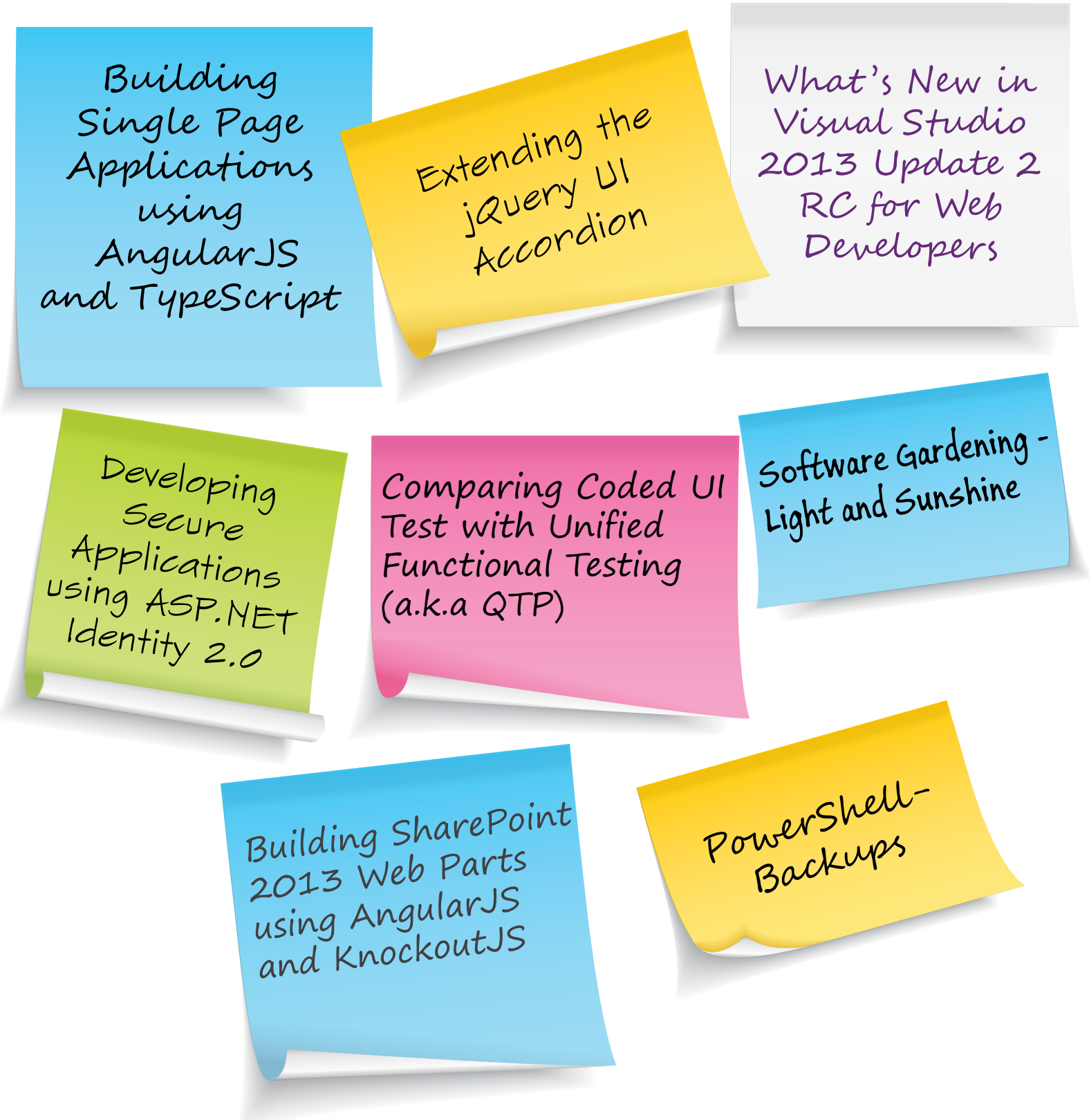
ISSUE 12 | MAY -JUNE 2014

DNCMagazine

www.dotnetcurry.com



CONTENTS

EDITOR'S NOTE

The lyrics were old, but the song was a new one - Windows Everywhere, Windows Everywhere ¶ . The recently concluded Microsoft Build conference portrayed a newer vibrant Microsoft, prepared to take on the future!

Cortana (a Bing powered, voice controlled assistant for Windows Phone and a Siri rival) took the centerstage as Microsoft updated Windows, Windows Phone, MSOffice, Visual Studio, Azure and more. With the news of 'Universal apps' running on every Microsoft device, a free version of the Windows OS for phones and tablets under 9-inches, a new version of Windows for 'Internet of Things' devices; it is indeed Windows Everywhere 2.0 - a comeback song, by a humbler Microsoft!

Good things come in pairs. In this edition, Mahesh Sabnis covers the latest Visual Studio 2013 Update 2 RC for Web Devs as well as ASP.NET Identity 2.0. In our Software Gardening column, Craig Berntson takes the Light and Sunshine analogy to explain why and how as a developer, you must take responsibility for your skills and career.

This month we have Ravi Kiran joining our exclusive club of authors, with a wonderful article showing how to build Single Page Applications (SPA) using AngularJS and TypeScript. Welcome Ravi!

Plus we have SharePoint Web Parts using AngularJS and Knockout, Extending jQuery UI, PowerShell for Backups and an awesome comparison of Coded UI Tests with UFT (a.k.a QTP), to pin to your soft board.

E-mail us at suprotimagarwal@dotnetcurry.com or tweet @dotnetcurry and keep the feedback coming!

Suprotim Agarwal
Editor in Chief

CREDITS

#dncmag www.dotnetcurry.com

Editor In Chief Suprotim Agarwal
suprotimagarwal@dotnetcurry.com

Art Director Minal Agarwal
minalagarwal@a2zknowledgevisuals.com

Contributing Writers Craig Berntson, Gouri Sohoni, Gregor Suttie, Mahesh Sabnis, Pravinkumar Dabade, Ravi Kiran, Suprotim Agarwal

Reviewers: Alfredo Bardem, Baishali Mandal, Carol Nadarwalla, Suprotim Agarwal

Next Edition 1st July 2014 (2nd Anniv Edition)
www.dncmagazine.com

If you are a web developer active on the social media and have read about the latest web trends, then you don't need an introduction to AngularJS. It is currently one of the hottest JavaScript frameworks for building Single Page Applications. AngularJS started catching the attention of many developers around the world because of its strong emphasis on code quality and testing and the ease with which one can get started with it and quickly build single page applications. AngularJS contains everything that one needs to build a full-fledged JavaScript based application, along with promoting good client-side coding practices.

Getting Familiar with AngularJS

In the 10th edition of DNC magazine, Sumit Maitra posted an excellent article on Building an app using Angular JS and ASP.NET MVC; you can take a look at it if you need an introduction to AngularJS.

BUILDING SINGLE PAGE APPLICATIONS USING ANGULARJS AND TYPESCRIPT

GETTING FAMILIAR WITH TYPESCRIPT:

TypeScript is a language created by Microsoft, which gets compiled into JavaScript. As the name itself suggests, TypeScript brings type checking capabilities into JavaScript. Since most of us are C# developers, TypeScript looks much like a combination of JavaScript and C#. TypeScript has most of the features of Object Oriented languages like C# and Java. The language designers didn't try to invent anything, so one doesn't need much time to get started with TypeScript.

TypeScript comes with a set of primitive types:

- number: to represent numeric values
- string: to represent string values
- boolean: to represent Boolean values
- any: to represent any type of value, to be used only when there are multiple possibilities

To create custom types, we need to define classes or interfaces depending on your application's needs. Types can be inherited to create child types. Type inheritance in TypeScript is compiled into prototypal inheritance in JavaScript.

A simple TypeScript class looks like the following:

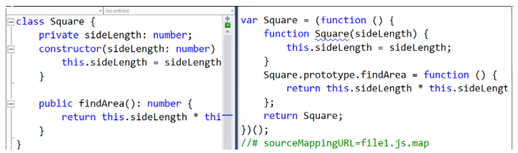
```
class Square {
  private sideLength: number;
  constructor(sideLength: number) {
        this.sideLength = sideLength;
  }

  public findArea(): number {
        return this.sideLength * this.sideLength;
  }
}
```

The compiled JavaScript of the above class is a self-executing function that returns a JavaScript constructor function, that takes one argument. The above class compiles to:

```
var Square = (function () {
    function Square(sideLength) {
        this.sideLength = sideLength;
    }
    Square.prototype.findArea = function () {
        return this.sideLength * this.sideLength;
    };
return Square;
})();
```

As we saw, we need to specify types for every field and method in TypeScript. This may pop a question in your mind on the possibility of using existing JavaScript libraries in TypeScript. Luckily, we don't need to rewrite the libraries in TypeScript; we just need to have a TypeScript declaration file for the library available. The type declaration file doesn't define anything; it just holds declarations of all objects that the library exposes, of which most of them are interfaces.

Visual Studio 2013 provides rich editing support for TypeScript. The file gets compiled and corresponding JavaScript file is generated as soon as the source file is saved. We also get a nice split view to see the TypeScript and JavaScript equivalent side-by-side.



If you need some more inputs on TypeScript, check out Hello TypeScript – Getting Started.

BUILDING THE ONESTOPTECHVIDEOS SAMPLE APP USING ANGULARJS AND TYPESCRIPT

Now that we got some background on TypeScript and AngularJS, let's discuss about using TypeScript to write an AngularJS application. Let's set up a project in Visual Studio 2013 to start creating the application. Open Visual Studio and create a new Empty Web Project named *OneStopTechVids*. As the name suggests, the application is used to maintain a list of training programs offered by a Video-based virtual training company. To the project, add the following NuGet packages:

- AngularJS.Core
- AngularJS.Route
- Bootstrap
- AngularJS.TypeScript.DefinitelyTyped

The fourth package mentioned in the above list is the type declaration file for AngularJS.

CREATING AN ASP.NET WEB API SERVICE TO SERVE DATA

As the data resides on the server, let's create an ASP.NET Web API service to serve data to the application. Since the focus of the article is to use AngularJS and TypeScript to build SPA's, I am going to store the data in the memory, in form of static collections. Let's create two simple classes first: TechVideo and Category to represent structure of the data.

```
public class TechVideo
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
    public int Category { get; set; }
    public string Description { get; set; }
    public int Rating { get; set; }
}

public classCategory
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Editorial Note: With the release of TypeScript 1.0, it has become a fully supported language in Visual Studio 2013 Update 2 and is available as a plug-in for Visual Studio 2012 and WebMatrix.

In a separate class, let's fill in some sample data that has to be used in the SPA:

```
public class TechVideosData
{
    public static List<TechVideo>TechVideos;
    public static List<Category> Categories;
    static TechVideosData()
    {
        Categories=new List<Category>()
        {
        new Category(){Id=1, Name=”JavaScript”},
        new Category(){Id=2,Name=”ASP.NET”},
        new Category(){Id=3,Name=”C#”},
        new Category(){Id=4,Name=”HTML”},
        new Category(){Id=5,Name=”CSS”},
        new Category(){Id=6,Name=”Patterns
        and Practices”}
    };

    TechVideos=new List<TechVideo>()
    {
        New TechVideo(){Id=1, Title = “JavaScript
        Patterns”,Author=”Ravi”, Category=1,
        Description=”Takes a close look at most of the
        common patterns in JavaScript”, Rating=4},
        new TechVideo(){Id=2, Title =
        “AngularJSFundamentals”,Author=”Suprotim”,
        Category=1, Description=”Teaches basics of
        Angular JS. Introduces the framework and
        dives into the concepts around.”, Rating=4},
        // More test data
    };
}
}
```

We have two simple API services to perform operations on the above data. They are as follows:

```
public class CategoriesController : ApiController
{
// GET api/<controller>
public IEnumerable<Category> Get()
{
    return TechVideosData.Categories;
}
}
```

```
public class TechVideosController : ApiController
{
    // GET api/<controller>
    public IHttpActionResult Get()
    {
        var videos = TechVideosData.TechVideos;
        return Ok(videos);
    }

    // GET api/<controller>/5
    public IHttpActionResult Get(string title)
    {
        var video = TechVideosData.TechVideos
        .FirstOrDefault(tv =>tv.Title.Equals(title,
        StringComparison.InvariantCultureIgnoreCase));
        if (video != null)
        {
        return Ok(false);
        }
        else
        {
        return Ok(true);
        }
    }

    // POST api/<controller>
    public IHttpActionResult Post([FromBody]TechVideo
    value)
    {
        var maxId = TechVideosData.
        TechVideos.Max(vid =>vid.Id);
        value.Id = maxId + 1;

        TechVideosData.TechVideos.Add(value);
        return Ok(value);
    }

    // PUT api/<controller>/5
    public IHttpActionResult
    Put(int id, [FromBody]TechVideo value)
    {
        for (int counter = 0; counter
        <TechVideosData.TechVideos.Count; counter++)
        {
        if (TechVideosData.TechVideos[counter].Id ==
        id)
        {
            TechVideosData.TechVideos[counter] = value;
            return Ok();
        }
        }
        return NotFound();
    }

    public IHttpActionResult
    Patch(int id, [FromBody] TechVideo value)
    {
        for (int counter = 0; counter
        <TechVideosData.TechVideos.Count; counter++)
        {
        if (TechVideosData.TechVideos[counter].Id ==
id)
        {
            TechVideosData.TechVideos[counter].Rating =
value.              Rating;
            return Ok();
        }
}

return NotFound();
}

    // DELETE api/<controller>/5
    public IHttpActionResult Delete(int id)
    {
        for (int counter = 0;
        counter <TechVideosData.TechVideos.Count;
        counter++)
        {
           if (TechVideosData.TechVideos[counter].Id ==
           id)
        {
            TechVideosData.TechVideos.
            RemoveAt(counter);
            return Ok();
        }
        }
return NotFound();
    }
}
```

BUILDING THE SINGLE PAGE APPLICATION (SPA)

Creating Angular module and configuring Routes:

The sample app that we are going to create in this walkthrough has three pages: List videos, Add a video and Edit a video. I am not going to cover the Edit video view here, but the code is made available in the downloadable file at the end of this article. I am leaving it to the readers to read the code and understand the functionality of the page.

To start with, add a new TypeScript file to the project and add references to Angular and Angular route to the file.

```
///<reference path=”../scripts/typings/angularjs/
angular.d.ts” />

///<reference path=”../scripts/typings/angularjs/
angular-route.d.ts” />
```

Delete all the default code in the file and create a module. A module in TypeScript is similar to a namespace in C#. It can hold definitions of a number of types. While compiling to JavaScript as a function, you can also add some logic directly to the module, but amount of logic should be minimized.

```
module OneStopTechVidsApp {
}
```

First thing that we add to any Angular JS application is creating a module and configuring routes. Statement for creating module remains the same in TypeScript as in plain JavaScript.

```
var app = angular.module(“techVidsApp”, [‘ngRoute’]);
```

Let's try to understand the way a configuration block is created in JavaScript:

```
app.config(function (dependency) {
//logic
});
```

The config block expects an executable function to be passed inside it. This function is going to be called immediately and it is not going to be instantiated. In the context of TypeScript, the function can be defined as one of the following:

• A class with a constructor and with no other functional components

• It can even be a class with a static function and the static function would be invoked while registering the config block

Let's use the first approach to define the config. We will use the second approach later for another component. The config block needs *$routeProvider* for registering routes of the application. We can use the *config.$inject* to inject the dependency or, specify the dependencies in array format while registering the component. Following is our config block:

```
export class Config {
    constructor($routeProvider: ng.route.
    IRouteProvider)
    {
        $routeProvider.when("/list", {
           templateUrl: "App/Templates/VideoList.html",
           controller: "TechVidsListCtrl" })
        .when("/list/:id", {
           templateUrl: "App/Templates/VideoList.html",
           controller: "TechVidsListCtrl" })
        .when("/add", {
           templateUrl: "App/Templates/AddVideo.html",
           controller: "AddTechVideoCtrl" })
        .when("/edit/:id", {
           templateUrl: "App/Templates/EditVideo.html",
           controller: "EditTechVideoCtrl" })
        .otherwise({ redirectTo: '/list' });
    }
}
Config.$inject = ['$routeProvider'];
app.config(Config);
```

Factory to interact with Web API:

Let's create a *factory* with a number of asynchronous operations to talk to either Web API or to the local data cache. A factory is a function that returns an object. In general, JavaScript implementation of a factory looks like the following:

```
app.factory('name', function(dependencies){
    //Logic
    return{
        field1:value1,
        method1: function1,
        //Other members to be returned
        };
});
```

Simulating the exact same behavior in TypeScript is a bit challenging. Let's rephrase the factory as follows to make it easier to create it in the form of a class:

```
app.factory('name', function(dependencies){
    function FactoryType{
        this.field1=<somevalue>;
        this.method1 = function(){
        //Logic
        };
        //Other members of the FactoryType
    }
return new FactoryType();
});
```

The object of the type *FactoryType* would look the same as the object returned in the first case; but it is created in a different way.

In the TypeScript class, we can create a static method that returns an object of the same class and register the static method as a type. The code skeleton looks like the following:

```
class MyClass{
constructor() {
    //Logic of constructor
    }
    method1(): return -type{
        //Logic in the method
    };
    public static MyClassFactory(){
        return new MyClass();
    }
}
app.factory('myFactory', MyClass.MyClassFactory);
```

Since our factory is going to talk to Web API endpoints, we need the dependencies of *$http* and *$q* in it. Also, as we will be dealing with custom objects for Video and Category, let's create simple classes to hold the properties that we need in these objects. I prefer creating a separate module for creating the custom types, this keeps the declarations and the definitions separated. Following is a module with Video and Category classes:

```
module Extensions {
export class Video {
        id: number;
        title: string;
        description: string;
        author: string;
        rating: number;
        category: number;
    }
export class Category {
        id: number;
        name: string;
    }
}
```

In the factory class, we will have a set of private fields that have to be available for all the methods. We will set values to these fields in either constructor or in the methods. Following is the factory class with the required private fields, constructor and the static factory method:

```
export class TechVidsDataSvc {
    private videos: Array<Extensions.Video>;
    private categories: Array<Extensions.Category>;
    private techVidsApiPath: string;
    private categoriesApiPath: string;
    private httpService: ng.IHttpService;
    private qService: ng.IQService;

    constructor($http: ng.IHttpService, $q:
    ng.IQService)
    {
        this.techVidsApiPath = "api/techVideos";
        this.categoriesApiPath = "api/categories";
        this.httpService = $http;
        this.qService = $q;
    }

    public static TechVidsDataSvcFactory
    ($http: ng.IHttpService,
    $q: ng.IQService): TechVidsDataSvc {
        return new TechVidsDataSvc($http, $q);
    }
}
```

All methods in the factory would be asynchronous and return promises. The private fields, videos and categories would be used to cache data locally and use them to serve data to the application whenever needed.

Let's start adding logic to the factory by adding a method to fetch all videos. This method will make a request to the API service if the videos are not already loaded. Otherwise, it would return the cached data. In both cases, the data would be wrapped inside a promise to make behavior of the method consistent. When it fetches data from the service, it updates the local cache with the data received. It is good to provide an option to forcefully refresh data from the server when needed; it can be done by accepting a Boolean value to this method. Following snippet shows the implementation:

```
getAllVideos(fetchFromService?: boolean)
: ng.IPromise<any> {
    var self = this;

    if (fetchFromService) {
        return getVideosFromService();
    }
    else {
        if (self.videos !== undefined) {
            return self.qService.when(self.videos);
            }
        else {
            return getVideosFromService();
        }
    }

    function getVideosFromService() : ng.IPromise<any>
    {
        var deferred = self.qService.defer();
        self.httpService.get
        (self.techVidsApiPath).then(function (result:
        any)
        {
            self.videos = result.data;
            deferred.resolve(self.videos);
        }, function (error) {deferred.reject(error);
    });

    return deferred.promise;
    }
}
```

As you can see, I created a function inside the method to avoid repeating the code to fetch videos from the API.

To add a video, we need to make an HTTP POST request. Once the API call is successful, the video object is added to the local cache after assigning id of the video received in response to the API call. This is done to keep the local copy consistent with the copy on the server. Following is the method that adds a video.

