C# 5.0 NEW FEATURES
ASYNC AND AWAIT IN C# 5.0

MEET ROSLYN
MICROSOFT'S COMPILER AS A SERVICE PLATFORM

WINDOWS AZURE
5 ROCKING TIPS ON WINDOWS AZURE IAAS

DNC MAGAZINE

www.dotnetcurry.com | ISSUE NOV-DEC 2012

PROCESSES IN TFS 2012
Embracing SCRUM and CMMI

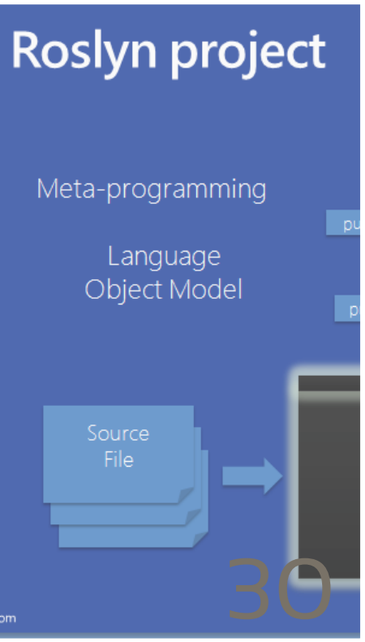
EXPLORING KNOCKOUT
Create a Snappy ASP.NET MVC UI with KnockoutJS

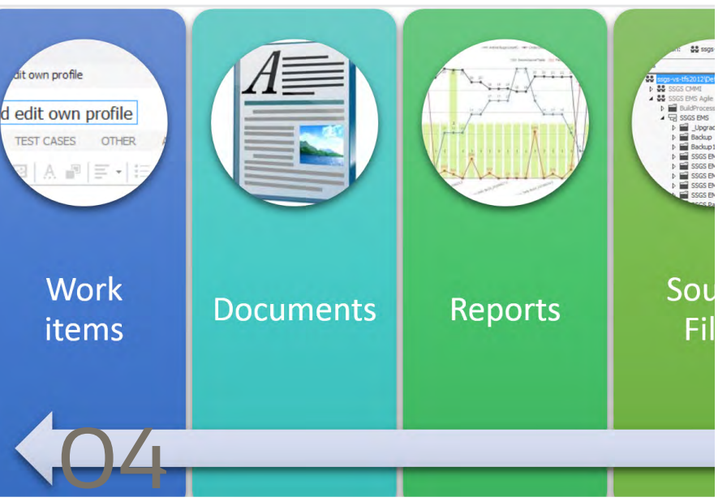
WINDOWS 8 APPS
Background Tasks and Live Tiles

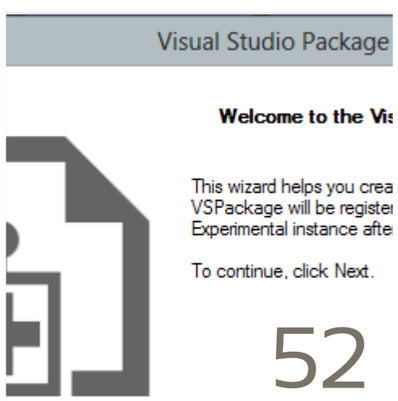
EXPLORING THE IGNITEUI TOOLSET
Building an Enterprise Web application from scratch

EXTENDING VS 2012

FEATURED Interview with Jon Galloway







Articles

stay

connected

www.Facebook.com/DotNetCurry

@dotnetcurry

www.DotNetCurry.com/magazine

Editor's Note*

Welcome once again to the third issue of DNC Magazine. As always, we have a cracker box full of good articles for you.

With the start of November, the 'Holiday Season' mood starts creeping in. November here in India is as much a Diwali (the festival of lights) month, as are the months of Thanksgiving and Christmas. We at 'DNC Magazine' wish all our readers a joyous Holiday Season.

We are extremely happy to welcome new Contributing Writers, Anoop Madhusudanan (MS MVP), Filip Ekberg (Author of the book C# Smorgasbord), Gregor Suttie (prolific blogger) and Niraj Bhatt (MS MVP). We start off with Subodh Sohoni's article on Software Development Processes in TFS 2012. Then we have Niraj telling us about Azure IAAS followed by Filip on the Async and Await pattern and then Gregor giving us a tour of KnockoutJS.

In the interview chair, we have Jon Galloway from Microsoft in his personal capacity. The interview with Jon reveals a fantastic personality behind the ASP.NET and Azure Evangelist, we already know and love.

Apart from being a Holiday Season, the Windows 8 season continues as well. Windows 8 is now available commercially and initial reports suggest it's doing awesome! One report says the Pre-sales for Windows 8 were 40 percent higher than Windows 7. Now that's something. Microsoft's first ever tablet hardware, the Surface RT is shipping and seems to be doing rather well.

With the impending success of Windows 8, it's hard to ignore the new Windows 8 Store app platform. In this episode, we have Mehfuz telling us how to do background tasks and tile notifications in a WinJS application.

We also have Anoop introducing us to the exciting world of Roslyn – The compiler as a service platform. We introduce a new section this episode where we explore a 3rd party Web development toolkit for developing Enterprise apps.

Finally we round off with a Visual Studio 12 automation article in which we develop a Visual Studio Extension that zips up a Solution and creates a single archive.

So once again, wishing everyone a fantastic Holiday Season, be safe, have fun, we'll see you again in the New Year. Keep your comments on DNC Magazine coming in. We read each one of them.

Sumit K. Maitra
Editor in Chief

www.dotnetcurry.com
dnc mag

Editor In Chief Sumit Maitra
sumitmaitra@a2zknowledgevisuals.com

Editorial Director Suprotim Agarwal
suprotimagarwal@a2zknowledgevisuals.com

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

Advertising Director Satish Kumar
business@dotnetcurry.com

Writing Opportunities Carol Nadarwalla
writeforus@dotnetcurry.com

Contributing Writers Anoop Madhusudanan, Filip Ekberg, Gregor Suttie, Mehfuz Hossain, Niraj Bhatt, Subodh Sohoni, Sumit Maitra, Suprotim Agarwal

Interview Feature Jon Galloway
Twitter @jongalloway

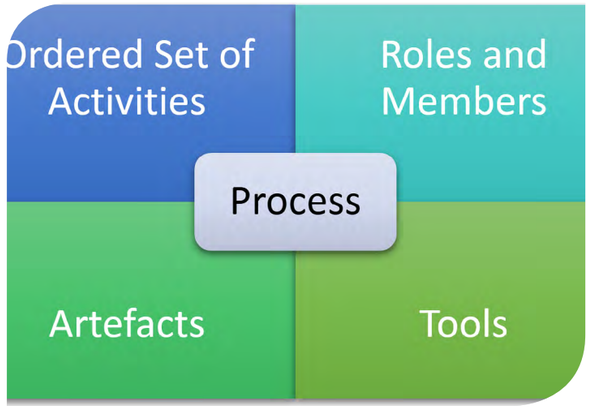
Next Edition 1st January, 2013
www.dncmagazine.com

POWERED BY
a2z Knowledge Visuals

EMBRACING SCRUM AND CMMI IN TFS 2012

VS ALM MVP Subodh Sohoni discusses how strongly Team Foundation Server 2012 supports the implementation of SCRUM and CMMI processes

Definition of the word 'process' is contextual to the subject in which it is used. The simplest generic definition of process found in dictionaries is '*a series of actions that produce a change or development*'.



After studying and gaining experiences in various facets of software development, I have arrived at a definition that is more comprehensive than the one ' mentioned above. I prefer to define Process as an '*Ordered set of activities that are performed once or optionally, iteratively in a predefined environment, by people playing different roles that produces desired artefacts, with the help of appropriate tools*'

Thus managing the entire 'Software Development process' is certainly a non-trivial task. Microsoft Team Foundation Server (TFS) facilitates execution of activities of the software development process and manages that result into desired artefacts. Those artefacts usually are featured products or services but may also include many other things like release notes, documentation, user manuals, installation & maintenance manuals, administration utilities etc.

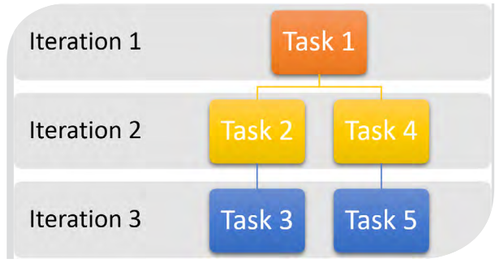
Let us explore how TFS 2012 facilitates implementation of two of the most popular software development process variants. One is CMMI, that is based upon the traditional waterfall model and the other is SCRUM, that is one of the most popular implementations of Agile methodologies.

ORDERED SET OF ACTIVITIES

Both CMMI and SCRUM are collection of many processes. For example, there is a set of activities that a software architect executes to transition the requirements into physical design. One example artefact is a UML diagram and Visio is a tool for that. Another example is creation and execution of a test case by a tester. Any of these processes need to be managed – planned, monitored and controlled. TFS 2012 facilitates this process management with the help of hierarchical work items.

Activities are always represented by tasks. In TFS 2012, tasks are an important type of work item. First level of classification of tasks supported by TFS 2012 is through iterations and area.

