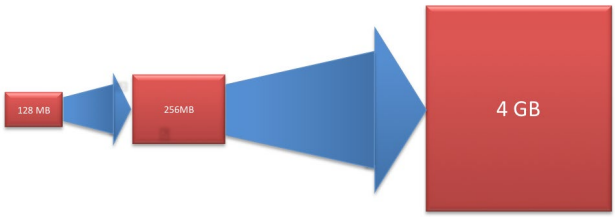


Units of Scale

Compute Scalability is one of the key enablers of Cloud's promise - "horsepower on demand". Its management capability through API makes increasing and decreasing of "horizontal numbers" pretty easy. Let us look at some of the challenges associated with this capability

- The first challenge in utilizing the horizontal scaling of Compute Resources is *localizing state*. In fact lower the dependency of shared state, easier it is to scale horizontally.
- Next Challenge is to think through "downscaling" in honorable manner, by accurately detecting if "compute" node is not doing any work. Having decoupled application architecture is the starting point for leveraging these features.
- Azure provides Scaling Adjustment options via both API and UI. These actions should be done in an auditable way. Right now Azure does not provide comprehensive audit-logs for changes (Azure specific/application specific/metadata change specific). It would be prudent to have a custom solution to persist/analyze these changes.
- Vertical scale in compute allows different capacity of horsepower and memory/attached storage resources (extra small to extra-large). Advantage of Big VM (extra-large) is possible isolation from interference from neighbors. But this decision is not economical and should be implemented with great care.
- ServiceBus and ACS both scale horizontally – They are optimized for throughput and available on-demand. Enabling these services is an additional cost and should be planned for.

Use of Caching



Caching is one of the traditional approaches to achieve scaling. On the cloud, Azure cache or host based cache (Azure's own host based cache or memcached or nosql solutions – mongodb etc.) are good fits to offload "browse/search" workload.

Azure caching has two models -

Shared caching - has been prevalent and provides large sizes up-to 4 GB. It is through a hosted/shared infrastructure and is subject to throttling similar to SqlAzure in terms of bandwidth, memory and connection usage. There is another kind of caching currently in preview. It gives the provision of caching on Azure roles of your choice. This prevents issues around throttling and allows scale limited by Role Instance resources (compute/memory). It also supports memcached protocols, thus allowing smoother migration of the content. It also has a feature where cache can be provisioned on existing web roles thus providing lower cost and locality.

Recently *Solr* has been pushed to Azure with ability to replicate and provide resiliency. It has some latency for an "update" scenario but in general should be given a nudge before settling down for regular web front end/database driven sites, (e.g. Ecommerce sites). Usual performance tips of caching/compressing static assets still holds good and works.

Cache Performance Testing

To measure performance issues, testing on the "test bed on Azure stack" is a trusted option (but be warned that no cloud offering can guarantee same numbers every time). SqlAzure does not provide many of the Dynamic Managament Views (DMVs) or perfmon log access or server side trace usage (all well documented). This means local emulation can indicate only big ticket items.

Throttling of resource requests is how it indicates pressure . Code changes will be required to handle retry logic.

Messaging

Azure provides two kinds of messaging/queuing platforms. Windows Azure Queue is storage services backed. It provides REST based interface and generally used for within application. Messages can live upto 5 days and go upto 64 KB.

Service Bus Queues on other hand provides publish/subscribe on simple queuing. Service Bus queues are directed towards integrating hybrid applications across different protocols and data serialization formats. It also provides longer time to live (7 days) for messages with size upto 256 KB.

Service bus queue also provides inorder delivery, local transaction support. The Service Bus Queue's publish/subscribe enables build applications that can distribute event notifications and data to occasionally connected clients, such as smart phones or tablets.

Multitenancy Pivots

Multitenancy has to be thought in terms of customization allowed at UI/Domain and Data Layers with respect to updates/patches applied to specific application. Isolation of units can result in increase of cost but allows improvement in terms of granular customization. Of course too much of customization is not a great idea. Choosing of update/fault domains for ensuring right uptime is essential.

This also means you need to think through namespaces abstractions over resources that can't be made multitenant (authentication) for metering purpose.

