Issue 08 | SEP-OCT 2013 EDITION

DNC Magazine

www.dotnetcurry.com

Windows 8.1 Media player App using Azure Media Services

Task Calendar using Knockout JS and ASP.NET WebAPI

Picture Collage Maker using WinRT 8.1

Questions Web Forms Developers ask about MVC

Building Store Ready Apps for Windows Phone

Web API, Async and Performance

Agile testing in Visual Studio 2012

Interview

SCOTT HUNTER

www.dotnetcurry.com

dnc mag

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

Editorial & Advertising Director • Suprotim Agarwal
suprotimagarwal@dotnetcurry.com

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

Contributing Writers • Gouri Sohoni, Mayur Tendulkar, Sumit Maitra, Suprotim Agarwal

Writing Opportunities • Carol Nadarwalla
writeforus@dotnetcurry.com

NEXT ISSUE - 1st November 2013

letter from the editor

Writing an editor's note is not an easy task, I always leave that to Sumit, our Editor-in-Chief. But with his travel plans and busy schedule this month, I decided to wear the Editorial hat and give it a shot. While I write this note, my brain is trying hard to make it an 'outstanding' note, by sounding poetic, discussing something cutting edge released in the high-tech section like the Lumia 1020 or gossiping about the news of Steve Ballmer retiring next year and who his successor would be. But I intend to do none of this and fall back to the KISS (Keep it short and simple) philosophy I always do, whenever I am feeling overwhelmed.

So let me start by telling you that we had a gala time in July hosting the Giveaway to celebrate DNC Magazine's First anniversary with you folks. We received plenty of questions, suggestions, adulations, criticisms, and everything that falls in between. It was very inspiring to read all of that, so Thank You, all of you, for everything! The Giveaway Winners were declared this month and they are being contacted as I write this note. Congratulations!

With the Windows 8.1 Preview release at the Build 2013 conference in June fresh in our mind, this month, we've put together two special articles on Windows 8.1 by Sumit. Leading off, Gouri explains to you some principles of Agile Testing and how Visual Studio 2012 can help solve some challenges faced during the process. Mayur introduces Windows Phone, a new segment we intend to cover in the magazine going forward, and talks about how to make marketplace ready apps.

Going back to the basics, I cover some questions that ASP.NET Web Form developers usually ask when moving to MVC. In a different article, I also talk about how to leverage async operations to gain performance and build a Web Application that scales better. If that's not enough for our MVC and Web API fans, Sumit cooks up a wonderful article on Knockout JS and Web API to implement a Task Calendar that looks similar to Outlook Calendar's daily View.

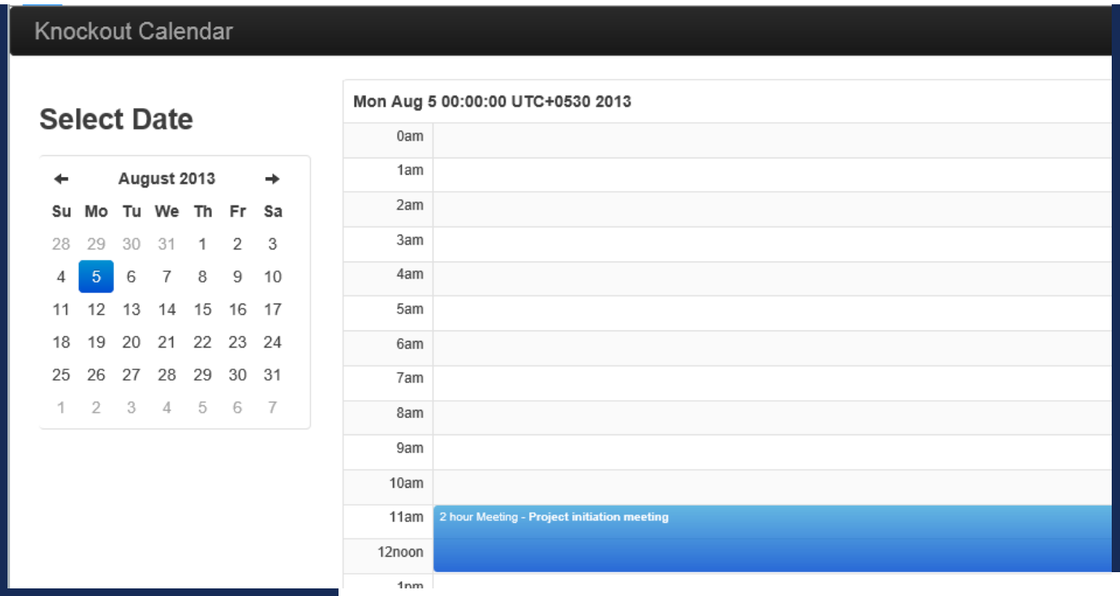
Saving the best for the last, this time we have the privilege to interview Scott Hunter, the Principal Program manager of the ASP.NET team. There's a boat load of exciting new stuff introduced at BUILD 2013 and we talk to him about it, as well as Microsoft's foray into OSS and his journey with Microsoft so far.

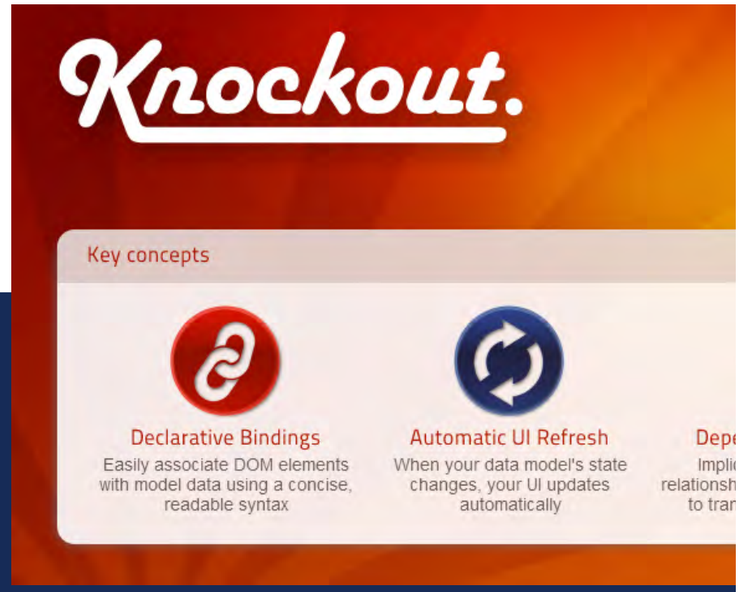
Well that pretty much covers what we have in store for you this month. So here we are, back with a new edition, solving little problems by programming them away. I believe we are all part of something amazing and are eagerly looking forward to what the software development industry has in store for us in the next 5-10 years. I truly believe we are living in the greatest development era ever, so let's enjoy it while we can!

What do you think of this edition or the magazine in general? Email me at suprotimagarwal@dotnetcurry.com

Suprotim Agarwal

Building a Task Calendar using Knockout JS and ASP.NET WebAPI





What is Knockout?

KnockoutJS simplifies dynamic JavaScript UIs by applying the Model-View-View Model (MVVM)

Sumit Maitra uses Knockout JS to implement a Task Calendar that looks similar to Outlook Calendar's daily View

> "This article should be treated as a reference project for putting together a Knockout based app

Managing Tasks is a common requirement and representing them in a Calendar style UI rather than a list of items, adds a level of finesse to your application. However, a calendar UI is non-trivial to build. Today we'll get a leg up with a task calendar. This will force us to use Knockout in a *near-real-world scenario* throwing up some challenges on the way.

Due to space constraints, I have not included significant chunks of markup and/or code. So treat this article as a reference for putting together a KO based app. We've data binding, binding handlers, calendars, computed values, date pickers and even a Time Picker. Getting all of these to work coherently, requires tweaking and tuning. This article is focused on those bits rather than being a complete walkthrough. That said, you can always download the code from Github and walk along as you read.

THE REQUIREMENT

The final application should have the following features:

1. A Date Picker to select a date

2. A Calendar View to show 24 hours of the day, with 60 minute blocks.

3. Clicking on the calendar should open a pop-up allowing - new task, task details edit or task detail deletion.

GETTING STARTED

We get started with an Empty ASP.NET MVC project and add dependencies as we go along.

Setting up Dependencies

We'll use a bunch of Nuget packages in our app and use the Package Manager Console to download them.

- jQuery

```
PM> install-package jQuery
```

- KnockoutJS

```
PM> install-package KnockoutJS
```

- Web Optimization so that we can use ASP.NET Bundling and Minification. This also installs Microsoft.Web.Infrastructure and WebGrease.

```
PM> install-package Microsoft.
AspNet.Web.Optimization
```

- We'll update Web API to the latest

```
PM> update-package Micrsoft.AspNet.
WebApi
```

- Get EntityFramework for the Data layer

```
PM> install-package EntityFramework
```

- Finally get MvcScaffolding to generate Repository based data layer

```
PM> install-package MvcScaffolding
```

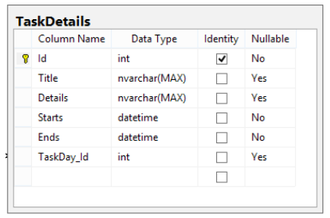
Setting up Connection String

In the Web.config, we'll add the following connection string to make use of SQL Server localdb. You can set it up to point to your SQL Server.

```
<add connectionString="
DefaultConnection"
providerName="System.Data.SqlClient"
 name="Data Source=(LocalDB)\v11.0;
AttachDBFilename=|DataDirectory|MvcK
oCalendar.mdf;
Integrated Security=True"/>
```

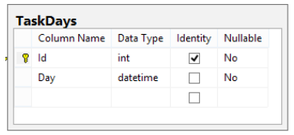
The Task Schema

We have two classes to store the task related data – TaskDay and TaskDetails. TaskDay simply stores the date on which we have an appointment whereas TaskDetails stores the Title, Name, Start Time, EndTime etc. Each TaskDay can contain one or more Task Details.



TaskDetails

Column Name	Data Type	Identity	Nullable
Id	int	☑	No
Title	nvarchar(MAX)	☐	Yes
Details	nvarchar(MAX)	☐	Yes
Starts	datetime	☐	No
Ends	datetime	☐	No
TaskDay_Id	int	☐	Yes
		☐	



TaskDays

Column Name	Data Type	Identity	Nullable
Id	int	☑	No
Day	datetime	☐	No
		☐	

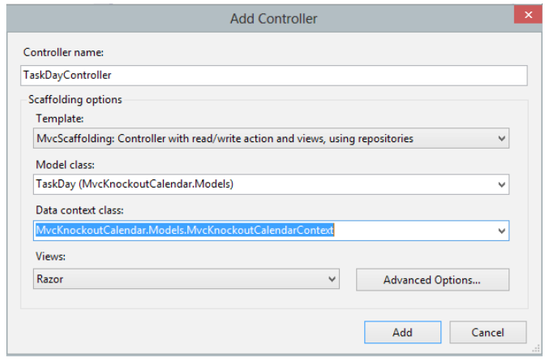
The classes are as follows:

```
public class TaskDay
{
    public int Id { get; set; }
    public DateTime Day { get; set; }
    public List<TaskDetail> Tasks { get; set; }
}

public class TaskDetail
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Details { get; set; }
    public DateTime Starts { get; set; }
    public DateTime Ends { get; set; }
}
```

Once the classes are in place, time for us to Scaffold some CRUD screens and generate some Sample data.

Scaffolding CRUD Screens

1. Right click on the Controller folder and select Add New Controller



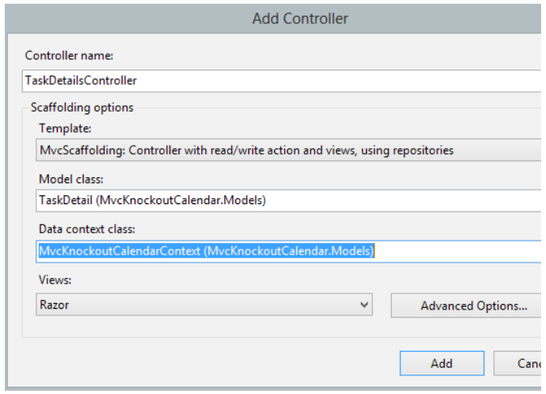
2. Select the TaskDay class to generate the controller. Note, the Template used is from MVCScaffolding using repositories. If you don't see the TaskDay class, hit Cancel, build the solution and start from Step 1 again.

3. Once the controller and views are setup, open the MvcKnockoutCalendarContext class and add the following constructor.

```
public MvcKnockoutCalendarContext():
    base(nameOrConnectionString: "DefaultConnection")
{
}
```

4. This passes the connection string that we declared earlier to EntityFramework. Without this, EF tries to find a local SQL Express and create the database based on the Context's NameSpace.

5. Next we Scaffold the TaskDetails as well.



6. If we run the Application now and navigate to the /Tasks or /TaskDetails URL, we should be able to create new TaskDay or TaskDetails entities that get persisted in the database.

As we can see, these screens look rather bland. So let's spruce it up by applying some BootStrap styling.

Styling with Bootstrap (2.x)

1. Creating a BundleConfig: In the App_Start folder, add BundleConfig.cs. We add jQuery, BootStrap, Knockout and jQuery Validation script and styles as follows

```
public class BundleConfig
{
 public static void RegisterBundles(
    BundleCollection bundles)
 {
  bundles.Add(new ScriptBundle("~/bundles/jquery").
  Include(
        "~/Scripts/jquery-{version}.js"));
  bundles.Add(new ScriptBundle("~/bundles/bootstrap").
   Include(
   "~/Scripts/bootstrap.js",
   "~/Scripts/html5shiv.js"));
  bundles.Add(new ScriptBundle("~/bundles/jqueryval").
  Include(
   "~/Scripts/jquery.unobtrusive*",
   "~/Scripts/jquery.validate*"));
  bundles.Add(new ScriptBundle("~/bundles/knockout").
  Include(
   "~/Scripts/knockout-{version}.js"));
  bundles.Add(new StyleBundle("~/Styles/bootstrap/css").
  Include(
   "~/Content/bootstrap-responsive.css",
   "~/Content/bootstrap.css"));
 }
}
```

2. Invoking the Bundle Config: Update the App_Start event to use the Bundle Config. This requires the System.Web.Optimization bundle.

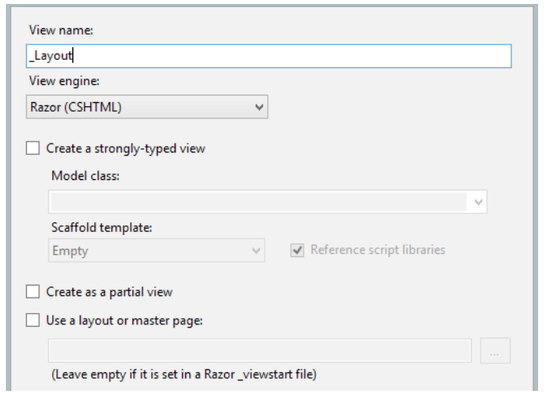
```
BundleConfig.RegisterBundles(BundleTable.Bundles);
```

3. Adding Master Pages:

a. Add a _ViewStart.cshtml in the View Folder with the following content.

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

b. Create a Shared folder under Views folder _Layout.cshtml



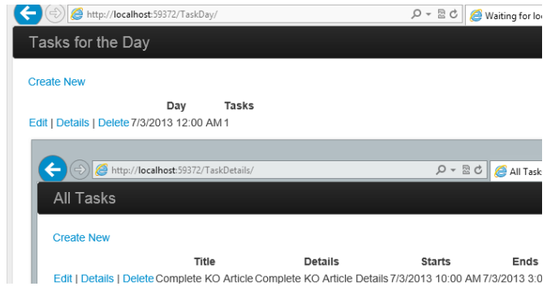
Update the markup as follows to include the jQuery, BootStrap and KO Bundles

```
<html>
<head>
 <meta name="viewport" content="width=device-width" />
 <title>@ViewBag.Title</title>
 @Styles.Render("~/Styles/bootstrap/css")
 @Scripts.Render("~/bundles/modernizr")
</head>
<body>
 <div class="navbar navbar-inverse">
  <div class="navbar-inner">
   <div class="brand">
    @ViewBag.Title
   </div>
  </div>
 </div>
 <div class="container-fluid">
 <div class="row-fluid">
  @RenderBody()
 </div>
</div>
 @Scripts.Render("~/bundles/jquery")
 @Scripts.Render("~/bundles/knockout")
 @Scripts.Render("~/bundles/bootstrap")
 @RenderSection("Scripts", false)
</body>
</html>
```

We have to add a reference to System.Web.Optimization in the Web.Config.

```
<system.web.webPages.razor>
 <pages pageBaseType="System.Web.Mvc.WebViewPage">
  <namespaces>   …
   <add namespace="System.Web.Optimization"/>
  </namespaces>
 </pages>
</system.web.webPages.razor>
```

4. All set, if we run the application and navigate to the /TasksDay or /TaskDetails pages, we'll see the following:



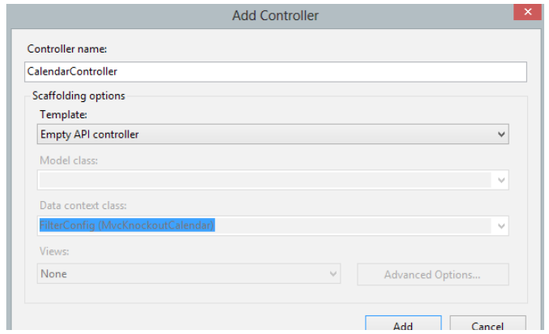
Note: There are some modifications to the scaffolded pages. I have updated the Title property in the ViewBag and removed the Header that said 'Index'.

With that, our basic infrastructure to save Task related details, MasterPages, Database is all setup. Let's build the calendar UI now.

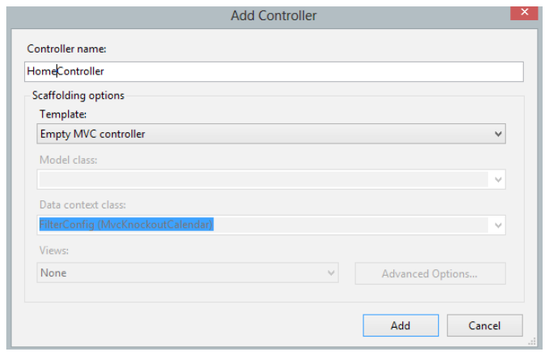
BUILDING THE CALENDAR

Setting up Controllers

We start off with a new Empty ApiController called CalendarController. We will route our DB calls via this controller.



To host the Calendar, we add an MVC Controller called HomeController.



Setting up the View and ViewModel

I usually take one piece at a time, but in this project, I have to interlace the ViewModel along with the View as I build it out. We start off with the View.

The Basic Layout

Add a Home Folder under views and add a blank cshtml called Index.csthml. This will be our calendar's view page.

First requirement of the day is to have an inline date picker. There are tons of them but we need one that matches bootstrap's styling. After a bit of searching, I have narrowed down to Stefan Petre's project with additions by Andrew Rowls. You can get it from here http://www.eyecon.ro/bootstrap-datepicker .

Next we setup our HTML markup. This is going to be simple because we will only be doing the layout and templating. KO will render rest of it dynamically.

As we see in the markup below, we have split the layout into two divs. The first one has a style span3. This is a bootstrap class that splits up the screen-width into 12 responsive columns as-in they change width as the browser changes width, till the extent possible.

```
<div class="span3" style="padding: 5px">
 <h3>Select Date</h3>
 <div id="inlineDatepicker" class="table-bordered"></
div>
</div>
```

So we have split our calendar page into two into 1/4th and 3/4th sized columns. The left column will contain the Calendar and the right column will have a table of 25 rows representing the day and the 24 hours in it.

```
<!-- Appointment Template goes here -->
<!-- Calendar UI -->
<div class="span9">
 <table class="table table-bordered table-striped"
  id="appointments-table">
 <thead>
  <tr>
   <td colspan="2">
    <strong>
     <span data-bind="text: selectedDate"></span>
    </strong>
   </td>
  </tr>
 </thead>
 <tbody data-bind="foreach: dateDetails">
   <tr data-bind="with: TaskDetails">
    <td style="width: 10%; text-align: right;
      vertical-align: top; padding-top: 0px">
     <small data-bind="text: TimePeriod"></small>
    </td>
   <td data-bind="click: create"></td>
  </tr>
 </tbody>
</table>
<!-- Calendar UI -->
<!-- Modal Dialog goes here -->
</div>
```

The div with the *inlineDatePicker* id is used to render the plugin. We use the following snippet to initialize it.

```
$(document).ready(function ()
{
 $('#inlineDatepicker').datepicker().on('changeDate',
function (ev)
 {
  vm.selectedDate(ev.date);
  vm.initializeDateDetails();
  vm.getTaskDetails(ev.date);
 });
}
```

Before we discuss the right hand side , let's look at the minimal Knockout View Model required to support it.

```
var viewModel = function ()
{
 var $this = this;
 var d = new Date();
 $this.selectedDate = ko.observable(new Date(
  d.getFullYear(), d.getMonth(), d.getDate()));
 $this.dateDetails = ko.observableArray();
 $this.appointments = ko.observableArray();
 $this.selectedTaskDetails = ko.observable(new
taskDetails(d));
 $this.initializeDateDetails = function ()
 {
  $this.dateDetails.removeAll();
  for (var i = 0; i < 24; i++)
  {
   var d = $this.selectedDate();
   $this.dateDetails.push({
    count: i,
    TaskDetails: new getTaskHolder(i, d)
   });
  }
 }
 …
}
```

The above ViewModel consists of:

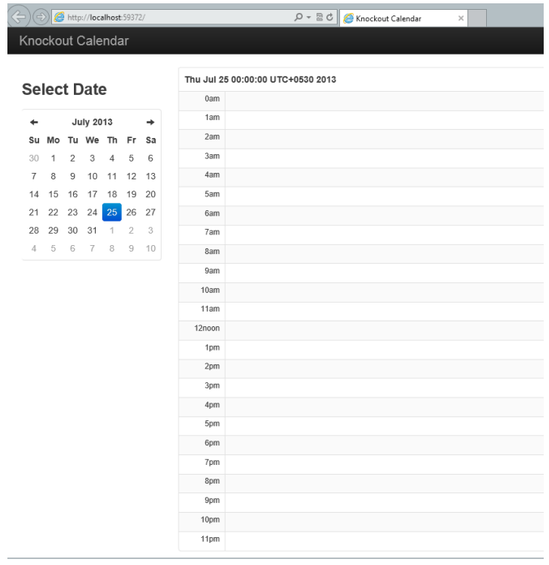
Selected Date ($this.selectedDate): This stores the current date selected in the inline calendar we just saw. On load, selectedDate is the same as current Date.