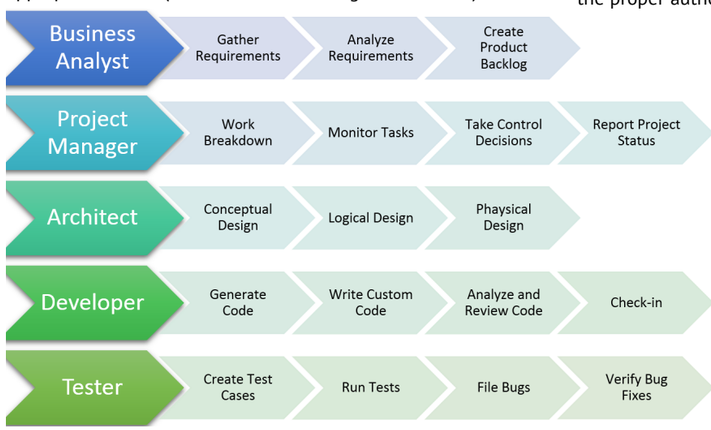
Iterations are natural part of Agile as well as adapted in CMMI. TFS 2012 allows us to classify each task into various time bound iterations. In SCRUM, they may be called Sprints but conceptually they remain same. Areas in TFS 2012 are logical grouping of work items. Freedom is given to the manager to decide the logic of grouping.



I have seen areas created for modules under development and physical tiers in which the developed component may go. Tasks can be managed using convenient software like MS Project or MS Excel. TFS 2012 seamlessly integrates with these software so that planning, scheduling, monitoring status and editing them becomes easy through the interface that is commonly used. Prioritization is done with the help of either priority given to the task work item or that given to the parent of tasks, which can be Requirements, User Stories or Product Backlog Items (PBIs).

MS Project also facilitates relationships like predecessor – successor that are effectively replicated in TFS 2012. In this way, the project manager in case of CMMI, or a team in case of SCRUM, defines the order in which the activities will be done. Tasks are assigned to various team members so that those team members will execute those tasks at the scheduled time. In case of CMMI, as the tasks are closed by each team member, their status is reflected in TFS and there onwards, in the MS Project, being used by the project manager.

SCRUM (task) board is a facility provided in TFS 2012 that allows the team members to move tasks to the columns of appropriate status (From 'To Do' to 'In Progress' to 'Done').



ROLES AND MEMBERS

In any process, each team member does not have the same ordered set of activities as others, since they play different roles. This role differentiation is more pronounced in CMMI than in SCRUM. TFS 2012 supports any role that is required to be present through groups. In TFS 2012, user grouping is present at three levels – at overall TFS level, Team Project Collection Level and Team Project level. Aim of grouping is to define different activity streams for each group and to provide appropriate permissions to features of TFS to them. Various services of TFS expose different permission types. Source Control service has permissions to access each object like branch, folder and file that can be granted or not granted to various groups. There is further granularity of permissions, for example in a project, it is possible to define who can check-out files and who can check-in, which group can edit labels and which group is not allowed to do that. Work Item tracking service has permissions based upon the queries that show results of filtered set of work items.

TFS 2012, to support SCRUM, also allows us to define teams that are brought together for implementing a set of features for a product. Defined teams can then be used to do Product Backlog planning, Sprint Planning and load balancing. TFS 2012 facilitates sprint planning and load balancing to be done by the teams with a graphical view that highlights the areas of overloading and under loading, while the load assignment is being done within the team.

Membership of groups and teams is based upon the security principals like domain (active directory) or windows where TFS is installed. This allows the authentication to be delegated to the proper authority and reduces any chances of user database being hacked. Authentication mechanisms supported are NTLM and Kerberos depending upon the operating systems and the option selected at the time of installation of TFS.

ARTEFACTS

Process of developing software generates many types of artefacts. Some of them are in the document form and others are in the source code files form. Documents further can be categorized as living documents which get changed during every phase of development

or static documents that are created and then never changed. Any artefact that is a living document can be defined in TFS 2012 as work item type.

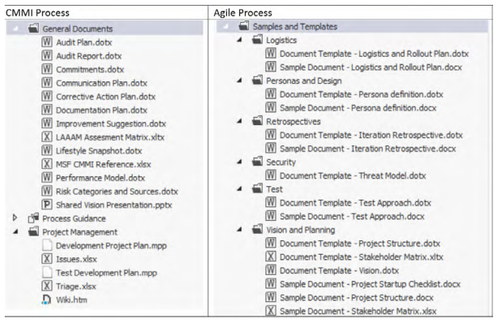
For example, a requirement document may contain requirements or stories for implementation. Both of these are created at certain point in time during the process of software development and once created they live an entire lifetime passing through many phases which finally terminates in Closed state when that requirement or story is completely implemented. This lifetime of the requirement or story is captured in the work item of those types. They pass through many states and while being in those states, collect a lot of relevant and related data.

TFS 2012 supports many types of work items that are commonly used. They are grouped for the methodology that the organization would want to follow. In case of Agile methodology ,it has built in definitions for User Story, Task, Test Case (and related Shared Steps), Bug and Issue. It supports SCRUM with work item types for Product Backlog Item, Task, Impediment, Test Case and Bug.

Under the process template of CMMI, it allows the creation of Requirements, Tasks, Test Cases, Bugs, Risks, Change Requests and Reviews. Other types of living documents are the reports that are created to show progress done and health of the project. These reports are based upon the data that is collected by Work Items that have lived or are living from the start of the project, till the moment that report is generated. This data is stored, analyzed and made available in report form by TFS 2012.

Some reports are also based upon the source that is under the source control. This is a very efficient way of reporting and effortlessly provides all necessary reports for the management of project, productivity and quality.

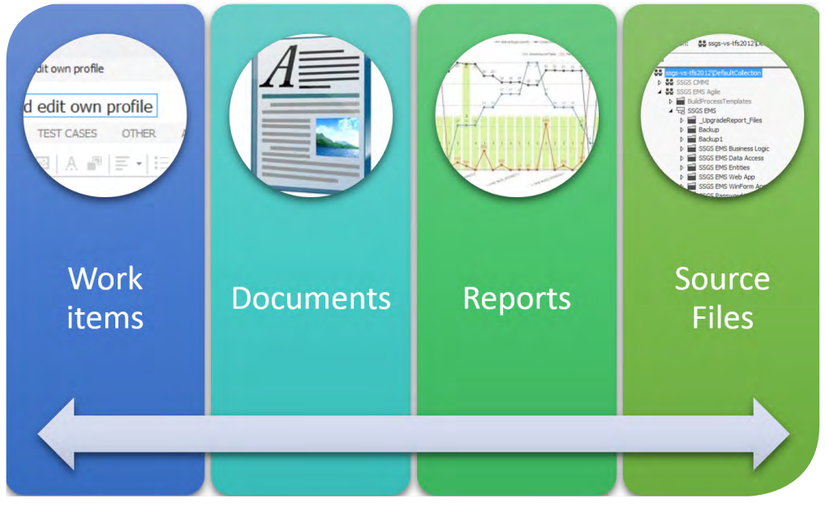
Some documents that are created but vary rarely changed, if at all, are provided by TFS 2012 in a template form. For example, there are documents for Project Management, Audit Plans, Documentation Plan, Risk Management, Security Management and many other related subjects. All these documents are



created and versioned in the libraries of portals created for each project on a linked SharePoint server. TFS 2012 is a complete set of ALM services. Source control is one of the major services out of that. Every source file of the project can be kept under the source control of TFS 2012.

TOOLS

TFS 2012 and Visual Studio 2012 are the largest set of tools that are created for supporting Application Lifecycle Management. MS Office components can also be integrated with TFS 2012. With so many tools to choose from, the user can use the tool of choice, and the ones with which they are comfortable to interact with TFS and to improve productivity without compromising quality.



REQUIREMENT GATHERING AND ANALYSIS

The task of requirement gathering and analysis is done by the Business Analysts. They are usually comfortable working in MS Excel so using the same tool, they can elicit requirements and stories as work items. They can also check the status of requirements and stories by synchronizing the same worksheet with the work items data from TFS 2012. In addition to work items for requirements or stories or PBIs, it also provides a tool for storyboarding in collaboration with MS PowerPoint. Using this tool, not only can you create storyboards that depict the flow of user interface, but also get a feedback with corrections in it from the customer.

PROJECT AND PROGRAM MANAGEMENT

Project and program management requires a manager to plan, monitor and control the efforts, quality (tests and bugs), risks, impediments and issues. Effort planning is facilitated by TFS 2012 through the work item type Task. Work break down can be done using either MS Project or MS Excel and then the result of that WBD is published in TFS 2012 as Task work items. These tasks are assigned to appropriate team members. These team members update status using other tools like Visual Studio 2012 to reflect the real efforts that have taken place. Managers get updates in their project plan since they are synchronized time to time to get the latest status from TFS 2012. Quality Managers may use MS Excel to do the test case planning. But, using Microsoft Test Manager 2012 (MTM 2012) is more productive and intuitive way of doing it. MTM 2012 has the ability to do the test case planning and monitoring the test results. It also has a built in feature to run the manual tests more efficiently. Risks, impediments and issues are managed using the Team Explorer or a browser based client of TFS 2012.

SPRINT PLANNING AND RUNNING

The biggest enhancement in TFS 2012 in comparison to TFS 2010 has happened in this area. TFS 2012 has built in support to do the sprint planning. While doing the sprint planning, it also allows the team to do capacity balancing. To run the sprint, TFS 2012 has a built-in, interactive sprint board, where team members can view the status of various tasks as well as move the tasks to appropriate areas.