Storage Scalability

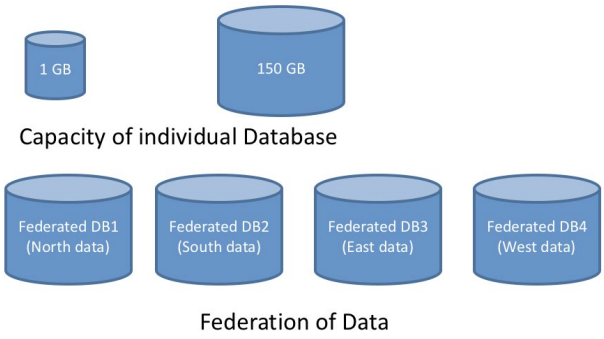
Azure Storage is designed for sustained throughput with good availability parameters. It is highly scalable. Generally if you err on higher provisioning and lower usage – it is a safer practice. It allows scale at provisioning of larger units of storage.

Database Size and Growth Patterns

SQL Azure allows growth of up to 150 GB. Most of the single DB applications are good to go with this limit. For larger volume, one needs to look at federation very carefully.

- The Idea of federation is to scale out data tier horizontally. This is achieved by splitting rows across federation members using distribution key.
- Federation also helps in multi-tenancy for SaaS applications that need to isolate data at database instance level. For multi-tenanted databases, after using the federation, the instance size limit will still be relevant.
- One can include several tables for federation but it is not advisable

- Tips for existing code - Using federation in an application has well documented limitations. Also, to get started with use of a Federated DB, certain changes are necessary. Key points being
  - Tables need to be created for Federation.
  - Identity columns are not allowed (now), so a careful key generation is important.
  - Reference table update management has to be done separately.
  - Currently vertical scaling in terms of memory/cpu resources in SqlAzure is not allowed.



- Usage of functional sharding (handled at the application level) is another brute force method to do sharding of the data and dividing the traffic, but involves lot of planning and maintenance. SQL Azure federation can do splits much more easily.

Planning for Large Data Storage

Once the growth of data is predicted to be more than 150 GB, one has to think of viable usage of Azure storage where limits are up to 100 TB for keeping old data. Key considerations are

- Download/Update and subsequent "pull/restore" have to be well designed and tested.
- At times, decision may have to be taken for "flattening" the database schema to allow efficient usage of the Azure storage.
- With new Azure Spring 2012 offering – SQL Server 2012 itself can be hosted inside VM. This offering makes it very easy to migrate as is existing application. It also helps to support a very large monolithic database that can scale up to 8 cores and Terabytes of data.
- Advantage of separate SqlAzure database instances for tenants allows individual updates/SLAs.

Query patterns/Data Movement and impact on Bandwidth and Transaction cost

In the 'On Premise' world, bandwidth usage and storage is what most of the people are worried about. In case of Cloud based solutions, it is bandwidth usage (sometimes ingress and egress) that contributes to costs. Usage of resources that were not traditionally on a transaction count may now be affected.

For example, using Windows Azure Blobs versus writing to disk OR using Access Control and Storage. This also means if you move stuff to/from SqlAzure to Storage/ Compute, too often and in bursts; it can have impact on cost. Another example is, if there is a "process to upload file locally and do work", migration to blob can introduce new costs.

Today all your assets are on physical disk/folder structure – once it moves to blob storage/CDN combine, there is clear costing attached to using them based on storage and transactions. It would be prudent to prune this collection and monitor the usage.

To give a simple example – Each message put on the queue is counted as one transaction. On the other hand, consuming a message, which is a 2-step process involving the retrieval, followed by a request to remove the message from the queue, will require 2 storage transactions. Even if No data is retrieved, a dequeue counts as billable transaction.

When using Queue – batching/grouping related messages, compressing them and storing just the reference in the queue would result in lower costs. When using a pull-based model, usage of only one queue listener per role instance, when a queue is empty, is desirable. This is to avoid escalation in cost as the cost of Windows Azure queue transactions increases linearly as the number of queue service clients increases.

New latency introduced due to the platform

Earlier, let us say you used file system to store data, now when you change this to blob – extra latency enters into the picture. This needs to be accommodated, verified and clarified (to your users) before you go live.

- Most of the time, this is related to file uploads and

processing or downloads of reports/data/files.