Date Details ($this.dateDetails): This is the observable collection that contains 24 Task Holder objects for each Day. These Task Holder objects are dummies, in the sense they are not persisted to the Database. They are used for rendering the day calendar's layout.

Task Details ($this.appointments): This is an observable collection of tasks that are actually present for that day. These are overlayed on the date details independently.

Selected Task Details ($this.selectedTaskDetails): This holds the task that is being edited.

Overall the Calendar Control looks as follows at this point.



The Modal Dialog

Next we need a Modal Dialog that comes up when we click on the Time Slots. To do this, we add a Click event handler to the View Model and then bind it to each Time Slot.

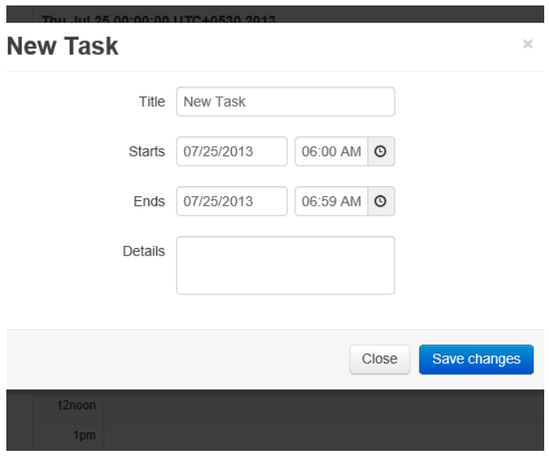
The Click handler function edit is defined as a part of the Task Holder view model as follows:

```
var getTaskHolder = function (i, d)
{
 var $this = this;
```

```
$this.TaskDetails = ko.observableArray();
$this.StartDate = ko.observable(new
 Date(new Date(d).setMinutes(i * 60)));
$this.EndDate = ko.observable(new
 Date(new Date(d).setMinutes(((i + 1) * 60) - 1)));
$this.TimePeriod = ko.computed(function ()
{
 var hr = $this.StartDate().getHours() > 12 ?
 $this.StartDate().getHours() - 12 :
 $this.StartDate().getHours();
 var amPm = ($this.StartDate().getHours() > 12) ?
 'pm' : ($this.StartDate().getHours() == 12) ?
 'noon' : 'am';
 return hr + amPm;
});
$this.edit = function (data)
{
 var selDateTime = new getTaskDetails(
  $this.StartDate());
 selDateTime.Starts(data.StartDate());
 selDateTime.Ends(data.EndDate());
 selDateTime.ParentTask(data);
 if (data instanceof getTaskHolder)
 {
  vm.selectedTaskDetails(selDateTime);
  $('#myModal').modal('toggle');
 }
};
}
```

Take a note of the getTaskDetails object used here. We'll take a closer look at it in the next section.

On click, we toggle a Bootstrap Modal dialog that looks as follows:



The dialog looks great and data in it is data-bound correctly. However we have done a few things that need more explanation.

1. Attaching Date Picker
2. Attaching a Time Picker
3. Using Knockout Computed values in our ViewModel

The Date Picker in Modal using Knockout BindingHandler

To make sure KO Data binding works properly between the date-picker, the input box and the view model best practice is to use a *Knockout Binding Handler*. A binding handler allows you to use custom binding syntax and put in code to execute on Initialization and Updation of the ViewModel.

The Date Picker Binding Handler is added to the day-calendar.knockout.bindinghandlers.js

```
ko.bindingHandlers.datepicker = {
 init: function (element, valueAccessor,
  allBindingsAccessor)
 {
  //bind the datepicker to the text box in the modal
  var options = allBindingsAccessor().datepickerOptions
|| {};
   $(element).datepicker(options);

  //update the view model when user updated the modal
  ko.utils.registerEventHandler(element, "changeDate",
  function (event)
  {
   var value = valueAccessor();
   if (ko.isObservable(value))
   {
    value(event.date);
   }
  });
 },
 update: function (element, valueAccessor)
 {
  var widget = $(element).data("datepicker");
  //when the view model is updated, update the widget
  if (widget)
  {
   widget.date = ko.utils.unwrapObservable(
   valueAccessor());
   widget.setValue();
  }
 }
};
```

To use the Binding Handler, we bind it to the StartDate field as follows

```
<input id="selectStartDate" data-bind="datepicker:
Starts" type="text" class="span12">
```

It's worth mentioning that we were able to use the same datepicker control that we used for the inline picker.

The Time Picker in Modal using Knockout BindingHandler

The date picker was easy, but we needed a time picker as well. A little bit of searching led me to this excellent plugin http://jdewit.github.com/bootstrap-timepicker. Note: I have used Apache or MIT licensed components, but always be aware of your OSS licenses and usage.

Just like the Date Picker, we need a Binding Handler for the Time Picker as well. The Handler for the time picker is as follows

```
ko.bindingHandlers.timepicker = {
 init: function (element, valueAccessor,
 allBindingsAccessor)
 {
  //initialize timepicker
  var options = $(element).timepicker();
  //when a user changes the date, update the view model
  ko.utils.registerEventHandler(element,
  "changeTime.timepicker", function (event)
  {
   var value = valueAccessor();
   if (ko.isObservable(value))
   {
    value(event.time.value);
   }
  });
 },
 update: function (element, valueAccessor)
 {
  var widget = $(element).data("timepicker");
  //when the view model is updated, update the widget
  if (widget)
  {
   var time = ko.utils.unwrapObservable(
   valueAccessor());
   widget.setTime(time);
  }
 }
};
```

The data binding attribute for the time picker is as follows:

```
<input id="selectStartTime" data-bind="timepicker:
StartTime" class="span8" type="text">
<span class="add-on"><i class="icon-time"></i></span>
```

The additional span is used to hold the 'picker-icon'. This is managed by the Timepicker plugin.

Saving new Appointments

Before we can Save the appointment, let's see the object that our Modal Dialog is binding too. The object is defined in the taskDetails function. We'll walk through this function step by step

```
var taskDetails = function (date)
{
 var $this = this;
 $this.Id = ko.observable();
 $this.Title = ko.observable("New Task");
 $this.Details = ko.observable();
 $this.Starts = ko.observable(new Date(new Date(date).
setMinutes(0)));
 $this.Ends = ko.observable(new Date(new Date(date).set-
Minutes(59)));
```

As we can see above, the taskDetails function is called or an instance of it is created by passing an initial date value. After initialization, the properties - Id, Title, Details, Starts, Ends all map to the server side TaskDetail entity. This is by design as this view model will be posted directly to the WebAPI controller, as we will see shortly.

Introducing KO Computed Observables

Next up, we have a property called deleteVisibility and it's set to a ko.computed(...) property. A Computed Observable in Knockout is one that derives its final value based on the value of some other property in the view model. If that property changes, the computed property is also updated automatically.

The deleteVisibility property depends on whether the Id property is 0 or not. The Id of 0 implies that the Appointment is new and it doesn't need a delete button, so the value returned is "hidden" which is set bound to the visibility style of a div that contains the delete button on the Modal.

```
$this.deleteVisibility = ko.computed(function ()
 {
  if ($this.Id() > 0)
```

```
  {
   return “visible”;
  }
  else
  {
   return “hidden”;
  }
});
```

Read and Writeable KO Computed Observables

The above Computed observable was only read only. This works fine for non-input type of values. However if we bind an input element to a computed field, then it is likely that when the value in the input changes, we have to set it back to ‘something’ in the view model. We have the exact scenario in case of StartTime and EndTime properties that we are using to bind to the Time Picker. Both these values are actually spliced off the Starts and Ends values saved in the database. As a result, we have to splice them when we want to show the time in the pickers and update the Starts property when the Time changes in the time pickers.

To do this, we pass an object with two properties read and write to the Computed Observable. KO internally calls these whenever it needs to get a value or whenever the bound field is updated. As we can see below, the read property is a function that returns a Time String from the Starts() date. The write property is defined as a function with an input value. The ‘value’ is the newly updated text. When the new value comes in, we create a new Date Object with the date in the Starts property and the Time as the value passed. We then set it back to the Starts() property.

```
$this.StartTime = ko.computed({
  read: function ()
  {
   return $this.Starts().toLocaleTimeString();
  },
  write: function (value)
  {
   if (value)
   {
    var dt = new Date($this.Starts()
      .toLocaleDateString() + “ “ + value);
    $this.Starts(new Date($this.Starts().getFullYear(),
      $this.Starts().getMonth(),
     $this.Starts().getDate(), dt.getHours(),
       dt.getMinutes()));
   }
  }
 });
```

The EndTime computed property is implemented in the same way as the StartTime property, except that it now uses the Ends Property to calculate the required value.

```
 $this.EndTime = ko.computed({
  // Similar to StartTime, uses $this.Ends() instead of
$this.Starts
  …
 });
```

Saving the Task Details

Next we have ‘Save’ in our taskDetails view model. It uses the model data, converts it to a JavaScript object and posts it using AJAX to the SaveTask controller Method.

```
$this.Save = function (data)
{
 var submitData = ko.mapping.toJS(data)
 submitData.Starts = (submitData.Starts.
       toLocaleString());
 submitData.Ends = (submitData.Ends.toLocaleString());
 var postUrl = “/api/Calendar/SaveTask”;
 $.ajax({
  url: postUrl,
  type: “POST”,
  contentType: “text/json”,
  data: JSON.stringify(submitData)
 }).done(function (data)
 {
  $(‘#myModal’).modal(‘toggle’);
  vm.getTaskDetails(vm.selectedDate());
 }).error(function (data)
 { // code not shown here for brevity
}
```

Adding SaveTask Controller Method

In the CalendarController.cs, we add the following code. We post a task JSON from the client and as we can see, it is bound to the *task* input parameter.

```
public bool SaveTask(TaskDetail task)
{
 DateTime targetDay = new DateTime(task.Starts.Year,
 task.Starts.Month, task.Starts.Day);
 TaskDay day = _taskDay.All.FirstOrDefault<TaskDay>(_ =>
 _.Day == targetDay);
 if (day == null)
 {
  day = new TaskDay
  {
   Day = new DateTime(task.Starts.Year, task.Starts.
   Month, task.Starts.Day),
   Tasks = new List<TaskDetail>()
  };
  _taskDay.InsertOrUpdate(day);
  _taskDay.Save();
 }
 task.ParentTaskId = day.Id;
 task.ParentTask = null;
 _taskDetail.InsertOrUpdate(task);
 _taskDetail.Save();
 return true;
}
```

Next we check if there are any other TaskDay entities for the given day. If there is one, we retrieve it and create a new TaskDetails with its ID. If there are no previous tasks in the day, we instantiate a new task instead of TaskDay and Save it. Finally we assign the TaskDay’s Id to the TaskDetail object, and send it off to the InsertOrUpdate method in the repository. We then call the Save method.

Loading Task Details

Now that we have started saving tasks, let’s reload them back as well. To do this, we need the following controller code

```
public List<TaskDetail> GetTaskDetails(DateTime id){
 TaskDay taskDay = _taskDay.All.
 FirstOrDefault<TaskDay>(_ => _.Day == id);
 if (taskDay != null)
 {
  return taskDay.Tasks;
 }
 return new List<TaskDetail>();
}
```

Retrieving Tasks on the Client

The getTaskDetails function in the main view model retrieves the tasks by calling the above server-side API.

```
$this.getTaskDetails = function (date){
 var submitValue = new Date(date.getFullYear(), date.
getMonth(), date.getDate());
 var uri = “/api/Calendar/GetTaskDetails”;
 $.get(uri, ‘id=’ + submitValue.getFullYear() + ‘-’ +
(submitValue.getMonth() + 1) + ‘-’
 + submitValue.getDate()).done(function (data)
{
 $this.appointments.removeAll();
 $(data).each(function (index, element)
 {
  $this.appointments.push(new appointment(element,
  index));
 });
}).error(function (data)
{ // code not shown here for brevity
}
```

Once the data is returned, we loop through each task and create an *appointment* view model object out of it. As we’ll see in the next section, the appointment object is used to render the tasks in the UI.

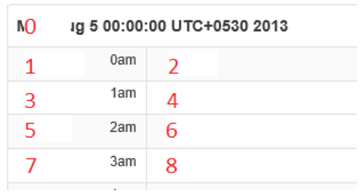
Displaying Tasks in Calendar

This is probably the most interesting part of the Calendar. Positioning of the appointment elements based on the Start Time and duration required some creative hacking. The appointments are loaded when a user changes dates on the date calendar. The getTaskDetails function calls the Web API to retrieve the task appointments. For each appointment, we create the following ViewModel.

```
var appointment = function (task, i){
 var $this = this;
 $this.id = ko.observable(task.Id);
 $this.starts = ko.observable(new Date(task.Starts));
 $this.ends = ko.observable(new Date(task.Ends));
 $this.title = ko.observable(task.Title);
 $this.details = ko.observable(task.Details);
 var trd = $(“#appointments-table tr td”);
 var trdIndex = ($this.starts().getHours() * 2) + 2;
 var top = $(trd[trdIndex]).position().top + ($this.
 starts().getMinutes() / 2) + 1;
 $this.top = ko.observable(top + “px”);
 $this.posleft = ko.observable(($(trd[trdIndex]).
 position().left + 1) + “px”);
 var diff = $this.ends() - $this.starts();
 $this.posHeight = ko.observable(((diff / 1000 / 120) -
 2) + “px”);
 $this.posWidth = ko.observable(($(trd[trdIndex]).
 outerWidth(false) - 7) + “px”);

 // Edit Appointment Function goes here
}
```

In the above code, we load the StartDate, EndDate, Title and Details from the TaskDetails we received from the database.

To calculate the position of each appointment, we first retrieve a list of the <td> elements in our appointments table (in the variable *trd*). The following image gives a better idea of the sequence of the elements. The first row has a colspan of 2, so the entire row comes in index 0, thereafter there are two columns, so the index (as shown in red) increases left to right, top to bottom.



If we look closely, the index is linked to the hourly tick. We have to place appointments in the second column, so the formula is

```
(hourly tick * 2) + 2
```

This will give us the index of the <td> on which we have to place our Appointment div. Say it is starting at 1:30 am, the hourly tick's value is 1, giving us the <td> index of 4. If you look at the image above, we see this is correct. From trd[4], we use jQuery to retrieve the position object and get the top and left of our Appointment div. The height is calculated by the formula

```
[(startTime – endTime) in milliseconds]/1000/120
```

Here 1000 is to convert milliseconds into seconds and 120 implies 2 minutes = 1px. Thus the above formula gives the number of pixels for the height.

Finally we calculate the width again using the trd[index] and subtract the padding from it. The top, height, width and left values are saved in the posTop, posHeight, posWidth and posLeft properties of the appointment view model. They are bound as follows:

```
<!-- ko foreach: appointments -->
<div
 style="position: absolute" class="appointment"
 data-bind="style: {
  top: posTop,
  left: posLeft,
  height: posHeight,
  width: posWidth }, click: editAppointment">
  <span data-bind="text: title"></span> - <span data-bind="text: details" style="font-weight:bold"></span>
</div>
<!-- /ko -->
```

Editing and Deleting Tasks

On the controllers side, we don't need much to do while editing. Our existing Save method will save updated tasks automatically. If the task's date is being changed, it will update the TaskDay foreign key reference as well.

On the client side, we need to create a taskDetail object when user clicks on the appointment div. For this, we've bound the click event to the editAppointment function in the view model.

The editAppointment function is as follows and is added to the appointment object

```
$this.editAppointment = function (){
 var selDateTime = new taskDetails($this.starts());
 selDateTime.Id($this.id());
 selDateTime.Starts($this.starts());
 selDateTime.Ends($this.ends());
 selDateTime.Title($this.title());
 selDateTime.Details($this.details());
 vm.selectedTaskDetails(selDateTime);
 $('#currentTaskModal').modal('toggle');
}
```

The delete function is added to the TaskDetails object and is as follows. It simply posts back to the Controller with an HTTP DELETE header and the Id of the task in the URL

```
$this.Delete = function (data){
 var postUrl = "/api/Calendar/" + data.Id();
 $.ajax({
  url: postUrl,
  type: "DELETE"
 }).done(function (data)
 {
  $('#currentTaskModal').modal('toggle');
  vm.getTaskDetails(vm.selectedDate());
 }).error(function (data)
 {
  alert("Failed to Delete Task");
 });
}
```

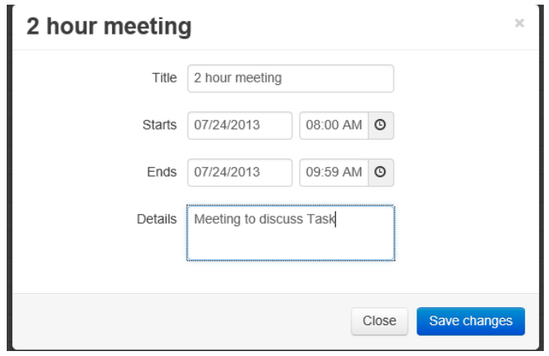
Note the URL doesn't actually need the Action Method's name because we've used the HttpDelete attribute on it in the Controller:

```
[HttpDelete]
public bool DeleteTask(int id){
 try  {
  _taskDetail.Delete(id);
  _taskDetail.Save();
  return true;
 }
 catch (Exception ex)  {
  return false;
 }
}
```

DEMO

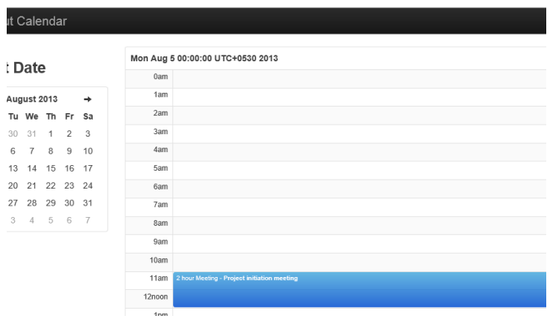
With the generic pieces in place, let's run the application and create some tasks to see how our calendar looks

Step 1: Run the Application and click on a time slot to bring up the 'New Task' dialog.



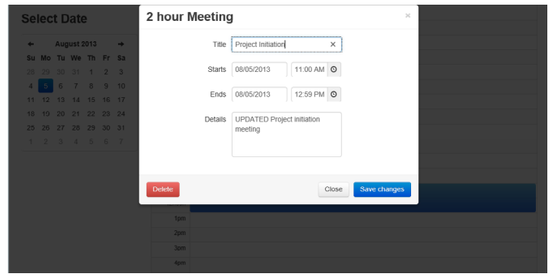
Step 2: Add task details as follows and Save changes.

Step 3: When we select the date, we'll see the Task in the



calendar as following:

Step 4: We can edit the appointment by clicking on it. Notice now you see the Delete button also. So you can update and Save changes or delete the appointment if no longer required.



Pretty neat!

There are a few caveats and gotchas still to go that is left as an exercise, like re-arranging the tasks when the browser is resized, Currently we depend on opacity to show overlapping tasks, we can make the algorithm smarter to resize the width if we have overlapping tasks and so on.

CONCLUSION

Though our calendar implementation is not 100% production ready, we saw the following integration pieces work perfectly

1. ASP.NET + MVC Scaffolding
2. ASP.NET Web API + KnockoutJS
3. Knockout Data Binding
4. Knockout Binding Handlers
5. BootStrap Styling
6. Absolute positioning using jQuery

We can treat this project as a reference project for Knockout related references and sample ■

Download the entire source code from our GitHub Repository at bit.ly/dncm8-kotaskcal

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

Questions Developers Ask When Moving From

ASP.NET Web Forms to MVC

Suprotim Agarwal discusses some common questions that arise when seasoned WebForms developers tread in ASP.NET MVC, for the first time. It's not an MVC vs. WebForms article, but rather focuses on what's different and how.

When WebForms developers move to ASP.NET MVC, it looks like a whole new world. Today we look at some of the first questions that come up in an attempt to look at MVC from the WebForms Developer's eyes.

Question 1: Where is the Page_Load event and the code behind? This separation of concerns thing is a little confusing, how does it work?

To answer the first question directly, there is no Page_Load 'event' in MVC. Fact is there are no 'Page' events in MVC at all. The MVC Pattern promotes separation of Model (data and logic), View (the HTML Page in case of ASP.NET MVC) and Controller (the glue between the View and Model) . To quote from my previous article on - The MVC Pattern and ASP.NET MVC - Back to Basics we can define Model, View and Controller as follows:

Models: Classes that represent the problem domain (the problem/requirement we are trying to solve/accomplish). These classes also have code and logic to serialize and de-serialize the data into dedicated data stores, as well as do validation and define domain-specific logic. In ASP.NET MVC, this is the place for ORM or Data Access frameworks to do their magic.

Views: Views are essentially templates (cshtml or vbhtml files) used to generate the final HTML at runtime. Though we say HTML, we could potentially return other types of Views like PDFs etc.

Controller: The class that manages the relationship between View and the Model. It is also responsible for accepting requests from the User and deciding which View to serve up, if any.

This is different from WebForms which was intrinsically tied to the Controller, that is the Code Behind. This resulted in mixing up responsibility of the View, the Controller (code to generate the HTML encapsulated in Server Side Controls) and the Model, often leading to strongly coupled implementation that were hard to extend. That said, you could write well-designed WebForms apps but you also had the additional responsibility of maintaining the discipline of separating concerns.

Question 2: What's the fuss about Stateless nature of the web? Where did the ViewState go in MVC and how does data from the Web Page go to the Server now?

In all discussions involving MVC and WebForms, you will hear how the Web is meant to be stateless. What it actually means is HTTP (the protocol over which Internet or the Web works) is a stateless protocol and it works on Requests from Clients to Server, and Responses from server, back to Clients. Beyond a Request/Response cycle, the server doesn't know/care about the state of the client.

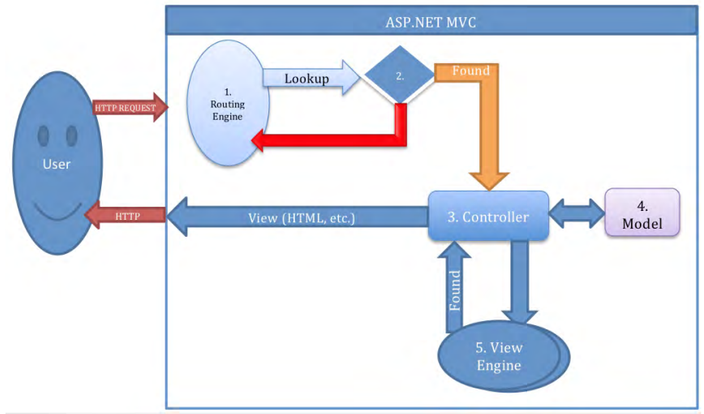
WebForms is an abstraction that was built on top of this, but it was primarily designed as an easy path for Developers from the Visual Basic world to migrate to the web world. So it mimicked things like Controls, events and event handlers, and State. This implied there was a hidden overhead to maintain and transfer client state to the server, every time some action was performed on client and handled on the server.