```
addVideo(video: Extensions.Video): ng.IPromise<any> {
    var self = this;
    var deferred = self.qService.defer();

    self.httpService.post(self.techVidsApiPath, video)
    .then(function (result) {
        video.id = result.data.id;
        self.videos.push(video);
        deferred.resolve();
    }, function (error) {
           deferred.reject(error);
        });
    return deferred.promise;
}
```

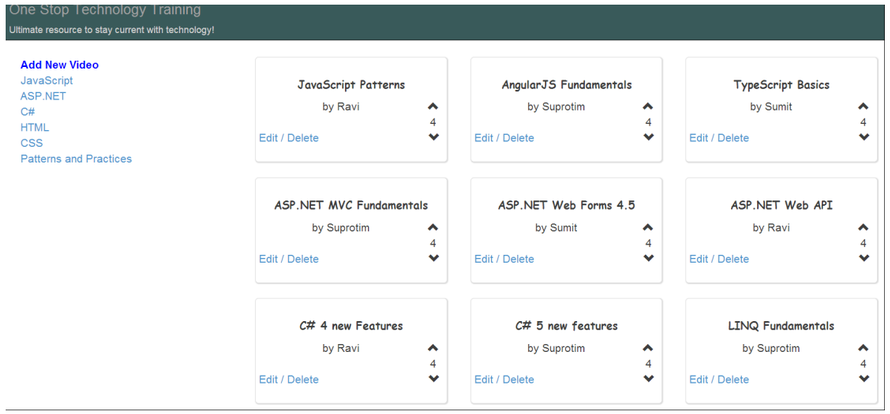
Other methods in the service follow a similar pattern. You can take a look at them in the source code.

Creating Main page and Menu

The page we are going to design has a menu on the left side with links to browse videos by categories or add a new video. At the centre, the page would contain the *ng-view* directive, the place where templates of other pages would be rendered.

Following is the mark-up in the main page:

```
<div class="navbarnavbar-inverse navbar-fixed-top"
style="background-color:darkslategray;">
    <div class="container">
        <div class="navbar-header">
        <a href="/TechVids.html" class="navbar-brand">
                One Stop Technology Training
        <h6>Ultimate resource to stay current with
        technology!</h6>
        </a>
        </div>
    </div>
</div>
<section class="content">
<div class="container" style="margin-top:10px;">
    <div class="col-md-3"
    ng-controller="TechVidsCategoryCtrl">
    <ul class="list-unstyled" style="font-size:14px;">
        <li><a href="#/add"style=" color:blue; font-
        weight:bold;">Add New Video</a></li>
        <li ng-repeat="category in categories">
        <a href="#/list/{{category.id}}">{{category.
        name}}</a></li>
    </ul>
</div>
<div class="col-md-9" ng-view>

</div>
</div>
</section>
```



We need a small Controller to serve data about categories that are used to generate a list of links on the left menu. In Angular, a controller gets instantiated. So, we can simply create a class and register it as a controller, we don't need to follow any conventions like we did in cases of the factory and the config blocks. But the objects used for data binding have to be made available on the Scope. A general interface type for scopes exists in the Angular type declaration file, but we need to create a child type to accommodate the new objects to be added to the scope specific to the controller.

Add the following interface to the Extensions module created above:

```
export interface ITechVidsCategoryScope extends
ng.IScope {
   categories: Array<Category>;
}
```

The controller for serving the categories is straight forward. It just makes a call to the method in the service and assigns the obtained results to scope.

```
export class TechVidsCategoryCtrl {
    private $scope: Extensions.ITechVidsCategoryScope;
    private dataSvc: TechVidsDataSvc;
    private init(): void {
        var self = this;
        self.dataSvc.getAllCategories().then(function
        (data) {
            self.$scope.categories = data;
        });
    }

    constructor($scope: Extensions.
    ITechVidsCategoryScope, techVidsDataSvc:
    TechVidsDataSvc) {
        this.$scope = $scope;
        this.dataSvc = techVidsDataSvc;
        this.init();
    }
}

TechVidsCategoryCtrl.$inject =
['$scope', 'techVidsDataSvc'];
app.controller('TechVidsListCtrl', TechVidsListCtrl);
```

View to list all videos

Let's design the *View* to list the videos. As you see in the above screenshot, the view consists of a list of *divs* that show details of the videos with an option to change rating of the video and a link for edit/delete. The voting buttons cannot be enabled for any value of rating. The buttons should prevent the user from going below 1 and going above 5. This can be achieved easily using Angular's data binding. Following is the mark-up in the video list view:

```
<div class="col-md-4 rowmargin" ng-repeat="video in
videos">
<div class="thumbnail" title="{{video.description}}">
<div class="caption"><strong>{{video.title}}</strong></
div>
<button class="right glyphiconglyphicon-chevron-up
voting-button" ng-disabled="video.rating == 5" ng-
click="upRate(video.id, video.rating)"></button>
<div style="text-align:center;">by {{video.author}}</
div>
<span class="right" style="margin-right: 9px;">{{video.
rating}}</span>
<button class="right glyphiconglyphicon-chevron-down
voting-button" ng-disabled="video.rating == 1" ng-
click="downRate(video.id, video.rating)"></button>
<br/>
<div><a href="#/edit/{{video.id}}">Edit / Delete</a></
div>
</div>
</div>
```

This view is created to act in two ways: one is to display all videos and other is to display videos based on the category received in the URL. As we already saw, we need to create a scope type specific to this view.

```
export interface ITechVidsScope extends ng.IScope {
    videos: Array<Video>;
upRate(id: number, rating: number): void;
downRate(id: number, rating: number): void;
}
```

The TechVidsListCtrl is responsible to fetch the videos based on route parameter and handle functionality to modify rating of the videos. Following is the implementation:

```
export class TechVidsListCtrl {
private $scope: Extensions.ITechVidsScope;
private $routeParams: Extensions.ITechVidsRouteParams;
private dataSvc: TechVidsDataSvc;

private init(): void {
var self = this;
//Fetching all videos if id is not found in route path
if (self.$routeParams.id !== undefined) {
    self.dataSvc.getVideosByCategory
    (parseInt(this.$routeParams.id))
       .then(function (data) {
           self.$scope.videos = data;
       });
}
//Fetching videos specific to category if id is found in
route path
else {
    self.dataSvc.getAllVideos().
    then(function (data) {
        self.$scope.videos = data;
    });
  }
}

constructor($scope: Extensions.ITechVidsScope,
$routeParams: Extensions.ITechVidsRouteParams,
dataSvc: TechVidsDataSvc) {
var self = this;

self.$scope = $scope;
self.$routeParams = $routeParams;
self.dataSvc = dataSvc;

self.$scope.upRate = function
(id: number, rating: number) {
    self.dataSvc.setRating(id, rating+1)
    .then(function () {
        self.init();
    });
};

self.$scope.downRate =
function (id: number, rating: number) {
    self.dataSvc.setRating(id, rating - 1)
    .then(function () {
        self.init();
    });
};

self.init();
}
}
TechVidsListCtrl.$inject =
['$scope', '$routeParams', 'techVidsDataSvc'];
```

View to add a new Video

The add video view is responsible to accept inputs, validate them and post the values to the Web API endpoint. We will use Angular's data validation features to validate the inputs and display the error messages. The list of validations performed on this page is:

- Video should have a title and the title shouldn't be already assigned to any of the existing videos (custom validation directive, will be discussed later)
- A category should have been selected
- Author's name should be entered and it shouldn't have numbers or special characters
- Video must have a description with at least 50 characters to 200 characters.

Mark-up of the page seems heavy because of the validations. Following is the mark-up:

```
<form name="techVidForm" novalidate>
<div class="form-group">
<label for="title">Title</label><br/>
<input type="text" name="title" ng-model="video.title"
required="" unique-video-title/>
</div>

<div class="form-group">
<label for="category">Category</label><br/>
<select name="category" ng-model="category" ng-
options="cat.name for cat in categories" required=""></
select>
</div>

<div class="form-group">
<label for="author">Author</label><br/>
<input type="text" name="author" ng-model="video.
author" required="" ng-pattern="name"/>
</div>

<div class="form-group">
<label for="description">Description</label><br/>
<textarea ng-model="video.description"
name="description" required="" ng-minlength="50"
ng-maxlength="200"></textarea>
</div>

<div class="form-group">
<input type="button" name="edit" value="Add"
class="btnbtn-default" ng-click="addVideo()" ng-disable
d="techVidForm.$invalid"/>
<inputtype="button"name="delete"value="Cancel"class="bt
nbtn-danger"ng-click="cancelVideo()"/>
</div>
</form>

<div style="color:red;">
<div ng-show="techVidForm.title.$dirty&&techVidForm.
title.$invalid">
<span ng-show="techVidForm.title.$error.required">Please
enter a title for the video</span>
<span ng-show="techVidForm.title.$error.
uniqueVideoTitle">Title already used</span>
</div>

<div ng-show="techVidForm.category.$dirty&&techVidForm.
category.$invalid">
<span ng-show="techVidForm.category.$error.
required">Please select a category</span>
</div>

<div ng-show="techVidForm.author.$dirty&&techVidForm.
author.$invalid">
<span ng-show="techVidForm.author.$error.
required">Please enter name of the author</span>
<span ng-show="techVidForm.author.$error.
pattern">Author's name cannot contain numbers or special
characters</span>
</div>

<div ng-show="techVidForm.
description.$dirty&&techVidForm.description.$invalid">
<span ng-show="techVidForm.description.$error.
required">Please enter a description for the video</
span>
<span ng-show="techVidForm.description.$error.minlength
|| techVidForm.description.$error.maxlength">Description
should have 50 - 200 characters</span>
</div>
</div>
```

Following is the controller for this view:

```
export class AddVideoCtrl {
    $scope: Extensions.IAddTechVidScope;
    $window: ng.IWindowService;dataSvc: TechVidsDataSvc;
    constructor($scope: Extensions.IAddTechVidScope,
    $window: ng.IWindowService, dataSvc:
    TechVidsDataSvc) {
        var self = this;

        self.$scope = $scope;
        self.$window = $window;
        self.datavc = dataSvc;

        self.$scope.name = /^[a-zA-Z ]*$/;

        self.$scope.addVideo = function () {
            self.$scope.video.rating = 4;
            self.$scope.video.category = self.$scope.
            category.id;
            dataSvc.addVideo(self.$scope.video)
            .then(function() {
                var category = self.$scope.video.
                category;

                self.$scope.video = {
                    id: 0,
                    title: "", description: "",
                    category: 0, author: "", rating: 0
                };
            self.$scope.techVidForm.$setPristine();
            self.$window.location.href = "#/list/" +
            category;
            });
        };

        self.$scope.cancelVideo = function () {
            self.$scope.video = newExtensions.Video();
            self.$scope.category = null;
            self.$scope.techVidForm.$setPristine();
        };
    self.init();
    }
```

```
private init(): void {
    var self = this;
    self.dataSvc.getAllCategories()
    .then(function (data) {
        self.$scope.categories = data;
    });
}
}
AddVideoCtrl.$inject =
['$scope', '$window', 'techVidsDataSvc'];
```

Directive to validate title

To check if the title assigned to the video is unique, we need to create a custom validation directive. In Angular, a *directive* is a function that returns a special object that has a set of pre-defined properties. In TypeScript, we can create a static method and make it return the required directive object.

For validation, we need two fields in the custom directive:

- require, to be able to use functionalities of *ngModel* directive
- link, to perform validation logic

The link function would invoke the *$setValidity* method of ngModel to set the validity of the input field based on certain condition. Following is the implementation of the directive:

```
export class UniqueVideoTitle {
public static UniqueVideoTitleDirective
(dataSvc: TechVidsDataSvc): ng.IDirective {
    return {
        require: 'ngModel',
        link: function (scope: ng.IScope,
        element: ng.IAugmentedJQuery,
        attrs: ng.IAttributes, ctrl:
        ng.INgModelController) {
            element.bind('blur', function() {
                var viewValue = element.val();
                dataSvc.checkIfVideoExists(viewValue)
                .then(function (result) {
                    if (result === "true") {
                       ctrl.$setValidity
                      ("uniqueVideoTitle", true);
                    } else {
                       ctrl.$setValidity
                      ("uniqueVideoTitle", false);
                    }
                }, function (error) {

ctrl.$setValidity("uniqueVideoTitle", false);
                        });
                });
            }
        };
    }
}
app.directive('uniqueVideoTitle', ['techVidsDataSvc',
UniqueVideoTitle.UniqueVideoTitleDirective]);
```

CONCLUSION

I think, by now you must have got a good understanding on implementing Angular components using TypeScript. As we saw, TypeScript makes strict type checking on the objects and it doesn't let the code compile, unless it resolves all objects. This makes the life of programmers a lot easier as we don't have to cross-check the properties referred on the objects, that would have otherwise remained unknown till runtime ■

Download the entire source code from our GitHub Repository at bit.ly/dncm12-angular

Ravi Kiran is a developer working on Microsoft Technologies. These days, he spends his time on the front-end JavaScript framework Angular JS and server frameworks like ASP.NET Web API and SignalR. He actively writes what he learns on his blog at sravi-kiran.blogspot.com. He is a DZone MVB. You can follow him on twitter at @sravi_kiran

Developing Secure Applications using ASP.NET Identity 2.0

> *ASP.NET Identity is the new membership system for building ASP.NET web applications, phone, store, or hybrid applications.*

Coping up with a variety of business needs and technical and security requirements of different domains like banking, insurance etc. can be a daunting task. In addition to this, we have social networking applications that have millions of users, and it is widely known that any service accessible to the public on the Internet is constantly probed for vulnerabilities. Therefore, it is recommended to build robust security into all of your Web applications and services.

Building a secure Web application is always a challenging task. Although Microsoft with ASP.NET 2.0 introduced a powerful provider-based architecture and a membership provider which could be customized; developers have been craving for something *simpler*. In addition to this, modern websites have become more social and now use social identities for authentication and authorization. Clearly a fresh look into the membership system was needed to cope up with the changes and growing demand.

ASP.NET Identity is the new membership system for building ASP.NET web applications, phone, store, or hybrid applications.

Editorial Note: *It's worth noting that the Simple membership API introduced with WebPages and WebMatrix has become a popular way of managing authentication but it's not very extensible, not compatible with OWIN and it is challenging to store membership system data using NoSQL databases.*

ASP.NET IDENTITY

ASP.NET Identity can be used with all ASP.NET frameworks, such as ASP.NET Web Forms , MVC, Web Pages, Web API, and SignalR.

ASP.NET Identity has been developed with some major security features like **Two-Factor Authentication, Account Lockout, and Account Confirmation etc.** In addition, you can use it to support multiple storage mechanisms like Relational Databases, SharePoint, Azure, NoSQL etc. It is Unit Testable, supports Social Login providers like FaceBook, Twitter, Google etc. and even supports Claim-based authentication. It is fully compliant with OWIN and can be downloaded from the NuGet Package Manager.

Here are the packages we need to download for ASP.NET Identity 2.0.0:

Microsoft.AspNet.Identity.EntityFramework Version 2.0.0 - Contains EF implementations for identity types. These types are used to manage information for identity users, roles, claim login etc.

Microsoft.AspNet.Identity.Core Version 2.0.0 - Contains classes and interfaces for managing users and roles in ASP.NET Identity. It contains classes for User validation, User login information etc.

Microsoft.AspNet.Identity.OWIN Version 2.0.0 - Contains classes used to manage identities associated with OWIN.

To implement ASP.NET Identity in an empty Web Application, the following samples can be installed using the NuGet Package:

Microsoft.AspNet.Identity.Samples -Version 2.0.0-beta2 –Pre

This will add the necessary classes for ASP.NET Identity 2.0.0 in your application and thus ease the development effort. It also provides freedom of changing the code wherever required by your application.

Implementing ASP.NET Identity

Step 1: Open Visual Studio 2013 and create a new MVC application targeting .NET 4.5. Name it as *MVC_Identity*. Once the project is created, you will find the references for ASP.NET Identity 2.0.0 (assuming you installed the NuGet package shown in Step 2)

- Microsoft.AspNet.Identity.Core
- Microsoft.AspNet.Identity.EntityFramework
- Microsoft.AspNet.Identity.Owin

Step 2: To get the basic infrastructure code ready for Identity 2.0.0, we need to install ASP.NET Identity sample. Run the following command:

```
Package Manager Console Host Version 2.8.50313.46

Type 'get-help NuGet' to see all available NuGet commands.