- For consumption of static assets (images/reports), CDN is good way to distribute the assets closest to the user.
- Uploads need to either go directly to blob/table (Azure storage), or local store of compute storage. Blob storage upload is always going to be more resilient in terms of availability. Co-location of storage/compute also plays important role in reducing the latency and cost impact.
- Azure Storage team has done excellent testing in terms of throughput and latency and results are published on the web and should be used as guidance.
- Multiple rounds of performance testing of various parts of Azure are important and necessary to remove any assumptions. For example, provisioning of resources for SqlAzure is done but authentication scaling is not thorough to accommodate large herd of people trying to do authentication at one end point.
- With Azure spring offering – cache offering comes in local form, where it can be co-located on web front ends. This allows faster lookup and size is limited by memory resources of the web roles plays important role in reducing the latency and cost impact.
- Azure Storage team has done excellent testing in terms of throughput and latency and results are published on the web and should be used as guidance.
- Multiple rounds of performance testing of various parts of Azure are important and necessary to remove any assumptions. For example, provisioning of resources for SqlAzure is done but authentication scaling is not thorough to accommodate large herd of people trying to do authentication at one end point.
- With Azure spring offering – cache offering comes in local form where it can be co-located on web front ends. This allows faster lookup and size is limited by memory resources of the web roles.

2 Application Lifecycle Management (ALM)

Development/Testing/Deployment is not very different on Cloud. In fact it becomes more stringent and prudent due to cost of running instances or databases.

You will need to have cloud based testing/staging/backup environments that need to be shutdown/deleted/recreated on-demand.

Client side development emulates storage and SqlAzure. But they are just emulations – you really can't emulate AppFabric etc. For performance testing, one needs a "close to the metal" environment – in this case as close to actual Azure environment. One of the issues in cloud environment is "non-repeatability" of exact performance test results, issues due to "neighbor presence/absence", resets due to hardware issues or throttling as in case of SqlAzure. Extra diligence is required to "repeat/rinse" over x iterations to ensure numbers are reliable. Tools like *LoadStorm* make this job easier and it would be a sin not to use it.

One has to depend on using scripts to store and retrieve code/schema/metadata in database or asset data in blob-storage. To allow quicker return to older state, usage of multiple labeled blobs, which stores older versions of application in Azure itself, is a better idea. This does not mean one should not use repository, but sometimes for large data, it may not be possible.

In future one can hope for a puppet kind of integration, which allows completely scripted deployment of the "infrastructure/configuration" to allow granular control over process.

Spring release of Azure also allows integration with hosted TFS/Git repositories, to allow continuous integration.

Change of Code

You need to change the code and test it on premise on the developer machine and hosted instance. This should include tests, which involve local storage emulation or physical disk as appropriate. At times, you may want to combine online storage too, just before you deploy the code. At present the code needs to be deployed in full; you can't just change one file and upload it by itself. This means you need to properly define upload domains at least two instances to ensure smooth upgrade.

Spring release of Azure allows creation of websites that share resources in template form, allowing almost instantaneous provisioning of the assets for use.

Create/Update/Delete Metadata

Appropriate tests confirming local changes that do not result in adverse output, need to be confirmed and these changes again checked into blob as SQL statements, so that you can easily roll them back. This is possible if data is pretty small, but if data change is massive between releases for databases, you will need to have a copy of the database on premise which you can sync and keep backed up and restored when required. Generally change (update) in data which is indexed, can result in different query plan and should be confirmed.

At present SQL Azure does not provide access to individual backup and restore. You can use COPY command to copy it to another DB before uploading and then run a process to extract affected data to blob. This is of course followed by tests to ensure everything is peachy. In case you use SqlAzure federation, reference tables are not automatically updated across federated members. This needs to be responsibility of the developer to ensure the table data is in sync.

With the Azure Spring offering, SQL 2012 allows one to take local backup that can be "replicated" to another data center.

Updating Schema

Depending on the kind of schema change, this might require change in code as well as reloading of data, and should be carefully considered.

Updating Blob Storage

You need to execute both local as well cloud tests to ensure everything is fine. To revert test, what will be the best way to revert back? Well ideally your blog update process maintains a log of changes which can be replayed later. Think of a CMS system which needs to delete an entry. Best way would be to delete the reference and let garbage collection happen in an asynchronous way. Depending on size or SLA, you may also decide that automatic garbage collection does not happen.

3 Operations

Azure at present provides a dashboard to indicate the general availability of the services in a datacenter. RSS feeds are used to provide the uptime of datacenters and major features (storage/compute/SqlAzure/ACS). System Center based single pane of monitoring, is making progress and till that time custom "interception/logging" should be enabled to provide peek into important operations.