For example, if you were handling a TextChanged event on the server, it meant that the entire Page with its contents had to be sent to the server along with the information about this server call, because Text on a particular control had changed. ASP.NET WebForms would then interpret this information and call the 'text changed' event handler to execute whatever code you had written for it. When the server side event handler returned, ASP.NET has to send back the exact same HTML that it had received along with the delta of whatever changes you made in the event handler.

All this overhead of information to manage state before and after the event handler is called, was all kept hidden from the developer, but was an overhead. ViewState was one of the mechanisms used to manage the overhead.

In MVC, we no longer have this overhead or the server side eventing model. The only server 'events' are the HTTP calls of GET, POST, PUT, DELETE etc. It's all at the HTTP protocol level.

The following diagram that I borrowed from an earlier article makes things clearer



Looking at the Diagram left to right, top to bottom, the User is sitting at a browser and makes a request over HTTP that goes to the Web Server (hosting ASP.NET and ASP.NET MVC). Once the request reaches the server, it goes through the routing engine, to the controller, which interacts with the Model, invokes the view engine if required and returns a View. If the routing engine doesn't find the URL that the user requested, it returns an error.

Instead of requesting a URL, if user was posting data from a HTML Form to the Server, then the browser (or custom AJAX) stuffs the form data into the Request body and packs it off to the server, which then does something called ModelBinding to create strongly typed Models of the data and send it to the Controller.

Each and every interaction between the User and the Web Application follows the same premise as above. No events, no ViewState or any other overhead. This segues us into the next question.

Question 3: Where are my Rich Server side controls? Do I have to make-do with HTML Controls only? What's this new Razor Syntax?

As WebForms developers, if you are already panicking about losing your favorite server controls, well, don't panic. There are options for pretty much every server side control you have used. But to answer your question straight, there aren't any 'Server Side Controls' so to speak in ASP.NET MVC. There are *HtmlHelpers* but their job is to make it easy to generate HTML on the server side, often using Model values etc. Since there is no 'eventing' per se, there are no 'controls'. Whatever serverside parsing you do, end of the day, its plain old HTML that's going to go to the user's browser. Once on the browser, you can definitely use JavaScript frameworks and libraries to add richness to the interactivity.

Coming back to Razor Syntax, as we saw in the diagram earlier, the Controller after interacting with the Model may want to send back HTML Views. The templates for these Views are in respective *cshtml* files. View Engines interpret the cshtml in context of the Model and return plain HTML. There are multiple ViewEngines available in ASP.NET, each supporting a different server side templating language. The *C# Razor engine* looks for cshtml files that contain C# for templating, the *VB.NET Razor engine* looks for vbhtml files that contain VB.NET for templating. The *ASPX engine* is the similar to the one used by WebForms and uses the <@ @> syntax for templating. Let's look at an example of Razor Engine in action using C#.

Lets say we have the following controller

```
public class HomeController : Controller {
 [HttpGet]
 public ActionResult Index()  {
```

```
  return View(new ScheduleTask());
 }
 [HttpPost]
 public ActionResult Index(ScheduleTask task)  {
  return View();
 }
}
```

The Index() method is called when you do a HttpGet and the Index(ScheduleTask) method is called when data is posted to the server, from the browser.

Let's look at the HttpGet method first; we'll look at the post while answering the next question. As we can see, the Index() method is calling the View(…) method and passing on a Model object to it. For sake of simplicity, it's just a new object without much data.

Things to note here:

1. The View method will return a ViewResult which will in turn have the HTML that needs to be returned to the Browser.

2. Since no name is passed to the View method, it will assume the name of the View to be "Index.cshtml". This is the convention that ASP.NET MVC follows.

So what does the Index.cshtml look like?

```
@model DateRangeValidator.Models.ScheduleTask
@using (Html.BeginForm())
{
 @Html.ValidationSummary(true)
 <fieldset>
  <legend>ScheduleTask</legend>
  <div class="editor-label">
   @Html.LabelFor(model => model.Title)
  </div>
  <div class="editor-field">
   @Html.EditorFor(model => model.Title)
   @Html.ValidationMessageFor(model => model.Title)
  </div>
  <div class="editor-label">
   @Html.LabelFor(model => model.Description)
  </div>
  <div class="editor-field">
   @Html.EditorFor(model => model.Description)
   @Html.ValidationMessageFor(model => model.
                                Description)
  </div>
  <div class="editor-label">
   @Html.LabelFor(model => model.StartDate)
  </div>
  <div class="editor-field">
   @Html.EditorFor(model => model.StartDate)
   @Html.ValidationMessageFor(model => model.StartDate)
  </div>
  <div class="editor-label">
   @Html.LabelFor(model => model.EndDate)
  </div>
  <div class="editor-field">
   @Html.EditorFor(model => model.EndDate)
   @Html.ValidationMessageFor(model => model.EndDate)
  </div>
  <p>
   <input type="submit" value="Index" />
  </p>
 </fieldset>
}
```

This is a quick peek to show basic Razor Syntax

1. The first line declares the fact this view uses an instance of the ScheduleTask object.

2. The @using (Html.BeginForm()) section simply wraps the containing HTML with a <form> element.

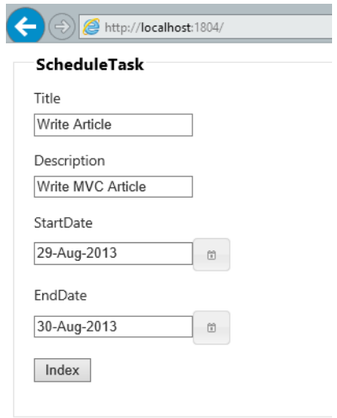
3. The @Html.Labelfor helper generates a <label> element and uses the DisplayName attribute of the Title property (if present) as the text.

4. The @Html.EditorFor(…) adds an appropriate HTML element based on the Model Property that is passed in. For example, a simple String property generates an `<input type="text"…> </input>` elment.

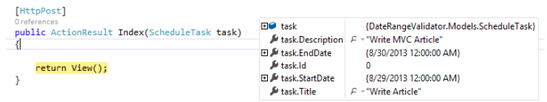
5. Similarly @Html.ValidationMessageFor adds a … tag that shows the validation messages applicable to the Element.

By now, you should have got an idea of how Razor Syntax, cshtml files and view engines are co-related in the MVC stack.

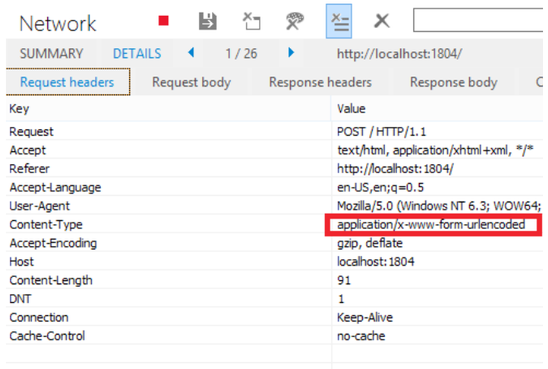
Question 4: Can you expand on Model Binding? Earlier you said data is posted in HTML Body, how does the controller end up with strongly typed objects?
Before we explain Model binding, let's visualize how data is actually getting sent. Let's say we are at the Index view and we fill up the following data in the Form



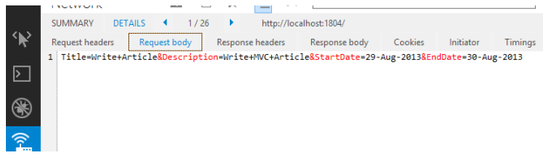
Next we enable Network monitoring in the Browser DeveloperTools (I am using IE11 here), put a break-point in the Controller's Index (HttpPost) method and click on Index button. The Form will do a POST and stop at the breakpoint at the Server. If we dig into the details of the Schedule Task object passed into Controller method, we'll see the data we updated above has been passed to the server



Now if we let the request go through and get back to the Browser, we can see the Request headers



As we can see, our Form is getting posted as an application/x-www-form-urlencoded. Now if we go to the Request body tab, we see our data is Url Encoded.



So somewhere in the Request Pipeline after MVC accepted the Request and before it called the controller, MVC converted the urlencoded data into a ScheduleTask object and filled in the property values. This is 'Model Binding'. MVC uses the content-type, the data in the 'Request body' and the input parameter type of the targeted Action method to create the Model instance and populate the values. Other accepted Content-Types out of the box are JSON and XML. In each case, we need to populate the Request Body appropriately.

Question 5: Okay, but I already have a massive legacy WebForms codebase, can I use ASP.NET MVC for new parts of my Web Application and ease into MVC?
For brown-field expansion projects, it is logical to introduce MVC slowly into the system for self contained new features. So does that mean we can mix and match WebForms and MVC. We did a really elaborate article on this one earlier and we strongly suggest you to refer to it for the details. The outline of the steps involved are as follows:

1. We should upgrade our WebForms project to .NET Framework 4.0 or 4.5, this adds first class routing support, which is essential to make this work.
2. Next we add all the MVC Framework components required, using Nuget package Manager.
3. Update the Web.settings to configure MVC components.
4. To add MVC Tooling support, we need to rejig the csproj file manually by adding a GUID to the <ProjectTypeGuids> element.
5. Add a BundleConfig class and initialize them in the App_Start.
6. Finally add a new Area to the project. An Area is a logical unit of the application that you can apply custom routes etc. to, as if it were a new application. To this new Area, add the Controllers, Views, Styles etc. as required to implement your new functionality.

CONCLUSION

With that, we wrap up this Q&A session for devs moving from Web Forms to MVC. If you have more questions feel free to shoot them our way and we'll try to collate them do one more article on the same ■

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

BUILDING A

WINDOWS 8.1 MEDIA PLAYER APP

USING AZURE MEDIA SERVICES

DOWNLOAD FILES >

bit.ly/dncm8-win8mp

Windows Azure Media Services provides a solid abstraction on top of which we can build our custom Media based solutions. ***Sumit Maitra*** *demonstrates how to use it to setup Adaptive Smooth Streaming video and a Windows 8 Store app to play the video*

Traditionally setting up a Media Streaming Service has multiple moving parts including taking care of Stream type, Media Encoding, High Performance Hosting and Clients to showcase the service. Each of these steps is non-trivial to setup.

However, Windows Azure Media Services provides a solid abstraction that takes care of Storage, Encoding as well as streaming of Media, and exposes easy to use APIs for multiple clients to help us quickly ramp up with a capable Media Service.

Let's see how we can go about leveraging the Windows Azure Media Services, as we try to setup our own Media Streaming Application. To start off with, lets quickly recap types of Media Streaming.

STATE OF MEDIA STREAMING OVER THE INTERWEBS

Media Delivery methods over the HTTP implies one of the following three streaming methods:

1. Progressive Downloads: Here a media file is progressively downloaded to the client and then played. However media quality/bitrate is constant irrespective of bandwidth.

2. Traditional Streaming: Here media is chunked in packets and sent over HTTP to client. Client can interrupt and send Play/Seek/Pause etc. commands and the server responds appropriately. Again media quality/bitrate is constant irrespective of bandwidth.

3. Adaptive Streaming: In Adaptive Streaming, media is present as a manifest of available streams of data. There are multiple streams of data available corresponding to a fixed set of bitrate values. When a client begins streaming an Adaptive Streaming media over HTTP, it exchanges bandwidth and rendering capabilities with the server periodically. This enables servers to adjust the streaming rate if bandwidth drops below required thresholds. This dynamic switching of bitrate keeps media flowing smoothly with dynamic upgrade or downgrade of media quality. Microsoft's implementation of Adaptive Streaming is called ***SmoothStreaming***.

WINDOWS AZURE MEDIA SERVICES – THE PAAS FOR MEDIA

Clients for Adaptive Streaming

As we saw above, Adaptive Streaming requires client side interaction with the server and this necessitates that there are Adaptive Streaming media clients on every conceivable platform. Apparently Windows Azure Media services team has most of this covered for you.

On the Web: You have Flash and Silverlight clients. HTML5 has a draft MSE/EME (W3C Extensions) proposal in the works for supporting Adaptive Streaming. IE11 is already working on support for this.

Native App Support: There is a Smooth Streaming Player Framework for Windows and Xbox. Microsoft also has a Smooth Streaming Porting kit to enable set-top boxes etc. implement their respective clients.

Mobile: On Windows Phone and iOS, we have Player Framework that has Smooth Streaming support available. Android support is enabled by Partner SDKs and also through the Open Source Media Framework (OSMF) plugin.

With wide ranging clients supported, the reach story of Azure Media services is nearly complete, so now let's get hands on and see what it takes to get started with Azure Media Service that is supposed to take care of the 'backend'.

Then we'll get hands on with Code and build a Windows Store App.

Windows Azure Media Services – The Platform

Windows Azure Media Services provides us with the Storage, Compute, Fabric and Database from Windows Azure. It builds on top of these by enabling Encoding, Secure Uploading, Publishing and On Demand Origin (basically ability to view media on demand over http). It provides REST APIs as well as Client SDK to do all of this programmatically and has a well fleshed out Web Portal to do these via an admin interface as well.

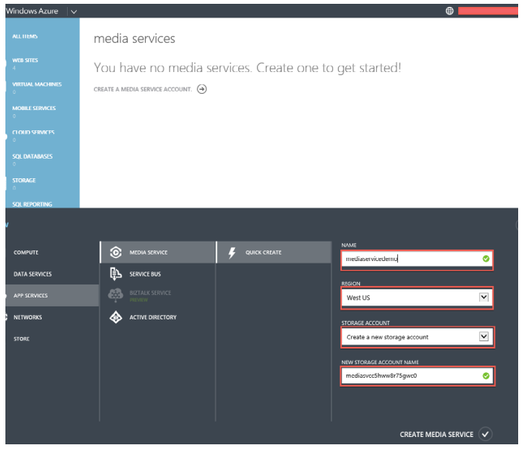
Today we will use the Web Portal for the Administrative functions and the Windows 8 client SDK for building a client.

SETTING UP YOUR AZURE MEDIA SERVICE VIA WINDOWS AZURE PORTAL

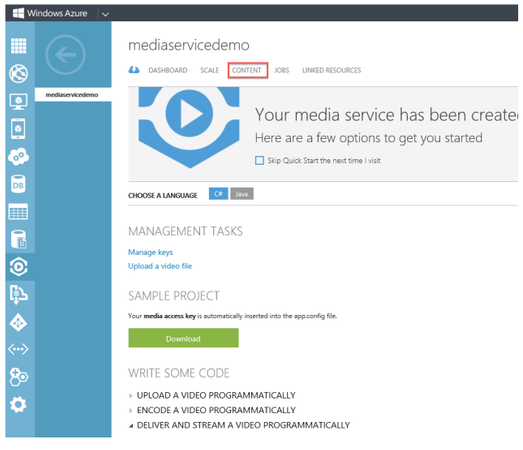
Step 1: Login to your Windows Azure portal at http://manage.windowsazure.com/

Step 2: Select New from the bottom toolbar and select App Services > Media Service > Quick Create:

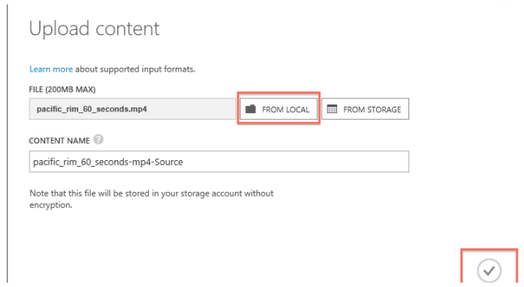
As seen here, we have to provide a Name for the Media Service, pick a Data Center Region, and select a Storage Account. I did not have a Storage Account, so it offered to create a new one for me.



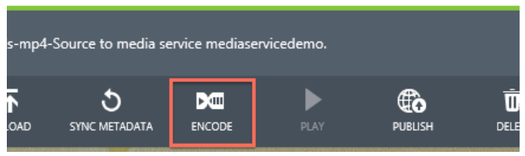
Once done, click on 'Create Media Service' button to initiate the Service. It takes a few minutes to provision the Service, especially if you do not have a Storage Account setup. It will indicate with a banner once the Service has been provisioned successfully, and the service will come up in the dashboard.



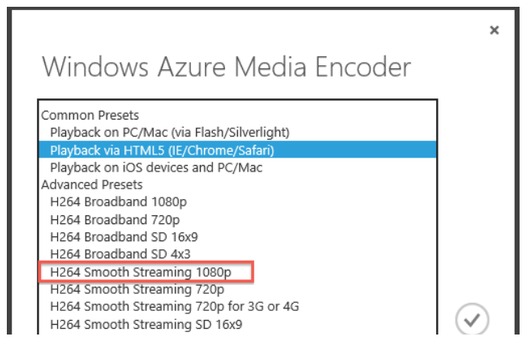
Step 3: Once the service has been setup, we can navigate to the Content Tab and upload a single format Media file. In our case, we'll upload an MP4 file from our local machine.



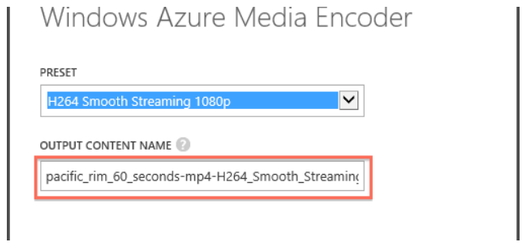
Note that you can pull in a file already stored in Azure Storage too. Click the 'Check' button to begin upload. Once it completes, you will see a Success banner at the bottom.



Step 4: Once the upload completes, select the file in the list and you will see the Encode button enabled for it. Click on Encode to select Encoding options. The first option is the Encoding type and we a see a boatload of presets optimized for the types of client you are targeting. I've selected 1080p as seen below.

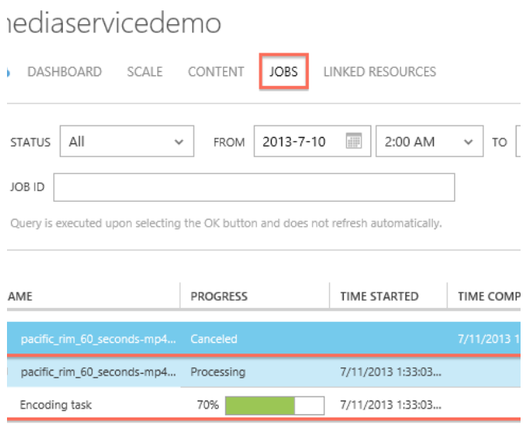


The next option in the Encode dialog is the Content Name. I left it as suggested



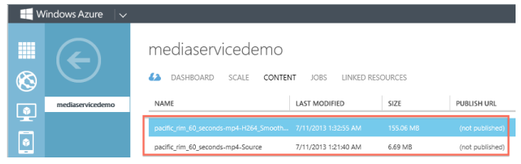
Click on the 'Check Mark' button to initiate the Job. Remember, encoding is a long running job and it takes some time to complete. Fact is it takes some time to even appear on the Jobs tab.

You can expand Job in the Jobs tab to see the Progress % of the encoding task



Note: I was initially confused as to whether the Job was initiated or not, so I initiated a second job, which you can see, I cancelled later. So if you don't see the job immediately, give it about a couple of minutes before refreshing.

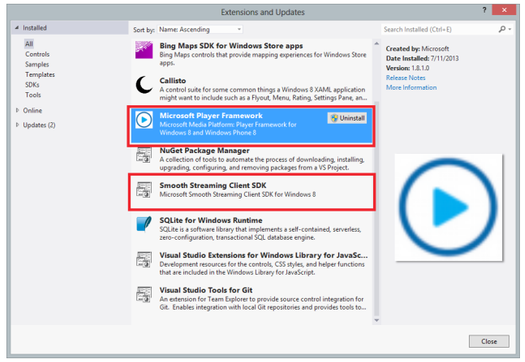
Once the encoding completes, you can see both the Single Bitrate and Multi Bitrate assets in the Dashboard



With this, we are nearly set. As you can see, both the videos are not published so they are not available to the outside world. Select each one of them and from the bottom tool bar, click Publish to get the publish URLs for each.

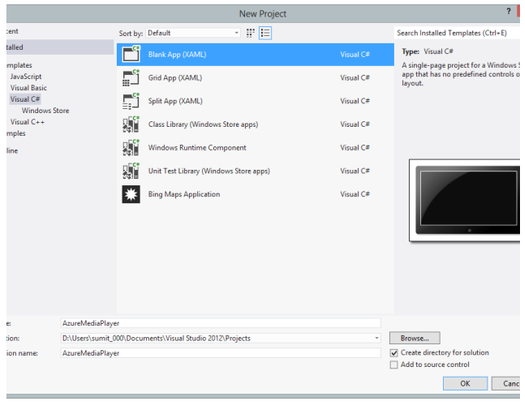
BUILDING A CLIENT WINDOWS 8

To build a Windows 8 app, we need the Smooth Streaming Extension SDK and a Player Framework. These are present as VSIX extensions and can be installed from the Visual Studio extensions gallery also.

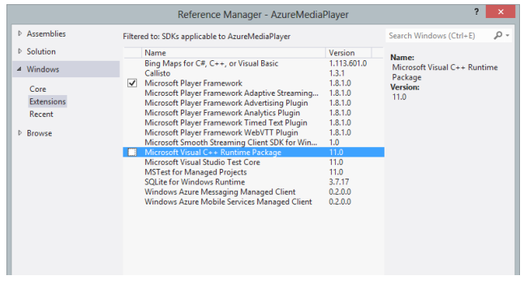


Building a Progressive Download Player

Step 1: Once the dependencies are set, we start off with a new Windows 8 Store app using C# and XAML.



Step 2: Once the project is initialized, we add reference to the Player Framework to our Application



Note: We have not selected the Adaptive Streaming component yet. Click OK to continue.