PM> install-package Microsoft.AspNet.Identity.Samples -Version 2.0.0-beta2 -Pre
```

This is an important step for getting the Identity infrastructure ready.

Step 3: After installing the sample, the project will have some additional Controller classes. Based on the Controller classes, new Views will be generated in the Views folder.

Step 4: Open the IdentityConfig.cs class file in the App_Start folder. This class file contains the following classes:

- ApplicationUserManager
- ApplicationRoleManager
- EmailService
- SmsService
- ApplicationDbInitializer

- SignInHelper

The *SignInStatus* enumeration provides the values for SignIn for the user.

So now the question is what are these classes responsible for? Let's explore them one by one along with the new security features provided in ASP.NET Identity.

TWO-FACTOR AUTHENTICATION

Two-Factor Authentication provides an extra security layer for an application's (web site) user account. This is a protection used in case the password of the user gets compromised. This feature uses the mechanism of sending the security code using SMS on the users phone or alternatively a verification email, if the user is not having access to his/her phone. The class *ApplicationUserManager* is inherited from the *UserManager* base class. This is a generic class with the type *ApplicationUser* parameter. The ApplicationUser class is responsible to work with the users identity. The class *ApplicationUserManager* defines a *Create* method. This method contains logic for validating user name and password as shown here:

```
var manager = new ApplicationUserManager(new UserStore<ApplicationUser>(context.Get<ApplicationDbContext>()));
// Configure validation logic for usernames
manager.UserValidator = new UserValidator<ApplicationUser>(manager)
{
    AllowOnlyAlphanumericUserNames = false,
    RequireUniqueEmail = true
};
// Configure validation logic for passwords
manager.PasswordValidator = new PasswordValidator
{
    RequiredLength = 6,
    RequireNonLetterOrDigit = true,
    RequireDigit = true,
    RequireLowercase = true,
    RequireUppercase = true,
};
```

ACCOUNT LOCKOUT

Account Lockout is another important feature provided by the ASP.NET Identity 2.0.0. This locks out the user's account if the user enters a wrong password for a specific number of times.

This can be specified by configuring maximum failed attempts and lockout time span as shown here:

```
manager.UserLockoutEnabledByDefault = true;
manager.DefaultAccountLockoutTimeSpan = TimeSpan.
FromMinutes(5);
manager.MaxFailedAccessAttemptsBeforeLockout = 5;
```

Once the validators and the lockout are set for the account, the Two-Factor authentication can be set using the code provided here:

```
manager.RegisterTwoFactorProvider("PhoneCode", new
PhoneNumberTokenProvider<ApplicationUser>
{
    MessageFormat = "Your security code is: {0}"
});
manager.RegisterTwoFactorProvider("EmailCode", new
EmailTokenProvider<ApplicationUser>
{
    Subject = "SecurityCode",
    BodyFormat = "Your security code is {0}"
});
manager.EmailService = new EmailService();
manager.SmsService = new SmsService();
```

In the above code, *PhoneNumberTokenProvider* and *EmailTokenProvider* classes are used. These classes are inherited from *TopSecurityStampBasedTokenProvider*. This class is responsible for generating time-based codes using the users security stamp, and sending it to users using the Phone number and Email address with the help of PhoneNumberTokenProvider and EmailTokenProvider respectively. Naturally to send Email or SMS, consent is needed. So to implement the consent service, the following classes are used:

```
public class EmailService : IIdentityMessageService {
    public Task SendAsync(IdentityMessage message)
    {
        // Plug in your email service here to send an
        email.
        returnTask.FromResult(0);
    }
}
public class SmsService : IIdentityMessageService
{
    public Task SendAsync(IdentityMessage message)
    {
        // Plug in your sms service here to send a text
        message.
        return Task.FromResult(0);
    }
}
```

This is just a sample. You can change the code here as per your apps requirement.

USING EMAIL SERVICE FOR AUTHENTICATION

Step 5: Add the following code in the *SendAsync* method of the *EmailService* class.

```
public Task SendAsync(IdentityMessage message)
{
    var mailMessage = new System.Net.Mail.
    MailMessage("TestAdmin.Admin@MyApplication.com ",
    message.Destination,
    message.Subject,
        message.Body
);

//Send the Message
SmtpClient client = new SmtpClient();
client.SendAsync(mailMessage, null);

return Task.FromResult(true);
}
```

Here the *MailMessage* class is used which accepts the following parameters:

- Senders Email Address
- Recipients Email address
- Subject
- Body

Using the *SmtpClient* class, a message can be sent. Note that you can use your email server settings here.

Step 6: In the development environment, an email drop folder needs to be configured in the web.config file as shown here:

```
<system.net><mailSettings>
<smtp deliveryMethod="SpecifiedPickupDirectory">
<specifiedPickupDirectory pickupDirectoryLocation="C:\
mailDrop"/>
</smtp>
</mailSettings>
</system.net>
```

The above configuration specifies the SMTP for the Email message pickup.

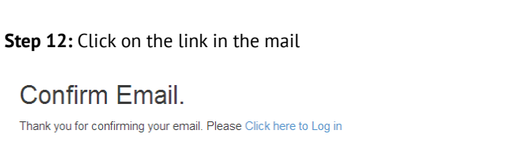
Step 7: The web.config file also adds a new connection string to store the user's information in SQL Server:

```
<add name="DefaultConnection" connectionString="Data
Source=(LocalDb)\v11.0;Initial Catalog=MVC_Identity-1-
14;Integrated Security=SSPI" providerName="System.Data.
SqlClient" /></connectionStrings>
```

Step 8: Open the AccountController.cs and locate the *Register* method with *HttpPost*. This contains the required code to send an email for account confirmation:

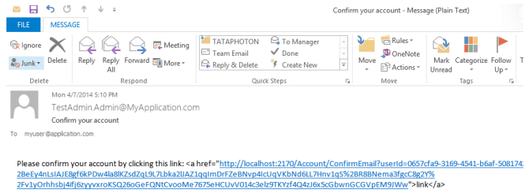
```
public async Task<ActionResult>
Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName =
        model.Email, Email = model.Email };

        var result = awaitUserManager.CreateAsync(user,
        model.Password);
        if (result.Succeeded)
        {
            var code = await UserManager.
            GenerateEmailConfirmationTokenAsync(user.
            Id);
            var callbackUrl = Url.Action("ConfirmEmail",
            "Account", new { userId = user.Id, code =
            code
        }, protocol: Request.Url.Scheme);
        await UserManager.SendEmailAsync(user.Id,
        "Confirm your account", "Please confirm your
        account by clicking this link: <a href=\"" +
        callbackUrl + "\">link</a>");
        ViewBag.Link = callbackUrl;
            return View("DisplayEmail");
        }
        AddErrors(result);
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

Step 9: Run the application and click the Register link on the page, the view rendered will be as shown here:



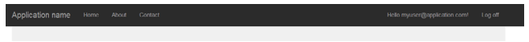
Step 10: Enter your email and other details and click on the register button. You will see a similar view:

Step 11: Navigate to the email drop folder configured in the config file. You will find the mail which can then be opened in Outlook or similar:



Step 12: Click on the link in the mail

Step 13: Now you can login as shown here:



IMPLEMENTING ACCOUNT LOCKOUT

Account Lockout as we briefly touched upon in the introduction section is one of the most important aspects of implementing security which is required in most commercial and finance sites.

Step 1: In the *Create* method of the ApplicationUserManager class, change the *DefaultAccountLockoutTimeSpan* and *MaxFiledAccessAttemptBeforeLockout* properties as mentioned here:

```
manager.UserLockoutEnabledByDefault = true;
manager.DefaultAccountLockoutTimeSpan = TimeSpan.
FromMinutes(1);
manager.MaxFailedAccessAttemptsBeforeLockout = 2;
```

Step 2: To implement the account lockout feature, we need to add the following code at the beginning of the *PasswordSignIn* method after application user is found by its name: (yellow marked code)

```
public async Task<SignInStatus>
PasswordSignIn(stringuserName, string password, bool
isPersistent, bool shouldLockout)
{
    var user = await UserManager.
    FindByNameAsync(userName);

    await UserManager.IsLockedOutAsync(user.Id);
    await UserManager.AccessFailedAsync(user.Id);
    await UserManager.SetLockoutEnabledAsync(user.Id,
    true);
if (user == null)
{
returnSignInStatus.Failure;
}

if (await UserManager.IsLockedOutAsync(user.Id))
{
return SignInStatus.LockedOut;
}

if (awaitUserManager.CheckPasswordAsync(user,
password))
{
return await SignInOrTwoFactor(user, isPersistent);
}

if (shouldLockout)
{
// If lockout is requested, increment access failed
count which might lock out the user
await UserManager.AccessFailedAsync(user.Id);
if (await UserManager.IsLockedOutAsync(user.Id))
 {
   return SignInStatus.LockedOut;
 }
}
return SignInStatus.Failure;
}
```

Here are some important points to note:

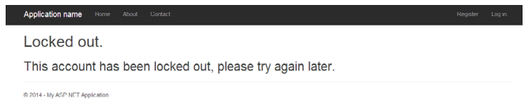
`await UserManager.IsLockedOutAsync(user.Id)` - Returns true if the user is lockout.

`await UserManager.AccessFailedAsync(user.Id)` - Increments the accessfail count for the user. If the failed access count is greater than or equal to *MaxFailedAccessAttemptsBeforeLockout*, then the user will be locked for the time span of the *lockouttimespan*.

`await UserManager.SetLockoutEnabledAsync(user.Id, true)` – Sets if the lockout is enabled for the user.

Step 3: Run the application and create a user as explained above in the Two-Factor authentication section.

Step 4: Try to login with a wrong password. After two failed attempts, you will get the following message:



Now wait for the completion of *DefaultAccountLockoutTimeSpan* and try logging in again with correct password; you will be able to login successfully.

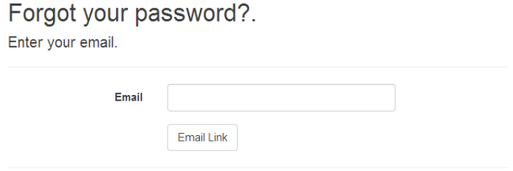
This is an awesome facility provided by the ASP.NET Identity 2.0.0 to protect the user credential information from unauthorized access.

PASSWORD RESET

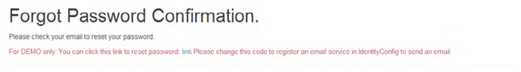
A *Password Reset* feature is also available in case the user forgets it. As we have seen in the register new user section, the verification mail is send to the users registered email address with a link for resetting the password.

In the project, we have existing methods in the *AccountController* class which contains logic for *ForgetPassword* and *ResetPassword*. This functionality is provided with the sample NuGet with ASP.NET Identity 2.0.0 package that we installed at the beginning of this article.

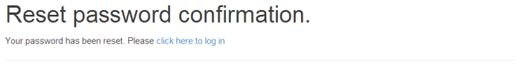
Step 1: Run the application and from the Login View, click on the 'Forget your password' link. You will navigate to the *ForgotPassword* view as shown here:



Step 2: Enter the account email and click on the Email Link button. The following View will be displayed:



Here the email will appear in the drop folder which we configured a short while ago in the web.config file with the attribute *pickupDirectoryLocation*. On clicking the link in the



email, the following view will be displayed:

and a new password can now be used to login.

CONCLUSION

As you might have observed, these new features provided in ASP.NET Identity 2.0.0 provides an enhanced mechanism to manage security of precious credential information in our data stores. A developer instead of customizing a provider from scratch, can now instead rely on the extensible API set provided with the new identity features for security.

I hope this article will help you integrate some robust security features in your new applications as well as help migrate your existing apps that use ASP.NET Membership to the new ASP.NET Identity system ■

Download the entire source code from our GitHub Repository at bit.ly/dncm12-aspidentity

Mahesh Sabnis 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 bit.ly/HsS2on

LIGHT AND SUNSHINE

In the past few issues, I've been setting the stage with the basic requirements for Software Gardening. In this issue, I present the last of the requirements, light and sunshine.

Most plants need lots of light and sunshine. I have a small hydroponic garden in my kitchen. It's easy to grow plants in this garden. I'm currently growing herbs. They were "planted" about three weeks ago. The seeds come pre-planted in plastic cups that you put into the device. You add water and nutrients to start, then each time the device indicates it time to add more. The garden also has a very bright light that runs on a time. It's off for about eight hours at night, otherwise it's on all the time. The light is essential for the plants to grow.

Image: A home hydroponic garden with bright grow light

Light is also essential to your software garden. Yes, I'm going to use an old cliché: you need to keep the light going through continuous learning. Many developers put responsibility of learning on their employer, saying they will only attend conferences or training classes when the employer pays for and requires it. This mentality will keep your skills stagnant and that's not a good thing in an industry that changes as fast as software.

The bottom line is, you must take responsibility to keep your skills and knowledge up to date. It is not the responsibility of your employer. You and you alone must take responsibility for your skills and career. Let's look at the various ways you can learn.

MAGAZINES

The fact that you're reading this column means you read magazines. An advantage of a magazine is they are generally edited and the content can cover the latest technologies, as deadlines are usually close to the publication date.

In addition to the DNC .NET Magazine, there are many others:

MSDN Magazine – This is the official Microsoft .Net magazine. It can be obtained in print through a subscription or read online at the MSDN website. It is published monthly

CODE – Published by EPS in Houston, Texas, CODE is published six times a year and has the second largest subscriber base, behind MSDN.

Visual Studio Magazine – Published monthly, it usually has less content that the other two, but still a good resource.

BOOKS

Books often have a long lead-time, meaning from the time the author starts writing until the book is in print, several months or even a couple of years may have passed. This seems to make books outdated almost as soon as they're available. However, books are good for getting a complete view on a topic as they often start at the basics and gradually work into more complex topics.

Don't think that you need to read an entire book. New Microsoft CEO Satya Nadella said he buys more books than he finishes.

I'm the same way. I have dozens of books that I've bought and started, but never finished.

Some publishers include:

Manning – Known for the old-time drawings on their covers, Manning offers a unique feature called the Manning Early Access Program, or MEAP. As an author finishes writing a chapter of a new book, it's released on the web and available for download. At this point, the chapter hasn't been edited, but an online forum is available for readers to post comments on the new chapters.

O'Reilly – You'll recognize O'Reilly books by the animal drawings on the cover. O'Reilly covers .Net , Java, and Open Source topics.

APress – Yellow and black covers tell you that you have a book from APress.

Pragmatic – PragProg has a handful of .Net books, but mostly produces books on Open Source topics.

LeanPub – Unique among the above publishers, LeanPub allows authors to self-publish. While this gives more of the purchase price to the author, the writing is often a lower quality than the big publishers.

BLOGS

It's easy to get a blog going. Go to wordpress.com and start one. Blog posts generally don't go into a lot of detail, but because new posts can be made at any time, new technologies are covered quickly. Some of my favorite blogs include:

MSDN Blogs – An umbrella place for official Microsoft blogs. You'll find blogs for teams or individual employees.

Los Techies – Home to many very knowledgeable technologists on a wide variety of topics.

MVPs.org – Many current and past MVPs host blogs here.

The Data Farm – Julie Lerman's blog on everything Entity Framework.

USER GROUPS

Surely there is a user group somewhere near you. It may not be on .Net, but you should be able to learn from groups on most topics. Most user groups are free or charge a small fee. Where I live, we have groups on .Net, SQL Server, SharePoint, Java, Ruby, mobile development, JavaScript, Hadoop, PHP, WordPress, architecture, Linux, game development, Agile, DevOps, interaction design, Oracle, Python, big data, and more.

CODE CAMPS

Code Camps are one or two day training events and almost always free and always on the weekend. They consist of several 60 or 75 minute sessions and cover a variety of topics. The largest code camp in the world is in Silicon Valley and has over 2000 attendees. Most code camps I've attended have 300-600 attendees. The other thing that is unique about a code camp is the sessions are proposed and given by members of the community.

REGIONAL CONFERENCES

Smaller regional conferences take place all over the world. They typically run two or three days and charge a fee for attending. Attendees come from the geographical area of the city that hosts the conference. Regional conferences include DevLink, CodeMash, ThatConference, DevWeek, and more.

NATIONAL AND INTERNATIONAL CONFERENCES

National and international conferences are a bit bigger than regional conferences. Attendees and speakers come from all over the world. Attendance can be 500 to 2000 or more. They usually last two to four days and charge fees for attending. Norwegian Developers Conference (NDC), DevConnections, VSLive, and others fall into this category.

MAJOR CONFERENCES

There are a couple of major conferences that come to mind. Microsoft Build and TechEd. Attendance can be up to 10,000 people. Build is generally held in the US. TechEd is usually held in the US, Europe, and Asia. These conferences have a high cost, but speakers are often Microsoft employees.

IN-PERSON TRAINING

The number of training classes seems to be decreasing, but this is still a viable option. There are a number of ways these classes are offered. You may hire a trainer to come to your office and give training to your staff. Sometimes training is held at the training company's office. A third option is where the training company rents a room at a hotel and opens up the class to anyone. Fees are usually quite high, but there is an advantage to having an instructor that can answer questions about specific issue you may have in your work. I've taken classes from several companies. One of my favorites is DevelopMentor.

ONLINE TRAINING

Taking a course online may be the way things are heading. These courses vary in price, but are generally quite affordable and cover a very wide variety of topics. Here are some sites that offer training:

Pluralsight.com – Perhaps the biggest of all the sites. It started out with just .Net training, but then went on a buying spree, purchasing several competitors. Their course now covers just about every technology you can think of including .Net, Java, Open Source, Database, Sales Force, Web, Networking, Creative skills and lots, lots more. Over 3000 courses are available.

Learninglineapp.com – This is the online version of DevelopMentor. They offer mostly .Net courses, but some others are available too. Courses on this site take a different approach from Pluralsight as they come close to duplicating in-person training classes.

MicrosoftVirtualAcademy.com – This is the official Microsoft training site, which means courses only cover Microsoft topics. One good thing is they're free.

KhanAcademy.org – Free courses on a wide variety of topics. The tech-related topics lean more to introductory than advanced, but some advanced topics are available.

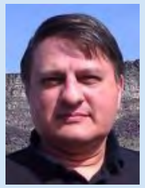
Coursera.org – These are university courses that are free. And the colleges are prestigious, including Stanford, Columbia, Duke, Johns Hopkins, Hebrew University of Jerusalem, Eindhoven University of Technology, and many more. These courses are taught by college professors and include study groups and homework. There is no college credit given, but courses cover a very wide variety of topics.

CodeSchool.com – Courses here cover iOS, HTML/CSS, Ruby, JavaScript, etc. You'll pay a small fee to attend.