BUILD AND DEPLOYMENT

Build service of TFS 2012 incorporates build administration as well as build workflow execution. It supports advanced concepts like Continuous Integration, Gated Check-in and verification of build using unit tests (either Microsoft or third party), send appropriate notifications when the build completes and call external tools to do additional tasks. We can also use custom activities of the workflow to package and deploy the built software.

FEEDBACK FROM CLIENT

In SCRUM, it is necessary to get feedback from client at the end of the sprint for the incremental release that was completed in that sprint. With TFS 2012, Microsoft has introduced a software tool to request feedback from client. Using this tool, it is possible to request a feedback post deployment, from the customer. A free feedback client part of the tool allows customer to view and run the deployed software and provide feedback in the form of text, voice, video, annotation on screens etc. This will complete the loop that is started for the sprint.

CONCLUSION

All this discussion has shown us how strongly Team Foundation Server 2012 supports the implementation of SCRUM and CMMI processes by providing the process templates that package these ordered set of activities, roles and artefacts. TFS 2012 with the help of Visual Studio 2012 and MS Office provides all the necessary tools to implement these process templates in real life. This support means an increase in productivity and quality for the customers who adopt those. ■

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

THE TOP 5 TIPS FOR WINDOWS AZURE IAAS

Microsoft MVP, Niraj Bhatt discusses five useful points for those starting off their IAAS journey on the Azure platform.

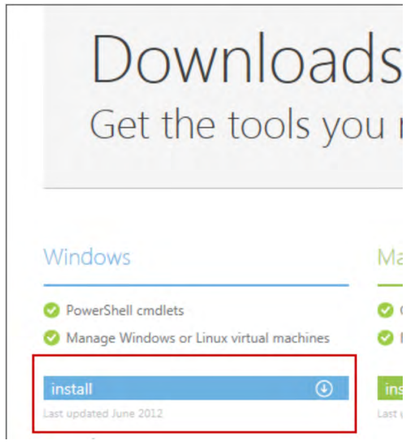
Windows Azure Virtual Machines, the Infrastructure-As-A-Service (IAAS) offering, is a big leap for Windows Azure platform. Virtual Machines, as the name suggests, offers multi-tenant virtualized infrastructure, wherein end user can deploy applications. This specifically simplifies the migration of legacy applications to Cloud, offering more optimization avenues to the enterprise IT.

When Virtual Machine preview was released in June 2012, I thought it would be a no-brainer - just spawn a new VM with few mouse clicks on Azure portal - and from there on, it's more like interacting with a remote server within your enterprise. But as I started working with it, aligning it to enterprise deployments, the experience turned out to be quite different and very educating. This article is an outcome of my dabbling, wherein I would be listing down five key points I wish I knew and understood well at the starting off my IAAS journey on the Azure platform.

THESE TIPS WILL HELP YOU TREMENDOUSLY

1 USE POWERSHELL

Being a developer writing code most of the time, I wasn't an advocate of PowerShell. But when I looked at Windows Azure PowerShell cmdlets (pronounced command-lets), I was completely sold. Using these cmdlets, you can customize almost every aspect of VM creation which otherwise you would manually do via Azure portal. In fact, few advance configurations are only possible via these cmdlets. For instance, basic configurations can be easily set with cmdlets like specify the VM name, instance size, base image name (refer to this article on how to add your own image to Windows Azure Image Gallery), login password, Cloud Service (the container for VM) name, affinity group, Virtual network and even the subnet in Virtual network. But if you are looking to support advanced scenarios like creating a VM that should be domain joined with an Active Directory installed on Azure, you can specify additional cmdlet parameters like fully qualified domain name, domain user name, domain password and necessary DNS settings. While you got to spend some time initially understanding various parameters cmdlets accept but once you are through with this initial phase, PowerShell enhances your productivity to the next level. Below is a simple PowerShell Script to create a VM. You can download cmdlets from Azure portal



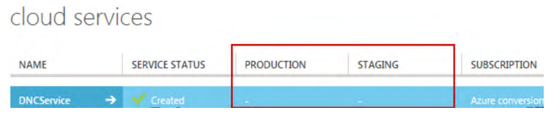

```
#Download publish settings using
#Get-AzurePublishSettingsFile

Import-AzurePublishSettingsFile
"C:\xyz.publishsettings"

#set subscription / storage for VM deployment
Set-AzureSubscription -SubscriptionName
'subscriptionName' -CurrentStorageAccount 'accountName'

#Set VM configuration parameters
$ServiceName = 'DNCService'
$imgName = 'MSFT__Win2K8R2SP1-Datacenter-201208.01-en.
us-30GB.vhd'
$vmConfig = New-AzureVMConfig -Name 'DNCService1'
-InstanceSize Small -ImageName $imgName |
Add-AzureProvisioningConfig -Windows -Password
'Password123'

#Create new VM
New-AzureVM –ServiceName $serviceName -VMs $vmConfig
-AffinityGroup 'NRAffGroup'
```

2 CONTROL VM LIFETIMES

Most of the Cloud computing resources are billable on an hourly basis and it's important that you release these resources when you no longer need them. This typically applies to Windows Azure Virtual machines not running 24x7, example - there could be business workloads which requires an application to run only twelve hours on week days. To avoid billing, most users stop their VMs, only to realize that Windows Azure bills you for VMs that are in a stopped state. So your only option to control costs is to delete the VMs. But wouldn't deleting a VM cause any issues?

The answer is both No and Yes. When you delete a VM, you are just deleting the VM instance, but the underlying OS and data disks are still intact (in fact, you still keeping paying for their storage which luckily is quite negligible). Hence you can easily resurrect your VM without much harm. The issue you might face though when you delete and re-create the VM, is the public IP address change. I work in an organization with strict IT security rules and locked down access. Static IP was necessary for me to raise an outbound RDP access request with my IT team. But with IP changing everyday, it was definitely turning into a challenge. At the end, the solution I adopted was to create an extra small VM running 24x7 and bounce from there to other VMs.

It's important to note that when you delete the VM, you still retain the underlying Cloud Service container and its associated Site URL (see '#3 Recover from VM failures' for more information on this topic). This snapshot from Azure Portal shows an empty Cloud Service Container post deletion of the VM. To delete a VM, you can use PowerShell cmdlets.



PowerShell cmdlets allow you to export your VM configuration, delete the VM and then re-create VM using exported configuration. A sample script is listed below:

```
#Export VM configuration to a file on local disk
Export-AzureVM -ServiceName '<cloud service>'  -Name
'<vm name>' -Path 'c:\vmconf.xml'
# Delete Azure VM
Remove-AzureVM -ServiceName '<cloud service>'  -Name
'<vm name>'
#Import / Recreate Azure VM using the exported
configuration file
Import-AzureVM -Path 'c:\vmconf.xml' | New-AzureVM
-ServiceName '<cloud service>' -VNetName '<vnet name>'
-DnsSettings '<DNS entry>'
```

Export configuration generated above is stored in a XML file. It contains various properties of a VM including Endpoints, disks,

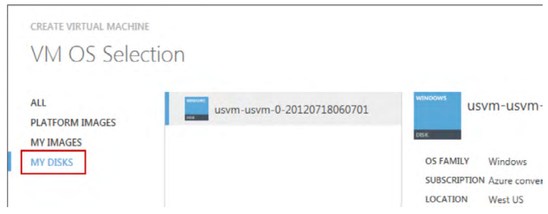
VM size, subnet, etc.

3 RECOVER FROM VM FAILURES

Your ability to handle failures is quintessential while working with any cloud platform and same applies to Windows Azure Virtual Machines. Being in preview mode, I expected some glitches with this offering. There were times when I would setup a VM and next day wasn't able to RDP (Remote Desktop) into it. I looked like my hard work of setting up the VM would go down the drain. But those initial struggles filled my knowledge gaps. While the stability of VMs has dramatically improved as we approach the General Availability, below are few pointers to deal with RDP failures.

Restart VM: First is to restart your VM. That should do the trick whenever your VM becomes unresponsive.

Delete VM: In case that doesn't work or the restart itself is failing, you can try deleting the VM and re-creating VM as detailed in 'Controlling VM lifetimes'. If you are following those steps and attempting a manual delete via Azure Portal, you should delete the underlying Cloud Service too. As explained later in 'Load Balancing VMs', Cloud Service – is the container holding your VM and would become visible when you delete the VM. Deleting your Cloud Service would allow you to reuse your DNS name (as best practices ensure that your VM and DNS names are prefixed with unique identifiers). I have seen quite a few developers struggle and provide different names to their VMs every time they delete it, failing to realize that they need to delete the Cloud Service too. The following figure shows how you can create a new VM by selecting an existing disk



Resize VM: An alternative to deleting your VM is to change the size of your VM. This is less intrusive, but might not be feasible all the times.

Unlock VHD / Disk: At times, after you delete your VM, the portal might show that Disk is still attached to the deleted VM. This would result into a difficult situation because you can't reuse the Disk until is detached. The easier option to resolve this is to delete the disk object. Deleting the Disk object would still retain the underlying blob. After deletion, you can re-create the disk object and use it to spawn a new VM. If you can't delete the disk, then your only option is to break the blob lease manually via PowerShell or using Storage Client Library (refer to this MSDN forum link bit.ly/errdelvm for further information).