Capacity Planning

Network bandwidth usage, IO profile and CPU usage profile are usually not available and rarely maintained for 'on premise' applications. It is suggested that Azure adopters get this profile. This also changes once the requirements of integrating with other systems through SSIS, Archival Systems or File based interactions. Earlier all these were assumed to be free. Now based on usage of compute resources and storage, all of this will need to be included in cost. Without doing proper application evaluation and migration discussion, your calculator can show wrong numbers.

Monitoring

At present, high-level information about availability is present across various pieces in forms of dashboards. Caching and compute, each have individual dashboards.

Compute allows shipping of diagnostic information to third parties or custom analyzers/visualizers. Azurescope, Paraleap, Splunk, Loggly all can be used in various ways to analyze the stack.

SQLAzure allows certain Dynamic Management Views (DMVs) and SqlAzure throttling exceptions should be logged and analyzed as part of the whole application. Storage analytics has been added and should be combined with compute or SqlAzure. ACS/ServiceBus require custom monitoring solution and at present do not allow "hooks". System center is improving its coverage of many aspects across Azure stack and should slowly emerge as the most comprehensive solution.

Reachability and "working condition" should be isolated and Gomez or similar solutions should be deployed. Idea of dummy endpoints to verify access and health of system should be encouraged across the stack. This would mean if there are worker roles, they should have a façade for "admin/health ping".

Backup Restore

Right now SQL Azure allows a copy of database and you can use data sync to create nearly same copy continuously. Configuration settings of Compute/ACS/Storage/Web.config etc, should be stored/retrieved from a repository. SLAs will be required to ensure "just feels right" process is in place and tested. If you plan to use caching for reference data, good idea would be to keep a script which can populate a new cache with this data.

In case of very large data which can be federated, a one time upload costs will need to be taken with change in schema. SqlAzure Federated Database right now does not allow taking copy of the database including members.

High Availability and SLAs

High availability has to be thought through as measure of the whole application and its moving components. Azure provides different SLAs across different components. RPO/RTO is not usually given for every one of them.

One has to design keeping in mind that Compute/Storage/Acs/Service Bus as well as SqlAzure can fail. This implies design has to be for fail resiliency. This has impact in terms of code-implementation and retry, timeout of compute/storage and throttling environment of SqlAzure.

It also means, no dependency on knowledge of certain Network resources (IP address to be specific) is to be kept. This also pushes decoupling of the moving pieces to ensure resiliency. Simple example would be processing of "work-flow sending email as part of notification". It need not be done synchronously and immediately. It can pile up in the Azure queue and reference data across storage or SqlAzure. A worker process can pick the work items and co-ordinate to send the mails. This also means use of Async mechanisms (Task and Friends in .Net Framework). Moreover, when one really think through the usage of CQRS in appropriate places.

Many people look for offloading of premise workload and look for load balancing. This is not possible with Azure Traffic manager for application front-ends hosted outside Azure. Geo-Replication is coming to various moving pieces as Azure team improves the coverage.

Scheduled downtimes vary for each component. For SqlAzure they are notified 5 days in advance. Scheduled downtime of fewer than 10 hours per calendar year is not considered downtime for purposes of the SqlAzure SLA.

Feature	SLA
Cloud services(compute)	99.95
Storage	99.9
SQL Azure	99.9 – billing month availability
SQL reporting	99.9 - billing month availability
Service Bus	99.9 - billing month availability
Access control	99.9 – billing month availability
Caching	99.9 - billing month availability
CDN	99.9 - billing month availability

SqlAzure Continuity

SqlAzure stores 3 replicas of user database and failover happens automatically with a throttling error. To provide more resiliency to handle data center failover, one can export data to blob (outside) or copy to local blob (and use it's geo-replication) or copy to on-premise location. Data sync services if used in this scenario, need to be tested for failover under stress condition in terms of acceptable data latency. Spring release of Azure allows SQL 2012, but High Availability features are not enabled in preview.

Compute Continuity

Azure infrastructure maintains a system to monitor, provision or decommission the dead or under-maintenance nodes. It also notifies the running code in terms of restart. It can do system updates in a controlled manner if backup domains are created to support the application.

Storage Continuity

Storage too stores three copies of data. It also has geo-replication feature to handle data-center failure.

All eggs in one basket

Migration of data from/to another cloud offering to provide Disaster Recovery solution has to be custom developed at this juncture. Migration of PaaS compute units is a different matter and has to be thought through. Units of operations change through the various offerings for different platforms (java/ruby/node.js etc.). In future, one can hope for some standards to help in this regard. Think of ODBC, JDBC, SQL standards and you get the picture.