It's impossible for me to provide a complete list in all these categories. I'm sure you'll find more magazines, online training, book publishers, etc.

Now you should create a learning plan for you. Don't limit yourself to a single source. Get your information from multiple providers. Pick topics of interest and dive in.

You'll find that the light you shine on your Software Garden will help it grow and stay lush, green, and vibrant ■



Craig Berntson is the Chief Software Gardener at Mojo Software Worx, a consultancy that specializes in helping teams get better. He has spoken at developer events across the US, Canada, and Europe for over 20 years. He is the co-author of "Continuous Integration in .NET" available from Manning. Craig has been a Microsoft MVP since 1996. Email: craig@mojosoftwareworx.com, Blog: www.craigberntson.com/blog, Twitter: @craigber. Craig lives in Salt Lake City, Utah.

COMPARING CODED UI TEST USING VISUAL STUDIO 2013 WITH UNIFIED FUNCTIONAL TESTING (A.K.A QTP)

In a previous Edition of the DNC .NET Magazine (Issue 5 - Mar-Apr 2013] , I had compared Automated Testing Tools Selenium, Coded UI Test and QTP in a broader perspective without going too deep into any one of them. In this article, we will do a thorough comparison between two of these tools - Coded UI Test and Unified Functional Testing (QTP).

Quick Test Professional (QTP) was introduced by Mercury which was later acquired by HP as an automated testing tool for testing various software's and environments. The first version was released way back in 1998 with the name as Astra QuickTest. In 2003, QTP 6.5 was released and now we have a combination of QTP and Service Test as *Unified Functional Testing* with the current version as 11.5. Note: Service Test is used to test Web services.

Coded UI Test (CUIT) is comparatively a new kid on the block. It was made available as a part of Visual Studio 2010 and since then a lot of enhancements have been made to it till the current version with Visual Studio 2013. Even though Visual Studio provides other automated test types like Unit Tests, Web Performance Test, Load Test; we are not getting into the details of any one of them other than CUIT. Note: Web Performance Test can be used to test Web Services.

SETTING UP THE SAMPLE

While doing the comparison, we are going to consider an example of an application that I developed as part of the Employee Management System. This application has two interfaces, one for the desktop (Windows) and the other for the web. With the Windows IDE, the HR personnel can login and enter or edit details of Employees. With the web interface, Employees can find their own information and maintain their passwords.

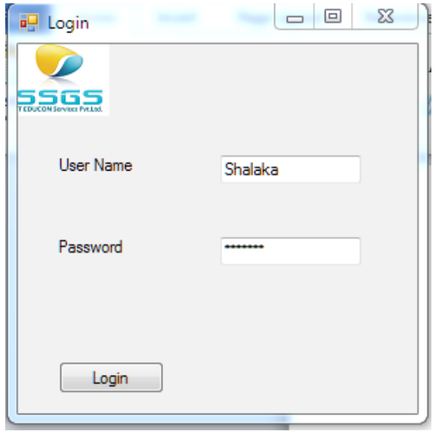
I am including some steps to start the desktop version of the app. You can download a zip file named "SSGS EMS App" from the source code mentioned at the end of this article.

Follow these steps to run the application which will help to reproduce the screenshots I have included in this article.

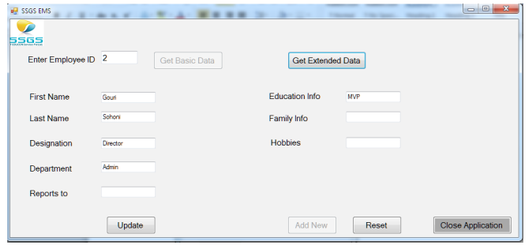
1. Attach the database files to SQL Server by copying it in the folder Program Files\Microsoft SQL Server\MSSQL11. MSSQLSERVER\MSSQL\DATA. Your SQL Server version may be different, so choose the folder accordingly.

2. Double click on "SSGS EMS WinForm App.exe" to start the Windows Application

3. The user name and password for the HR Personnel is "Shalaka" and "Shalaka". Click on the Login button.



4. Enter Employee Id as 2 in the next screen and click on *Get Basic Data*, followed by *Get Extended Data* to get following details:5. The Reset button will clear all textboxes and you can enter any other Employee code.



6. In order to enter a new Employee, enter all details except employee id and click on Add New. As you may have observed, I have not created any validation or followed any coding practices as this application is solely created to help you use the two testing tools, we will be discussing shortly.

Note: Employee Id will be automatically generated. I have used the application only up to the first 4 steps in the article i.e. Record and Playback.

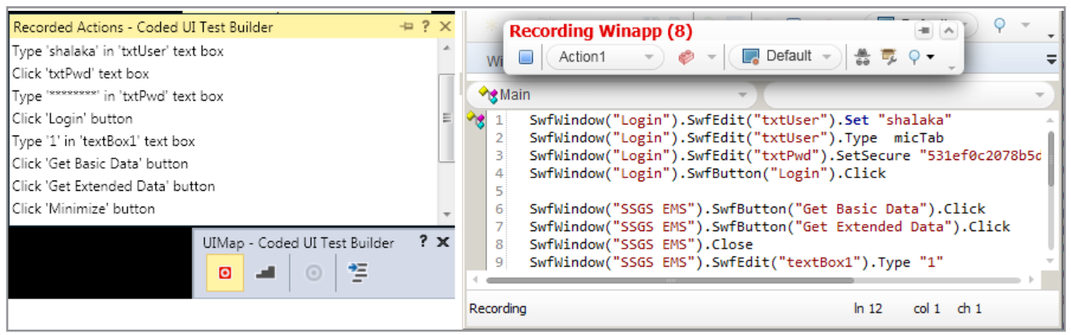
> ***Manual testing*** is a slow process as human factor is involved in it, for execution as well as determining whether the test case passes or fails. Manual testing requires more testers to execute, but less skill. ***Automated testing*** is less time consuming but requires more resources for creating it. With Automation testing, the tool helps in execution and the assertion can be checked programmatically. When a test is needed to be executed in recurrence for checking quality of build or regression, it is more productive to run it as an automated test rather than running it manually every time.

RECORD AND PLAYBACK

UFT as well as CUIT support record and playback feature. Application under test is executed by the tester and every action of the tester as well as changes in the UI objects, is recorded by the testing framework. This recording is then played-back number of times to ensure that every time for the same action, the change is the same in the UI objects. The actions performed are recorded with a tool built into the testing framework and the playback is done by another tool from the same framework to execute the automated tests.

With UFT, we can use the Record option, while CUIT provides Coded UI Test Builder to do the same. The settings for Windows application or Web application can be set using UFT's record and run settings. There are 4 icons with CUIT Builder as - record, show steps, add assertions and generate code. The code will be automatically generated in high level languages like C# or VB.NET. There is another option for creating CUIT by using the already recorded actions while doing manual testing. Usually any test is first run as a manual test and when we realize that it now needs to be re-executed quite a number of times, say as a regression test; it is then automated. In Microsoft Test Manager (MTM), it is possible to record the actions of testers when they are executing a test. While creating CUIT, we can select an option to use that recording, for generating the test code.

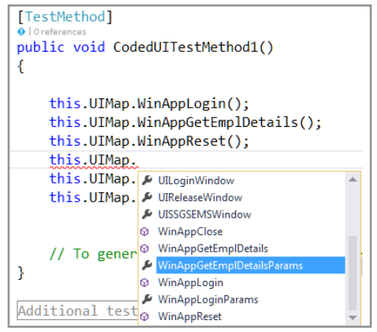
I personally like the feature of MTM for converting manual actions to its code equivalent. It reduces the efforts required by the automated test developer to record the test. Other than that, Record and playback feature is equally easy to use in both the tools.



LANGUAGES USED WITH UFT AND CUIT

The first and foremost difference between these two Automated Testing Tools is the underlying code that is generated or written. While UFT uses a scripting language (VBScript), CUIT is supported by high level language like C# or VB.NET. UFT also provides an additional option of descriptive programming. As we all know, VBScript supports classes but does not provide polymorphism and inheritance, which is the basis of object oriented technology. VBScript also does not have inherent event handlers.

Another difference is the way the test code can be debugged. VBScript code cannot be debugged as such. Though HP added a debugger in its package of UFT, it comes with very limited functionality as compared to integrated IDE, which we have with CUIT as a part of Visual Studio. Visual Studio also provides excellent IntelliSense which makes code writing much faster. The IntelliSense with UFT is a minimal feature.



I like the high level language support provided by CUIT. In my experience, hard core testers prefer VBScript as the efforts for them to learn the language like C# or VB.NET are considerably higher, but those who have little knowledge of coding in object oriented languages, appreciate CUIT more. Support of object oriented programming clearly gives more flexibility although it is more difficult, if not impossible, to write such a code from scratch, which many testers have got accustomed to do, rather than using a recorder.

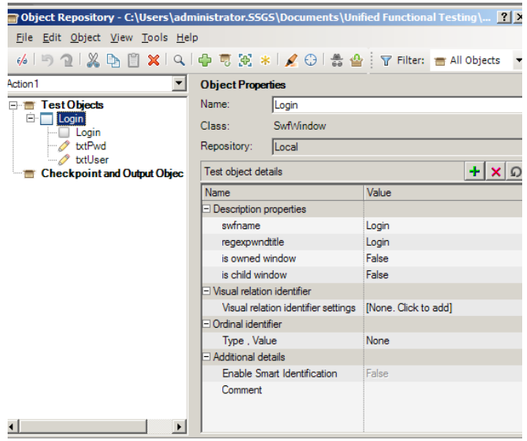
OBJECT REPOSITORY

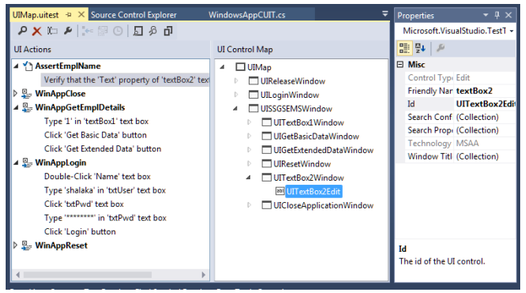
UFT stores all application related objects in Object Repository and modifying it is quite simple. Once the object repository is ready, it can be exported to XML. We can even see the objects getting stored while recording. The object repository can either be local or shared. As the name suggests, shared repository can be shared amongst multiple tests if the tests have properties which can be re-used. Shared repository can also be edited if required. UFT provides a feature called as object spy which helps to list runtime properties of GUI and test objects. It also lists the methods which can be used in implementing the particular object.

Visual Studio stores the application related objects in its own repository called UIMap. The CUIT Editor is provided in Visual Studio 2010 Feature Pack 2 onwards, which edits UI Map. You can select any object and view its properties. You can change some of the search properties or expressions. You can also expand recorded action and view the steps. It is possible to remove the steps; split a bigger method into another but you cannot add a step to an action. You can use CUIT Builder to record all previous steps and your new step and also overwrite the changes in action. These actions are re-usable within the project. For re-using repository across the projects, you can set a reference to the assembly from which you want to use the code. The default UIMap becomes very heavy if we add a number of large methods to it. In order to overcome this, we can add non-default UIMap. If we add another UIMap, we need to take care of calling the methods after we have recorded the actions with CUIT Builder.

Object spy is a great feature provided by UFT. This clearly shows the stability of the tool.

The following two images show the object repository with both the tools:

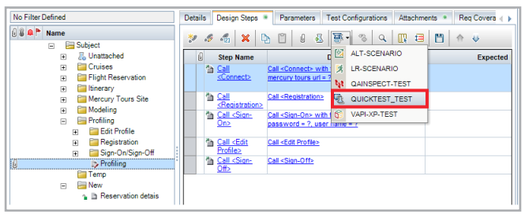




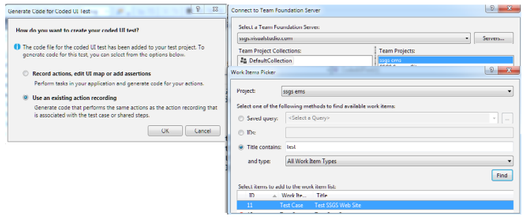
INTEGRATION WITH TEST MANAGEMENT TOOL

Test Management Tool provides a container for storing information about testing to be done, planning of various artifacts and helps in providing reports for quality. The tool can provide different configurations for testing. The tool can be used to execute manual test cases, provide data in terms of parameters and gather information about the result. It can also be used to run automated tests, manage different environments under which tests need to be executed etc. This works as a collaborative platform for communications within teams. The defects can be filed and the traceability from defect/ bug to test case to requirement can be established.

UFT can be connected with HP ALM (Application Lifecycle Management) which is an enhanced version of Quality Centre (QC) with a complete application lifecycle support. HP ALM provides option for converting Test Case to script (provided you have HP ALM and UFT installed). Once the script is generated you can open it with UFT.



CUIT can be connected with Team Foundation Server, a product provided for application lifecycle management. Microsoft Test Manager is a tool for test case management. Using this tool, any test case can be recorded and then converted to its code (CUIT) by using Visual Studio. Visual Studio provides us with automated build and CUIT can be included as a Build Verification Test. We can use distributed testing by using Test Controller and Test Agent.



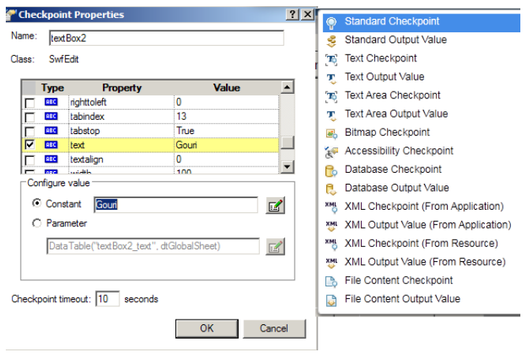
Visual Studio can be integrated seamlessly with Team Foundation Server with the help of Team Explorer.

CHECKPOINTS/ ASSERTIONS

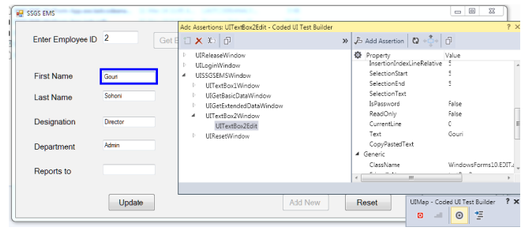
Checkpoint or assertion is used to find out if the Application Under Test (AUT) is providing a particular value or a property. The value can be asserted against expected value to decide whether to pass or fail the test. These assertions or checkpoints can be added while recording the application or it can be added after the recording is completed.

Once a checkpoint is added in UFT, it gets added to the current row in the keyboard view. There are various types of checkpoints like - standard, bitmap, image, text, database and table.

Standard checkpoint can be added in a keyboard view, expert view or using active screen. It checks objects like radio buttons, tables, combo boxes, links etc. The Bitmap checkpoint is used in an application or a web page to check its area. It checks the area pixel by pixel, either partially or completely. The Text checkpoint checks to see if the expected text matches. The Database checkpoint creates a query and the values are verified against the database values stored as expected values.



Normally checkpoints are called as *assertion* in Coded UI Test. Assertions can be added using Coded UI Test Builder. The cross hair for assertion can be followed by the property to be selected for assertion.



In the UI Control Map, we can add control, delete a control or rename a control. Once the assertion is added, we can set the comparison operator and the value to compare. We can create assertion by directly adding a method for it in the partial class UIMap which is in file UIMap.cs.

UFT provides more types of checkpoints when compared to CUIT.

INTEGRATION WITH SOURCE CONTROL

Application Lifecycle Intelligence (ALI) is a technology which is embedded in HP ALM which helps in providing complete ALM traceability. ALI provides support to Source Control and Build Management. Since I was working with the evaluation version, this feature could not be tested; however those having a licensed version of the product can test this feature.

Coded UI Test has complete traceability with ALM features of Team Foundation Server (TFS) as it totally integrates with it. We can convert a recorded test case from Microsoft Test Manager to CUIT, for which the recording is stored in TFS. We also have the source control service of TFS at our disposal.

As I could not try the support with UFT, I prefer CUIT in this scenario, although my choice here is *biased*.

CONCLUSION

Here's a quick recap of a comparison of CUIT with UFT (QTP):

	Coded UI Test (CUIT)	UFT / QTP (Unified Functional Testing)
Record and Playback	Easy to use, full support	Easy to use, full support
Object Repository	A bit challenging to work with UIMap	Very easy to use, object spy is a great feature
Integration with Test Management Tool	Seamlessly integrates with Team Foundation Server	Not so easy to integrate with QC ALM
Checkpoint / Assertions	Easy to use	Easy to use with lots of default checkpoints
Integration with Source Control	Very useful	No practical experience to mention

I hope you enjoyed going through these feature comparisons of both these tools. UFT clearly is the market leader and has been there since a long time. Coded UI Test (CUIT) is being made feature rich with each new version of Visual Studio, thus making it a great competitor tool, that I look forward to! ■

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

Building SharePoint 2013 Web Parts using **AngularJS and KnockoutJS**

This article talks about how to make use of JavaScript libraries to fetch data from SharePoint List and bind that data to some controls using SharePoint Web parts. For this demonstration, we will make use of AngularJS and KnockoutJS JavaScript libraries.

SharePoint versions prior to 2010 have been using the *SharePoint API* (a.k.a. Server-Side Object model) to develop SharePoint web parts. Although the API allowed CRUD operations on SharePoint data, the restriction was that it had to be run on the SharePoint server or a developer would alternatively use SharePoint's web server interface, which had its own limitations from a design perspective.

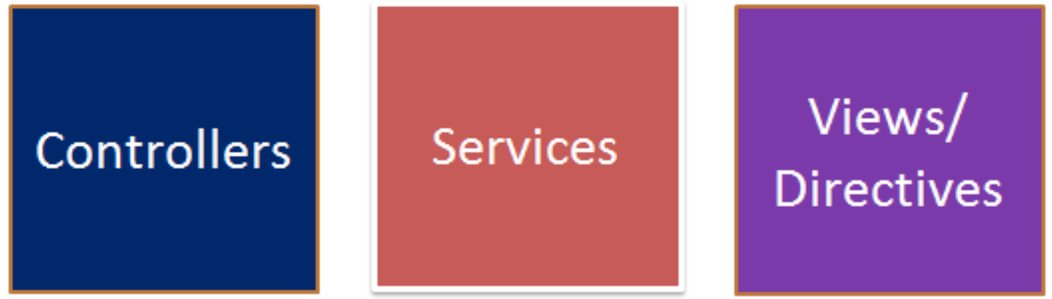
With SharePoint 2010, developers were now able to use *Client Side Object Model* to develop Web parts. The Client Side Object Model is a subset of the SharePoint API which mimics a large portion of the server-side API. The advantage of the client-side API is it can be called from .NET apps, Silverlight or using JavaScript libraries. In SharePoint 2013, the client-side API has been extended with REST/OData endpoints.