Step 3: In the MainPage.xaml, add the following namespace reference to the Page

```
xmlns:mmppf=”using:Microsoft.PlayerFramework”
```

Step 4: Next we add the Player using the following markup

```
    <Grid Background=”{StaticResource
ApplicationPageBackgroundThemeBrush}”>
        <mmppf:MediaPlayer Source=”https://
mediasvcc5hww8r75gwc0.blob.core.windows.net/
asset-7bc60521-5b61-4a8d-847f-623b8e2e6111/pacif-
ic_rim_60_seconds.mp4?sv=2012-02-12&amp;st=2013-
07-11T10%3A38%3A29Z&amp;se=2015-07-11T10%3A38%
3A29Z&amp;sr=c&amp;si=11d5e306-e338-4643-9b94-
6ec21d13c0d3&amp;sig=FJZ%2FoiP1vYJq%2BA4HjSIhUfFrvWnuj
dj6GHpoHENXCHg%3D” />
    </Grid>
```

This adds the player framework. The Source value is the URL for the direct MP4 file asset that we had uploaded (not the smooth streaming source we encoded).

Note: Paste the Source Url in the Property Window and not the XAML markup. If you don't copy it into the Property Window, the ampersands won't be escaped and you will end up with 'Invalid Markup' error.

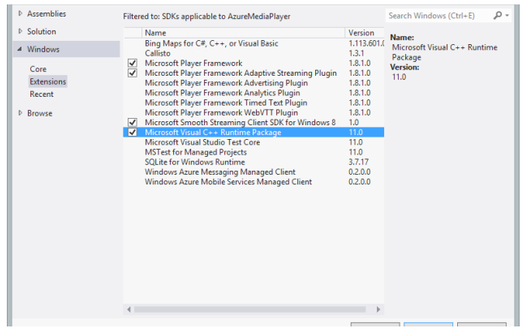
If we run the application now, we'll see the Player play our video in the App.

(Frame from the movie Pacific Rim, taken from its trailer)

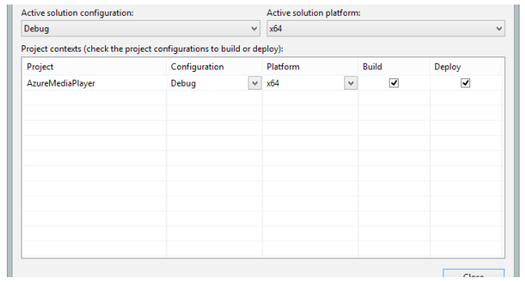
However if you observe closely, you'll note the Progressive download bar is slightly ahead of the current position. That covered the Progressive Download player using PlayerFramework and Windows Azure media service backend. Player Framework's player control makes it trivial to show the video. Now let's see how smooth streaming works.

Adding Smooth Streaming Player

Step 1: Select 'Add Reference' to the previous application and select 'Microsoft Player Framework Adaptive Streaming', 'Microsoft Smooth Streaming Client SDK for Windows 8", 'Microsoft Visual C++ 2013 Runtime Package…'



Step 2: Change build mode from any to x64 because VC++ libraries need to be targeted to a specific architecture. (Select the Dropdown that shows Debug and choose the option 'Change Framework')



Step 3: Now let's add the reference to the MainPage.xaml's namespaces to add reference to the Adaptive Rendering Plugin.

```
xmlns:adaptive="using:Microsoft.PlayerFramework.
Adaptive"
```

Step 4: Update the Player Settings. As we can see here, the SmoothStreaming player adds a few additional features like Signal Strength etc.

We replace the src property with the URL to our encoded asset. As you can see, the URL doesn't end in MP4 but instead ends in /manifest which describes all the available bitrates etc.

```
<mmppf:MediaPlayer
 AutoPlay="True" IsFullScreenVisible="True"
 IsSignalStrengthVisible="True" IsResolutionIndicator
Visible="True"
 Source="http://mediaservicedemo.origin.media-
services.windows.net/b16fb973-8c2b-49d6-a27f-
646058194dbe/pacific_rim_60_seconds.ism/Manifest">
 <mmppf:MediaPlayer.Plugins>
  <adaptive:AdaptivePlugin />
 </mmppf:MediaPlayer.Plugins>
</mmppf:MediaPlayer>
```

Step 5: Run the app.

Woah!!! Smooth Streaming up and running in practically zero code (all markup).

KEEPING AN EYE ON THE COST

Not often do you find realistic costs displayed in service promotions. Well we are not promoting anything, so let's see what it costed us to write the above application

We did the following:

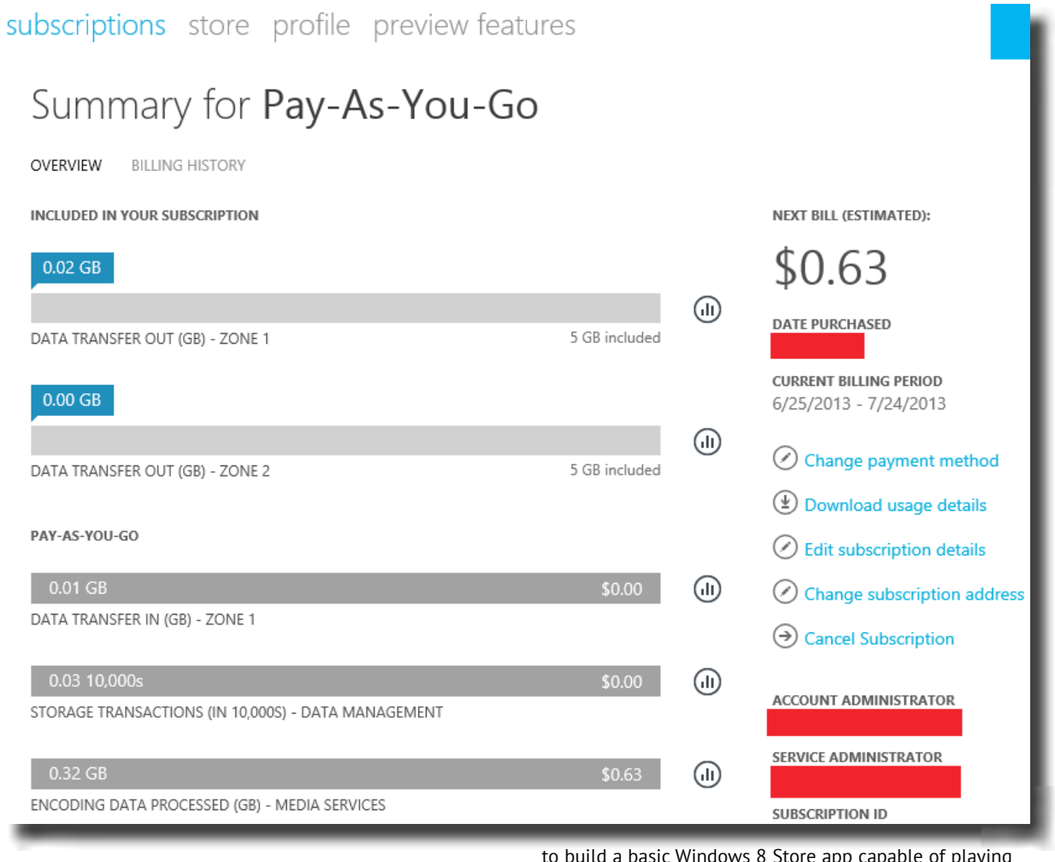
1. Uploaded data IN-to Azure, one time 6MB

2. Ran encoding thrice, two completed one cancelled. I ran the second time because I kept getting Unable to Play Video for the Streaming Player. I initially imagined the encoding to have gone wrong, but that wasn't the case. It was more to do with Internet quality at my end.

3. Viewed the 60 second clip about five or six times.

I have a Pay-as-you-go plan and the total cost incurred was $0.63 or 63 cents. If we look at the chart below closely, you'll see the entire cost is for encoding. Data in and Data out remained in my monthly allowance limits.

I'll have to run a video in loop to see how quickly I run out of my monthly allowances, but up-front, whatever plan you are on, you have a buffer at hand and the only cost you will incur is cost of encoding, which I believe is pretty fair given the minimal effort we had to expend.



MAKING MONEY FOR YOUR VIDEO PUBLICATIONS

By now you would have realized that Azure Media Services is for building services like YouTube/Vimeo or even the Twitter's Vine. So if you are building a media based service, one likely source of income is revenue from Advertisement.

The PlayerFramework has extensive support for inserting Advertisement in between the main video. It supports VMAP, MAST, FreeWheel SmartXML, AdSchedulerPlugin as well as custom Scheduling schemes via plugins. You can read more about how to place Advertisements in a Player Framework media control here.

WRAP UP

With that, we'll wrap up this introduction to Windows Azure Mobile Services. We also got a nice look at the excellent Open Source Player Framework from Microsoft. We were able to build a basic Windows 8 Store app capable of playing Videos from Azure Mobile Services, practically without writing any C# code. We also got a glimpse of the 'initial investment' involved in our exercise ■

Download the entire source code from our GitHub Repository at bit.ly/dncm8-win8mp

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

> Right from the platform (iOS, Android, Windows Phone), you need to choose the right technology and architecture which can help you to build store ready apps in short time!

Building Store Ready Apps for **Windows Phone**

With mobile device proliferation, there is a large number of mobile devices available in the market, exceeding even laptops & desktops. As a developer, creating and listing your app in the Marketplace offers you the opportunity to reach millions of Apps users. But you have to make sure that these apps meet the criteria set by respective stores. Windows Phone MVP Mayur Tendulkar shares some mantras to make Widows Phone Store Ready Apps

In this article, we're going to build an end-to-end application which will meet all windows store requirements so that you can publish it for end-users to download. We'll build a '*MovieQuotes*' application which will show popular quotes from famous movies. We'll keep this application very simple and focus more on getting this application certified on Windows Phone store.

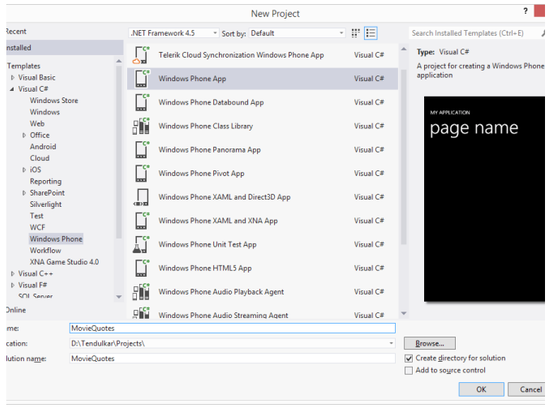
WINDOWS PHONE DEVELOPMENT ENVIRONMENT

To create and test Windows Phone 8 applications, we'll need a Microsoft Windows 8 Pro 64 bit machine with Second Level Address Translation (SLAT) support. More information about this requirement can be found here: http://msdn.microsoft.com/en-US/library/windowsphone/develop/jj863509%28v=vs.105%29.aspx

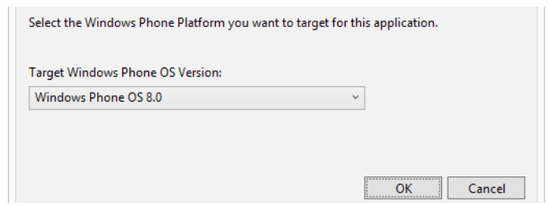
Once this developer machine is ready, we can install Windows Phone 8 SDK. This SDK will provide a Free edition of Microsoft Visual Studio Express for Windows Phone, which can be used to create our 'MovieQuotes' application. If Microsoft Visual Studio 2012 is already installed on the dev machine, this SDK will integrate into it, providing more benefits.

CREATING MOVIEQUOTES APPLICATION

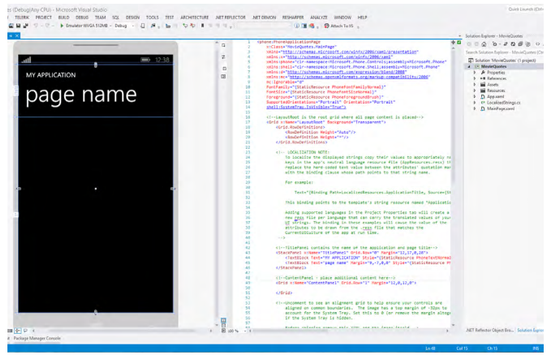
To create 'MovieQuotes' application, start Visual Studio and navigate to File > New > Project. In the next dialog box, select Windows Phone as a project type under C# Language. In Project Template select 'Windows Phone Application'. Give 'MovieQuotes' name to the project. The dialog box will look similar to one shown below:



Once we click on 'OK', Visual Studio will ask us to select Target Windows Phone OS version. We're going to build this application for Windows Phone 8. Select this OS version and click on 'OK' again.

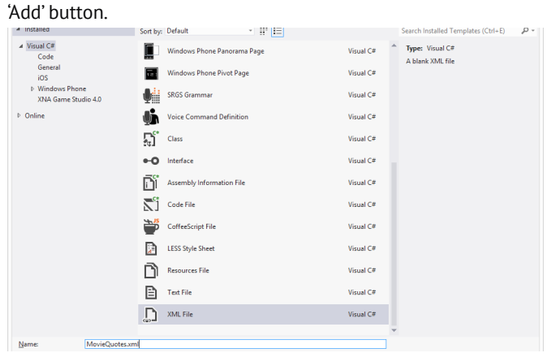


At this time, the project is created and Visual Studio will load this project. Visual Studio will also provide a WYSIWYG interface for creating Windows Phone projects.



MOVIEQUOTES DATA

To keep things simple and focus on the Store Ready aspect of this article, we're going to use an XML file which will hold popular movie quotes. To add this XML file to your project, click on Project Menu > Add New Item to bring up a new dialog box. Select XML file type, name it as 'MovieQuotes.xml' and click on 'Add' button.



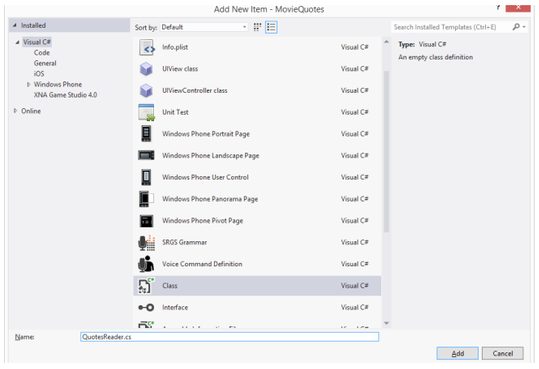
To start with, let's add some records in this file. Type the following code into this file.

```
<?xml version="1.0" encoding="utf-8" ?>
<Quotes>
  <Quote>
    <Text>I'm gonna make him an offer he can't
     refuse.</Text>
    <Movie>The Godfather</Movie>
    <IMDB>http://www.imdb.com/title/tt0068646/</IMDB>
  </Quote>
  <Quote>
    <Image>Godfather.jpg</Image>
    <Text>No Sicilian can refuse any request on his
    daughter's wedding day.</Text>
    <Movie>The Godfather</Movie>
    <IMDB>http://www.imdb.com/title/tt0068646/</IMDB>
```

```
  </Quote>
  <Quote>
    <Text>Up there with the best of the best.</Text>
    <Movie>Top Gun</Movie>
    <IMDB>http://www.imdb.com/title/tt0092099</IMDB>
  </Quote>
  <Quote>
    <Image>Apollo13.jpg</Image>
    <Text>We just put Sir Isaac Newton in the driver's
     seat.</Text>
    <Movie>Apollo 13</Movie>
    <IMDB>http://www.imdb.com/title/tt0112384</IMDB>
  </Quote>
  <Quote>
    <Image>Apollo13.jpg</Image>
    <Text>Houston, we have a problem.</Text>
    <Movie>Apollo 13</Movie>
    <IMDB>http://www.imdb.com/title/tt0112384</IMDB>
  </Quote>
</Quotes>
```

READING QUOTES

Now let's add another class, which can allow us to read this XML file. Click on Project Menu > Add Class. Give it a name as 'QuotesReader.cs' and click on 'Add' button.



In this class, write the following code. To make things simpler, I have added some inline comments in the code.

```
/// <summary>
/// MovieQuotes Class to represent MovieQuote from XML
file
/// </summary>
public class MovieQuote
{
public string Text { get; set; }
public string Movie { get; set; }
public string ImdbUrl { get; set; }
}

/// <summary>
/// Reader Class to read quotes from XML file
/// </summary>
public class QuotesReader
{
public List<MovieQuote> GetMovieQuotes()
        {
            var xdoc = XDocument.Load("MovieQuotes.
                                              xml");
            var quotesList = from x in xdoc.
                                Descendants("Quote")
                select new MovieQuote
                {
                    Text = x.Element("Text").Value,
                    Movie = x.Element("Movie").Value,
                    ImdbUrl = x.Element("IMDB").Value
                };
            return quotesList.ToList();
        }
}
```

```
/// <summary>
/// Extension Class to load Random/Next Quote
/// </summary>
public static class Extenders
{
public static T GetSingleRandom<T>(this IEnumerable<T>
target)
        {
            Random r = new Random(DateTime.Now.
                                          Millisecond);
            int position = r.Next(target.Count<T>());
            return target.ElementAt<T>(position);
        }
}
```

Modifying MainPage

Once the Reader class is ready, let's jump-in and modify MainPage.xaml to show quotes from the Reader. In MainPage.xaml file, locate TitlePanel and modify the code as shown below. This will show the Movie name in the page title.

```
<StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
<TextBlock Text="MovieQuotes" Style="{StaticResource
PhoneTextNormalStyle}" Margin="12,0"/>
<TextBlock Text="{Binding Movie}" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

In the ContentPanel on the same page, add a TextBlock using the following code. This will display movie quote in the content area.

```
<Grid x:Name="ContentPanel" Grid.Row="1"
Margin="12,0,12,0">
<TextBlock Text="{Binding Text}" Style="{StaticResource
PhoneTextExtraLargeStyle}" TextWrapping="Wrap"></
TextBlock>
</Grid>
```

In this page, before Page tag ends, add an ApplicationBar which will hold buttons to show Next & Previous quotes. The code at the end of the page should look like the following:

```
<phone:PhoneApplicationPage.ApplicationBar>
<shell:ApplicationBar>
<shell:ApplicationBarIconButton IconUri="/Assets/back.
png" Text="Next" Click="BackClicked"/>
<shell:ApplicationBarIconButton IconUri="/Assets/next.
png" Text="Next" Click="NextClicked"/>
</shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
</phone:PhoneApplicationPage>
```

This is the only thing which is required to show movie name and quote from reader. Now, let's modify the code behind to 'actually' show quotes on the page.

Modifying MainPage Code Behind

In MainPage.xaml.cs, modify the code as shown below. Again for simplicity, comments have been added.

```
//Private variables to hold movie quotes
private List<MovieQuote> _movieQuotes;
private QuotesReader _reader;
// Constructor
public MainPage()
{
InitializeComponent();
       Loaded += MainPageLoaded;
}
//When page is loaded, read quotes using reader
//And bind it to main page
private void MainPageLoaded(object sender,
RoutedEventArgs e)
{
_reader = new QuotesReader();
       _movieQuotes = _reader.GetMovieQuotes();
       this.DataContext = _movieQuotes.
FirstOrDefault();
}
//Handle previous quote button click
private void BackClicked(object sender, EventArgs e)
{
this.DataContext = _reader.GetMovieQuotes().
GetSingleRandom();
}
//Handle next quote button click
private void NextClicked(object sender, EventArgs e)
{
this.DataContext = _reader.GetMovieQuotes().
GetSingleRandom();
}
```

Test Project

At this point, our application is ready. Now run the project and see the output. Inside emulator, you'll see different quotes.

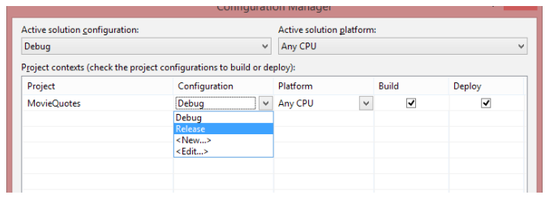
BEFORE PUBLISHING

Now, the 'MovieQuotes' app is complete (from a demo context). However, it is not yet ready to pass certification process. There are few more steps that need to be followed to make sure this app meets certification guidelines and passes all the tests. Now let's visit all these steps and run them over through this application.

01. Build in 'Release' Mode

It is mandatory to build this application in 'Release' mode to submit it to the App Store for download. To change the build mode from 'Debug' to 'Release', click on Build menu >

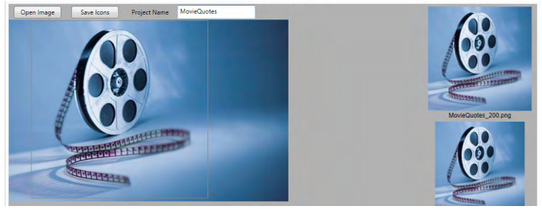
Configuration Manager. In the next dialog box, make sure that the configuration is set to 'Release'



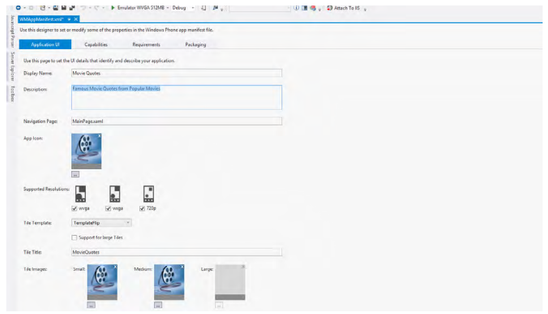
02. Application Icons & Splash Screen

It is mandatory to provide application icons with application. Default Star-Burst icon is unacceptable and application may fail during certification tests. To create application icons, you can use any graphics editing tool like Adobe Photoshop, Illustrator or even Microsoft Paint. I recommend using Windows Phone Icon Maker, which is freely available at codeplex and link is: http://wpiconmaker.codeplex.com/

For Splash Screen, as mentioned earlier, any graphics editing tool can be used. This tool will generate icons of different sizes which is required during application design and submission process.



Once these icons are created and saved, open Project > Properties > WMAppManifest.xml. This will open a tool, which will allow you to associate the icons to the project. Along with icons, do provide a complete description of the application in this tool.



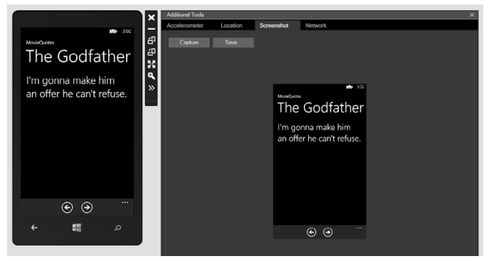
03. Disable Debug Information

During debugging, application shows frame-rate counter and other debug information. This functionality is enabled by default. When app is ready to publish, disable this functionality. To disable, open App.xaml.cs and comment following line of code