4 LOAD BALANCE VMs

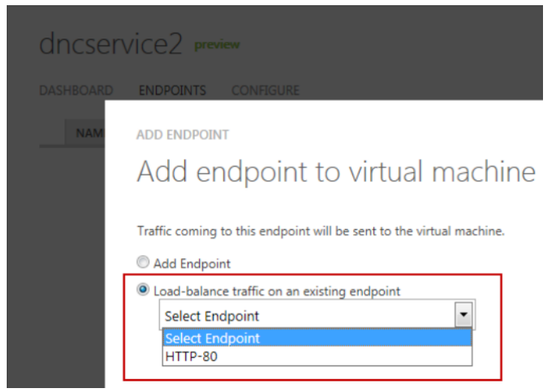
Load balancing is essential to leverage Cloud's elasticity. If you want to maximize your Cloud migration ROI, you should be able to scale your application linearly to the corresponding demand. While establishing load balancing was trivial with Platform-As-A-Service (PAAS) offerings like web role, IAAS requires you to understand how individual Virtual machines are connected. When you create a Windows Azure VM, it's placed inside a container called 'Cloud Service' – the fundamental unit of deployment on Windows Azure Platform. The name of this Cloud Service is the DNS name you specified while creating your VM in the Azure Portal.

Interestingly, the Cloud Service container is hidden and you see this container only when you delete the VM or when you deploy more than one VM in the same Cloud Service. It's important to note that portal currently doesn't support adding multiple VMs inside a Cloud Service, and the same has to be carried via PowerShell cmdlets. VMs placed inside the same cloud service are connected by default. This is a primary requirement to Load Balance VMs and it really doesn't matter whether your VMs are part of a Virtual network (Virtual networks are used to connect Cloud Services in the same affinity group) or not. Bottom line - you can't load balance VMs that don't belong to a single Cloud Service. The PowerShell cmdlets to create a VM with a new Cloud Service and to create a VM to join existing Cloud Service are quite similar. For latter, you just don't provide an affinity group / location and your VM will be created in specified Cloud Service. Sample script is listed here:

```
#Create VM with a New Cloud Service
New-AzureVM –ServiceName $cloudService -VMs
$vmConfig -VNetName $virtualNetwork
-AffinityGroup $affinityGroup
```

```
#Create VM into an existing Cloud Service
New-AzureVM –ServiceName $cloudService -VMs $vmConfig
-VNetName $virtualNetwork
```

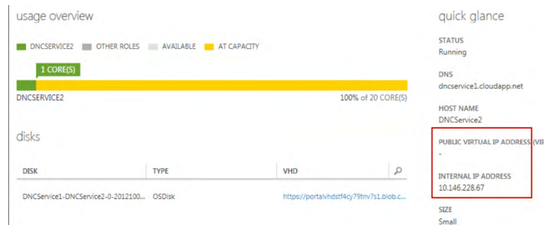
CloudService variable should be common for creation of both VMs. After multiple VMs are placed inside the same Cloud Service, you need to setup load balancing endpoint. For web based applications, this would typically be on port 80. For each VM, you need to select the load balanced endpoint as shown below:



5 RESTRICT VM'S PUBLIC ACCESS

Every Virtual machine in Windows Azure is contained inside a Cloud Service. For accessibility to inside VMs, every Cloud Service has public IP address and an internal IP address. At times though, you might require different levels of access. Consider a 3-tier deployment scenario, where you want only your front end web servers to be accessible publically and rest of your deployment including database servers to be reachable only via those front end servers (internal access).

To achieve this, you got to control your VM endpoints. VM endpoints are publicly accessible endpoints of a VM. To restrict public access to a VM, it's as simple as deleting those endpoints. Note you can still reach out to these VMs using internal IP address provided they are in the same Cloud Service or connected via Virtual Network. You can RDP into the VMs using internal IP address and allow access to specific ports as required. This snapshot shows you a VM with no public endpoints and hence Azure portal doesn't display its public IP address.



SUMMARY

To summarize, this article listed few aspects of Windows Azure Virtual Machines that you need to keep in mind. As shown in the article, you can use PowerShell for most of your tasks and if your VMs deployed are deployed in the same Cloud Service container, it's easy to load balance them. You can also restrict public access to VMs by controlling VM endpoints. Finally, it's important that you manage the lifetimes of your VM to maximize your Azure Cloud ROI and in case your VM becomes unresponsive, you can try the various routes including restarting it. Hope that will make your adoption of Azure Virtual machines lot easier. Please mail me your feedback. ■

Niraj works as a Senior Architect for a Fortune 500 company and has an innate passion for building / studying software systems. He is a top rated speaker at various technical forums including Tech Ed, MCT Summit, Developer Summit, and Virtual Tech Days, among others. He enjoys working on – IT innovations that impact enterprise's bottom line, architecture and integration of systems, performance tuning, and review of enterprise applications. He has received Microsoft MVP award for ASP.NET, Connected Systems and Windows Azure. He maintains a blog at http://nirajrules.word-press.com and can be reached at niraj@indiamvps.net.

ASYNC & AWAIT IN C# 5.0

Filip Ekberg introduces the new Async and Await pattern for asynchronous development in C# 5.0 and highlights the differences from the traditional practices in place so far.

As of C# 5.0 which comes with .NET 4.5 and with Visual Studio 2012, we can use the new asynchronous pattern involving use of the async and await keywords. There are many different point of views regarding if this is a better approach than what we have seen before with regards to readability and usability. We will walk through an example and see how it is 'different' from the current practices.

LINEAR VERSUS NON-LINEAR CODE

Most software engineers are used to programming in a linear manner, at least that is how they are taught when they begin their careers. When a program is written in a linear manner, that means the source code will be read somewhat like what the diagram in Figure 1 visualizes. This assumes that we have an order system in place that will help us to fetch a collection of orders from somewhere.

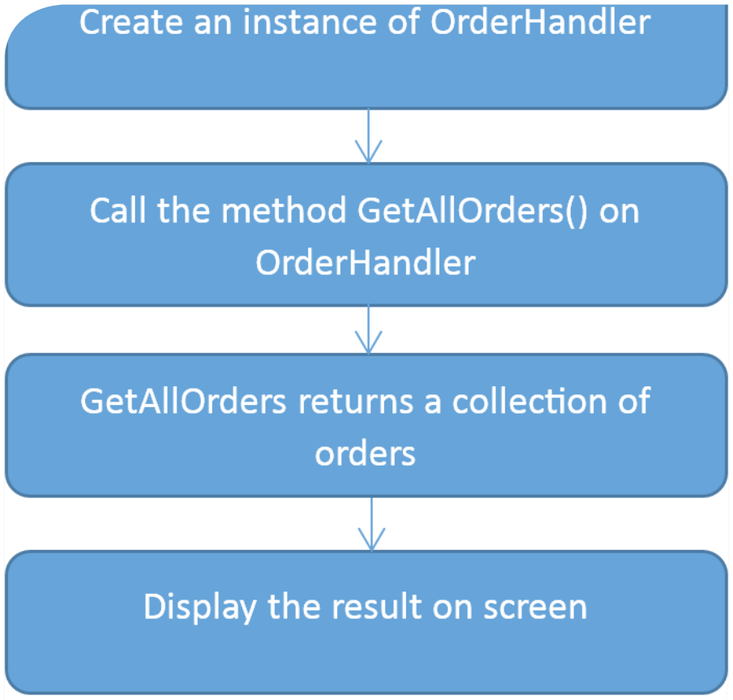


Figure 1

Humans are used to reading from top to bottom, even if it starts from left or right. If we have something that disturbs this flow of content, we are going to be confused and spend more efforts than what is really necessary. Event based applications often have these non-linear constructs.

The flow of an event-based system is such that it fires off a call somewhere and expects the result to be delivered through a raised event, can be visualized like the diagram in Figure 2. At a first glance, the two sequences might not seem very different but if we assume that GetAllOrders returns void, it would not be very straight forward how we retrieve the list of orders.

Without looking at the actual code, we can identify that the linear approach is much more pleasant to handle and it is less error prone. Errors in this case might not be actual runtime errors or compilation errors, but errors in usage; since the lack of lucidity.

There is one big advantage with the event based approach; it will let us conform to an event-based asynchronous pattern.

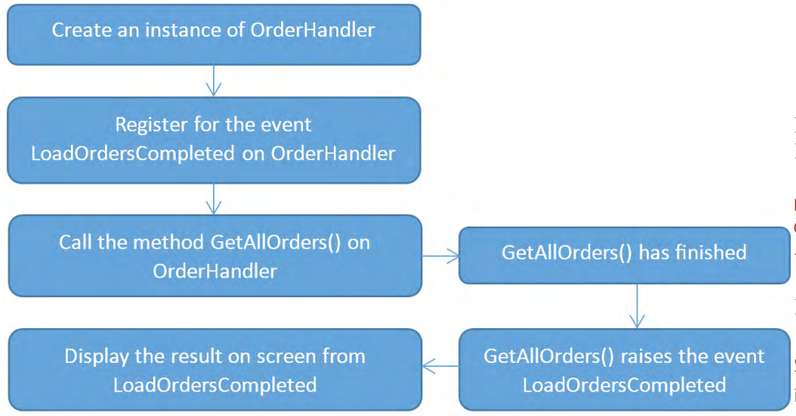


Figure 2

When you look at a method, you want to understand everything about the method's purpose. This means that if you have a method that is called ReloadOrdersAndRefreshUI, you want to understand where the orders are loaded from, how it's added onto the UI and what happens when the method ends. That can be hard to achieve with an event-based approach.
Another benefit from this is that we can write asynchronous code in GetAllOrders; as long as when we raise the LoadOrdersCompleted event, we are back on the calling thread.