A QUICK LOOK INTO ANGULARJS AND KNOCKOUTJS

Before getting started, let's quickly introduce the two JavaScript libraries – Angular JS and Knockout JS.

ANGULARJS

AngularJS is an open source MV* JavaScript framework maintained by Google and community, for building rich, interactive, single page applications (SPA). AngularJS uses HTML as your template language and lets you extend the HTML vocabulary for your application. The * in AngularJS represents MVC, MVVM or the MVP pattern which makes both development and testing of AngularJS applications easier. The main components of AngularJS are shown here –



Controller – Controller is key to AngularJS. A Controller is a JavaScript constructor function that is used to create a scope. A controller is attached to the DOM via *ng-controller* directive. You can use controllers to initiate the state of the *$scope* object and add behaviours to $scope object. $scope – is a glue between a Controller and DOM and just watches the changes to models and also applies changes to the model. One thing to note about Controller is they do not perform DOM manipulations or maintain state. AngularJS uses *Services* to maintain state.

Services – Services are used to perform common tasks on the web applications which are consumed by AngularJS via *Dependency Injection*. Services are registered with a Module and are usually singular to the application.

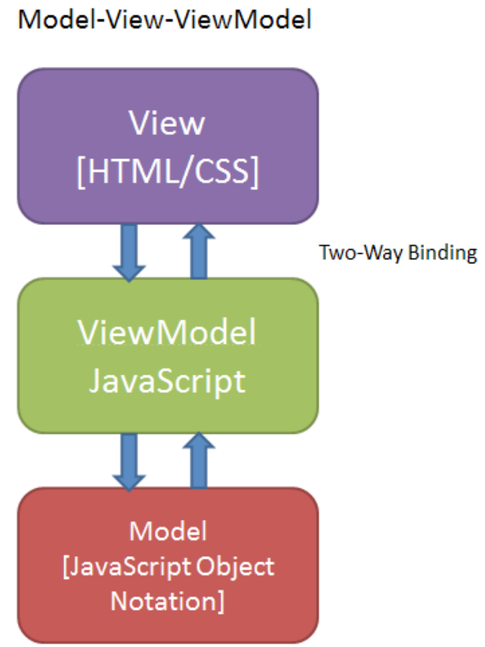
Views – Views are complied DOM of Angular JS. A view is produced using *$compile* with HTML templates and $scope. With clientside implementation, Angular can have integration with the server using REST and $http service.

KNOCKOUTJS

KnockoutJS is a fantastic library when you are looking for a drop-in enhancement that brings in client side data-binding and applies the Model-View-View-Model design pattern to your websites.

Knockout works with a ViewModel that can either be a JS object or a JS function. Either ways, your ViewModel is your data source. Using Knockout, you can bind DOM elements to your model using a declarative syntax. Knockout also provides Templating (for repetitive structures in a web page) and dependency tracking using ko.observableArray. With dependency tracking, if a property is changed, it will automatically notify the UI. The UI reflects these changes and can also change the value to automatically update the source object again.

This was just a small introduction of Angular JS and Knockout



JS libraries. We can get more details on both the libraries here –

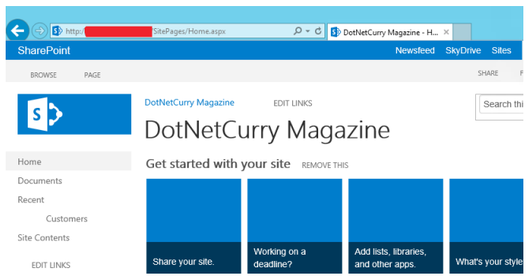
- Angularjs.org
- Knockoutjs.com

You can also learn more about AngularJS in Getting Started with AngularJS in ASP.NET MVC and about Knockout at Getting Started with KnockoutJS in ASP.NET MVC

USING ANGULARJS AND KNOCKOUTJS IN SHAREPOINT

Using these libraries in SharePoint development can make a big difference as we will see shortly. We will use these libraries with SharePoint Web parts using our very favorite IDE – Microsoft Visual Studio. I am using Visual Studio 2013 for these demonstrations.

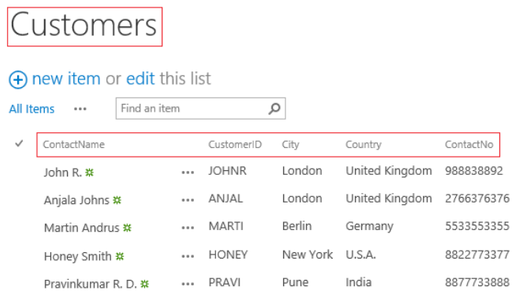
I have already created a SharePoint Site using Team site Template –



We will create a Custom List in our SharePoint site with the name Customers and some columns as described here –

- CustomerID
- ContactName [Rename title column]
- City
- Country
- ContactNo

Add some default data in our customers list. The list should look like the following –



Before we start building the Web Part, we will query SharePoint list data using OData queries. Let's write some queries in our browser as described here –

- http://localhost/sites/dncmag/_api/web/lists/getByTitle('Customers')/items

- http://localhost/sites/dncmag/_api/web/lists/getByTitle('Customers')/items?$select=CustomerID,Title,City,Country,ContactNo

The output of the above query which uses $select operator is as shown here –

As already mentioned earlier, we will be using Visual Studio to build our Web Part using the AngularJS library. Before that, we will need to add the scripts into our /_layouts/15/Scripts folder. Go to Windows Explorer and browse the following path –

```
C:\Program Files\Common Files\microsoft shared\
Web Server Extensions\15\TEMPLATE\LAYOUTS
```

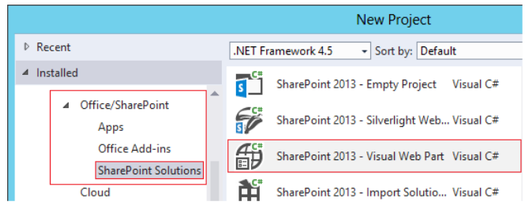
Now create a *Scripts* folder under *Layouts* folder and add the following script files into the same –

- Angular JS.
- Bootstrap JS.

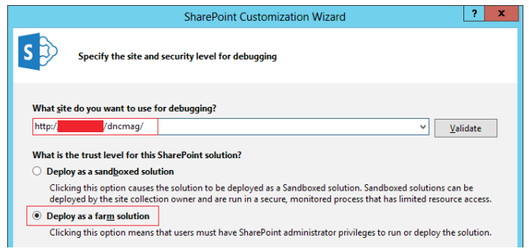
I will also be using Twitter Bootstrap to make my UI look better. To those new to Bootstrap, it is a package that consists of predefined CSS styles, Components and jQuery plugins. It is licensed under Apache 2.0 and is free for commercial use. *For this demo, I have added the Bootstrap CSS into Scripts folder, although ideally you should create a separate CSS folder if you will be dealing with too many CSS files.*

Also add some empty JavaScript files with the names SharePointListModule.js and CustomerController.js in the Scripts folder. We will use these two files when we will be writing the script for fetching the SharePoint List Data using Angular JS.

Now we are ready to build the SharePoint Visual Web Part using AngularJS. Open Visual Studio and create a New Project. Choose Office/SharePoint > SharePoint Solutions and choose SharePoint 2013 – Visual Web Part template as shown here –



Click on the OK button. You will now see a dialog box which will ask you for a site URL and security level for debugging.



Once the Web Part project is created, we will first import the AngularJS, Bootstrap JS, Bootstrap CSS, SharePointListModule.js and CustomerController.js in our User Control as shown here –

```
VisualWebPart1UserControl.ascx
<link href="/_layouts/15/Scripts/bootstrap.min.css" rel="stylesheet"/>
<script src="/_layouts/15/Scripts/angular.min.js"></script>
<script src="/_layouts/15/Scripts/bootstrap.min.js"></script>
<script src="/_layouts/15/Scripts/SharePointListModule.js"></script>
<script src="/_layouts/15/Scripts/CustomerController.js"></script>
```

We will now create an Angular *Module*. You can think of a module as a container which wires the different parts of your application like controllers, services, directives, etc. To create an Angular Module, write the following code in our SharePointListModule.js file –

```
SharePointListModule.js*
var myAngApp = angular.module('SharePointAngApp', []);
```

Add a *Controller* which will fetch the data from our SharePoint List using $http service. We will use the SharePoint OData query and bind the data in our user control. Let's write some code in CustomerController.js file as shown here –

```
var webSiteURL = "<%= SPContext.Current.Web.Url %>";
var myAngApp = angular.module('SharePointAngApp', []);
myAngApp.controller('spCustomerController', function ($scope, $http) {
    $http({
        method: 'GET',
        url: webSiteURL + "/_api/web/lists/getByTitle('Customers')/items?
          $select=CustomerID,Title,City,Country,ContactNo",
        headers: { "Accept": "application/json;odata=verbose" }
    }).success(function (d, s, h, c) {
        $scope.customers = d.d.results;
    });
});
```

In the script we just saw, we have first created an Angular Controller with the name 'spCustomerController'. We have also injected $scope and $http service. The $http service will fetch the list data from the specific columns of SharePoint list. $scope is a glue between Controller and a View. It acts as execution context for Expressions. Angular expressions are code snippets that are usually placed in bindings such as {{ expression }}.

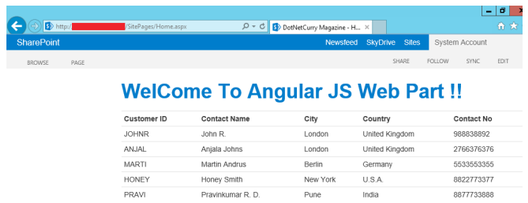
You might have observed that we are creating separate JavaScript files for creating Angular JS Module and Controller. As already mentioned, the AngularJS module acts as a container for Controllers, Services, Filters, and Directives etc. Modules declaratively specify how an application should be bootstrapped. It offers several advantages like –

- The declarative process is easier to understand
- Modules can be reusable
- Modules can be loaded in any order or parallel
- Unit testing and end-to-end testing is easier

We will now build the UI which will display SharePoint list data in our User Control. Let's write the following code after the script –

```
<div ng-app="SharePointAngApp" class="row">
    <div ng-controller="spCustomerController" class="span10">
        <table class="table table-condensed table-hover">
            <tr>
                <th>Customer ID</th>
                <th>Contact Name</th>
                <th>City</th>
                <th>Country</th>
                <th>Contact No</th>
            </tr>
            <tr ng-repeat="customer in customers">
                <td>{{customer.CustomerID}}</td>
                <td>{{customer.Title}}</td>
                <td>{{customer.City}}</td>
                <td>{{customer.Country}}</td>
                <td>{{customer.ContactNo}}</td>
            </tr>
        </table>
    </div>
</div>
```

In above HTML, we are using table and ng-repeat to iterate through the customers. Now deploy the Web Part and add the same in your SharePoint site. You should see the following output –



USING KNOCKOUTJS IN SHAREPOINT

We will repeat the same demonstration that we did earlier with AngularJS, now using KnockoutJS. We will create one more webpart, but this time we will go with KnockoutJS JavaScript library to bind the data in our User Control. Before creating the Web Part, let's add the following class libraries to our Script folder –

- jquery-2.1.0.min.js
- knockout-3.1.0.js

Follow the same steps to create a Web Part using Visual Studio as described in AngularJS Web Part section. Once the Web Part is created, we will now add the references of script files into our User Control as shown here –

VisualWebPart1UserControl.ascx

```
<link href="/_layouts/15/Scripts/bootstrap.min.css" rel="stylesheet" />
<script src="/_layouts/15/Scripts/jquery-2.1.0.min.js"></script>
<script src="/_layouts/15/Scripts/knockout-3.1.0.js"></script>
<script src="/_layouts/15/Scripts/bootstrap.min.js"></script>
```

To fetch the data from SharePoint list [Customers], we will make use of jQuery AJAX. Let's write the following script in our user control –

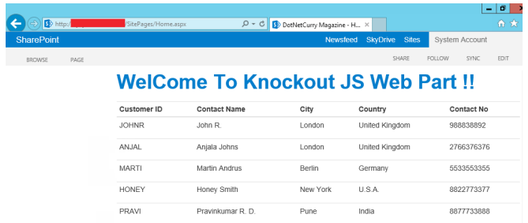
```
<script>
    var webSiteURL = "<%= SPContext.Current.Web.Url %>";
    $(function CustomersModel() {
        var self = this;
        self.Customers = ko.observableArray([]);
        $.ajax({
            method: 'GET',
            url: webSiteURL + "/_api/web/lists/getByTitle('Customers')/items?
              $select=CustomerID,Title,City,Country,ContactNo",
            headers: { "Accept": "application/json;odata=verbose" }
        }).success(function (d) {
            self.Customers = d.d.results;
            ko.applyBindings(self);
        })();
    });
</script>
```

Note: *Replace jQuery success() function with done() for jQuery 1.8 and above.*

The above script, fetches the data from SharePoint List and exposes it as a Knockout observable array. We will now build the UI for our user control in which we will use the KnockoutJS binding capabilities. Let's write some code in our User Control after the script –

```
<h1>WelCome To Knockout JS Web Part !!</h1>
<div class="row">
    <div class="span10">
        <table class="table table-condensed table-hover">
            <tr>
                <th>Customer ID</th>
                <th>Contact Name</th>
                <th>City</th>
                <th>Country</th>
                <th>Contact No</th>
            </tr>
            <tbody data-bind="foreach: Customers">
                <tr>
                    <td>
                        <p data-bind="text: CustomerID"></p>
                    </td>
                    <td>
                        <p data-bind="text: Title"></p>
                    </td>
```

In the above HTML, we are using Knockout JS foreach loop to iterate through the list of customers. Now build the Web Part and deploy it to a SharePoint site. After deployment, add the Web Part on SharePoint site and the output should be similar to the following –



There you go! You now have a number of possibilities open to build different SharePoint Applications like Web Parts or SharePoint Apps using various client-side libraries.

SUMMARY

In this article, we saw how you can use the client side JavaScript libraries like AngularJS and Knockout JS in SharePoint ■

Download the entire source code from our GitHub Repository at bit.ly/dncm12-spwebparts

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

EXTENDING THE JQUERY UI ACCORDION

The jQuery UI accordion allows you to organize related content into distinct sections, with any one section visible at a time. The collapsed sections are revealed by clicking on the corresponding section heading. The jQuery UI accordion is very easy to setup and configure.

Download the source code which contains the jQueryUIAccordion.html in it. This html already contains the library references to the jQuery theme CSS, jQuery library, jQuery UI library and some underlying markup.

Here's a preview of the HTML markup:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Simple jQuery UI Accordion</title>
    <link href="CSS/jquery-ui.min.css" rel="stylesheet"
    />
    <script src="../scripts/jquery-1.11.0.min.js">
    </script>
    <script src="../scripts/jquery-ui.min.js"></script>
    <script type="text/javascript">
        $(function () {
            $("#faq").accordion();
        });
    </script>
</head>

<body>
    <div id="faq">
        <h3>What Does This Book Contain?</h3>
        <p>
            Scores of practical jQuery recipes that you
            can use in your projects.
        </p>
        <h3>Where can I download Sample chapters?</h3>
        <p>You can download sample chapters over
        here.</p>
        <h3>Who is the Book Author?</h3>
        <p>
            Suprotim Agarwal is the author of this Book.
        </p>
    </div>
</body>

</html>
```

The outer container ' ' has an id of 'faq'. The heading of the accordion has a '<h3>' element and the content section has a para '<p>' as a container. The content section can contain any kind of data. In order to make an accordion of this markup, just add a single line of jQuery code:

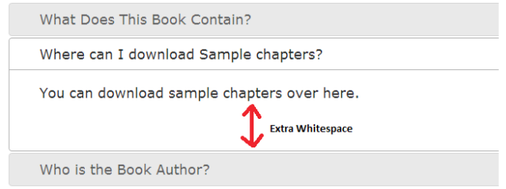
```
<script type="text/javascript">
    $(function () {
        $("#faq").accordion();
    });
</script>
```

and that's it. You have an Accordion control!

You can pass some configuration options to change the way the default accordion behaves. To set configuration options, we will pass an object literal into the accordion as shown below:

```
$(function () {
    $("#faq").accordion({
        "optionname": "value"
    });
});
```

For example, the Accordion sets an equal height for all the Panels by setting the *heightStyle* to auto, which means all panels will be of the same height as the tallest panel. This leads to extra whitespace in those content panels where the content is less.

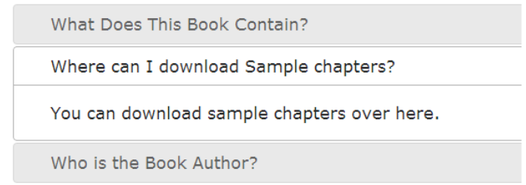


<s3-accordion-whitespace.png>

In order to fix this, set the *heightStyle* to content which means each panel will be only as tall as its content.

```
$("#faq").accordion({
    event: "mouseover",
    collapsible: true,
    active: 2,
    heightStyle: 'content' });
```

View the page in the browser and you will see that the 2nd panel is now only high enough to contain its own content.



EXTENDING THE JQUERY UI ACCORDION

We just saw how to save whitespace by using the *heightStyle* property to enable the panel to be only as tall as its contents.

Alternatively we can also *extend* the existing jQuery UI Accordion widget and provide a *consistent* way for users to enable/disable resize functionality. We will extend the jQuery UI Accordion to provide users with the option to drag and resize the content panel.

Understanding the jQuery UI Widget Factory

The jQuery UI Widget Factory is simply a function or factory method ($.widget) on which all of jQuery UI's widgets are built. It takes the widget name and an object containing widget properties as arguments and creates a jQuery plugin; that provides a consistent API to create and destroy widgets, encapsulate its functionality, allow setting options on initialization and provide state management. In simple words, it allows you to conform to a defined standard, making it easier for new users to start using your plugin.

Note: *All jQuery UI's widgets use the same patterns as defined by the widget factory. If you learn how to use one widget, you'll know how to use all of them.*

Creating a Widget Extension

You can create new widgets with the widget factory by passing the widget name and prototype to $.widget().