4 Azure Access Control

Azure ACS 2.0 allows for federated claims based authentication with third party entities (LiveId, Google, Yahoo, Facebook). This allows consumer applications to quickly adopt social media identities. For enterprise applications, ADFS is an option. Azure connect has been used experimentally to allow AD domain controller to join the connect group but usually frowned upon from security sense. OAuth 2.0 draft is supported via ACS.

With Windows Azure Virtual Network, you can both create secure site-to-site connectivity, as well as create protected private virtual networks in the cloud. The new offering has certain limitation in terms of "modifiability" once it is implemented – it usually means – redoing many things. So this has to be thought through very well.

SQL Azure Spring 2012 offering also has stepped up to provide a name resolution service which provides support for third party DNS support.

5 Image based Deployment

Windows Azure Virtual machine services of June 2012 release allows much easier solution for quickly moving an existing workload from on-premises to the cloud or for building new applications that have dependencies which require a server with persistent local storage. There are three different ways of provisioning VMs (Library, custom Image or own VHD). The disks in a virtual machine are stored as page blobs in Azure Storage. This brings benefits of Azure storage in terms of reliability/availability (geo-replication). Only difference in deployment is that we have product slot minus the staging slot for this role. Local D drive is not persistent across reboots/crashes. Load balancer can be configured to load balance the traffic across set of VMs based on endpoints (name, protocol, Internal IP, public IP etc). LoadBalancedEndpointSetName (LBSetName in powershell) allows this configuration.

Feature migration

Attribute	Suggestion
SMS	Integrate with SMS aggregator using HTTP rest APIs. There is no static ip available but domain address is available. Persistent IP is available as part of the Windows Azure Virtual Network ability.
EMAIL	Integrate email via relay functionality, Azure does not allow sending emails. SendGrid and similar vendors do a decent job.
FTP	FTP is now available in the platform. With regard to the content - when your server is restarted or moved, all locally stored files are deleted, too. So you need to store the FTP uploads somewhere persistently
Broadcast/TCP-IP	By default, cloud network is closed for safety and isolation requirements. Azure does not support broadcast/IP protocols internally.
COM Components	Please remember that ASP.Net in Azure is 64bit only - relatively few ATL-COM objects were written as 64bit. One could try to find RegFree process works, and it will not work for dlls which are bundled with certain apps (ms office, ie – dlls like SHDocVw.dll, MSXml.dll or ocr capabilities). This rules out class of applications which use them and distribute them at their own will.
Visual Basic Support	There is no native support for visual basic on its own. At present all visual basic component (dll) code should be migrated.
Crystal Report Support	http://forums.sdn.sap.com/thread.jspa?threadID=1626975 - will get errors like OutOfMemoryException, file not found, please be very careful in promising support for Crystal reports on Azure. Another option is to migrate to Azure Reporting services, after careful evaluation of features.
Sticky Sessions	First suggestion would be to move session persistence out of process and measure the impact on cost. Then remove the dependency on sticky sessions.
Migration from existing OS/database	Migration is easiest if application runs on Windows 2008 and SQL server 2008 at least.

6 Non MS Platform Support

Azure aims to be a first class host for frameworks like node.js, PHP, Java, Ruby etc. It provides client libraries for Azure storage and management in native platform.

Frameworks - Spring release of Azure allows use of PHP, Ruby, Node.js natively. Microsoft is also making significant inroads in supporting popular technologies (redis/mongodb/solr/AMQP) natively on Windows platform.

OS support - Centos, Ubuntu, SUSE are supported through libraries currently.

Database - Spring release of Azure allows use MySql with Linux Virtual Machines and plans are on to allow increase in size of the database instances. Hosted multi-tenant database like SqlAzure is not available for other databases.

IN CONCLUSION

We saw a big list of things that are important for people to keep in mind when considering Azure adoption. This list has been compiled over time as Customer Feedback and questions have come in with respect to Azure. Azure holds a lot of promise as a cloud based platform and now you are well aware of the multiple pivot points that will help you either realize your first Azure project or help you migrate your in-house/self hosted project to Azure, successfully. Welcome to the Cloud! ■

Govind Kanshi works as a Technical Director in the Microsoft Technology Center (MTC). You can follow him on twitter at @govindk