Introducing a new pattern

Let us assume that we are working on our system that uses the OrderHandler mentioned before and the actual implementation will use a linear approach. To simulate a small portion of a real order system, the OrderHandler and Order will look like the following:

```
class Order
{
    public string OrderNumber { get; set; }
    public decimal OrderTotal { get; set; }
    public string Reference { get; set; }
}

class OrderHandler
{
    private readonly IEnumerable<Order> _orders;
    public OrderHandler()
    {
        orders = new[]
        {
            new Order {OrderNumber = "F1",
                       OrderTotal = 100,
                       Reference = "Filip"},
            new Order {OrderNumber = "F1",
                       OrderTotal = 100,
                       Reference = "Filip" }
        };
    }

    public IEnumerable<Order>
    GetAllOrders()
    {
        return _orders;
    }
```

Since we don't use a real data source in this example, we need to make it a little bit more interesting. As this is about asynchronous programming, we want to have something to ask for in an asynchronous manner. In order to simulate this, we can simply add

```
System.Threading.ManualResetEvent(false).WaitOne(2000)
in GetAllOrders:
public IEnumerable<Order> GetAllOrders()
{
    System.Threading.ManualResetEvent(false).
                                WaitOne(2000);
    return _orders;
}
```

The reason that we aren't using Thread.Sleep here is because this is going to be inside a Windows 8 Store Application. The goal here is to have a button that we can press in our Windows 8 Store Application which loads our orders into a list. Then we can compare the user experience and the code prior to the asynchronous code being added.

If you've created a new blank Windows 8 Store Application project, you can add the following XAML to your MainPage.xaml:

```
<Grid Background="{StaticResource
ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="140"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

<TextBlock x:Name="pageTitle" Margin="120,0,0,0"
  Text="Order System" Style="{StaticResource
```

```
PageHeaderTextStyle}" Grid.Column="1"
IsHitTestVisible="false"/>
    <StackPanel Grid.Row="1" Margin="120,50,0,0">
        <TextBlock x:Name="Information" />
        <ProgressBar x:Name="OrderLoadingProgress"
       HorizontalAlignment="Left" Foreground="White"
       Visibility="Collapsed"
       IsIndeterminate="True"
       Width="100">
            <ProgressBar.RenderTransform>
                <CompositeTransform ScaleX="5"
                      ScaleY="5" />
            </ProgressBar.RenderTransform>
        </ProgressBar>
        <ListView x:Name="Orders"
       DisplayMemberPath="OrderNumber" />
    </StackPanel>
    <AppBar VerticalAlignment="Bottom" Grid.Row="1">
        <Button Content="Load orders"
       x:Name="LoadOrders"
       Click="LoadOrders_Click" />
    </AppBar>
</Grid>
```

Before we can actually run the application, we need to add some things to the code file as well. There needs to be an event handler for the click event and we also want to set a default value on the information text block:

```
public MainPage()
{
    this.InitializeComponent();

    Information.Text = "No orders have been loaded
                        yet.";
}


private void LoadOrders_Click(object sender,
RoutedEventArgs e)
{
    OrderLoadingProgress.Visibility = Visibility.
                                      Visible;
    var orderHandler = new OrderHandler();
    var orders = orderHandler.GetAllOrders();
    OrderLoadingProgress.Visibility = Visibility.
                                      Collapsed;
}
```

This will give us a nice looking application that looks like the following image when we run it in the built-in simulator of Visual Studio 2012:

To see the application bar at the bottom, simply enable the basic touch mode by pressing this image  in the right hand menu, then swipe up from the bottom up.

Now when you press the Load orders button, you will notice that you will not see any loading indicator and that the button is left in its pressed state for 2 seconds. This is because we are locking up the application.

In previous C# versions, we could have solved this by wrapping the code in a BackgroundWorker. That when finished would have raised an event in which we had to invoke a delegate in order for us to change the UI. This is a non-linear approach which tends to mess up the readability of the code. In an older application that is not WinRT, using a BackgroundWorker would have looked something like this:

```
public sealed partial class MainPage : Page
{
    private BackgroundWorker _worker = new
        BackgroundWorker();
    public MainPage()
    {
        InitializeComponent();

        _worker.RunWorkerCompleted +=
               WorkerRunWorkerCompleted;
        _worker.DoWork += WorkerDoWork;
    }

    void WorkerDoWork(object sender, DoWorkEventArgs e)
    {
        var orderHandler = new OrderHandler();
        var orders = orderHandler.GetAllOrders();
    }
```

```
private void LoadOrders_Click(object sender,
RoutedEventArgs e)
    {
        OrderLoadingProgress.Visibility = Visibility.
                                          Visible;
        _worker.RunWorkerAsync();
    }

void WorkerRunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
    {
        Dispatcher.BeginInvoke(new Action(() =>
        {
            // Update the UI
            OrderLoadingProgress.Visibility =
                                Visibility.Collapsed;
        }));
    }
}
```

The BackgroundWorker is what is known as event-based asynchronicity, the pattern is called event-based asynchronous pattern (EAP). This tends to make the code messier than it could be and since it is also written in a non-linear manner, our brains will have a harder time getting a complete overview of it as fast as it could.

In WinRT however, there is no BackgroundWorker so we have to adapt to the new linear approach, which is only a good thing! Our solution to this is to adapt to the new pattern introduced in .NET 4.5, async & await. When we use async & await, it is mandatory that we are using it together with the task parallel library (TPL). The principle is that whenever a method needs to run asynchronously, we mark it as such. This means that the method will have something that we are waiting for to get back to, a continuation. The marking for where the continuation block will be, is made by defining an 'awaitable' section, hence we ask to await the task to finish.

Based on the original code, without the BackgroundWorker we can just make some small changes to the click handler in order for it to be used in an asynchronous manner. First we need to mark the method as async, this is as easy as just adding the keyword to the method signature:

```
private async void LoadOrders_Click(object sender,
RoutedEventArgs e)
```

Be very careful when using async and void together, the only reason that marking a method with async when the return type is void, is because of event handlers. Never mark a method as async when the return type is void if it is not an event handler! Async & await are always used together, if a method is marked as async and does not have something awaitable in it; it will simply run synchronously.

So the next thing that we want to do is actually having something that we can await, in our case this is the call to GetAllOrders. As this is what takes up the most of the time, we want this to run in a separate task. We can simply wrap the method call in a task that expects to return an IEnumerable<Order> like this:

```
Task<IEnumerable<Order>>.Factory.StartNew(() => { return
orderHandler.GetAllOrders(); });
```

The above is what we want to await, let's take a look at what we had from the start and how it compares to what we have now:

```
// Before
var orders = orderHandler.GetAllOrders();

// After
var orders = await Task<IEnumerable<Order>>.Factory.
StartNew(() => { return orderHandler.GetAllOrders(); });
```

When we add await in front of a task, the type that the variable orders will be is what the task is expected to return; which in this case is IEnumerable<Order>>. This means that all we had to do in order to make this method asynchronous was to mark it as such and just wrap the long running method call in a task. What happens internally is that we will get a state machine that keeps track of when the task is finished. Everything below of the awaitable block will be put into a continuation block. If you are familiar with TPL and the continuation on a task, this is similar to that except that we are back on the calling thread when we reach the continuation! This is an important distinction, because that means that we can have our method look like this, without any dispatcher invocations:

```
private async void LoadOrders_Click(object sender,
RoutedEventArgs e)
{
    OrderLoadingProgress.Visibility = Visibility.
                                      Visible;

    var orderHandler = new OrderHandler();

    var orderTask = Task<IEnumerable<Order>>.Factory.
       StartNew(() =>
       {
```

```
        return orderHandler.GetAllOrders();
    });

    var orders = await orderTask;

    Orders.Items.Clear();
    foreach (var order in orders)
        Orders.Items.Add(order);

    OrderLoadingProgress.Visibility = Visibility.
                                      Collapsed;
}
```

As you can see, we can simply change things on the UI after the awaitable block without using the dispatcher as we previously would have when using EAP or TPL. We can now run this application and load the orders without the UI locking up and then having a nice list of order numbers presented to us.

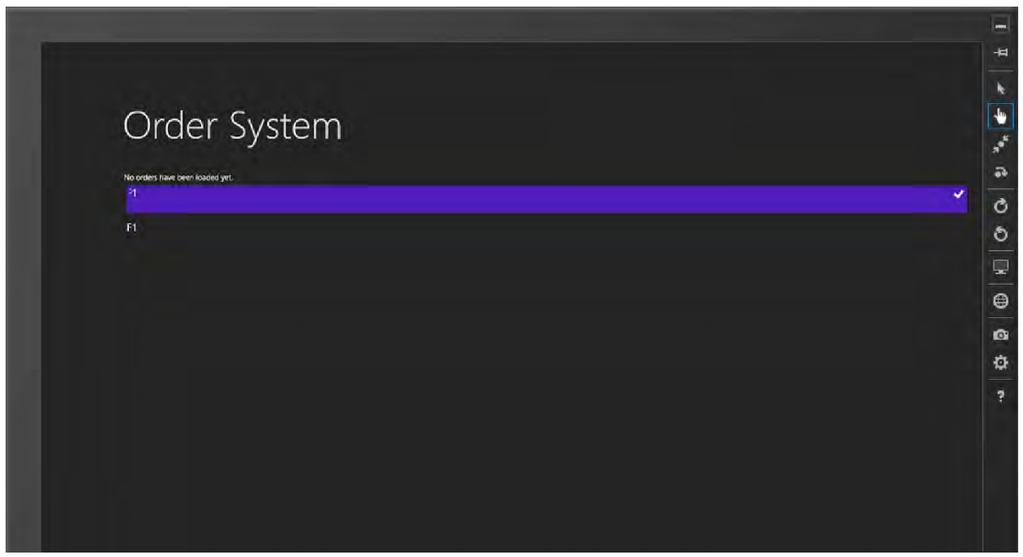
We can thus see that using this new approach will give us more readable and linear code; this is what we are most use to seeing. Of course, there are always ways to write less readable code even with the best of patterns. Async & Await sure helps along the way to create more readable and maintainable code.