```
$.widget("custom.newAccordion", {});
```

This will create a *newAccordion* widget in the custom namespace.

However in our case, we want to *extend* the *existing* Accordion widget. To do so, pass the constructor of the jQuery UI Accordion widget ($.ui.accordion) to use as a parent, as shown here:

```
$.widget("custom.newAccordion", $.ui.accordion, {
});
```

By doing so, we are indicating that the *newAccordion* widget should use jQuery UI's Accordion widget as a parent. In order to make our new widget useful, we will add some methods to its prototype object, which is the final parameter passed to $.widget(). The shell of our widget will look similar to the following:

```
(function ($) {
    $.widget("custom.newAccordion", $.ui.accordion, {
        options: {
        },
        _create: function () {
        },
        destroy: function () {
        },
        disable: function () {
        },
        enable: function () {
        },
    });
})(jQuery);
```

Note: *If you observe, the jQuery UI plugin has been encapsulated in a self-executing anonymous function (function ($) {}). This aliases $ and ensures that jQuery's noConflict() method works as intended.*

Let's add some functionality to our widget. Observe the following code:

```
(function( $ ) {
$.widget( "custom.newAccordion", $.ui.accordion, {
    options: {
        resizable: true
    },
    _create: function () {
        this._super();
        if ( !this.options.resizable ) {
            return;
        }
        this.headers.next().resizable({ handles: "s" })
            .css({
                "margin-bottom": "5px",
                "border-bottom":"1px dashed",
                "overflow": "hidden"
            });
    },
    _destroy: function () {
        this._super();
        if (!this.options.resizable) {
            return;
        }
        this.headers.next()
        .resizable("destroy")
        .css({
            "margin-bottom": "2px",
            "border-bottom": "1px solid",
            "overflow": ""
        });
    },
});
})(jQuery);
```

We start by providing flexible configuration through *options* and initialize them with default values. In this case, we are setting *resizable* to true by default. The _create() method is the widget's constructor. In this method, we are replacing the original _create() method with our new implementation. this._super() ensures that the default setup actions of the accordion widget happens.

Note: *Widget properties which begin with an underscore such as _create, are considered private.*

The next step is to find all content sections of the accordion and apply the resizable() widget.

```
this.headers.next().resizable({ handles: "s" })
```

The "s" handle redirects the resizable widget to only show a *south* handle which means the user can use the cursor at the bottom of each Accordion section to resize it up or down.

We are also using some CSS to set the margin-bottom and border-bottom and overflow properties. Instead of adding CSS properties manually, you can even use a class like this

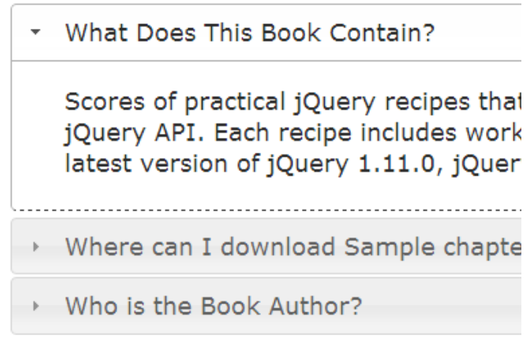
```
this.headers.next().resizable({ handles: "s" })
}).addClass( "mycss" );
```

The *destroy()* method removes the widget functionality and returns the element back to its pre-init state.

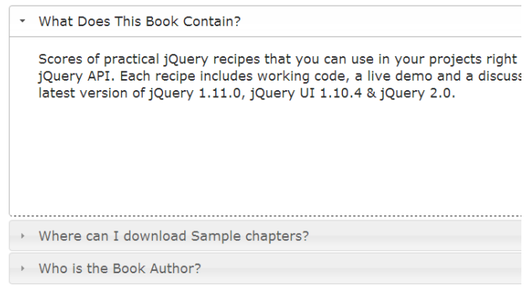
The final step is to call our *newAccordion* widget on the faq div and passing *resizable* to true

```
$(function () {
    $("#faq").newAccordion(
        { resizable: true }
    );
});
```

Run the example and you will see the following Accordion with a dashed bottom border, 5px spacing and a resizable panel:



You can resize the panel by placing the cursor at the bottom of each section. Once you place the cursor on the dotted line, the cursor changes using which you can drag it up and down to change the height of the panel. Here I have increased the panel size by holding the cursor on the dotted line and pulling it downwards:



Further Reading: http://api.jqueryui.com/jQuery.widget

This article was taken from my upcoming ***The Absolutely Awesome jQuery Cookbook*** at www.jquerycookbook.com ■

Download the entire source code from our GitHub Repository at bit.ly/dncm12-jqaccordion

Suprotim Agarwal, ASP.NET Architecture MVP, is an author and the founder of popular .NET websites like dotnetcurry.com, devcurry.com and the DNC .NET Magazine that you are reading. You can follow him on twitter @suprotimagarwal or learn more about his new book www.jquerycookbook.com

POWERSHELL-BACKUPS

Over the last month or two I have been looking into a way to automate different kinds of backups, including RavenDB, SQL Server and files backups such as IIS logs. Originally I was backing up RavenDB using a batch file but decided to change this and instead use Windows PowerShell, a command and scripting language from Microsoft. The goal was to create a suite of PowerShell reusable scripts which can be run on any server to backup RavenDB, and then move the backup files to a centralized location.

RavenDB is a document-oriented database build from the ground up on the .NET platform.

I had no previous knowledge of PowerShell before I embarked on this project. This article focuses on how I went about it, the tools I have come across and hopefully show just how easy it can be to get up and running with PowerShell. I'll cover the basics in this article and leave you with something hopefully you can look at, learn from and create your own set of reusable PowerShell scripts.

We will discuss some theory behind the scripts shown in this article using which you should be able to apply what you'll learn to other areas just as easily.

WHY POWERSHELL?

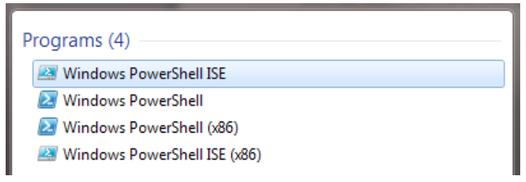
Why would you want to learn PowerShell you may ask? Well it's always good to learn a new skill. PowerShell is more than just a scripting language, and you can do a number of things quickly which ultimately improves the productivity of administrators and power users alike. For eg: using Powershell script, you can check all the certificate's expiry dates on servers or how much disk space is available on your servers or simplify the management of data stores using its provider model, and so on.

Powershell works with Windows commands and applications, and it has been designed to leverage what you already know, making it easier for you to learn it.

Editorial Note: *For those interested, you may also want to read Using PowerShell in SharePoint 2013 for a good introduction to Powershell and how to use Powershell in SharePoint 2013.*

POWERSHELL 101

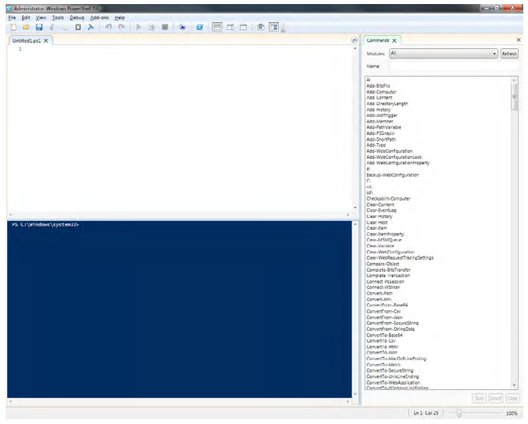
Make sure PowerShell is installed on your machine. Go to the PowerShell page at www.microsoft.com/powershell and click on the download link to install it. To get started with Powershell, there are 2 ways to write your scripts. On my windows machine, I click Start and then type PowerShell, in the list I can see the following:-



- *Windows Powershell* is a PowerShell command prompt, think of this as a command prompt on steroids.

- *Windows PowerShell ISE* is the PowerShell Integrated Scripting Environment. This environment will allow you to debug scripts that you write. This is the option I tend to start off with, so I can get Intellisense and use the documentation help, when writing my scripts. You should choose this option, especially if you're a beginner like I was. I always run the ISE (integrated Scripting Environment) as an Administrator and you can do this by right clicking on the menu option above and select run as Administrator.

From here onwards, I will be using the Windows PowerShell ISE and running as an Administrator. You can tell you're running as an Administrator if you look at the top left hand side of the bar as shown here. It displays "Administrator: Windows PowerShell ISE".



On the right hand side, we can see a list of available commands we can use. On the left hand side at the top, we have the script pane where we will write our script and debug them and the bottom pane is a PowerShell window where we can see our output, as well as execute commands if we want to.

I tend to close the right hand side as it gives me a little more screen estate to work with. Using the PowerShell ISE, we can set breakpoints and step through our scripts, just like you would in Visual Studio when debugging some C# code.

LET'S GET SCRIPTING

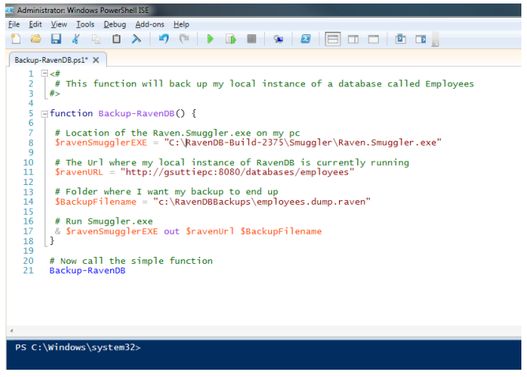
Ok let's dive right in and write our first script. This script will show how to backup a local instance of a RavenDB test database. No real knowledge of RavenDB is required for this, and the reason for this is that to backup RavenDB, we run a program called *Smuggler* – which is an exe that we run to do the job.

From the RavenDB help, we can Export our data using this command:-

```
Raven.Smuggler out http://localhost:8080 NorthWind.
dump.raven
```

Basically what we have done here is run Raven.Smuggler.exe using the *out* parameter, passing it the url for our RavenDB instance and also supplying a filename to export to. This command will export all indexes, documents and attachments from the local RavenDB instance to a file named NorthWind.dump.raven.

We will create a function just like we would in C# . Our script will initially look like the following:



We have a very simple function declaration with a comment at the top informing us of what the function will do. This simple function will just backup the Employees database I have running on my local instance of RavenDB on my PC. To execute the function, we call it on line 21 as Backup-RavenbDB.

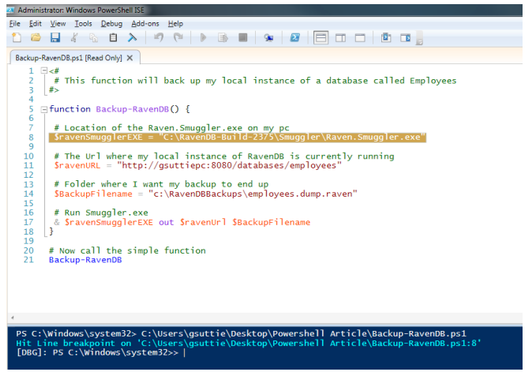
We can add breakpoints and debug this function in the same manner as you would in Visual Studio - highlight a line, press F9 to add a break point and then to call and step through the function, here is what to do:-

- Highlight the entire function (lines 5-18) and then click on the icon to the right of the green play button (green icon with white background) – this is the run selection button.

- Let's add a break point to line 8. Put the cursor on line 8 and hit F9, this will add our breakpoint and the entire line will be highlighted in brown.

- Now to debug our function, click on the function call on line 21 and then press F5 (Run/Continue). You'll now see line 8 with the breakpoint being hit and we can now step through the code by pressing F10. (You can also use the Debug menu if you wish)

Below we can see the script debugging in action. Line 8 has our breakpoint and the debugging of our script has begun:-



Points of Interest

& is the way to run a command (like execute an exe), so this is the same as writing the following:-

```
powershell.exe $ravenSmugglerEXE out $ravenUrl
$BackupFilename
```

Parameters in powershell start with the dollar sign $, so we are storing the URL to Raven in the variable $ravenURL

FILE BACKUPS

Another example of using PowerShell scripts would be backing up files in a folder. Let's assume we are running IIS and our websites is logging to a specific folder, which we want to back up. To accomplish this is very easy using PowerShell, so let's take a look at how we can go about this.

The complete function is listed below:-

```
function Backup-IISLogs() {

  Param(
    # Source location folder for logs we wish to backup
    [parameter(Mandatory = $true)]
    [ValidateNotNullOrEmpty()]
    [string]$IISLogsDir
  )

  # This function will backup IIS log Files

  # Import the Powershell Community Extensions add-on
for (using Write-Zip cmdLet)
  # can be downloaded from here:- http://pscx.codeplex.
com/
  Import-Module PSCX

  # Name our zip file
  $zipFileName = "IIS_LOGS_$(Get-Date -f yyyy-MM-dd)" +
'.zip'

  # Email settings
  $sendEmailSMPTServer = 'my.mail.server.com'
  $sendEmailFrom = 'logbackups@gregorsuttie.com'
  $sendEmailTo = 'gregor@gregosuttie.com'
  $subjectSuccess = 'IIS logs backed up successfully'
  $bodySuccess = 'The IIS logs were backed up
successfully'
  $subjectFailure = 'IIS logs failed to backup'
  $bodyFailure = 'The IIS logs failed to backup'

  try
  {
    # Sets the current working location to the specified
location
    Set-Location $IISLogsDir

# Loop through all the files (not folders) in the folder
and add them to a zip file
Invoke-Command -ScriptBlock { Get-ChildItem
$IISLogsDir -Recurse -File | where-object { -not ($_.
psiscontainer)} |
      Write-Zip -OutputPath $zipFileName
    } -ArgumentList $zipFileName

    # Check for errors
    if ($?)
    {
      # Send an email with success message
Send-Mail -smtpServer $sendEmailSMPTServer
-from $sendEmailFrom -to $sendEmailTo -subject
$subjectSuccess -body $bodySuccess
    }
    else
    {
      # Send an email with failure message
Send-Mail -smtpServer $sendEmailSMPTServer
-from $sendEmailFrom -to $sendEmailTo -subject
$subjectFailure -body $bodyFailure
    }
  }

  # Catch exceptions
  catch {
    Write-Host "System.Exception on:- $(Get-date) -
$($Error[0].Exception.Message)"
  }
  finally
  {
    Write-Host "Backup-IISLogs finished
at:- $(Get-date)"
  }
}

#region Send-Mail
function Send-Mail{
  param($smtpServer,$from,$to,$subject,$body)

  $smtp = new-object system.net.mail.
smtpClient($SmtpServer)
  $mail = new-object System.Net.Mail.MailMessage
  $mail.from = $from
  $mail.to.add($to)
  $mail.subject = $subject
  $mail.body = $body
  $smtp.send($mail)
}
#endregion
```

```
# Test our function using  a supplied directory
$sourceDir = 'C:\inetpub\logs\logfiles'

Backup-IISLogs $sourceDir
```

In the script we just saw, we start off with one parameter called *$IISLogsDir*. We have marked the parameter as mandatory and we have validation to make sure the parameter cannot be Null or Empty by using `ValidateNotNullOrEmpty`.

```
Param(
    # Source location folder for logs we wish to backup
    [parameter(Mandatory = $true)]
    [ValidateNotNullOrEmpty()]
    [string]$IISLogsDir
  )
```

The next thing we do in the script is to import a module – this is a pre-written PowerShell module written by the PowerShell Community and has a lot of great content.

```
Import-Module PSCX
```

To read more on the PowerShell Community Extensions, head over to http://pscx.codeplex.com. We are using the *Write-Zip* commandlet which yes, you guessed it right, creates a zip file.

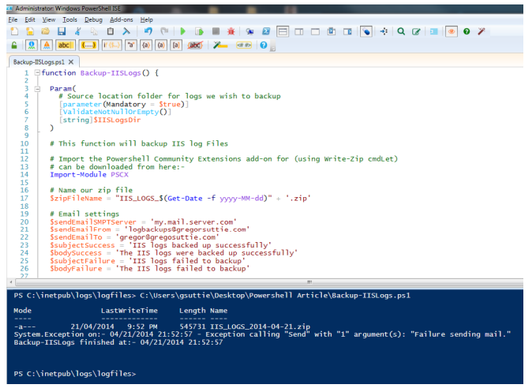
Next we define some variables and then begin the real work. Notice we place the next part of code within a try-catch-finally block just like C# has.

```
try
  {

    ...
  }
  # Catch exceptions
  catch {
    Write-Host "System.Exception on:- $(Get-date) -
$($Error[0].Exception.Message)"
  }
  finally
  {
    Write-Host "Backup-IISLogs finished
at:- $(Get-date)"
  }
}
```

We invoke a command which isn't strictly necessary but will still work and then check for any errors with the last line of code that executed. Lastly we send an email with a success or failure message. The script is basic but enough to give you something to look at and to *whet your appetite*.