```
// Display the current frame rate counters.
//Application.Current.Host.Settings.
EnableFrameRateCounter = true;
```

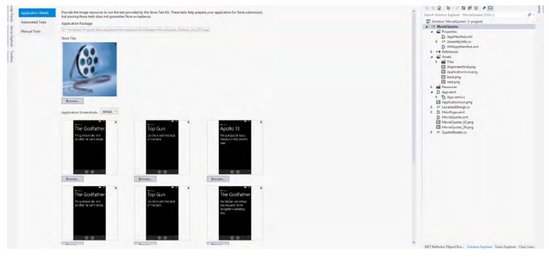
04. Take Screenshots

Screenshots are required during app testing and whilst submitting the app to the store. Windows Phone SDK & Emulator makes it easy to take screenshots with appropriate resolution. Run the project again and this time, use Emulator tool to take the screenshots. Take at least 3 screenshots for further use.

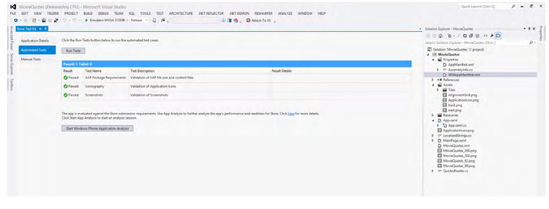


05. Set Test Parameters

Once again make sure that the Build mode is set to Release, all icons are in place and some screenshots are taken. Now click on Project menu > 'Open Store Test Kit'. Associate all images to respective blocks. The window should similar to the following



Don't forget to build the project at this stage. It will associate all information and create a new build for test. Now, click on Automated Test on left pane and click on 'Run Tests' button.



Once, all the tests are run on the project, it will show the test results.

06. Windows Phone Application Analysis

MovieQuotes application doesn't involve lot of processing. Also, the data is stored within the application. If application is using intensive graphics, animations, frequent web service calls or lot of computations, it is always a good idea to run Windows Phone Application Analysis tool on the app.

WHILE PUBLISHING

Once above mentioned steps are followed and test results are passed, this application is ready to submit to the App Store. However, there are few things, which need to be considered while processing submission. Let's visit them here.

01. Provide Correct Information

Even though it is not mandatory, it makes sense to match the application name inside the app with the application name on display. It is also advisable to select specific category for the application. An educational application can be submitted inside education as well as books + reference. So choose the category wisely, where people will actually look-out for the application.

02. Pricing

Developer/Publisher can price the application according to their expectations. However, make sure that W8 form (required by the USA) and other formalities are completed before the app is submitted to the store. Else, it can be a challenge to withdraw/transfer money earned from app sales.

To get more audience, it is always a good idea to provide free/trial or freemium application version. This will give a chance to the audience to test your app and if they like, they can purchase the higher version.

03. Market Distribution

Unless the app is really targeted to certain markets (country regions), be restrictive about publishing to that market. Every region has their own certification requirements. For example, different countries have different rules for content certification, minimum age requirements, localization, etc... If the app is submitted to all the markets, it will take time to certify and sometimes, additional information or changes in the app may be required to meet those certification requirements.

04. Certification Notes

MovieQuotes application doesn't require any authentication. However, there can be some apps which may require authentication. There also can be some apps with special cases like localization on some pages, some unpredictable content served from web service. In this case, this information (may be a test-user account credentials) can be provided under certification notes. These notes can help testers to validate your app and certify it.

05. Legal & Privacy Statement, Support Email Address

It is mandatory for the publisher to provide valid Legal and Privacy statements during submission process. Publisher can host 2 different html files e.g. site.com/legal.html and site.com/privacy.html. For simplicity and in cases where the publisher doesn't have a website hosted, these files can be put on SkyDrive with public read access or on a blog as well.

Along with this information, there must be a support email address provided to submit the app. This can be used by end-users if they need any support. Creating separate Distribution List or Email ID will help.

AFTER SUBMISSION

With these basic points in mind while developing and submitting the app to the store, it is very unlikely that your app will fail the certification. However, if the app fails for some reason, the publisher will receive a detailed error report which can help track-down the exact details of the error for the developer to fix it.

CONCLUSION

With these mantras, I'm quite sure that many of you will find building Windows Phone applications a fun activity and easy to publish on the store. With this, happy coding :) ■

Download the entire source code from our GitHub Repository at bit.ly/dncm8-wpstore

Mayur Tendulkar works as a Local Type Inference (i.e. Consultant) at Zevenseas. He has worked on various Microsoft technologies & is part of Pune and Mumbai User Groups. Since 2008, he has been a speaker at various Microsoft events. You can reach him at mayur.tendulkar@hotmail.com

Scott HUNTER

Dear Readers, we are very excited to talk to the Principal Program Manager of the ASP.NET team, Scott Hunter.

Hello Scott, thanks for joining us! We at DotNetCurry focus on all .NET technologies, but it is no secret our primary audience is the Web Dev community. This makes it all the more exciting to have someone from the ASP.NET team with us.

First up, congratulations on the boat load of new stuff introduced at BUILD 2013. Just when we think we are plateauing, in comes a set of exciting new stuff. We are talking about OWIN and Self Hosting plus things like Artery in Visual Studio and how SignalR is becoming a mini-framework inside .NET itself! But we'll come back to that in a moment.

DNC: Before we launch into a barrage of what's next in ASP.NET, let's step back a little. Tell us a little bit about your journey at Microsoft?

SH: Before joining Microsoft I was an avid user of ASP.NET and I remember when I interviewed, I said I only wanted to interview for the ASP.NET team. I had tried to interview for the team once before and found myself being interviewed by a completely different team which I was not interested in. Once joining I remember starting on a Tuesday and my boss Simon Calvert asking me to write a spec and present it to the team that Friday, after only 2-3 days on the job. There is no real manual or description of your job when you start at Microsoft; you just have to feel your way around the system. Luckily Phil Haack had just joined as well and we figured our way around the system as we built .NET 3.5 SP1 and MVC 1 and coming from the outside, we brought with us a push to be more open with blogs, social networking and finally open source. But it isn't just about me, this is a team effort and it is fun to be surrounded by so many hardworking and talented people that truly care about the products and our customers. It has been fun!

DNC: The ASP.NET team started the trend of being more open at Microsoft with respect to development road map, using external/OSS components as a part of the framework which eventually resulted in Open Sourcing of ASP.NET MVC and Web API. This has permeated into the Azure team as well. Tell us about how this all started? Some of the maybe lesser known battle stories on how you had to convince people at Microsoft to go OSS.

SH: This effort started many years ago when we were developing MVC 1 in 2008. After we released that version, we also released access to the source code with a license that even allowed customers to build their own custom copy of the code. That was the first baby step we took. Next we started shipping third party libraries that we thought were important for our customers to use in the projects such as jQuery, jQuery UI and many more. And then finally in March of 2012 we open sourced MVC, Web Pages and Web API taking contributions. Looking back we just took a small step each time and kept taking another one.

> My favorite backstory is the day before we announced the open sourcing, Scott Hanselman and I were worried someone might call us or email us and tell us they changed their minds. So the night before, we both turned off our phones and email until after we had announced it on stage at DevConnections.

DNC: That was quite an interesting backstory you shared with our readers :) Personally, do you think OSS development model affects profitability?

SH: On the Web team we don't think of OSS in terms of profitability, we think of it in terms of empowering our customers. Now our customers have easy access to the source code to help them understand our product, they have access to our nightly builds if they want to try the latest changes before we release a major update. Also we develop fully in the open with our roadmaps available online at aspnetwebstack.codeplex.com. So to me the real benefit here is a much more transparency to our customers in terms of how we develop our products. And we have gotten some great contributions from our customers as well. Both the attribute routing and CORS support are major features in MVC 5 and Web API 2 that came from the community.

DNC: Scenario for ASP.NET is different where ASP.NET platform was always free on top of Windows Server licenses. However, say there is an increasing demand for Mono compatibility on Linux, do those become tough calls?

SH: We have open sourced MVC, Web Pages, Web API and Entity Framework which allows them to be compiled and run on non-Windows platforms. But our goal will be to make sure *they always run best* on the Windows Platform and we can do that because we own the operating system, the .NET framework and the ASP.NET frameworks. And moving forward one of the easiest ways to run ASP.NET on the latest greatest hardware and operating systems will be through the cloud such as Windows Azure.

DNC: We were listening to your DNR podcast from about a year ago where you mentioned that NodeJs today feels light and barebones, but soon things get tacked on and before you know, it's as good or bad as some other stack it intended to 'better'. With OWIN, it feels like this time we are doing the same thing. So tell us more about the idea behind OWIN, why kind of redo the IIS stack in a more granular way. How do users (more so Enterprise Users) benefit from adopting OWIN standards?

SH: I think there are two directions here. First, when we started building ASP.NET, we put the kitchen sink of features into the platform and they were enabled by default. This was great when the web was young and people were first moving to it. Now that we developers are more experienced, the market has shifted a model where developers opt in just the features they want which can allow us to build a smaller, higher performance stack. Secondly, due to the first point, this made ASP.NET not the greatest platform to build your own web framework on top of. The work in OWIN will now let people, that want to build their own .NET frameworks, to build on top of us without inheriting all of ASP.NET plumbing that they might not want or might have their own replacements.

DNC: Any words of advice to the WCF hardcore fans who are reluctant to move to WebAPI?

SH: My advice here is the same advice I give a Web Forms customer looking at MVC. If the technology you are using is working well for you then continue to use it. Don't switch just because we have shipped a newer thing, switch if there is a value proposition to switch. In the case of Web API one of the big reasons to use it is when you are targeting platforms that don't have rich support for WCF such as iOS or Android.

DNC: That's a real good piece of advice! Could you explain the idea behind 'One ASP.NET'? What are the obvious and not so obvious things pending on the 'One ASP.NET' todo list?

SH: The idea behind ONE ASP.NET is that today our tools force you to make a choice of Web Forms, MVC, Web API at the start and once you make that choice, you might feel that you can't use the other parts of ASP.NET. With One ASP.NET, you just create an ASP.NET application and choose which frameworks you want to use. Once you have created a project you can now easily add other frameworks and we will add the libraries, folder and tooling for those. But this is only the beginning, there will be much more coming after VS 2013.

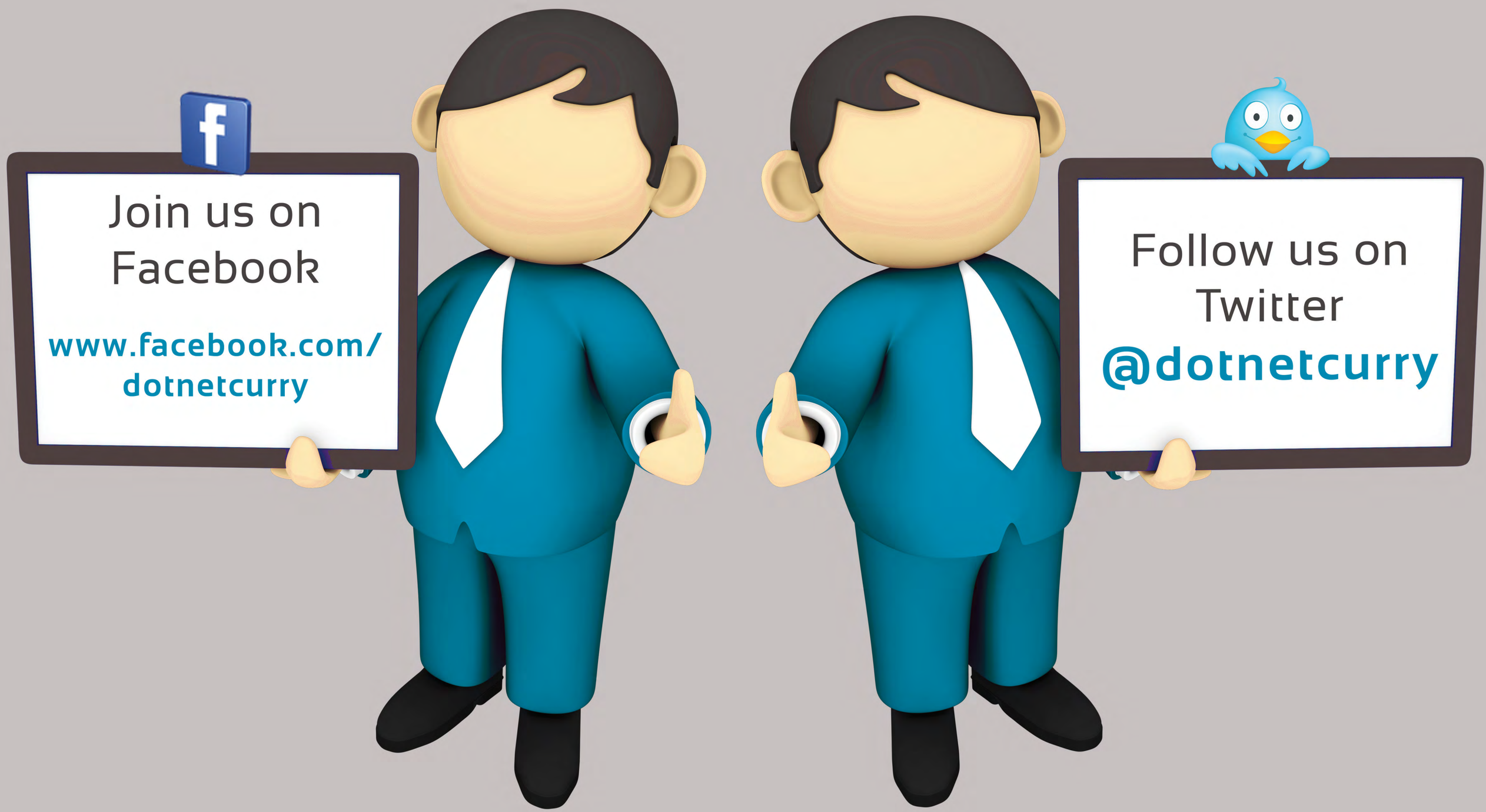
DNC: What are the things that are pending or being worked upon to make ASP.NET a first class HTML5 dev platform?

SH: We think we already have a great tooling story around HTML 5, CSS and JavaScript and it will continue to get better in VS 2013. The area that I think we can grow in is providing a great tooling story for people that want to use Visual Studio to build applications using client frameworks like Angular.js, Ember, Knockout, Backbone and more. In this case I don't think we need new frameworks in ASP.NET but the right tooling to let you easily build a page that uses Angular.js to display some data that is exposed via Web API using Entity Framework. This is an area we plan to invest in the future.

DNC: What's a typical day like for Scott Hunter when he is not working?

SH: I've got two kids aged 9 and 10 and when I'm not working you will probably find me at the soccer field, swimming pool or biking with my kids. I also love hiking and the Seattle area is awesome for this so I try and do that when the weather is permitting. I'm also a fan of American Football and the Seattle Seahawks should have a good year and I love Formula One racing and am an avid Ferrari fan.

DNC: Thanks again for your time Scott, and kudos to you and your team for creating such wonderful products ■

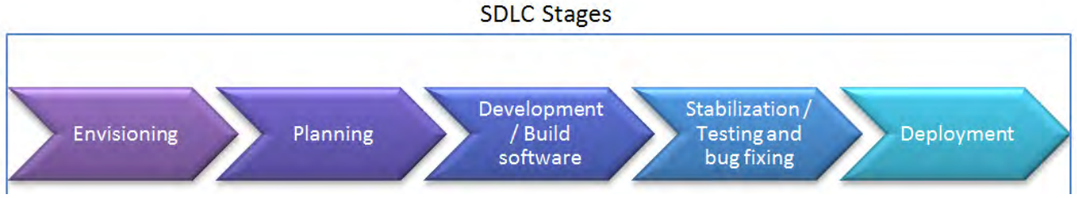

Join us on
Facebook
www.facebook.com/
dotnetcurry
Follow us on
Twitter
@dotnetcurry

AGILE TESTING IN VISUAL STUDIO 2012

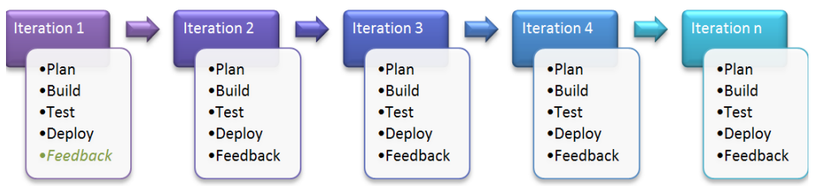
VS ALM MVP **Gouri Sohoni** discusses Agile Testing principles, along with some challenges faced while implementing Agile principles and resolving them using the tools provided as part of Visual Studio 2012

The success rate of a development project which provides a solution, satisfies the customer and is also robust, is very low. Some reasons why a software development project may not be successful are: **(A)** Perception of the team developing software and of customer, about requirements, may defer. This may lead to development of a great software that does not match customer requirements and needs. **(B)** Customer's requirements have changed over time. As the understanding of business processes improve, customer may change the requirements. These changes may become difficult to implement and certainly will increase the time required for implementation.

In both the cases, the customer remains unhappy with software that is built at the end of entire cycle of development. In traditional waterfall model, there are clear cut stages in Software Development Life Cycle (SDLC).



It is always a challenge to incorporate the changing requirements of customers in this process. For this reason, a new approach to software development was introduced in terms of Agile methodology. Agile methodology recommends small chunks of functionality to be built in a short time span (iteration) and then get the feedback from customer, before starting the next iteration. Each of the iterations comprises of the activities as follows



These activities may run in parallel, many-a-times. The order of activities and who does what activity are not important. What is most important is that team builds a demonstrable version of software that can be viewed by customer and customer can give feedback on that. It should have incremental value in comparison to the earlier version. Team members together share the responsibility to create demonstrable software once the requirements to be completed in each iteration (sprint), are frozen. Every iteration involves deploying a workable bit which is tested as a part of the activities of that iteration. Since each iteration has some amount of testing, it is necessary to plan and execute this testing systematically. This testing is what we will call **Agile Testing**.

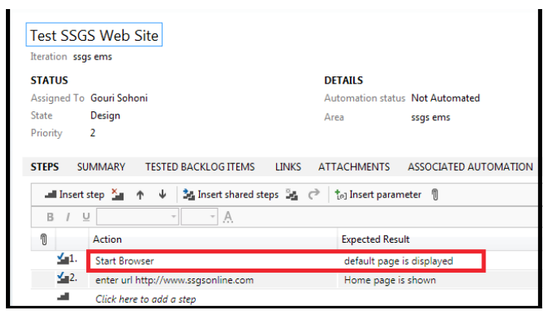
Normally when a team is developing software following the agile methodology, they are applying agile principles. The following list specifies a list of agile principles, challenges faced in implementing these principles in terms of testing and the kind of practice from agile testing that can be applied to address it.

AGILE PRINCIPLES	CHALLENGES FACED	AGILE TESTING TO BE APPLIED
Satisfy customer through early and continuous delivery	Meeting Customer's requirements	Continuous Testing
Customer's requirements are always welcome	Always keep customers in loop	TDD (Test Driven Development), Acceptance criteria with Test Cases
Stakeholders and team members must work together daily	Testers involvement early in the life cycle of software development	Test Plan, Test Suite
Working software is frequently delivered	Deliver tested software at the end of each iteration	Continuous Integration, Report bug in shorter time
Face to face interaction	Understand spec changes faster	Ensure testers in complete life cycle
Team members always look for being more effective	How do I do regression testing?	Find regressions earlier

I will now discuss how each of these challenges can be addressed by various tools provided as part of Visual Studio 2012.

HOW DO I TAKE CARE OF CUSTOMER'S REQUIREMENT

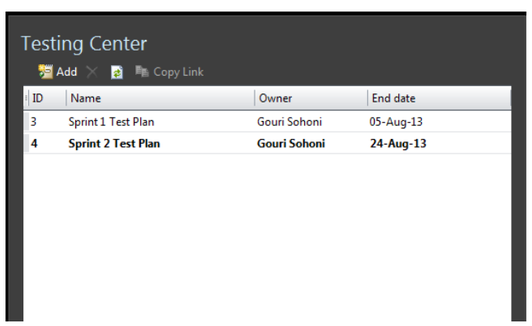
We can involve customers while creation of test cases. With Microsoft Team Foundation Server 2012, management of Test cases is taken care of with the help of the IDE called Microsoft Test Manager 2012 (MTM). MTM provides IDE for creating and executing test cases. Test cases can provide acceptance criteria in the form of expected results. A Test Case can be linked to a requirement. A Requirement can be tested by one or more test cases.



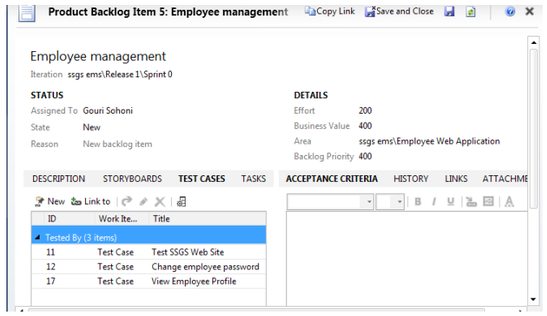
TDD or Test Drive Development is also one of the ways of modeling customer's requirements. In this process, the test will be first defined, executed and later the functionality to satisfy test is written. In Visual Studio 2012, a developer can follow the approach of TDD where he/she writes the unit test first and later writes the code to satisfy that test.

HOW DO I ENSURE TESTER'S INVOLVEMENT

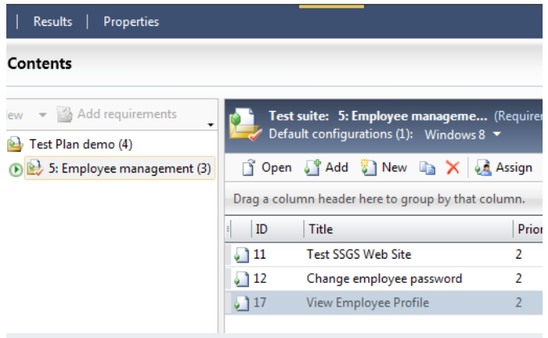
As there are no boundaries of stages while working with Agile Methodology, Agile Testing will provide tester's involvement in the development cycle from beginning of iteration. A Tester has to define what will be tested in the iteration and which test cases will be run in that iteration. This definition is documented in a Test Plan which is linked to that iteration. Microsoft Test Manager 2012 provides Test Plans which are associated to Team Projects. The Team Projects can be based on process templates provided by Team Foundation Server in terms on Agile, Scrum or CMMI. Each Team Project can have a number of Test Plans associated. Each iteration may have one Test Plan. The following diagram shows 2 test plans available with current team project.



Each Test Plan will act as a container for Test Suites made up of test cases. These Test Suites can be requirement based, static or query based. Creating a requirement based suite will automatically fetch all the test cases which are associated with the requirement.



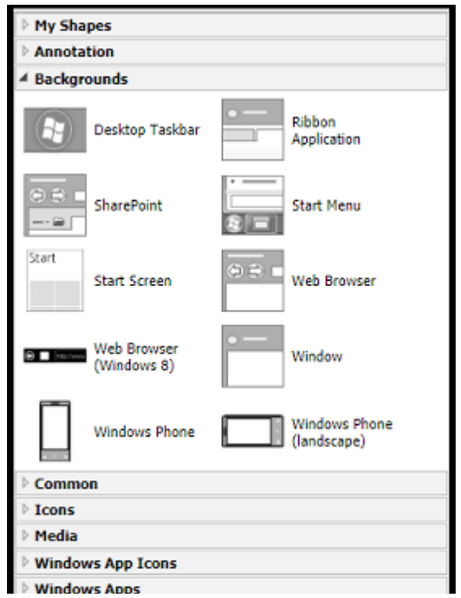
A PBI has three test cases associated with it. If I create a requirement based test suite with this PBI, then it will start showing all the 3 test cases in the plan tab of MTM.



HOW DO I KEEP CUSTOMERS ALWAYS IN LOOP WHILE DEVELOPING THE SOFTWARE?

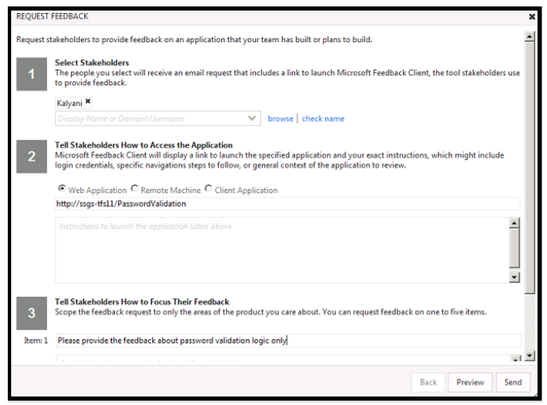
Customers should know and stay updated with the look and feel of the software project. At the same time, customer should also be able to view the software and its functionality before shipping.

• Visual Studio 2012 provides storyboard feature which will give an idea of the look of an application, to the customer. With requirement as work item, there is a provision of connecting to a storyboard which can be created with the help of PowerPoint. There are a lot of shapes and icons provided with PowerPoint which support in creating a storyboard. Following diagram shows some of them.



The customer can provide inputs about the workflows in the software based upon the storyboard that is shared with the customer representative.

• After the software is ready for demonstration, team creates a feedback request which comprises of all the information, as to what is to be checked, how the application is going to execute etc. An email is sent to the customer mentioning those guidelines. The customers can install free Feedback client tool on his/ her machine and provide the feedback that will be recorded in feedback response work item.



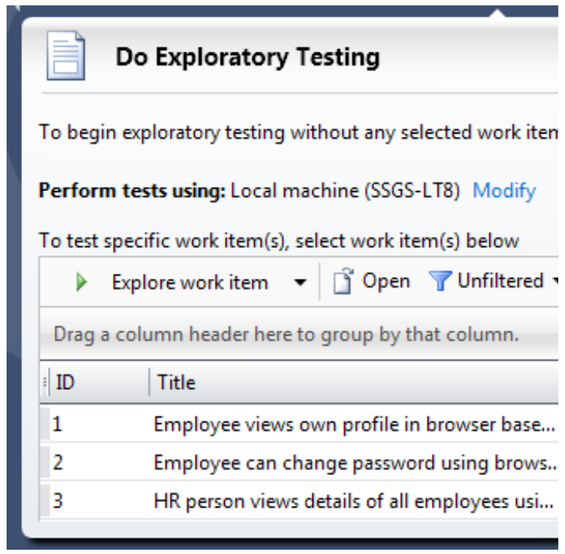
Customer can provide feedback in terms of audio, video or screenshots.

• There is a concept of creating a Work Item View only for customers. With this view, the customers will be able to view various work items, but cannot update it.

HOW DO I TAKE CARE OF EXPLORATORY TESTING DURING DEVELOPMENT?

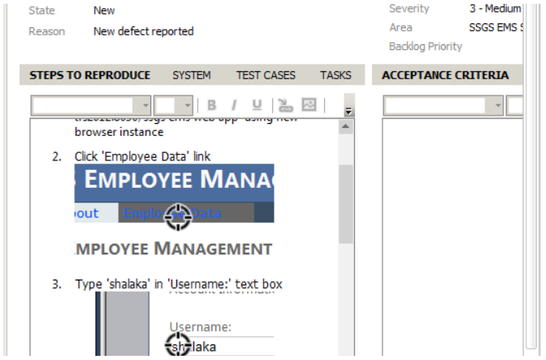
Normally testing is carried out by using pre-defined test cases. These test cases provide detailed information about each step to be executed and the result expected from it. Sometimes these set test cases are not available. If you have sufficient domain knowledge, then you can explore the application without going through fixed steps and still find out about any flaws if they exist. Visual Studio 2012 provides exploratory testing feature with a lot of in built facilities.

Exploratory testing does not require test case for execution. You can either explore the application without any association to work item or explore an existing work item. While doing so, actions can be recorded with some comments or video or audio information about exploration can also be provided. Later you can decide whether to create a bug or a test case based on the actions recorded during exploration.

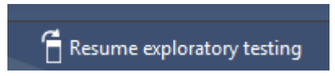


Explore with option can capture a lot of information behind the scenes which helps in creating a rich bug. A rich bug is the bug which provides a lot of data about itself.

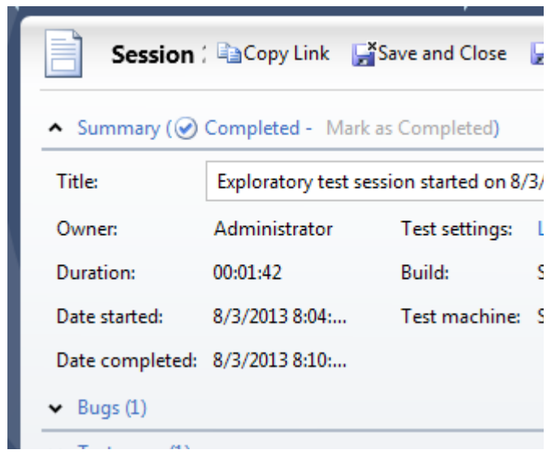
Following bug shows exploration actions and we can create test case along with the bug as shown below:



If the application exploration is going to take a lot of time and I have to pause in between, I can resume the application exploration at a later stage. Note that I cannot close the MTM instance running.

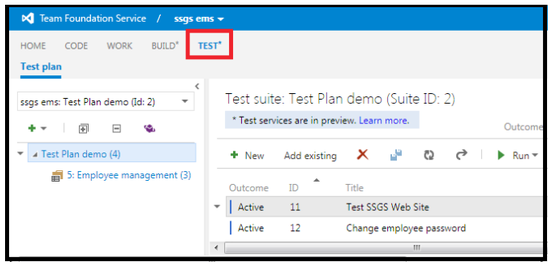


With exploratory session result, I can observe that I have created a bug as well as a test case. This test case will now have pre-defined steps which can be modified to suit my acceptance criteria.



I started exploring application without any defined set of test cases, but after the exploration is over, I can have a test case with actions and expected results. This test case can then be used for testing by a tester who may not have sufficient domain knowledge.

The lightweight tool for testing is provided with the Test hub as a part of Web Access. It is available from Team Foundation Server 2012 Update 2 onwards.



HOW DO I TAKE CARE OF AUTOMATED TESTING?

It is required to test the software to find out if the bugs have been actually fixed and also to ensure that no other functionality has failed because of bug fixes and newly added features. We need to test previously executed tests. Using Microsoft Test Manager 2012, we can create action recordings of a test case while manually executing. These recorded actions can be used to automate the test cases.

Visual Studio 2012 also provides various automated test types namely Unit Tests, Web Performance Tests, Load Tests and Coded UI Tests. Out of these types, normally unit tests, web tests (which are also functionality tests) are mainly executed by developers or programmers to check the functionality. Once functionality testing is over, load testing can be conducted. Visual Studio 2012 provides a unique way of testing UI in terms of Coded UI Tests (CUIT). CUIT can be created in 2 ways. In the first option, we can use the actions we have already recorded while conducting manual testing using MTM. The other option is to record actions and add assertions from Visual Studio 2012 with the help of Coded UI Test Builder (CUIT Builder). With CUIT, we can also track the quality of the output provided.

The code generated is with the help of .NET framework supported languages like C# as shown here.