CONCLUSION

Async & Await makes it very easy to create readable and maintainable asynchronous solutions. Prior to the release of this, we had to fall back on event-based methods which tended to introduce confusion. As we are in an era where almost any computer has at least two cores, even the mobile phones, we will be seeing a lot more parallel and asynchronous code. As this is getting much easier with async & await, it no longer has to be an issue in the development phase to introduce it. We can avoid cross-thread problems that often arise when using tasks or event-based asynchronicity due to not using the dispatchers or invocation capabilities. With this new pattern, we can step away from that way of thinking and focus on creating responsive and maintainable solutions.

Of course this is not the answer to everything, we still have problems that can arise and complex scenarios where this approach also can cause confusion. But as with anything, use it where it is appropriate and where it benefits the application life-cycle. ■

Filip is a Software Engineer with a big heart for programming in C#. Most recently Filip is the author of the book C# Smorgasbord, that covers a vast variety of different technologies, patterns & practices. You can Follow Filip on twitter @fekberg and read his articles on fekberg.com



CREATE A SNAPPY UI WITH KNOCKOUTJS

Gregor Suttie introduces KnockoutJS, a JavaScript library that bring tons of richness to your web application with features like 'Declarative Bindings', 'Dependency Tracking', 'Templating' and 'Observables'. Find out how you can bring all this goodness in your ASP.NET MVC apps.

KnockoutJS (KO) is a JavaScript library written by Steve Sanderson who works for Microsoft and is the author of Pro ASP.NET MVC Framework.

It is a JavaScript library that helps build apps conforming to the Model View View-Model pattern. KO makes it easier to create rich, desktop-like user interfaces with JavaScript and HTML.

It uses observers to make your user interface automatically stay in sync with an underlying data model, along with a powerful and extensible set of declarative bindings to enable productive development.

In short if you want your website to have a snappy, slick user interface, with less code, then KO is certainly a good library to use.

KNOCKOUTJS FEATURES AND EXTENDING KNOCKOUTJS

KO works along-side any JavaScript library, so you can use jQuery and other JavaScript libraries along with KO as your application starts to grow. We first take a look at the features of KO and then dive into a sample that helps explain these features.

DECLARATIVE BINDINGS

KnockoutJS gives us declarative bindings which means we can easily get access to HTML DOM elements on our page and use it with any web framework including PHP, Ruby on Rails and even with ASP. It's free, open source with no dependencies and supports all mainstream browsers IE6+, Firefox 2+, chrome, Opera and Safari.

DEPENDENCY TRACKING

Dependency Tracking comes with KnockoutJS so that you can chain relationships, between your model data which lets you transform your data when a part of the relationship change gets updated.

TEMPLATING

KnockoutJS has inbuilt templating, as well as the ability to use jQuery templating or your very own custom templating.

KnockoutJS comes with a number of built-in bindings and these make life really easy and straight forward. These include bindings for controlling text and appearance, control flow, and form fields.

With KnockoutJS, you can create your own custom bindings and this really could not be easier, so with KnockoutJS we extend it with using custom bindings and templates and these templates can include jQuery templates or plain old HTML templates.

Note that using external templates can slow down the rendering slightly and in my opinion, it's advisable to try to use inline templating where possible.

CUSTOM BINDING

As developers, we are always looking for something that adds that little bit more value to our websites to make them snappier, have a nicer user interface, with less code. KnockoutJS gives us the ability to load our data for the page and have KnockoutJS bind the data to the user interface using simple elegant bindings.

If the data changes or the user makes a change to the webpage, KnockoutJS has 2-way bindings that updates the page and reflect the changes on the user interface, all very seamlessly for you.

If you have used jQuery, you might be thinking - hang on jQuery does this for me already! Yes you'd be correct, but KnockoutJS can simplify things even further. With KO, you can use a number of bindings for a whole range of things and also extend these to create your own easily. Shown below is how you define a custom binding.

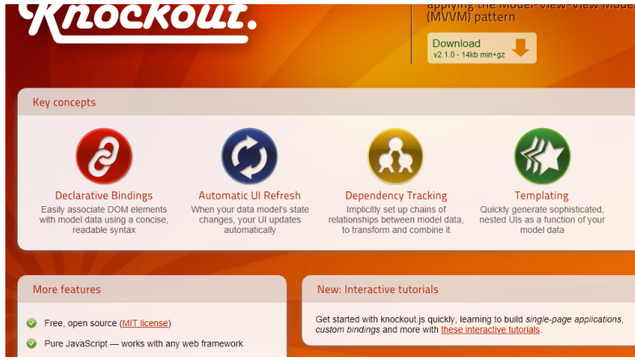
```
ko.bindingHandlers.myCustomBinding = {
   init: function(element, valueAccessor,
            allBindingsAccessor, viewModel)
   {
      //init logic
   },
   update: function(element, valueAccessor,
        allBindingsAccessor, viewModel)
   {
        //update logic
   }
};
```

> *"Knockout simplifies dynamic JavaScript UI's by applying the MVVM pattern"*

HOW DO I GET KNOCKOUTJS?

Microsoft is shipping KO as a part of the ASP.NET project templates in Visual Studio. So any new project will have KO added to it via a NuGet package reference. If you're not working on a new MVC 4 application, you can add it into any web application by using NuGet as below :

```
PM> Install-Package knockoutjs
```



Or by downloading the .js file from its home on Github https://github.com/SteveSanderson/knockout/downloads, add the knockout .js file to your application and reference it in your webpage and you are good to go.

USING KNOCKOUTJS IN ASP.NET MVC

When using KnockoutJS within a web application, there are a couple of things you can do to make life easier and these are really some best practice advice points from using it, which I have found useful.

If you create a new MVC web application in VS 2012, the solution loads and if you take a look at the contents of the packages.config file, we can see that as part of the solution, we are pulling in a number of NuGet packages, one of which is:-

```
<package id="knockoutjs" version="2.1.0"
targetFramework="net45" />
```

Here we are using NuGet to pull in version 2.1.0 of the KnockoutJS library, which at the time of writing this article was the latest version. So straight out of the box, our solution has a reference to KnockoutJS and we haven't had to do anything – we are off to a great start.

THE DEMO

Our demo application is a simple shopping cart order page where we can update the quantity of items in our shopping basket and see a running total being updated. The page has the data passed in from our MVC Controller Action Method and this is then presented to the page.

Demo has a ProductController, a set of entities like CartItem, Category, Model, OrderItems and Product, and the Index.cshtml for the UI. Custom scripts are in the JS folder and we have ajaxservice.js, dataservice.shopping.js, index.js and utils.js.

To start with, when using KnockoutJS, it's a good idea to create a namespace and use it within your JavaScript to keep things nice and tidy, same as you would within your C# codebase. To create a namespace in JavaScript is as simple as:

```
var OrdersApp = OrdersApp || {};
```

THE VIEW MODEL

Next we need to create our ViewModel. It's in the index.js file. A typical example of the ViewModel we might use for our shopping cart, is as follows:-

```
$(function () {
    OrdersApp.Product = function () {
        var self = this;
        self.id = ko.observable();
        self.price = ko.observable();
        self.category = ko.observable();
        self.description = ko.observable();
};

// The ViewModel
OrdersApp.vm = function () {
    products = ko.observableArray([]),
    shoppingCart = ko.observableArray([]),
    addToCart = function (product) {
        // Stub
    },
    removeFromCart = function (cartItem) {
        // Stub
    }
},
grandTotal = ko.computed(function () {
    // Stub
}),
loadProducts = function () {
    // Stub
};

return {
    products: products,
    loadProducts: loadProducts,
    shoppingCart: shoppingCart,
    addToCart: addToCart,
    removeFromCart: removeFromCart,
    grandTotal: grandTotal
};
}();

    OrdersApp.vm.loadProducts();
    ko.applyBindings(OrdersApp.vm);
});
```

This is a gist of the complete ViewModel, but what we have here is a pure JavaScript representation of the model data (i.e. products and a shoppingCart) and actions to be performed.

The ko.applyBindings() statement is used to tell KnockoutJS to use the object as the ViewModel for the page.

DATA BINDING AND TEMPLATING

Products are defined as an observableArray and this means that KnockoutJS will track what's in this array. For example, we can push() and pop() items onto the products observableArray and the front end will automatically show us the updated data due to the 2-way binding KnockoutJS has – you can even use the console within say the Chrome browser to manipulate the items in your ViewModel and KnockoutJS will take care of updating the user interface for you - it's that simple.