```
function Send-Mail{
  param($smtpServer,$from,$to,$subject,$body)

...
}
```

The screenshot here shows the output to the screen after I have run the function we created:-



We have run the script and we can see an error occurred due to the mail server setting being incorrect – change this to a real SMTP server and the script will run and backup our folder as a zip file, and then email the success message.

COPYING FILES

Once we have our backups created, we may need to copy these to a backup server. To begin with, I looked into using the Background Intelligent Transfer Service also known as *BITS* http://en.wikipedia.org/wiki/Background_Intelligent_Transfer_Service. The reason for looking into this was to have the ability to copy files from SQL Server to a centralized backup server and if the copying of the backup file was interrupted for any reason, then file transfer would resume, surviving a server reboot for example.

BITS is a windows service which runs on XP and later operating systems, so I had to have the service started initially and if everything worked well without interruptions, files would be transferred using a job. If the server was rebooted during transfer, they would resume after the server restarted. Great!

BITS worked well but I did run into some issues with permissions as a result of trying to copy files inside a PowerShell session. So I looked for a different option. A colleague suggested using *Robocopy* http://en.wikipedia.org/wiki/Robocopy instead and having seen many an example of PowerShell scripts refer to this tool, I looked into it a bit further.

Robocopy actually comes with Windows Vista and later operating systems, something I did not know and it turns out, is pretty awesome. If you need to transfer files, I recommend you look into this further. This tool also has the ability to copy files with a resume option similar to BITS and can copy files and folder structures with permissions applied to them. If you haven't checked out Robocopy, then go to your command prompt and simply type:-

```
Robocopy /?
```

Copying files using RoboCopy it's as simple as:-

```
RoboCopy c:\inetpub\logs\logFiles c:\backups *.log /Z
```

which is basically RoboCopy <Source> <Destination> <filetypes> <options>

It's very simple to use this tool and there are a number of good reference websites out there covering how to use it. And that pretty much sums up a good use case of PowerShell if you are getting started with it!

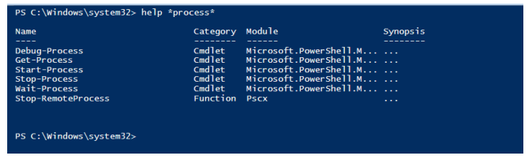
HELP WITH POWERSHELL

If you're new to PowerShell like I was until recently, then the best way to learn PowerShell is actually *not* to just use Google – instead PowerShell has inbuilt help which is exceptionally useful. The best way to learn is to investigate further, so let's have a look at how the help works and how to go about using it. But just before that, a quick bit of information which will make a lot more sense.

PowerShell uses something called *commandlet's* and there are a huge number of them already written. As an example, on the PowerShell prompt type:-

```
Get-Help *process*
```

This will list all of the commandlet's which refer to processes. Commandlet's are usually very well named and as you can see from the list below, you can guess what most of them do:-



We can then get help on each commandlet in the following manner -

Whilst using the in-built help, we also have some very helpful options, at the bottom of the help screen. It basically says the following:-