```
public void CUITWebFunc()
{
    this.UIMap.startbrowser();
    this.UIMap.enterurlhttpssgsvstfs20128090ssgsemsweb();
    this.UIMap.LoginDetailsParams.UIUsernameEditText = TestContext.DataRow["Userna
    this.UIMap.LoginDetailsParams.UIPasswordEditPassword = Playback.EncryptText(Te
    this.UIMap.LoginDetails();
    this.UIMap.clickonGetBasicData();
    this.UIMap.ClickonGetExtendedData();
    this.UIMap.ClickonLogout();
}
```

As the code generated is in a high level language like C#, it can be easily modified as per the needs.

SUMMARY

With this article, I have put forth the concept of Agile Testing. While applying Agile methodology, the testing done is termed as Agile Testing. Visual Studio 2012 provides tools with which a lightweight testing feature is available as Test tab with web Access. We also discussed how testing involves the whole team and that customers play a very important role in the whole development life cycle. We also discussed how exploratory testing is one of the useful feature provided with Visual Studio 2012. Using automated tests with Visual Studio 2012 will lead to reliable automation which will further result into continuous quality of the software produced ■

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-gso

Web API, Async and Performance in an ASP.NET MVC application

> Async tasks can be compared to waiters in a restaurant
> - Rowan Miller

Async and performance are often considered to go hand in hand because performance is bracketed with scaling. ASP.NET Architecture MVP *Suprotim Agarwal* explores the differences and shows how to best leverage async operations to gain performance and build a Web Application that scales better.

With the multifold increase in processor performance as well as number of processor cores in a system, software systems are expected to scale horizontally with system hardware. However, software systems have the tough job of working with disparate systems that work at their own speeds (e.g. Network latency, Disk latencies, peripheral device latency and so on). So if we are building a system that does things synchronously, our throughput is always going to be limited by the slowest system in the chain.

Thus, the ability of our software to initiate a task and do something else till that task completes, goes a long way to ensure we don't 'bottleneck' or 'wait doing nothing' on a slowly running component.

Rowan Miller had an excellent analogy of Async tasks to waiters at a restaurant, in his TechEd NA (2013) talk. To paraphrase – A system working asynchronously is like a waiter at a restaurant. More often than not, a waiter serving a table will be at Table 1, take an order, explain a menu item, deliver an order and then move away, free to do the same at Table 2. When Table 1 is done (deciding the order, requiring a refill, requesting the check), they will draw the waiter's attention and the waiter would come back to Table 1 as soon as they become available (or immediately if they are available). This mechanism of a waiter serving multiple tables ensures that you don't need as many waiters as tables to maintain optimal performance (in case of restaurant – experience).

If we consider a computation unit (CPU + Coprocessors + Cache + System Bus etc.) to be a waiter serving customers, then async operations is the way to make sure that they don't waste time waiting while the customer decides what to order.

Enough analogies, let's see some real code.

THE SAMPLE APPLICATION

Let's say we have an application whose home page is split into four parts showing Emails, Tasks, Bookmarks and Notes. Ideally these would probably come from different feeds from disparate systems, but to keep things simple, we'll get them from four tables in a database. We've filled up the tables with some sample data that I generated.

This demo uses Visual Studio 2013 Preview, but you can use VS 2012 as well. For the performance testing pieces, you'll need the Ultimate SKU, or you can use the free WCAT tool as well.

Since I want to focus on the performance bit, I will not do a step-by-step walkthrough of how to build the application. You can directly get the code and try it out. However this is how the application is laid out

The Web API services App

The Web API services app is using the latest Web API binaries. If you are using Visual Studio 2012, you can create a Web API application and upgrade to Web API preview version using the –pre parameter in the Package Manager Console.

If you are using Visual Studio 2013 Preview, you are already setup with the latest preview binaries for Web API 2.0.

Once the project is setup, we use the following command to install the latest version of Entity Framework (which is version 6 beta at the time of writing).

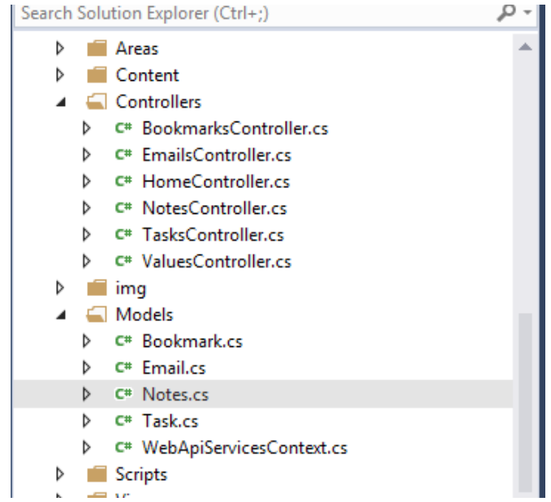
```
PM> install-package EntityFramework –pre
```

Next we add the four entities that will serve as our data source – Email, Task, Bookmark and Note, each with the following properties:

```
public class Email
{
    public int Id { get; set; }
    public string Mail { get; set; }
    public string From { get; set; }
    public string To { get; set; }
    public string Title { get; set; }
}

public class MyTask
{
    public int Id { get; set; }
    public string Starts { get; set; }
    public string Ends { get; set; }
    public string Title { get; set; }
    public string Details { get; set; }
}

public class Bookmark
{
    public int Id { get; set; }
    public string Url { get; set; }
    public string Title { get; set; }
}

public class Note
{
    public int Id { get; set; }
    public string Content { get; set; }
}
```

Finally we scaffold Entity Framework based Web API Controllers for each of the above entities, giving us the APIs required to do CRUD operations for these entities.

Our solution structure looks like the following:



The Get APIs for each controller is as follows:

```
public IEnumerable<Bookmark> GetBookmark()
{
 return db.Bookmarks.AsEnumerable();
}
public IEnumerable<Email> GetEmail()
{
 return db.Emails.AsEnumerable();
}
public IEnumerable<Note> GetNotes()
{
 return db.Notes.AsEnumerable<Note>();
}
public IEnumerable<MyTask> GetTask()
{
```

```
 return db.Tasks.AsEnumerable();
}
```

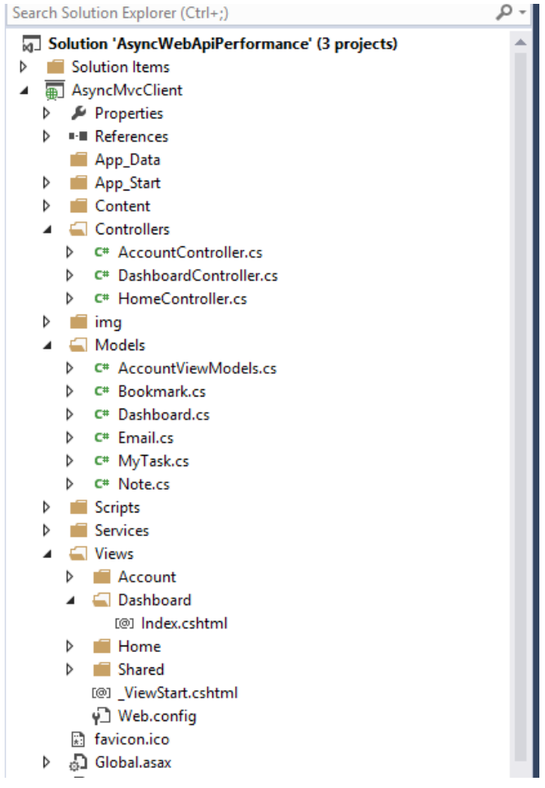
Instead of entering 100 rows of data manually, I used the GenerateData.com site to generate INSERT queries for the dummy data. They are included in the source code of the app.

The MVC Client App

In the same solution, we can add an ASP.NET MVC Client app that will use these web services. Now using Web Service in ASP.NET MVC app is trivial if you use AJAX to query service directly. But that would require CORS and other concerns like authentication etc. To keep things simple for us, we will instantiate a WebClient object and query each Web Service directly. The Client app will use its own proxy entities and return one Model object that we will bind in our MVC View.

Now this may not be the most elegant way to consume Web Services but it will help us identify performance changes as we go from synchronous to asynchronous implementation.

Overall we end up with the following classes in our Client App.



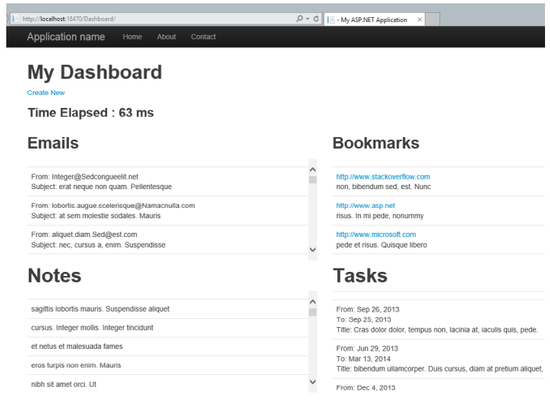
The DashboardController collates all the data from the WebAPI in the Index GET action method and returns an instance of a Dashboard Model to the Index view.

The Synchronous Application

The synchronous version of the Dashboard controller is as follows:

```
public ActionResult Index()
{
 Stopwatch timer = Stopwatch.StartNew();
 timer.Start();
 var feeds = GetFeeds();
 timer.Stop();
 feeds.TimeTaken = timer.ElapsedMilliseconds;
 return View(feeds);
}