In order to display a list of Products, we would be able to use the built in for-each binding and display our products as follows.

```
<ul class="saleItems leftFloat" data-
bind="foreach:products,
        beforeRemove:hideItem, afterAdd: showItem">
 <li class="mediumProductSquares" >
  <div>
    <div class="dialogTitleBorder">
     <span class="borderTitleText">Details</span>
    </div>
    <div class="leftFloat">
     <div>
      <span>Make: </span><span data-bind="text:
         model().brand"
        class="textValues"></span>
     </div>
     <div>
      <span>Model: </span><span data-bind="text:
         model().name"
        class="textValues"></span>
     </div>
     <div>
      <span>Price: </span>
      <span data-bind="text: OrdersApp.
        formatCurrency(price())"
        class="textValues">
      </span>
     </div>
    </div>
   <button data-bind="jqButton: { }, click: $root.
        addToCart">Add
        Item</button>
  </div>
 </li>
</ul>
```

Note that KO also uses the data-bind tag to specify the type and field to bind to in the ViewModel.

The elements inside the foreach data-bind are treated as the 'row-template' and repeated for each 'product' in the Products collection.

CHANGE TRACKING

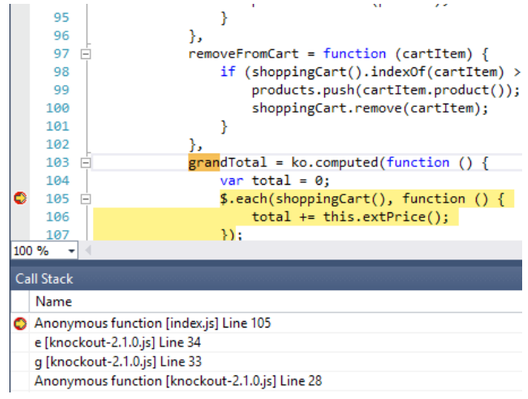
As we just saw, each of the 'Add Item' buttons invoke the $root.addToCart method. However on addition, the Total gets updated automatically. If we look at the markup for rendering the cart total, we will see the following

```
<div class="cartSummaryContainer">
 <span>Total Items</span><span data-
bind="text:shoppingCart().length"></span>
 <span>Total Price</span><span data-
bind="text:OrdersApp.formatCurrency(grandTotal())"></
span>
 <button data-bind="enable: shoppingCart().length >
0, click: $root.placeOrder">Place Order</button>
</div>
```

As we can see here, Total Items and Total Price is bound to the calculated fields of shoppingCart().length and value returned by the grandTotal() function. In Index.js, we will see the grandTotal method is defined as follows.

```
grandTotal = ko.computed(function () {
 var total = 0;
 $.each(shoppingCart(), function () {
  total += this.extPrice();
 });
 return total;})
```

Essentially we have defined it as a KO computed value that's calculated for all the elements in the shopping cart. KO 'observes' for the changes in the number of shoppingCart items and on change, computes the grandTotal. Once the grandTotal value is computed, KO updates the UI because of the binding. If we put a breakpoint in the above function and add an item to the cart, we can see how KO calls the computed function automatically.



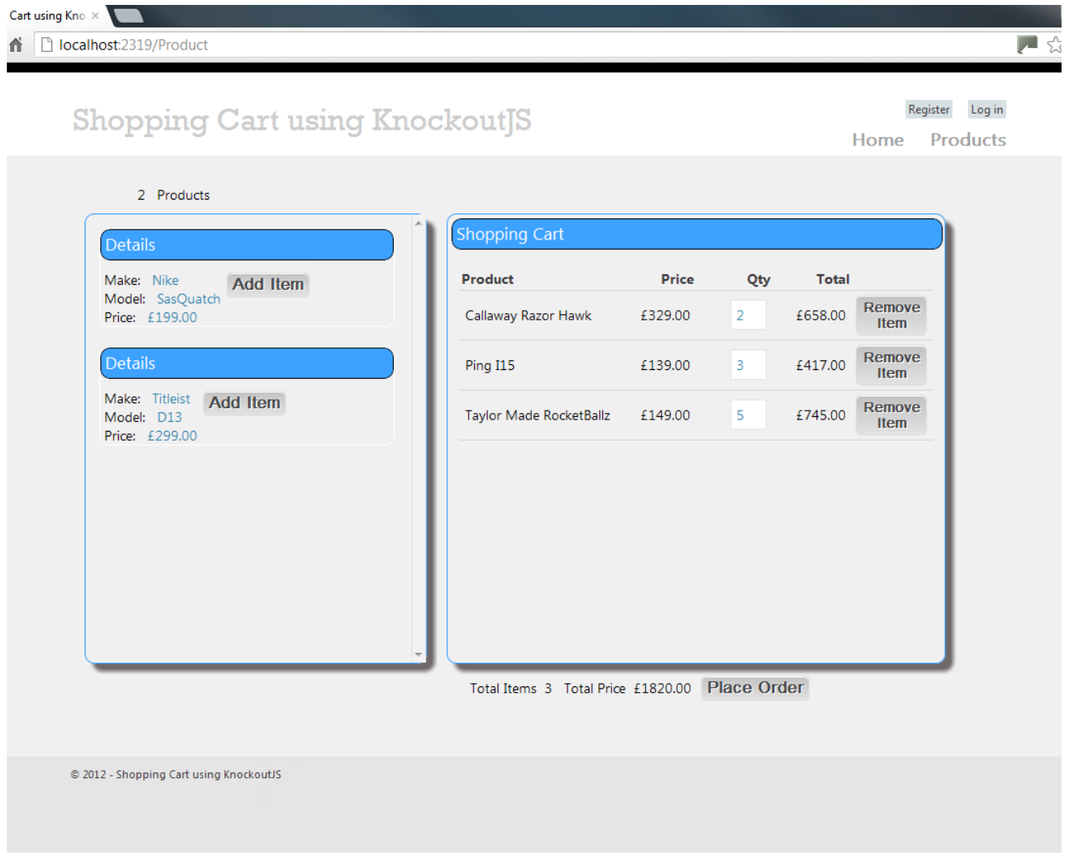
Thus change tracking and two way data binding provide a rich and responsive behavior where changes in one area of the application gets reflected immediately elsewhere.

The demo application to show the basic flow using KnockoutJS code can be found on Github here: http://bit.ly/dncmag-snapko

WHAT KNOCKOUTJS IS GOOD AT?

KnockoutJS is ideal for creating a user interface that responds immediately to the user and that includes adding and removing data. You don't have to wait on server postbacks and hack away with viewstate or use any of the older tricks such as Update Panels and similar ones.
You set up your ViewModel, then fill it with your data and then the user interface is updated using the 2-way binding done by KnockoutJS. It's quick, responds immediately and makes the user experience a whole lot better than it was before the introduction of KnockoutJS.

KnockoutJS gives us the added benefit of separation of concerns and the fact that it allows for better unit testing of the user interface code – we can now test our JavaScript with a tool such as QUnit http://qunitjs.com/ allowing us to add into our build server so we can run the user interface unit tests before deployment.



The complete demo shows a shopping cart webpage as shown above where you add products to your basket and can update the totals and a basket total is calculated. You can also remove items from the basket and the totals are all kept in sync using KnockoutJS.

The example code covers use of observables and observable arrays, computed functions, namespaces in your JavaScript and how to use callbacks.

SOME GOTCHAS TO AVOID WHEN USING KNOCKOUTJS

Having discussed what KnockoutJS is good at, we should cover what problems you might run into using KnockoutJS - it's actually something to be careful of within JavaScript.

Scoping in JavaScript is a minefield and can really give you sleepless nights if you're not careful, so a little discipline is required in order to try to avoid falling into some scoping issues when using JavaScript.

When using JavaScript, there are thankfully a few JavaScript patterns you can use and one is the Module Pattern which is useful for organizing independent, self-containing pieces of JavaScript – you can read more about the Module Pattern here http://elegantcode.com/2011/02/15/basic-javascript-part-10-the-module-pattern/

Be careful when using the 'this' keyword in JavaScript as you can run into issues easily with the context of the keyword, depending on how you structure your JavaScript.

For a very good tutorial on how to go about structuring your JavaScript that you will use when working with KnockoutJS as well as other great tips, I recommend you take a look at Rob Connery's Tekpub course here: http://tekpub.com/productions/knockout

WHERE CAN I LEARN MORE?

In this article, I covered an introduction to KnockoutJS. Although we covered a fair amount of work, it hopefully left you wanting to know more. I have listed some additional learning resources over here bit.ly/dncmag-snapko in the Readme section.

SUMMARY

In summary, KnockoutJS is a fantastic addition to your arsenal as a web-developer when trying to create a slick user interface that responds immediately. You can use it with any web framework and it's a one file addition to your solution which makes it easy to update if newer versions come out.

With KnockoutJS, you get superb tutorials and there are more articles and tutorials popping up all the time. There is no reason not to give it a try as its super easy to get going with, just be mindful of scoping. Also try to refactor your JavaScript code at all times. If you find yourself writing a lot of JavaScript when using KnockoutJS, the chances are there is a better cleaner way (normally methods are only a couple of lines).

Spend a couple of hours using KnockoutJS and you'll wonder why you're not using it on every web project – you can even go back and add it into your old web applications and improve the user experience with ease. ■

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

INTERVIEW WITH JON GALLOWAY

Dear readers, this is the third issue of the 'DNC Magazine' and so far we have had the privilege of hosting a .NET Community personality in each of our issues. First it was Ayende Rahien, then it was Jon Skeet. In this issue, we are very glad to 'converse' with Jon Galloway from the Windows Azure Evangelism Team at Microsoft. Though we are an out and out tech magazine, we try to theme the interviews more as *'know your favorite techie better'*. This time is no different, and as we 'talk' to Jon, he opens up and talks about Nuclear Submarines to Bass guitar and of course ASP.NET and Windows 8. Without further delay, let's jump in.