```
Remarks
   To see examples, type "get-help Start-Process
-examples"
   For more information, type: "get-help Start-Process
-detailed"
   For Technical Information, type: "get-help Start-
Process -full"
   For Online help, type: "get-help Start-Process
-online"
```

This information is available for every commandlet and even if you create your own, you can create the exact same level of help and support, very easily indeed.
This blog entry will show you how to go about this:-

http://blogs.technet.com/b/heyscriptingguy/archive/2010/01/07/hey-scripting-guy-january-7-2010.aspx

WHERE CAN I GET SOME MORE HELP?

PowerShell has a huge number of good resources, some of which I have listed below. Just remember some of the articles cover different versions of PowerShell. I used PowerShell 3 in this article, PowerShell 4 is out and PowerShell 5 is due to be released very soon.

In my honest opinion, try to avoid using Google to search for PowerShell scripts, as you will learn far more experimenting within PowerShell itself, using the help, which is excellent. Here are some resources I referred to.

http://powershell.org – Resources, Q& A and more
http://poshcode.org – PowerShell Code Repository
http://blogs.technet.com/b/heyscriptingguy/ - Scripting Guy Blog

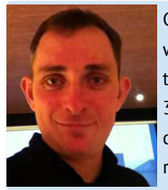
SUMMARY

Due to the nature of the Noun-Verb naming of commandlets in PowerShell and once you've tried a couple of commandlets, you will soon see how easy it is to do things with PowerShell.

If you have any old batch files running out in the wild, try writing a PowerShell script which you can replace the batch file with, and you'll learn just how easy PowerShell is to pick up. The help within PowerShell is superb and very easy to use, it gives you examples on every commandlet as well as full documentation on all of the switches and parameters. There are a number of great resources out there for learning more, and last but not least, learning PowerShell is fun and quite powerful. You can achieve quite a lot with very few lines of code.

PowerShell is fun, it's always adding more and more, you can do a lot quickly and it's worth learning. Oh and it's not something that will go away anytime soon. As far as I know the Azure Portal runs using PowerShell and you can download the scripts to create Virtual Machines and lots more, so go check out PowerShell and I think you'll be pleasantly surprised ■

Download the entire source code from our GitHub Repository at bit.ly/dncm12-powershell



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

WHAT'S NEW IN VISUAL STUDIO 2013 UPDATE 2 RC FOR WEB DEVELOPERS

Visual Studio is an enormous product. With every release and update, it includes a host of improvements, tools, new features and functionality; which ultimately increases the productivity of developers. Every update is aligned with the latest development trends in the market and with Microsoft's premier technologies and languages.

Since the launch of Visual Studio 2013 in October 2013, Microsoft has released two updates referred to as Visual Studio 2013 Update 1 and 2 (2013.1 and 2013.2) which adds functionality to VS 2013. The updates are cumulative, which means installing the newest package will make sure you get benefits of all previous updates.

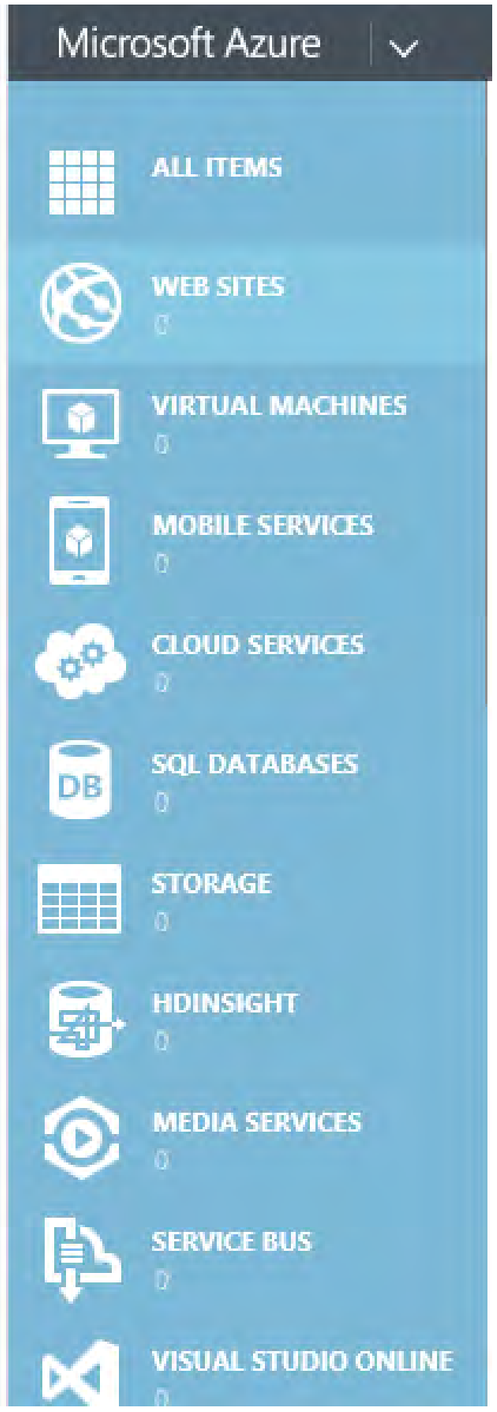
Microsoft released Team Foundation Server 2013 Update 2 RTM and Visual Studio 2013 Update 2 Release Candidate (RC) on April 2, 2014 in its Build Conference. Amongst many newly released features in Visual Studio 2013 Update 2, the highlights were the new capabilities in Windows & Phone 8.1 with the support to integrate apps with Cortana (WP new personal assistant) in VS, support for Universal Projects which allows you to build apps that can target multiple devices in the Windows ecosystem, TypeScript 1.0 becoming a fully supported language in VS, Native Code Compliation, Azure integration with Visual Studio, some cool tooling support as well new features for Web Developers and much more.

The Visual Studio 2013 Update 2 RC can be downloaded from here.

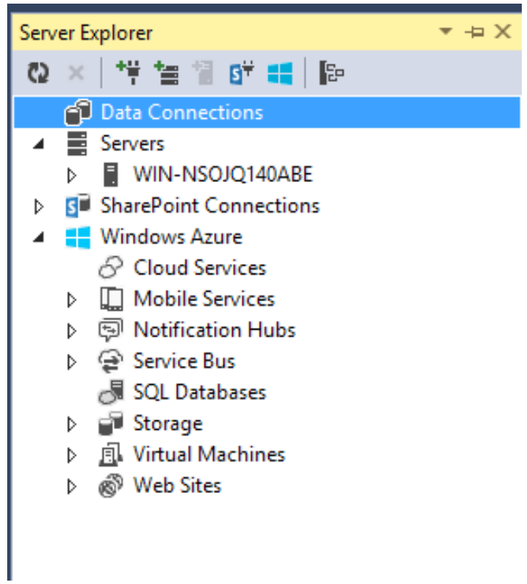
In this article, we will discuss some of the new features targeted towards Web Developers. We will also see how Azure devs can now manage and deploy their apps from Visual Studio.

WHAT IS NEW IN WINDOWS AZURE SDK 2.3?

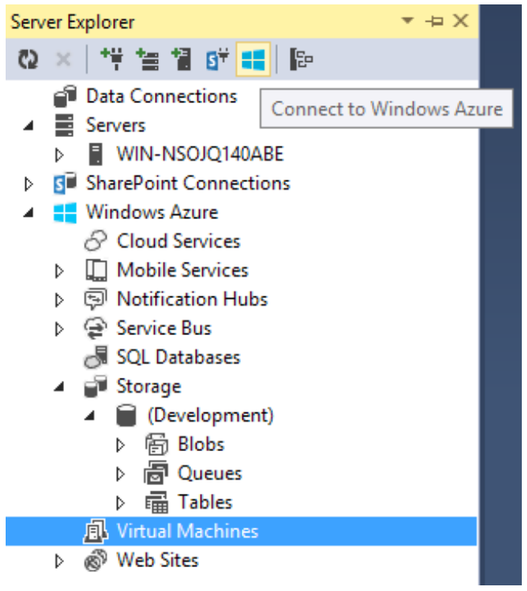
If you are a cloud based application developer, you must be having an Azure subscription for using various cloud features like Cloud Services, Mobile Services, Service Bus, SQL Database, Storage etc. If not, get one! To use these features, you need to make use of the Windows Azure portal as shown here:



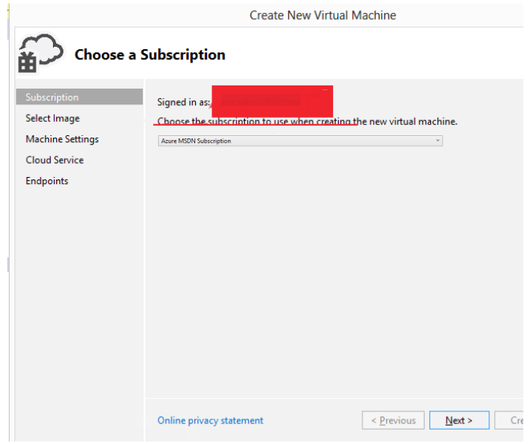
Visual Studio 2013 Update 2 now comes with the newly designed 'Server Explorer' to manage and interact with Windows Azure:



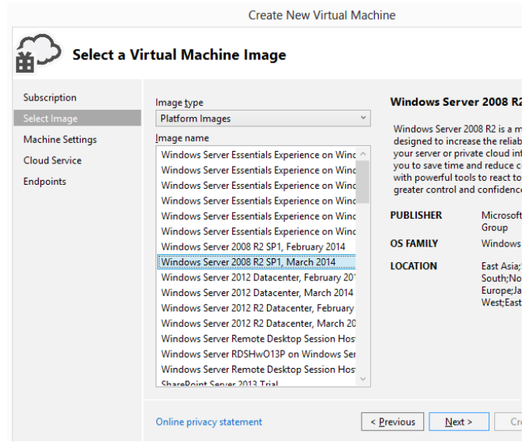
You can now create Virtual Machines directly from Server Explorer by right clicking *Virtual Machines* > *Create Virtual Machine* option.



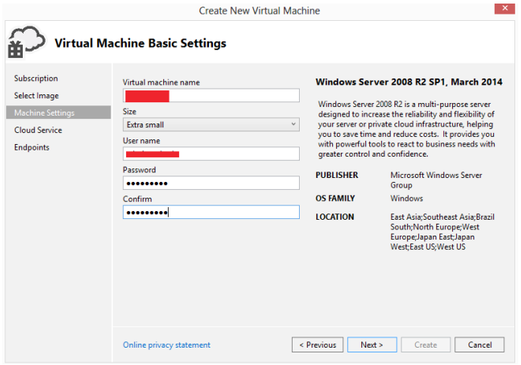
In the Create New Virtual Machine wizard, you can choose the subscription to be used when the VM is created. The subscription dropdown contains all subscriptions you have access to, with your signed in credentials. If you want to sign in with a different set of credentials, just select <Manage...> from the subscription dropdown to do so.



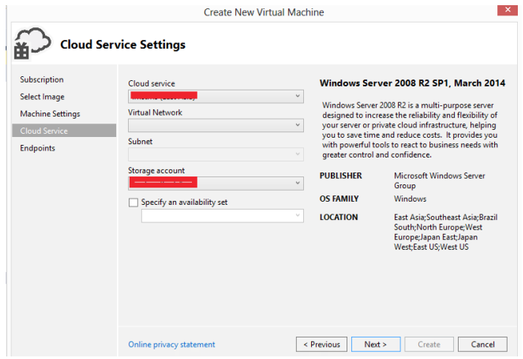
By clicking on Next, the wizard will fetch a list of Virtual Machine Image type to create the VM. You can choose from platform images, MSDN images or your own set of custom images as per your requirement:



On clicking the Next button, we get another window where we can supply the Virtual Machine Basic settings for the VM such as the name, size and the admin username, password:

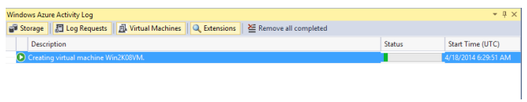


Click Next to configure Cloud Service Settings which essentially means to choose how the VM is exposed to the internet.

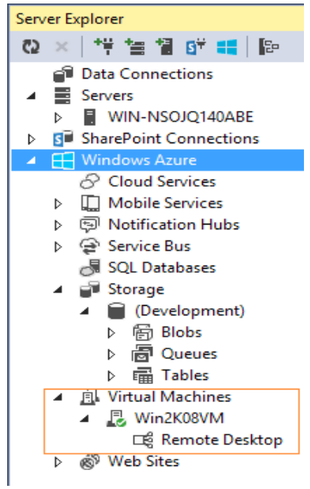


The next step of the wizard is where we can configure EndPoints for the VM. The Remote Desktop and PowerShell endpoints are created by default to run remote PowerShell commands. You can also add additional endpoints if you are running any other services on the VM. After configuring these EndPoints, click on *Create* to create the Virtual Machine.

Once the VM is created, Visual Studio will show the Virtual Machine details in the server explorer as below:

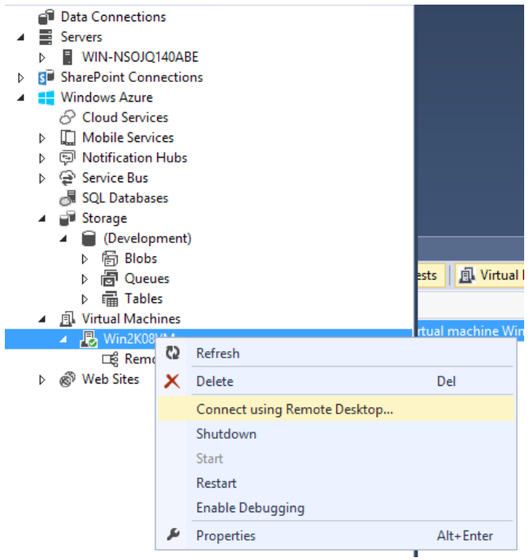


Visual Studio displays the status of VM creation as seen here:

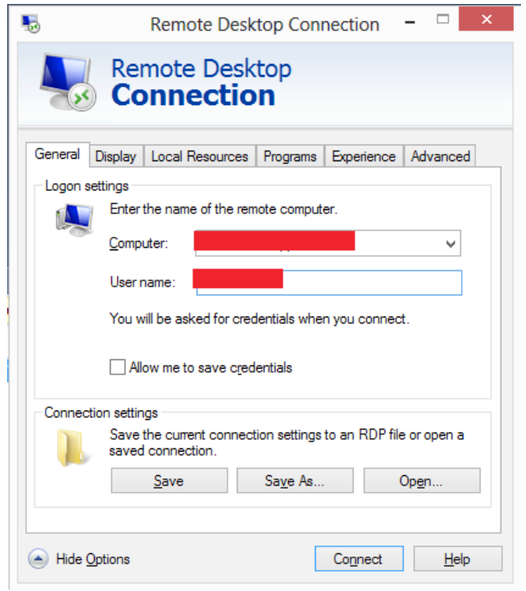


If you visit the portal again, you can see the VM details in it as shown here:

We can also connect to the VM using Visual Studio 2013 Server Explorer. Right click on the VM name and select the option 'Connect using Remote Desktop':



Enter the Computer name and user name in the Remote Desktop Connection window and click on Connect:



And that's it. I am sure as a Windows Azure Developer, you are as excited as I am to be able to interact with Azure in Visual Studio 2013. We can make use of this virtual machine for hosting our services, web sites etc. To manage the VM for installing software and services, please refer to the article Using Azure Virtual Machines for Hosting WCF Services and Communicating with On Premises Applications

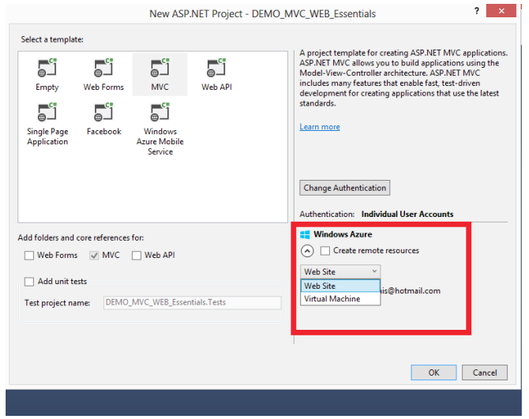
Likewise, we can create a SQL Server Database, Mobile Service etc. using the Server Explorer of Visual Studio 2013 Update 2.

SOME NEW FEATURES FOR WEB APPLICATION DEVELOPERS IN VISUAL STUDIO 2013 UPDATE 2 RC

Visual Studio 2013 Update 2 RC now contains features like creating Azure web sites when creating new ASP.NET applications, a SASS and JSON editor, new project templates etc. We will explore some of these new features in this article.

WINDOWS AZURE OPTION

Open Visual Studio 2013 and create a new ASP.NET Web Application. You will see a new Windows Azure functionality in the ASP.NET Project Dialog options as shown here:



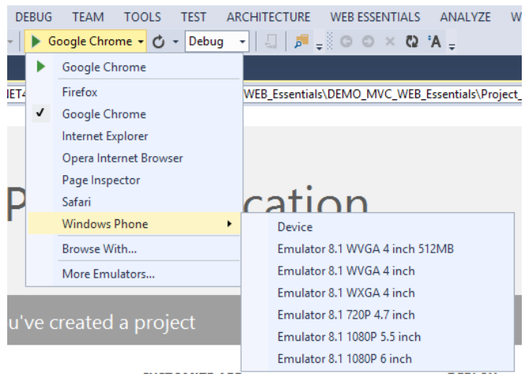
This option enables creating a Web Application on Windows Azure as a 'Web Site' or can be created on the 'Virtual Machine.

For the time being, do not choose any of the Windows Azure option described above.

WINDOWS PHONE BROWSER SUPPORT

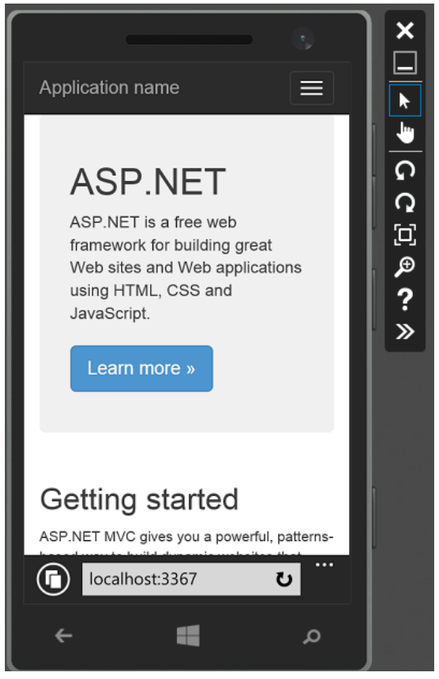
In the screenshot above, create a MVC application and click OK. The application gets created.

We can test this MVC application in some browsers as seen here:



But what's new is that apart from the various available browsers, we can now view the page on device browsers as well (naturally Windows Phone by-default). This is simply great!

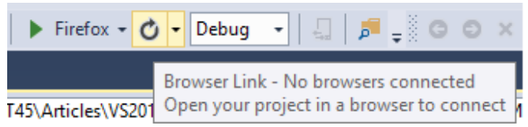
You can see the Windows Phone option, since the Windows Phone 8.1 development tools are included with Visual Studio 2013 (Update 2 or later). The Windows Phone 8.1 Emulators package adds six emulator images using which developers can test how their apps will look on Windows Phone 8.1 devices. Once we select the browser emulator and view the page on Windows Phone, it will get displayed as shown here:



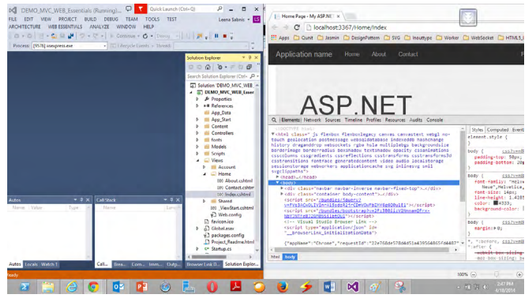
We can also test our applications on other device emulators as well. Observe that there is an option for *More Emulators*, clicking on which will take us to this link where various emulators can be downloaded.

LINKING BROWSER WITH CURRENT PROJECT

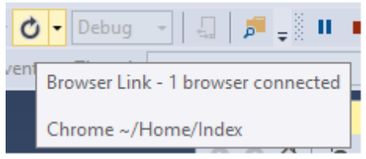
The browser link feature was introduced in Visual Studio 2013. Visual Studio 2013 Update 2 RC now supports HTTPS connections too and even lists these connections in the Browser Link dashboard. Let's go through this cool feature in case you have never tried it out. The browser link feature provides an option of linking the browser with the current project using which we can change the page elements, CSS etc. using the browser developer tools and synchronize it with Visual Studio 2013.



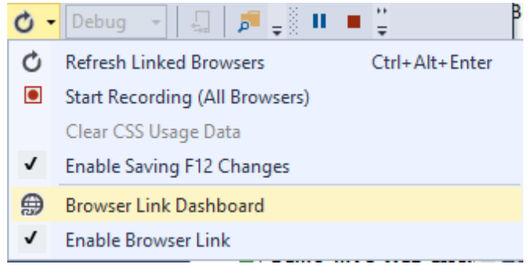
The Linked Browser shows up with a little Refresh icon image in your toolbar, next to debug drop down. Run the application and view the page in Chrome. Make sure that Visual Studio and Chrome are displayed in side-by-side mode and open the Chrome developer tools as shown here:



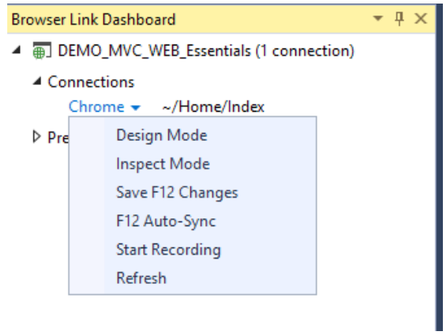
The browser link button displays a message saying *1 browser is connected*:



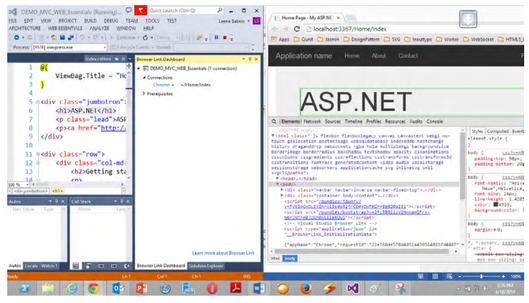
From the browser link button, select the 'Browser Link Dashboard' option as shown here:



This provides the following options:

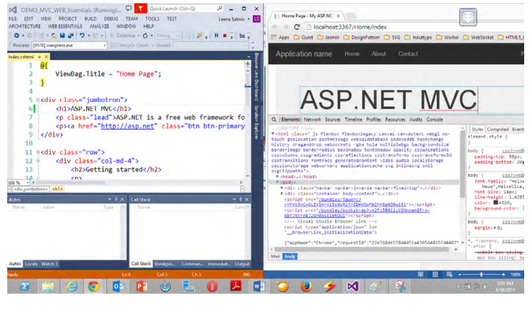


From the above options, select 'Inspect Mode' and move your mouse cursor to the ASP.NET Title of the Page. In Visual Studio, the corresponding view will be opened along with the selected markup synchronized with the selection of the section in Chrome, as shown here:



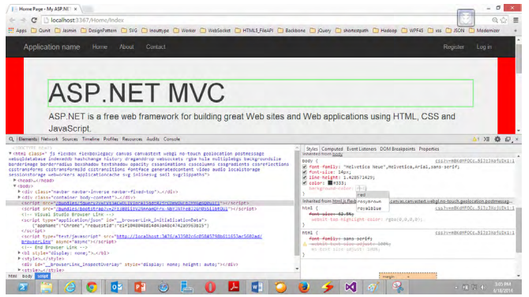
The above image shows that when the 'ASP.NET' Title section is selected in Chrome, the 'Index.cshtml' is opened with <h1>ASP.NET</h1> marked.

Select 'Design Mode' from browser link dashboard and change the Title in Chrome. Visual Studio will show the changes made simultaneously:



After refreshing the page, the changes will be saved in Visual Studio.

Likewise in Chrome, select 'Styles' as shown here and change the 'background-color' to some other color. The page will show the corresponding changes:

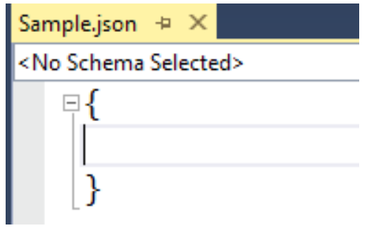


That's it. This synchronization provides more manageability for the Views, for their look and feel across browsers.

INTELLIGENT JSON EDITOR

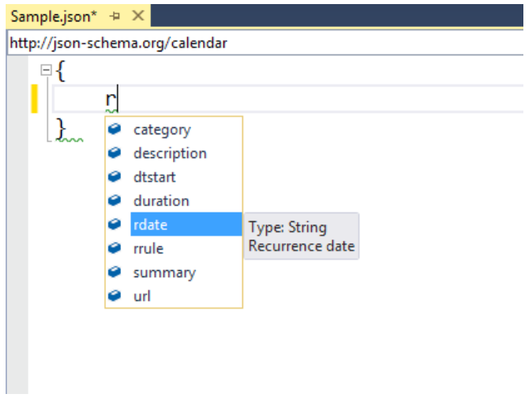
Most web applications (also device apps) communicate with external services (REST Services/WEB APIs) using the JSON data format. In Visual Studio 2013 Update 2 RC, a new JSON Project item is provided. This JSON editor supports JSON syntax validation, colorization, brace completion etc. Intellisense now supports JSON Schema v3 and v4.

In the MVC project we just created using Visual Studio, add a new JSON file. This file will show a drop-down with text as 'No Schema Selected' as you can see here:



Expand the drop-down and some standard schemas will be displayed:

Select the Calender schema (json-schema.org/calendar) and design the schema. The schema details will be as shown here:



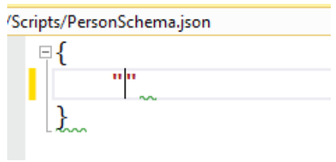
Creating Custom JSON Schema using JSON Editor

In the Scripts folder of MVC project, add a new JSON file and name it as 'PersonSchema.json' and add the person details in it like PersonId, Name, Address etc.:

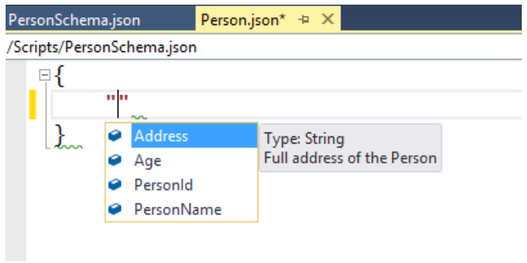
```
{
    "type":"object",
    "properties":{
      "PersonId" :{
        "description": "Person Id",
        "type":"integer"
      },
    "PersonName" :{
        "description":"Full name of the Person",
        "type":"string"
      },
    "Address":{
        "description":"Full address of the Person",
        "type":"string"
      },
    "Age":{
        "description":"Full address of the Person",
        "type":"integer",
        "minimum":18
      }
    },
    "required":["PersonId","PersonName","Address","Age"]
}
```

The above JSON schema defines the type for the Person details as 'object'; the properties section defines *properties* for the Person along with the description and type.

Add a new JSON file in the project under the Scripts folder, name it as 'Person.json'. In this file, refer the PersonSchema.json in it:



Now when we define the details of the Person in the Person.js, we will get an intellisense as follows:

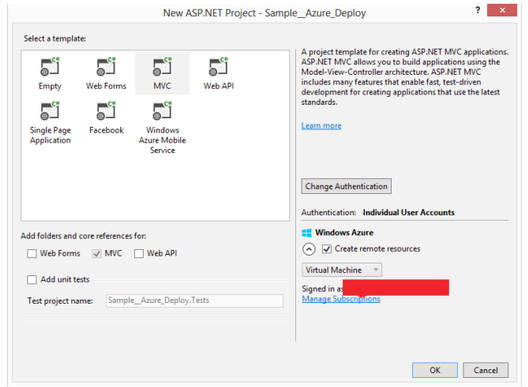


Pretty cool!

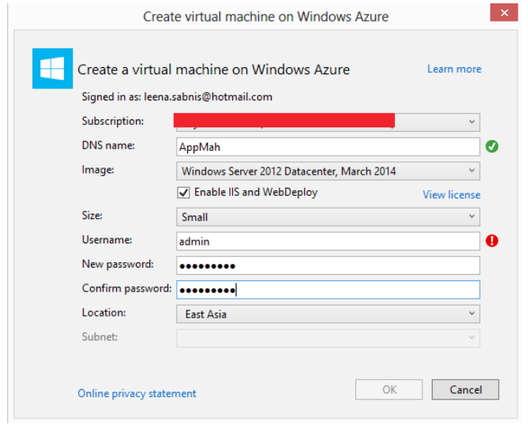
DEPLOY YOUR WEB SITE ON THE WINDOWS AZURE VIRTUAL MACHINE

Windows Azure SDK 2.3, with Visual Studio 2013 Update 2 RC, allows us to publish our Web Site directly on the virtual machine. This allows us to debug the web site remotely with some simple steps to follow.

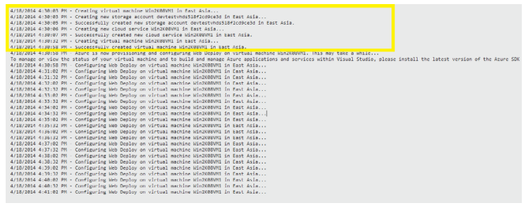
Step 1: Open Visual Studio 2013 and create a new ASP.NET Web Application. While creating this web application select 'Create Remote Resources' checkbox and select Virtual Machine from the Drop-Down as shown here:



Step 2: On clicking 'OK' we will be asked to create the Virtual Machine where the Application will be hosted. We need to set the Virtual Machine Details:

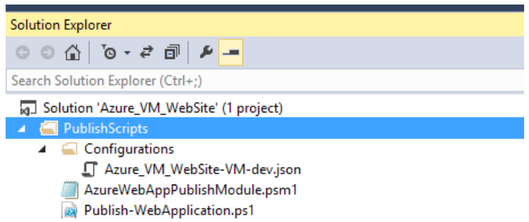


This will create a Virtual Machine, the details of the steps followed will be shown in the output window:



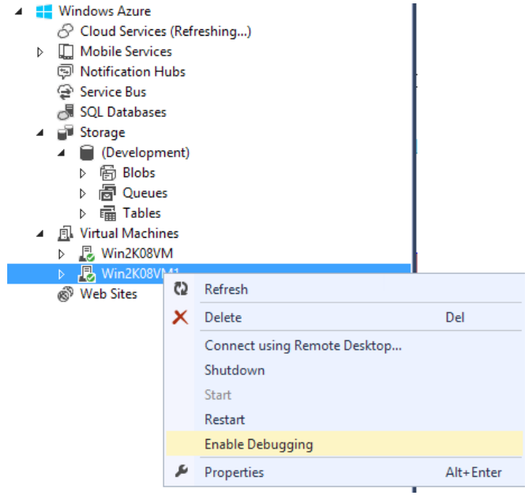
In the Server Explorer, we can see the new Virtual Machine created.

Step 3: In the Solution Explorer we can view the files used for publishing the Web Site:

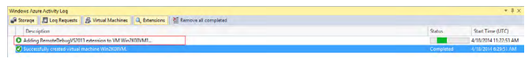


The Publish-WebApplication.ps1 file is the PowerShell file. This is used to create some Windows Azure resources like SQL Database, Storage etc. The 'Azure_VM_WebSite-VM-dev.json' file contains configurations for the web site, like virtual machine administrator, size of VM, endpoints etc.

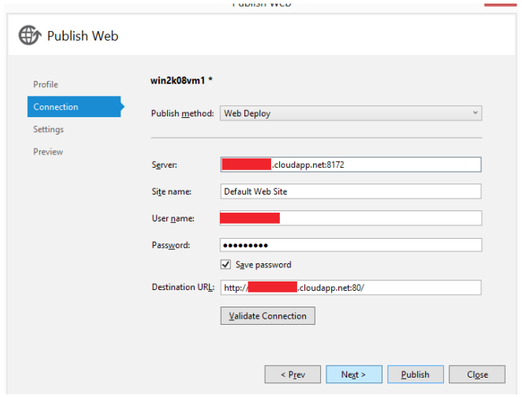
The remote debugging on the VM can be enabled as shown here:



This option will provide the following information specifying status of enabling remote debugging on the VM:



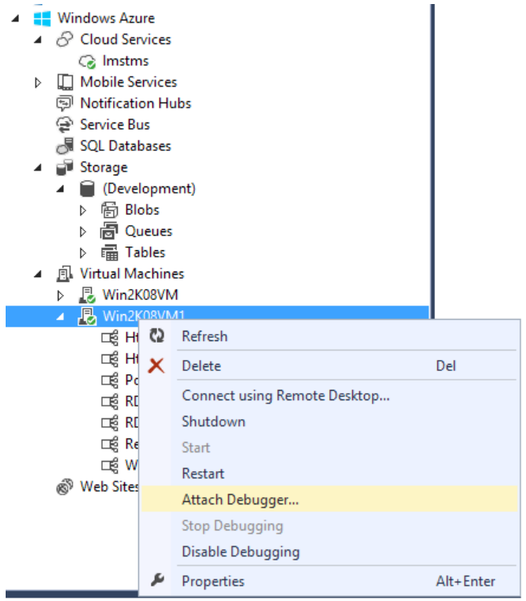
Step 4: Publish the Web Site on the VM. The following window gets displayed with the information as shown here:



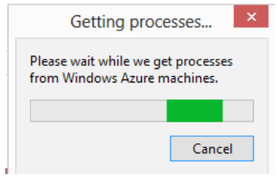
Click on 'Next'.

Step 5: On the Publish Web window, select Configuration as 'Debug' and click on Next and publish. The site will be published.

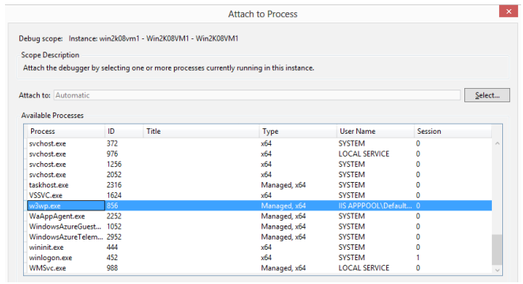
Step 6: Right-click on the VM in the Server Explorer and select 'Attach Debugger' as shown here:



This step will fetch the processes from the Windows Azure machine:



Select w3wp.exe on the VM as shown below:



Put a BreakPoint in the code, (load event of the page or the Index method of the Home controller), and view the page in the browser with the remote address as 'App'.cloudApp.net. The code hits the breakpoint:

```
0 references
public class HomeController : Controller
{
    0 references
    public ActionResult Index()
    {
        ViewBag.Message = "I am the Web Site on the Azure";
    }
```


This is an awesome feature for the devs who have experienced remote debugging in the past. Apart from the changes we explored, we also have a new SASS project item and editor, some changes in ASP.NET Scaffolding, some ASP.NET template changes including ASP.NET Identity.

Microsoft's is certainly on the right path giving developers a single set of tools that spans devices and the cloud. Exciting times ahead! ■

Mahesh Sabnis 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 bit.ly/HsS2on