public Dashboard GetFeeds()
{
 string emails = new
  WebClient().DownloadString("http://localhost:18545/
   api/Emails");
 string myTasks = new
  WebClient().DownloadString("http://localhost:18545/
   api/Tasks");
 string notes = new
  WebClient().DownloadString("http://localhost:18545/
   api/Notes");
 string bookmarks = new
  WebClient().DownloadString("http://localhost:18545/
   api/Bookmarks");
 Dashboard dash = new Dashboard();
 dash.Emails = Deserialize<Email>(emails);
 dash.Bookmarks = Deserialize<Bookmark>(bookmarks);
 dash.Notes = Deserialize<Note>(notes);
 dash.Tasks = Deserialize<MyTask>(myTasks);
 return dash;
}
```

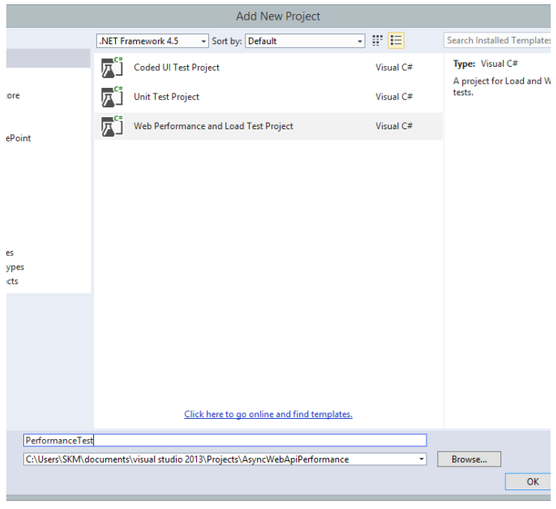
The Index.cshtml for the Dashboard is marked up such that it splits the screen into 4 parts each showing the Email, Bookmarks, Notes and Tasks. We also have a stopwatch that keeps track of how much time it takes to fetch the data on the page.



SETTING UP A PERFORMANCE TEST PROJECT

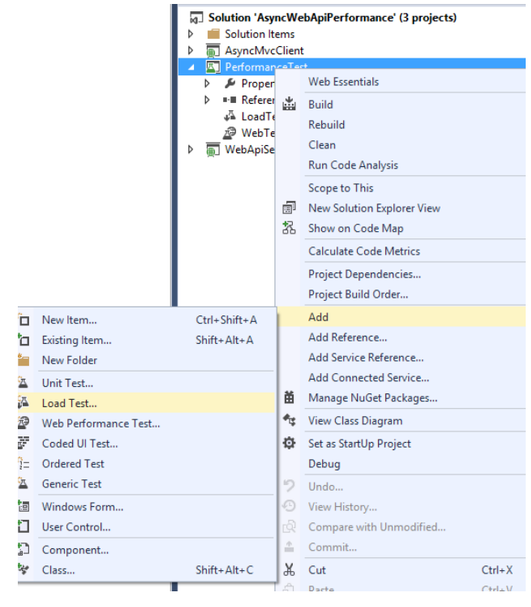
To positively quantify the change in performance characteristics, a simple timer like we have above is often misleading because going async almost never gives you an improved page load time on the client. However what it does give you is better page response on higher load. The only way to load test is use a load testing tool. Today we'll use a Performance Test project in Visual Studio and setup a load of concurrent users. The key characteristics to watch out for are the *Average Page Time* and the *Count of page hits* while the test was running. Lets setup the project first.

Step 1: Add a new Project and select the Test project template "Web Performance and Load Test Project".

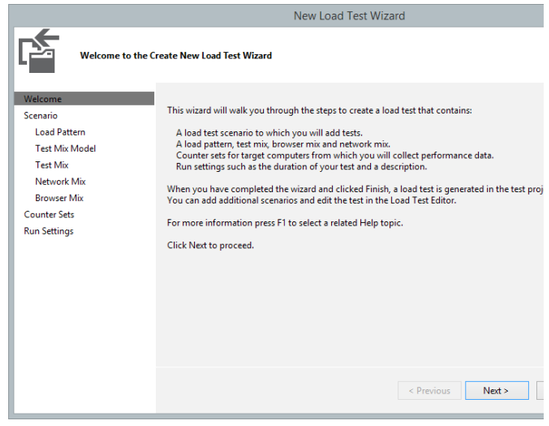


This will add a WebTest to the project automatically.

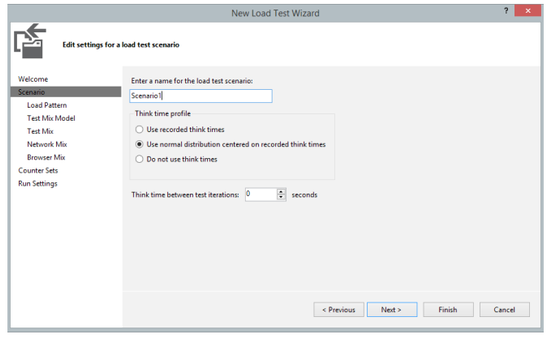
Step 2: Add a new Load Test to the project.



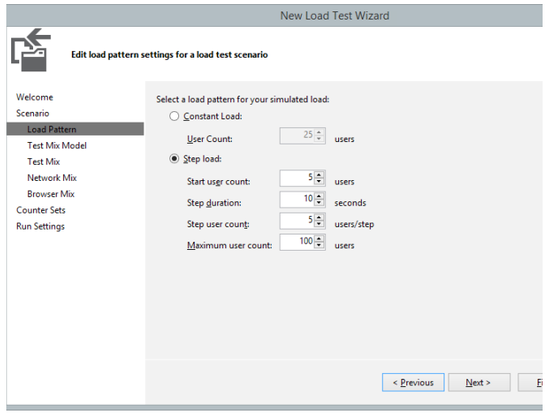
This will launch a new Load Test wizard.



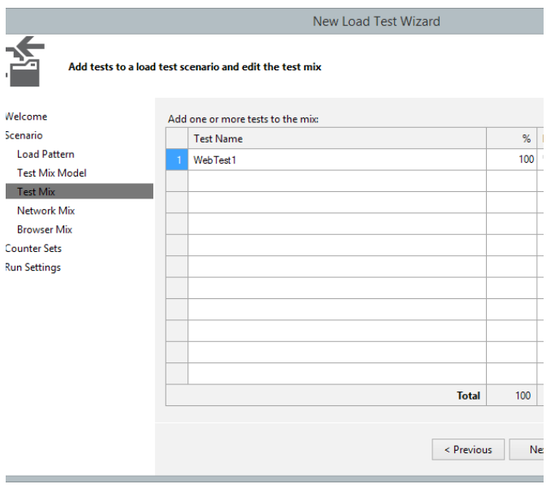
Click "Next" to setup the Scenario, you can keep the default here



The "Next" page sets up the Load pattern. You can use the following settings but I had to tune it down to 10 maximum users, Start User count of 1 and Step User Count of 1. This completely depends on how far you can push your host system based on its hardware configuration. I hit the CPU and Memory thresholds pretty quickly as you will see in a bit.

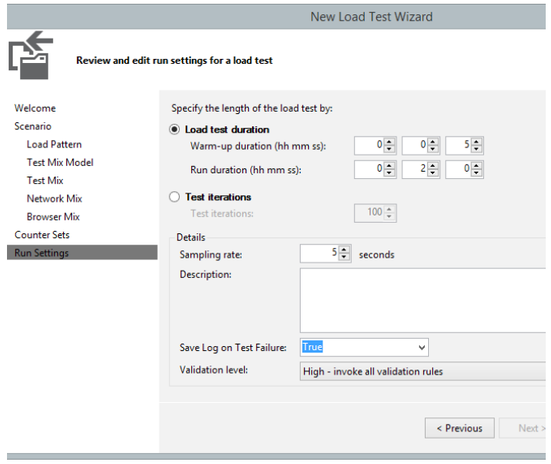


Once the load is setup, next step is to setup the Test Mix Model. Here we've to add the WebTest1.webtest to the mix.



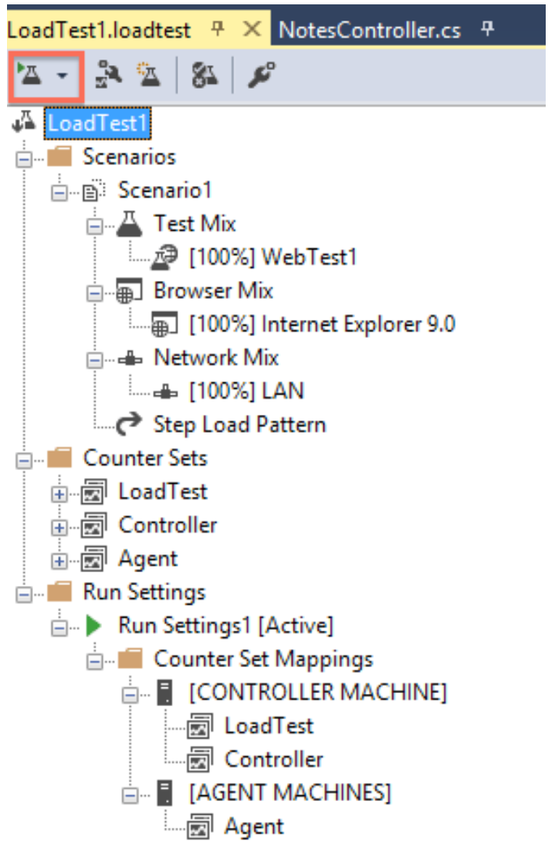
You can keep the defaults for the Network Mix, Browser Mix and Counter Sets. For the Run Settings, change the defaults to the following:

Load Test Duration, Warm-up duration: 20 seconds
Run Duration: 1 minute
Sampling Rate: 5 seconds



Click Finish to finish setting up the Load Test.

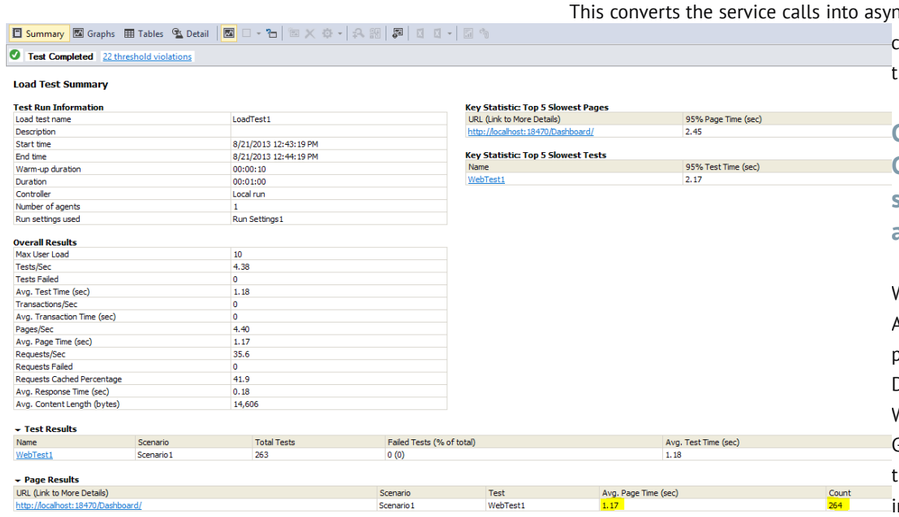
Step 3: Double click on the LoadTest1.loadtest file to open the Test configuration.



Step 4: Hit Ctrl+F5 to run the application without the Debugger (running tests with Debugger gives far worse results so start without Debugger).

Once IIS Express starts, start TaskManager and wait for the CPU spike to settle down.

Step 5: Now click on the 'Run Load Test' button to start the load test. Remember our code is synchronous at the moment. Once the test completes, you'll get a test Summary as follows:



The important figures as I mentioned earlier are highlighted above. As we can see, *Average Page Time* is 3.23 seconds and we were able to serve up 79 pages in the 1 minute that the test ran (with progressively increasing loads from 1 user to 10 users). This gives us a baseline. Now lets' convert our entire call stack to Asynchronous code and run the load test again to see what kind of performance we gain.

CONVERTING TO AN ASYNCHRONOUS IMPLEMENTATION

We will convert our Web API Services and EF DB calls to async first. EF6 has provided Async counterparts to all calls that return Data (not the Query). The updated Api Controllers are as follows.

```
public async Task<IEnumerable<Note>> GetNotes(){
 return await db.Notes.ToListAsync<Note>();
}

public async Task<IEnumerable<MyTask>> GetTask(){
 return await db.Tasks.ToListAsync<MyTask>();
}
```

```
public async Task<IEnumerable<Bookmark>> GetBookmark(){
 return await db.Bookmarks.ToListAsync<Bookmark>();
}
public async Task<IEnumerable<Email>> GetEmail(){
 return await db.Emails.ToListAsync<Email>();
}
```

As we can see, we are awaiting the ToListAsync calls and we have annotated our Controller methods as async and returning a Task of the Enumerable instead of the Enumerable itself. This converts the service calls into async but what about the client? It is still calling the service synchronously.

Converting the MVC Client to call the services asynchronously

We switch to our AsyncMvcClient project and get to the DashboardController. We first update the GetFeeds method to use the HttpClient() object instead of the WebClient object. The HttpClient has only asynchronous APIs, so we call the GetStringAsync API.

The keen eyed would note we are not awaiting the calls, so we are getting a Task<string> from the GetStringAsync method. Next we have an await for Task.WhenAll(…) and pass all the Task instances to it. What this does is, it retrieves all the Tasks and kicks them off together and waits for all of them to return. Once they all return, we create the Dashboard object and return it to the Action Method.

```
public async Task<Dashboard> GetFeeds(){
    var emails = new HttpClient().
GetStringAsync("http://localhost:18545/api/Emails");
    var myTasks = new HttpClient().
GetStringAsync("http://localhost:18545/api/Tasks");
    var notes = new HttpClient().
GetStringAsync("http://localhost:18545/api/Notes");
    var bookmarks = new HttpClient().
GetStringAsync("http://localhost:18545/api/Bookmarks");
    await Task.WhenAll(emails, myTasks, notes,
bookmarks);
    Dashboard dash = new Dashboard();
```

```
    dash.Emails = Deserialize<Email>(emails.Result);
    dash.Bookmarks = Deserialize<Bookmark>(
      bookmarks.Result);
    dash.Notes = Deserialize<Note>(notes.Result);
    dash.Tasks = Deserialize<MyTask>(myTasks.Result);
    return dash;
}
```

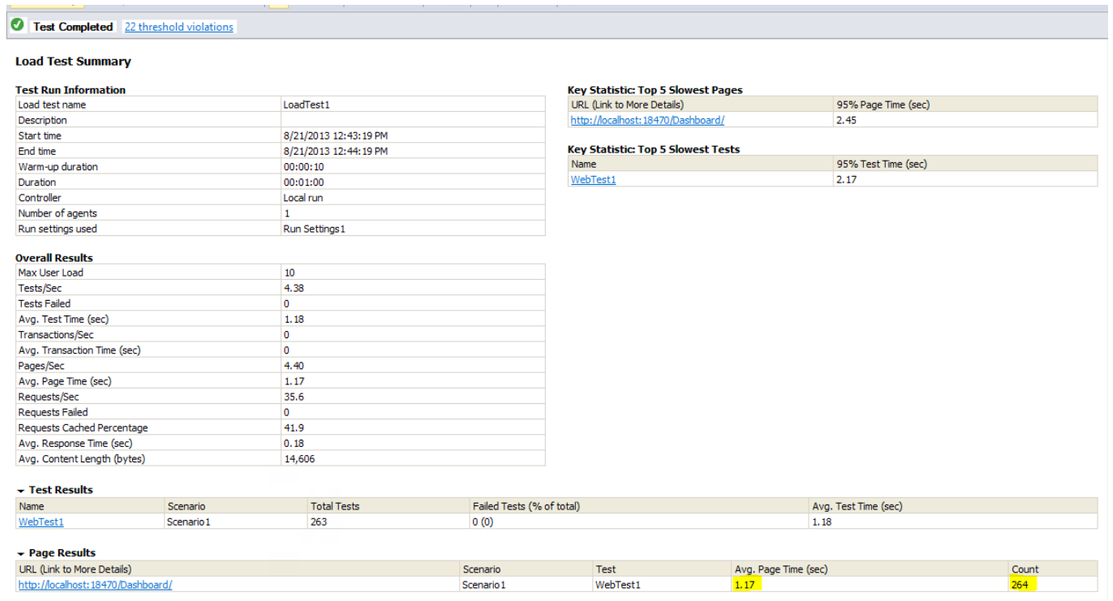
We update the Index action method to be called async as well, and it now returns a Task<ActionResult> instead of ActionResult.

```
public async Task<ActionResult> Index(){
    //Services.FeedService feedService = new
     Services.FeedService();
    Stopwatch timer = Stopwatch.StartNew();
    timer.Start();
    var feeds = await GetFeeds();
    timer.Stop();
    feeds.TimeTaken = timer.ElapsedMilliseconds;
    return View(feeds);
}
```

With that, we've converted our application to Async all throughout. Moment of truth, Time to test the outcome.

Load Testing Asynchronous implementation

Do a Clean Build and hit Ctrl + F5 to run the application without debugger. Open the LoadTest1.loadtest file and Run the Load Test. Check the screenshot.



Test Completed 22 threshold violations

Load Test Summary

Test Run Information

Load test name	LoadTest1
Description	
Start time	8/21/2013 12:43:19 PM
End time	8/21/2013 12:44:19 PM
Warm-up duration	00:00:10
Duration	00:01:00
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Key Statistic: Top 5 Slowest Pages

URL (Link to More Details)	95% Page Time (sec)
http://localhost:18470/Dashboard/	2.45

Key Statistic: Top 5 Slowest Tests

Name	95% Test Time (sec)
WebTest1	2.17

Overall Results

Max User Load	10
Tests/Sec	4.38
Tests Failed	0
Avg. Test Time (sec)	1.18
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	4.40
Avg. Page Time (sec)	1.17
Requests/Sec	35.6
Requests Failed	0
Requests Cached Percentage	41.9
Avg. Response Time (sec)	0.18
Avg. Content Length (bytes)	14,606

Test Results

Name	Scenario	Total Tests	Failed Tests (% of total)	Avg. Test Time (sec)
WebTest1	Scenario1	263	0 (0)	1.18

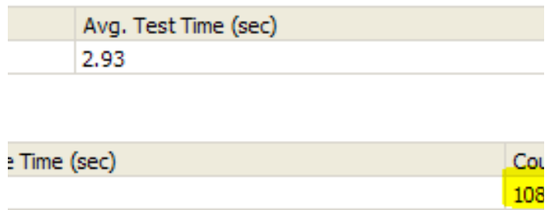
Page Results

URL (Link to More Details)	Scenario	Test	Avg. Page Time (sec)	Count
http://localhost:18470/Dashboard/	Scenario1	WebTest1	1.17	264

As we can see, the Average Page Time is 1.17 and the Page Count is 254 - almost threefold increase if the number of pages hit 1/3 the Average Page time. This validates our belief, but there is more than meets the eye here.

Understanding Performance Changes

The improved scaling of the app resulting in the higher Count of requests handled is understandable, but how did the page time drop? Well, remember the bunching up of Tasks and then firing them off with the Tasks.WhenAll? Yup, that's where I cheated by launching Async tasks in parallel (more threads so BEWARE). If I awaited each request to the web service, I would still get improved performance, but the Average Page Time would be similar to Synchronous operation as the following image shows (Page time of 2.92 and Request Count of 108).



	Avg. Test Time (sec)
	2.93

e Time (sec)	Cou
	108

So yes, Async and Parallel Async are different and affects scaling and performance differently.

The Final Confession (I cheated, but only to prove a point)

Performance testing is tough, performance testing on a restricted VM is even tougher. To be honest, I couldn't find enough performance difference for the Four Database calls (there is a caveat regarding database calls and async but that's coming in a minute), hence to get some visible difference, I put a Thread.Sleep(500) for the synchronous calls and Task.Wait(500) for the asynchronous calls. Hence you see the 2 second+ Page times. Apart from making the performance difference obvious, I believe it doesn't affect the conclusions.

- Synchronous calls to 4 services takes 2+ seconds and only 70 odd Page Requests were serviced
- Asynchronous calls with await for each service calls still takes 2+ seconds but handles 100+ Page requests.
- Asynchronous calls fired off in parallel take < 2 seconds (1.17 seconds) because there are 2 cores in action, so effectively the 4 requests were handled by two parallel threads reducing the page load time and handling more Page Requests.

The Final Set of Caveats

Okay, so async and multi-threading usually gets all hairy and controversial if the caveats are not called out. I already called out one where the Task.WhenAll was potentially spinning up more than one physical thread to service one request. This can get dangerous and lead to Threadpool starvation in case of very high load.

Next, by making your Service calls async, you are moving the performance bottleneck from the Web Tier to the DB Tier (in our specific example). This could come back to bite you with increased DB server load. So use async on DB server cautiously. However, if your services are going out to the Internet to fetch data (like from RSS feeds) you are good, because in that case, you will be firing more requests to an external service (which may be subject to its own rate limits) and won't be killing any of your internal systems. In a nutshell, if the process is really long running and not stressing out some other part of your architecture, async is a good candidate to improving scaling of your web tier.

CONCLUSION

With that I conclude this article. Hopefully you've had a few takeaways from the demonstration, primarily:

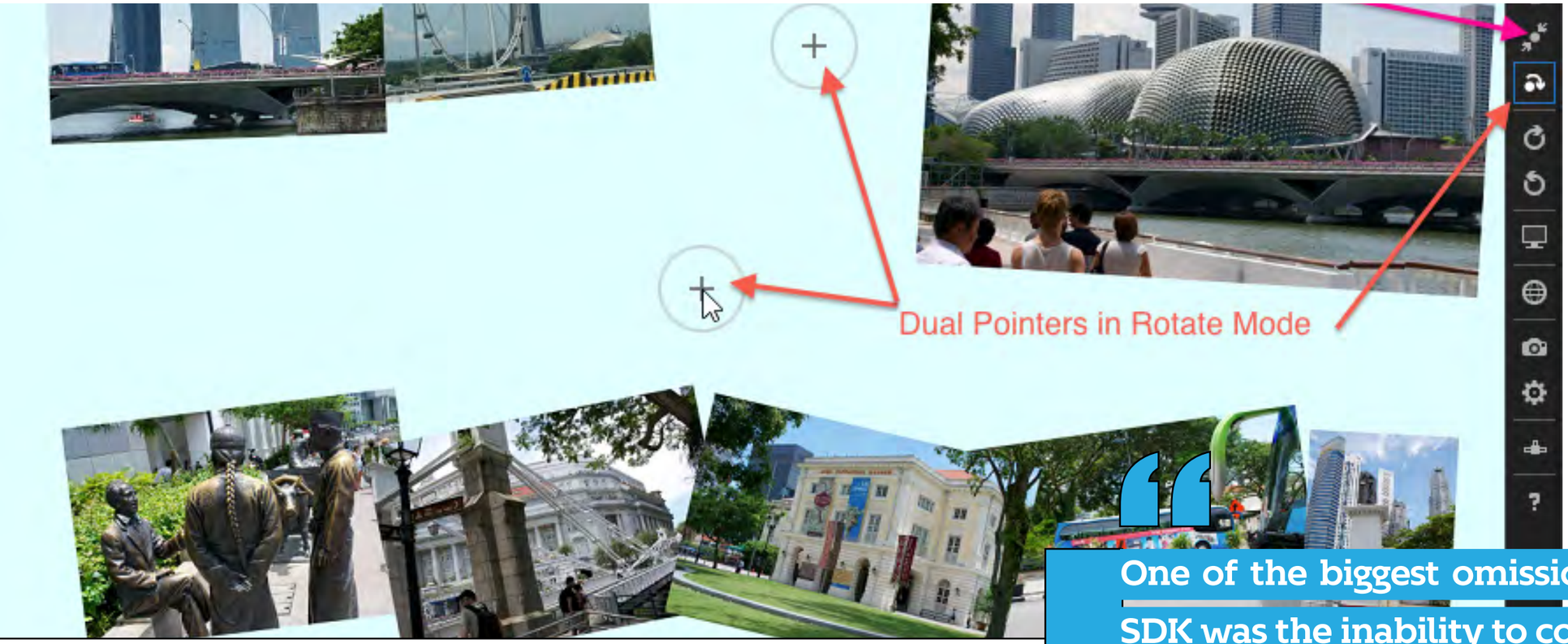
- How to use aync APIs end to end starting with MVC Controller right down to Database calls via Async Web API services

- How to create a LoadTest project and fire load tests at your application

- Difference between scaling and performance gains when using Async ■

Download the entire source code from our GitHub Repository at bit.ly/dncm8-webapiperf

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 read about his latest book at www.jquerycookbook.com



Create a Picture Collage Maker Using

WinRT 8.1

Sumit Maitra demonstrates an application that allows us to arrange pictures on a Canvas to create a collage and then save the control stack as a Bitmap, using the new RenderToBitmap API in Windows 8.1.

One of the biggest omissions in Windows RT XAML SDK was the inability to convert a controls stack into a Bitmap Image. The upcoming Windows 8.1 fixes this by providing us with the ***RenderToBitmap*** API.

Fact is this app that I will showcase today, has been two Windows versions in making. I started it with Windows 8 RTM but got stuck when I realized I couldn't save the Control Stack as an Image. If your projects or features were stuck due to same reasons, Windows 8.1 couldn't come soon enough!

Just to state the obvious, you'll need Visual Studio 2013 and target your application for Windows 8.1. This is default if you are using it on Windows 8.1 Preview, as I am.

Note: We have left out verbose code and XAML markup snippets and focused on code samples for specific features only. Please download the code and take it for a spin if you want to get hands on.

THE APPLICATION

The app we are building will use a File Picker to select multiple images we want to include in our 'Collage'. Images will then be loaded and arranged in a default pattern on the Canvas control. Once loaded, we can Zoom in/out, rotate and place the images to our liking. Once done fixing our Collage, we can save the Collage as an Image using the new RenderBitmap API. Optionally we can share our Image using the Share Charm.

One of the biggest omissions in Windows RT XAML SDK was the inability to convert a controls stack into a Bitmap Image. The upcoming Windows 8.1 fixes this by providing us with the RenderToBitmap API

Our application design is simple for the sake of this demo. We are not using any MVVM framework, like we should for a production app.

Our main focus is on

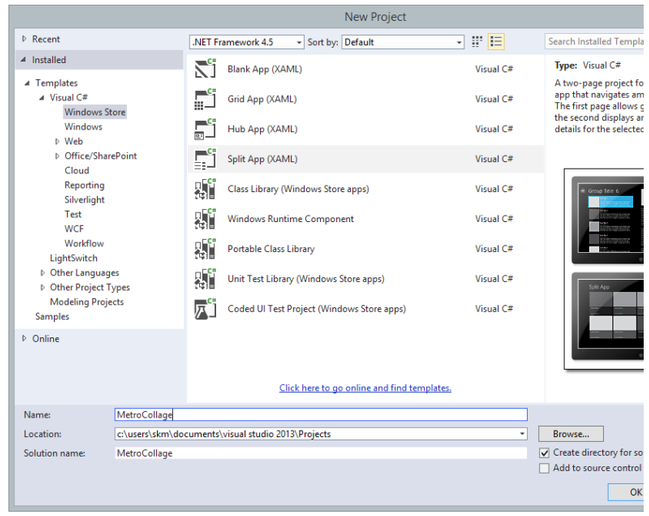
- Getting to use the RenderBitmap API to save an Image of the Canvas.

- Interpret the Pinch and Zoom gestures and apply to individual images so that they can be resized appropriately.

- Interpret the two finger rotate gesture and apply to individual images so they can be placed appropriately.

BUILDING THE APPLICATION

To start off with, we create a new Split Application in Visual Studio 2013.



The default application template uses a sample dataset and data model. You will find this in the DataModel folder.

The SampleData.json file contains dummy data for the SampleDataSource entity structure. We rename them to DefaultData.json and DefaultDataSource. We'll use these to create the first HomePage tile.

But it is worth mentioning that we have trimmed the DefaultData.json to just the following:

```
{"CanvasProjectsMRU":[
  {
    "Id": "-1",
    "Name": "New Collage",
    "Description": "Select group of files to use in Collage",
    "ImagePath": "Assets/DarkGray.png",
    "Pictures":
    [
      {
        "Id": "1",
        "ImagePath": "Assets/LightGray.png",
        "Top": 100,
        "Left": 100,
        "Rotation": 0
      }
    ]
  }
]
}
```

Note the change from generic properties like Items to the more specific - **Pictures**. We have to update the XAML bindings properties in ItemsPage.xaml as well.

We have added two entities CanvasPicture and CanvasProject to encapsulate information about our 'Canvas Project'. Today we'll not focus on serialization and deserialization of these, but let's breeze through the structure anyway.

The CanvasProject has default properties like the Name, a Thumbnail Image's path and the list of Pictures being used in the Canvas.

```
public class CanvasProject
{
 public int Id { get; set; }
 public string Name { get; set; }
 public string ImagePath { get; set; }
 public List<CanvasPicture> Pictures  { get; set; }
}
```

The CanvasPicture class encapsulates the Position and the Source File of the image. This meta-information will be useful when we support the "Save Project" functionality, so users can reload, edit, add, remove pictures and re-arrange them again if desired.

```
public class CanvasPicture
{
 public int Id { get; set; }
 public string ImagePath { get; set; }
 public double Top { get; set; }
 public double Left { get; set; }
 public double Width { get; set; }
 public double Height { get; set; }
 public double Rotation { get; set; }
 public int CanvasProjectId { get; set; }
 [XmlIgnore]
 public StorageFile SourceFile { get; set; }
}
```

Using a FilePicker in the ItemPage and loading images

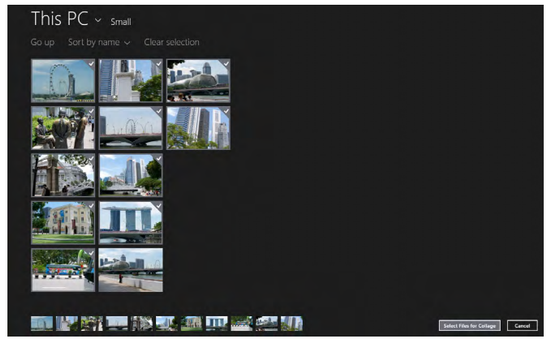
In WinRT, you cannot access files using their paths directly in code, unless you explicitly express intent to use the 'known' folders like Picture Library, Music Library etc. The correct way is to fire up a FilePicker and allow users to select one or more files using the Picker.

As of now, our app is setup to use the DefaultData.json in the Home page, so if you fire it up, you'll see a single Tile asking you to Select Files to create a new Collage.

Clicking on this will launch a file picker and return us the files selected. We'll take the selected files, create CanvasPicture instances out of them and pass them on to the Canvas page as we navigate to it.

The code basically checks if the Tile clicked on was the first tile clicked, and if so, creates an instance of the FileOpenPicker. We add png, jpg and jpeg as the filter types, set the ViewMode to Thumbnail, the startup location suggestion to PicturesLibrary, give it an identifier and the text of the Commit (OK) button. Finally we display the picker using PickMultipleFilesAsync call.

```
async void ItemView_ItemClick(object sender,
ItemClickEventArgs e)
{
 var project = ((CanvasProject)e.ClickedItem);
 if (project.Id == -1)
 {
  FileOpenPicker filePicker = new FileOpenPicker();
  filePicker.FileTypeFilter.Add(".png");
  filePicker.FileTypeFilter.Add(".jpg");
  filePicker.FileTypeFilter.Add(".jpeg");
  filePicker.ViewMode = PickerViewMode.Thumbnail;
  filePicker.SuggestedStartLocation = PickerLocationId.
   PicturesLibrary;
  filePicker.SettingsIdentifier = "imagePicker";
  filePicker.CommitButtonText = "Select Files for
   Collage";
  IReadOnlyCollection<StorageFile> files = await
   filePicker.PickMultipleFilesAsync();
 if(files.Count > 0)
 {
  project.ImagePath = files.First().Path;
  project = new CanvasProject
  {
   Name = "NewCanvas.canvas",
   FilePath = files.First().Path,
   ImagePath = files.First().Path + ".canvasimage",
   Id = -1
  };
  int count = 1;
  project.Pictures = new List<CanvasPicture>();
  foreach (var item in files)
  {
   CanvasPicture pict = new CanvasPicture
   {
    Id = count++,
    ImagePath = item.Path,
    SourceFile = item
   };
   project.Pictures.Add(pict);
  }
 }
}
this.Frame.Navigate(typeof(SplitPage), project);
```



Once the user returns from the Picker with the selected files, we create a CanvasProject object and then add one CanvasPicture instance for each file into the CanvasProject's Pictures collection and then pass it on to the SplitPage.

The SplitPage - XAML Changes

We update the SplitPage by removing the entire markup except for the Back Button. We simplify it into a Grid with a single Row and Column that hosts a Canvas.

```
<Grid.RowDefinitions>
 <RowDefinition Height="*"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
 <ColumnDefinition x:Name="secondaryColumn" Width="*"/>
</Grid.ColumnDefinitions>
<!-- Vertical scrolling item list -->
<Canvas Background="LightCyan" Name="collageCanvas"
Grid.Row="1">
</Canvas>
```

The Back Button is moved inside an AppBar that slides in from the Top when user right clicks or does the slide from the bottom gesture. It has a total of three buttons, **Back, Save** and **Remove**. In the scenario where we Save the Project also, this Save button should be called Export because that's what it does - exports the Canvas as an image using the RenderBitmap function.

```
<Page.TopAppBar>
 <CommandBar>
  <CommandBar.SecondaryCommands>
   <AppBarButton x:Name="backButton" Icon="Back"
    Height="95" Margin="10,25,10,0"
    Click="backButton_Click"
    Command="{Binding NavigationHelper.GoBackCommand,
ElementName=pageRoot}"
     Visibility="{Binding IsEnabled,
```

```
Converter={StaticResource
       BooleanToVisibilityConverter},
       RelativeSource={RelativeSource Mode=Self}}"
      AutomationProperties.Name="Back"
      AutomationProperties.AutomationId="BackButton"
      AutomationProperties.ItemType="Navigation
Button"/>
  </CommandBar.SecondaryCommands>
  <CommandBar.PrimaryCommands>
   <AppBarButton x:Name="Save" Label="Save" Icon="Save"
Margin="10,25,10,0" Click="Save_Click" />
    <AppBarSeparator Margin="10,25,10,0"/>
    <AppBarButton x:Name="Delete" Label="Remove Image"
Icon="Delete" Margin="10,25,10,0" Click="Delete_Click"
/>
  </CommandBar.PrimaryCommands>
 </CommandBar>
</Page.TopAppBar>
```

At design time, the Canvas looks as follows:

Note: The new XAML Designer shows hidden components at design time when you select them in the markup. In the above screenshot, the Back Button was selected in the markup making the entire AppBar visible with the Back Button selected.

The TransformableContainer Control

The TransformableContainer Control encapsulates the Translation, Scaling and Rotation behavior for our Images. It is inspired from the ManipulableContainer control you will see in some of the SDK Samples.

It essentially sets up a CompositeTransform object as the default RenderTransform on Load. It also sets it up to handle TranslateX, TranslateY, Scale and Rotate Manipulation Modes. By excluding the 'Inertia' modes, we ensure that the components don't fly out of the viewable area.

The OnManipulationDelta override updates the Transformation settings as required. The other Manipulation events are overridden and marked as handled.

```
public class TransformableContainer : ContentControl
{
private CompositeTransform _transform;
public CompositeTransform Transform
{
 get
 {
  return _transform;
 }
}
public TransformableContainer()
{
 _transform = new CompositeTransform();
 this.Loaded += ManipulableContainer_Loaded;
}

private void ManipulableContainer_Loaded(object sender,
Windows.UI.Xaml.RoutedEventArgs
{
 this.RenderTransform = _transform;
 // Enable standard manipulations, without inertia to
    ensure they stay in view
 this.ManipulationMode = ManipulationModes.TranslateX |
  ManipulationModes.TranslateY |
  ManipulationModes.Rotate |
  ManipulationModes.Scale;
 }

 protected override void
  OnManipulationStarting(ManipulationStartingRoutedEven
tArgs e)
 {
  base.OnManipulationStarting(e);
  e.Handled = true;
 }

 protected override void OnManipulationStarted(Manipulat
ionStartedRoutedEventArgs e)
 {
  base.OnManipulationStarted(e);
  e.Handled = true;
 }

 protected override void OnManipulationDelta(
 ManipulationDeltaRoutedEventArgs e)
 {
  base.OnManipulationDelta(e);
  // Update render transform to reflect manipulation
    deltas
  _transform.Rotation += e.Delta.Rotation;
  _transform.ScaleX += e.Delta.Scale - 1.0f;
  _transform.ScaleY += e.Delta.Scale - 1.0f;
  _transform.TranslateX += e.Delta.Translation.X;
  _transform.TranslateY += e.Delta.Translation.Y;
  e.Handled = true;
 }
 protected override void
  OnManipulationCompleted(ManipulationCompletedRoutedEv
entArgs e)
 {
  base.OnManipulationCompleted(e);
  e.Handled = true;
 }
}
```

SplitPage – The Implementation

Once we have navigated to the SplitPage, the OnNavigatedTo method is invoked

```
protected override void OnNavigatedTo(NavigationEventAr
gs e){
 _project = (CanvasProject)e.Parameter;
 _navigationHelper.OnNavigatedTo(e);
 Task returnTask = BindImages();
 returnTask.ContinueWith(_ => { });
 DataTransferManager dataTransferManager =
  DataTransferManager.GetForCurrentView();
 dataTransferManager.DataRequested +=
DataTransferManager_DataRequested;
}
```

We extract the CanvasProject sent to us by the home page and use it to layout the images in the BindImages method. The DataTransferManager is used for sharing data via the Share Charm. The DataRequested event is fired when the Share charm is invoked, we'll see its implementation in a bit. First let's see how we layout the images by default in the BindImages() method.

The BindImages Method

This method takes the CanvasProject and loops through all the Pictures in it. For each picture, it does the following:

- It loads the actual image from the disk and sizes it to 300x200 with stretch property set to Uniform implying that the aspect ratio will be maintained.

- Instantiates an instance of the Transformable container sets the Image as its content.

- Positions the image on the canvas. It divides the canvas into Screen Width/300 columns and calculates the current row/column accordingly.

- It also adds a Random rotate transformation between -15 degrees to +15 degrees. This gives the images a carelessly laid out look to start with.

- Finally adds the CanvasPicture to the TransformableContainer and adds the container to the Canvas.

The full code listing is as follows:

```
private async Task BindImages(CanvasProject project){
 collageCanvas.AllowDrop = true;
 Random degrees = new Random(5);
 int maxWidth = (int)Window.Current.Bounds.Width;
 int maxCols = (int)maxWidth / 300;
 int row = 0, col = 0;
 for (int i = 0; i < project.Pictures.Count; i++) {
  var canvasPicture = project.Pictures[i];
 var canvasPicture = project.Pictures[i];
 Image image = new Image();
 using (var fileStream = await canvasPicture.
  SourceFile.OpenAsync(FileAccessMode.Read)  {
  var sourceImage = new BitmapImage();
  sourceImage.SetSource(fileStream);
  image.Source = sourceImage;
  image.Tag = canvasPicture;
  image.Height = 300;
  image.Height = 200;
  image.Stretch = Stretch.Uniform;
  image.IsHitTestVisible = true;
  image.IsTapEnabled = true;
  TransformableContainer container = new
    TransformableContainer();
  container.Content = image;
  container.Transform.Rotation = degrees.Next(-15,
                                              15);
  col = (col + 1 > maxCols) ? 0 : col + 1;
  row = (col + 1 > maxCols) && (i != 0) ? row + 1 :
   row;
  container.Transform.TranslateX = col * 300;
  container.Transform.TranslateY = row * 200;
  container.Transform.CenterX = 150;
  container.Transform.CenterY = 100;
```

```
    container.CanvasPicture = canvasPicture;
    collageCanvas.Children.Add(container);
  }
```

With that, we are actually all set to run our app and create our Collage. But let's implement the save as well.

Saving the Canvas

We handle the Click event of the Save AppBar Button. The steps for saving the Canvas are rather simple:

- Create instance of the Windows.UI.Xaml.Media.Imaging.RenderBitmap object.
- Call the RenderAsync function and pass it the collageCanvas
- Extract the Pixel information into an IBuffer instance
- Get the target file information by requesting the user to provide a file name via the FileSavePicker()
- Open the Target File
- Create a Encoder instance for PNG encoding
- Stuff the Pixel information we saved in the IBuffer earlier, into the encoder and flush it to disk.

Voila! Canvas converted into Image. The full code listing for the Save click event is as follows

```
RenderTargetBitmap renderTargetBitmap = new
RenderTargetBitmap();
await renderTargetBitmap.RenderAsync(collageCanvas);
var pixelBuffer = await renderTargetBitmap.
GetPixelsAsync();
var savePicker = new FileSavePicker();
savePicker.DefaultFileExtension = ".png";
savePicker.FileTypeChoices.Add(".png", new List<string>
{ ".png" });
savePicker.SuggestedStartLocation = PickerLocationId.
PicturesLibrary;
savePicker.SuggestedFileName = "snapshot.png";
// Prompt the user to select a file
var saveFile = await savePicker.PickSaveFileAsync();
// Verify the user selected a file
if (saveFile == null)
    return;
// Encode the image to the selected file on disk
using (var fileStream = await saveFile.
OpenAsync(FileAccessMode.ReadWrite)){
 var encoder = await BitmapEncoder.
CreateAsync(BitmapEncoder.PngEncoderId,
  fileStream);
 encoder.SetPixelData(
  BitmapPixelFormat.Bgra8,
  BitmapAlphaMode.Ignore,
  (uint)renderTargetBitmap.PixelWidth,
  (uint)renderTargetBitmap.PixelHeight,
  DisplayInformation.GetForCurrentView().LogicalDpi,
  DisplayInformation.GetForCurrentView().LogicalDpi,
  pixelBuffer.ToArray());
 await encoder.FlushAsync();
}
```

Implementing Share Charm Integration

Now that we are done with Saving, let's see what it takes to share the Collage. This is a subtle way to demonstrate how to render the image in-memory without saving to disk. This scenario will come in handy if you want to implement Dynamic Thumbnail type UI Widgets.

If you remember, we had attached an EventHandler – DataTransferManager_DataRequested in the OnNavigatedTo method in SplitPage.xaml.cs.

This event handler sets up the Title and Description that we see when the Share Charm is invoked for our App. It also sets up the OnDefferedImageRequestedHandler delegate to be called when Windows requests for the Image.

```
private void DataTransferManager_DataRequested(
DataTransferManager sender, DataRequestedEventArgs e){
 e.Request.Data.Properties.Title = "MetroCollage";
 e.Request.Data.Properties.Description = "Collage by
Metro Collage";
 e.Request.Data.SetDataProvider(StandardDataFormats.
Bitmap, new DataProviderHandler(
this.OnDeferredImageRequestedHandler));
}
```

How does Windows know that our App can share Images? The StandardDataFormats.Bitmap enum value lets Windows know that we'll be sharing a Bitmap.

Grabbing the Rendered Bitmap in Memory

The OnDefferedImageRequestedHandler delegate uses the RenderTargetBitmap. This time instead of writing to a FileStream for a file selected by the user, we create an InMemoryRandomAccessStream and flush the PNG encoded pixel data into the memory Stream.

Once done, we use the DataProvideRequest object to set the data that's going to be used by the app selected via the share charm (target app). The complete code listing is as follows:

```
private async void OnDeferredImageRequestedHandler(
DataProviderRequest request){
 DataProviderDeferral deferral = request.GetDeferral();
 await Dispatcher.RunAsync(Windows.UI.Core.
CoreDispatcherPriority.Normal, async ()
  =>
  {
   try   {
    RenderTargetBitmap renderTargetBitmap = new
     RenderTargetBitmap();
    InMemoryRandomAccessStream stream = new
     InMemoryRandomAccessStream();
    // Render to an image at the current system scale
    //  and retrieve pixel contents
    await renderTargetBitmap.RenderAsync(collageCanvas);
    var pixelBuffer = await renderTargetBitmap.
       GetPixelsAsync();
    // Encode image to an in-memory stream
    var encoder = await BitmapEncoder.
      CreateAsync(BitmapEncoder.PngEncoderId,
      stream);
    encoder.SetPixelData(
     BitmapPixelFormat.Bgra8,
     BitmapAlphaMode.Ignore,
     (uint)renderTargetBitmap.PixelWidth,
     (uint)renderTargetBitmap.PixelHeight,
     DisplayInformation.GetForCurrentView().LogicalDpi,
     DisplayInformation.GetForCurrentView().LogicalDpi,
     pixelBuffer.ToArray());
     await encoder.FlushAsync();
     // Set content of the DataProviderRequest to the
     // encoded image in memory
     request.SetData(RandomAccessStreamReference.
     CreateFromStream(stream));
   }
   finally   {
    deferral.Complete();
   }
  });
}
```

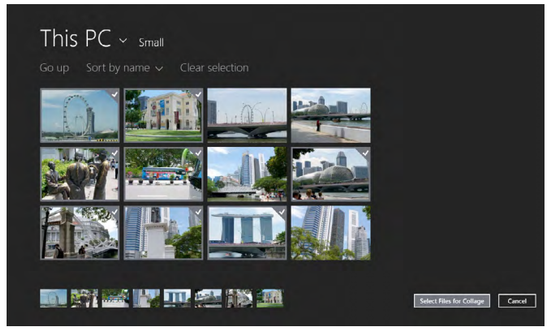
That's actually a wrap for our application.

DEMO TIME

With our app all set, it's time for us to see an end-to-end Demo of all the features. So off we go:

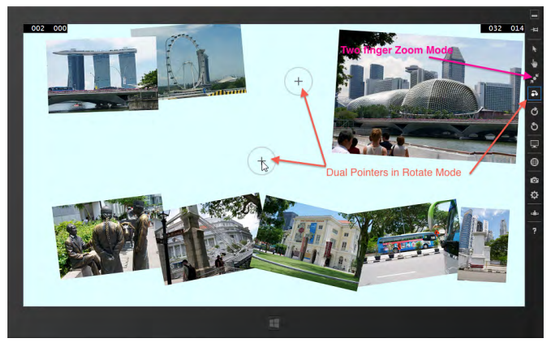
Step 1: Run the App and on the Home Page click on the New Collage Tile

Step 2: Select a bunch of images. You can see, I've selected about 8 images from my last Singapore visit.



Step 3: Once we click on "Select Files for Collage", the app navigates to the Collage page and the layout is something like the following. Remember the rotation degrees are random so your results might be slightly different

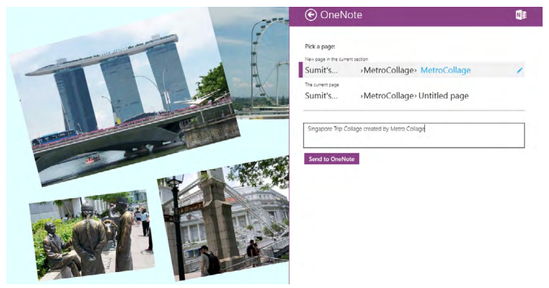
Step 4: Now re-arrange the Images, rotate and size them as you want. If you are running the app on a Desktop without touch controls, you can use the Simulator to test out the Rotate and Zoom effects. Note the dual pointers when you select Zoom or Rotate Modes



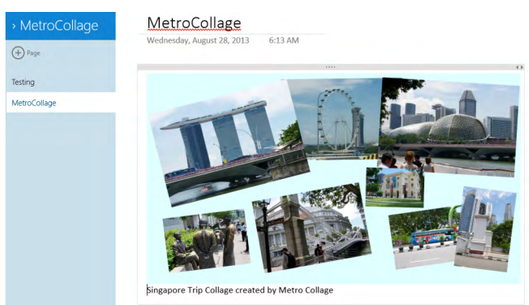
Step 5: After rearranging, Scaling and Rotating images to my liking, my final collage is as follows.

Step 6: Now when I invoke the Share Charm, I get one app that can accept in-memory Bitmaps and that is One Note.

Step 7: Clicking on OneNote brings up additional information that I can tag to One Note.

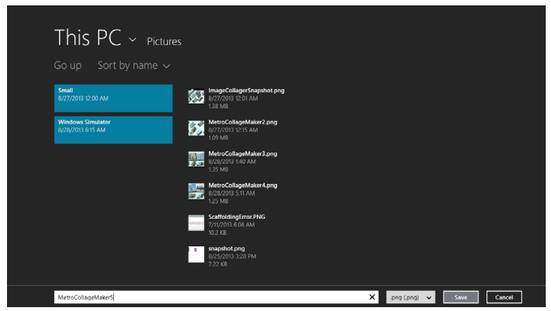


Step 8: If I switch to One Note and check, the image would have been copied into it



Step 9: To save to a File, we invoke the AppBar and click on Save

Step 10: Click/Tap Save to save the file and open it from Explorer to verify!



CONCLUSION

That brings us to the end of this article. We saw a bunch of stuff happen today. Of these, saving a control-stack as an Image to a File as well as in memory, was new to Windows 8.1. We used the Canvas as the container today, but you could pretty much use any Container control and render the Control stack inside it. We also saw how to use Pinch to Zoom and two finger 'Tap and rotate' functions which was neatly encapsulated in the TransformableContainer control.

We also saw how to share Bitmaps between applications using the Share Charm. Windows 8.1 brings a host of new APIs that makes it much more attractive to developers as well as the end user of the platform ■

Download the entire source code from our GitHub Repository at bit.ly/dncm8-piccol

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