DNC: Hello Jon, thanks for taking some time out for the interview. To start off with, we heard, before getting into software you were in the U.S. Navy, submarines no less. Tell us more about how that happened (maybe some memorable adventures...) and how did the career in Navy morph into a career in Software?

JG: My older brother went to the US Naval Academy (Annapolis), so when I started looking at colleges it was on my list. I decided to go there for a few reasons: I could afford it without a college loan (it's free with a 5 year Navy commitment after graduation), they have great technical programs, and it sounded like an adventure. It sure was – even before graduating, I'd backpacked around Europe, flown a helicopter, and ridden on submarines and ships. After graduating, I somehow got through Navy Nuclear Power School (the hardest thing I've ever done) and ended up on the USS Jefferson City, an attack submarine home ported in my home town, San Diego. We did two Western Pacific deployments, so I got to see Japan, Hong Kong, Thailand, Korea, Singapore, Guam and Bahrain.

There was a lot of excitement while I was on board. Just a few months after I'd reported aboard, we scraped a seamount going fast and deep (http://nyti.ms/U87fQi). No one was hurt, but they said that if we'd been just ten feet deeper, things wouldn't have turned out so well.

Editor's Note: A 'Seamount' is an underwater mountain that does not rise above the ocean's surface, and thus is not an island. 'Scraping a seamount' for a submarine can be a very dangerous thing.

Overall it was an exciting job, but it was hugely stressful – a lot more stress than fun. I got out when my five years were up. My last few years on the ship I saved up all the money I could so I could find a job I liked – I called it The Galloway Freedom Fund. I'd always enjoyed computers – I taught myself to program four languages in high school, tested out of all the computer requirements at the Naval Academy, got booted off the Cray at Los Alamos National Lab for running a nuclear fission simulation that went supercritical, and so on, but until leaving the Navy I'd never thought of it as a job – it was just too fun to be a job, right? I read a lot of books to brush on what seemed like marketable languages – C++, Java, and Visual Basic. When I got out, I called around and started at an intern level job making $5 per hour working on Visual Basic desktop applications. After a few years, I shifted over to internet development and finally felt completely at home.

Even before graduating, I'd backpacked around Europe, flown a helicopter, and ridden on submarines and ships.

DNC: That is certainly a remarkable journey, thanks for sharing. Now, as a Software Engineer first and a Tech Evangelist next, what's a typical day for you like?

JG: I worked in a variety of computer jobs for twelve years before joining Microsoft. During that time, I worked in a lot of sectors, especially financial, real estate and medical. I spent that time writing a lot of code and as team lead, coordinating teams of three to fifteen people.

When I joined Microsoft, that all changed. I was no longer shipping large internet applications, and I didn't have anyone working for me. I'd spent a decade getting good at coordinating technical teams to achieve a goal – well, almost two if you count my submarine time, where I was in charge of smart people like nuclear and radio technicians. Now that team lead thing was over, and anything I want done, I pretty much had to do myself. That's been a change.

On a typical day, I start by reviewing my weekly and daily goals – I try to follow J.D. Meier's Getting Results system as much as possible. The particular work I'm doing each week – or even day – varies quite a bit. Some main areas I focus on:

• Creating content for ASP.NET developers. I work on things like tutorials, screencasts, and blog posts.
• Speaking at conferences. Really doing this right requires a good amount of prep time before the conference to deeply understand the material, write some good demos, etc.
• Contributing to the ASP.NET web site. I work with the ASP.NET team and site dev team to strategize on content, fix things that are broken, coordinate content for releases, selecting daily community spotlight posts for the home page, and things like that.
• Lining up, recording, and editing podcasts. We usually record in the evenings, but there's a good amount of coordination to schedule interviews, and I often edit the audio while working on other things.
• Supporting field evangelists and support staff. I answer a lot of questions on the internal ASP.NET mailing lists, which has led to a lot of opportunities to help our internal staff prepare presentations or recommendations for customers. I also get requested pretty regularly to meet directly with large ASP.NET customers to give recommendations on migrations, specific issues they're facing, best practices, and things like that.
• Trying to support the ASP.NET – and .NET – community in general. I try to help people find answers, promote exciting projects, and just generally try to help make it fun to be a .NET developer.

But there's other random fun stuff that pops up, too. Sometimes Scott Hanselman calls to talk about a blog post he's working on, or there's an internal briefing about something that's not going to be released for several months, or I'll get invited to review something that another team's working on.

DNC: That's an amazing amount of stuff, I will definitely be looking closer at the 'Getting Results' system soon. Moving on to your latest move inside Microsoft, tell us more about your new role in the Azure team.

JG: I joined the Windows Azure Evangelism Team in May 2012. It's an amazing team, packed with pros who know their technology areas backwards and forwards, speak at conferences all the time, and work closely with both product teams, conference keynote teams, etc. It's a high powered team.

They've been incredibly supportive. My focus remains on ASP.NET as a platform, with the idea that if people are building ASP.NET applications, they'll likely want to host a lot of them on Azure. I originally expected I'd be slowly shifted to selling Azure door to door or something, but that hasn't been the case at all. They're happy as long as I'm building the ASP.NET community, as am I. So it's a great fit.

What's nice is that the evangelism teams have some actual budget, so we can work with vendors to get additional content and other work done. I'm still wrapping my head around it, but it's nice to start to head back to a world where I'm not the bottleneck.

DNC: That's cool. Digging a little deeper into ASP.NET; in the latest release of Web Forms, we see lots of new features that actually are on par with what MVC has. So where do you see Web Forms headed, are we going to see a gradual convergence of features?

JG: I worked in Web Forms from version 1.0 and have enjoyed watching it mature through the years.

> ***I think MVC is one of the best things that's happened to WebForms, and the improvements in ASP.NET 4.5 is a proof to that.***

The new model binding and strongly typed data controls feel kind of like ASP.NET MVC, but scoped to the control level rather than the entire page. Individual controls call methods that look basically like controller actions, the control markup fragments are like the views, and you can use the same models you'd use in an ASP.NET MVC app, even including the validation attributes. It's not exactly the same, of course, because the controls have properties and events that can affect their behavior, and the model binding support is completely optional. But if you want to use the MVC pattern within Web Forms, you can.

And there's a lot of benefit to using the MVC pattern in Web Forms. You get a better separation of concerns, you can get better code reuse by focusing on code that operates on models rather than control properties, and you eliminate tedious (and error prone) mapping variables to control properties. No more var firstName = txtFirstName.Text!

Maybe the biggest advantage is that it's now a lot more realistic to think about hybrid Web Forms / MVC apps. If you've got a Web Forms application that's got models and services that operate on those models, you really can consider adding a new controller that's strongly typed to that model. I've given some recent talks about hybrid applications, and my top recommendation was to get to Web Forms 4.5 first.

DNC: As someone who has used MVC and been closely associated with the team that has developed it, what would your wish-list for MVC (vNext/5) be?

JG: Hmm, that's hard to say. A lot of the exciting things that first come to mind, like Single Page Applications and SignalR are already on the roadmap (http://aspnetwebstack.codeplex.com/wikipage?title=Roadmap). A lot of things I'm personally interested in are already available in NuGet packages (most from the community) and I don't see much benefit in cramming them into the MVC core. MVC has been leveraging popular open source libraries via NuGet in the past few releases, so there might be a benefit in pulling in some libraries like AttributeRouting and FluentSecurity. Then again, they're easy to install via NuGet, so the only real benefit would be that more people would be exposed to them.

I'd be interested in an HTML5 focused template – something conceptually similar to Twitter bootstrap with great CSS3 and HTML5 support, some HtmlHelpers, etc. That's one thing that could possibly be better as a full project template.

Oh, one big area is localization. There are some community projects out there – I'm partial to http://code52.org/aspnet-internationalization/tutorial.html – but it'd be nice to see localization in the core.

DNC: SPAs and SignalR are definitely very exciting, their official inclusion in the ASP.NET stack will be the cherry on top. Coming to authoring books (you already have co-authored a couple), most technical authors give the impression that the efforts outweigh the returns. What's your take on authoring books? Are you currently authoring any books?

JG: None of the technical authors I know make much or any money on books anymore. I'm actually a very slow writer, so it's even worse for me. I'm certain if I did the math I'd be making far less than minimum wage in a straight dollar per hour accounting.

But that's not why I write technical books. I write books because it forces me to learn a technology in depth. To explain something well, I really have to get to know it. First I have to understand the big picture – how the parts fit together, what's most important, how one thing affects another – in order to figure out how to tell the overall story. That drives the book table of contents, then the chapter outline. Then as I describe things in more detail, write the code samples, and dig into the source code, I get to know how things really work behind the scenes, and sometimes that bubbles up to changing my high level understanding and thus the table of contents. So writing a book is a very structured approach to gaining both broad and deep knowledge of a technology. It's the same reason I write blog posts and speak at conferences: to do it right, you have to know your stuff. It keeps me sharp.

I write books and speak at conferences because I want to know enough to be able to write a quality book or deliver a great presentation. And, of course, because I like to teach people. I love learning and then sharing what I've learned – and learning more from that experience.

I just got done with Wrox Professional ASP.NET MVC 4. It's been great to work with Phil Haack, Brad Wilson, and K. Scott Allen on that series. I'm planning to work on the next release of the book.

Jesse Liberty recently talked me into co-writing Pro Windows 8 Development with XAML and C# (Apress). I'm not sure how he did that, but I'm excited about it. I've been learning a lot about Windows 8 development in my spare time, and this is going to push me to get to that expert level. And Jesse's a phenomenal author, so I'm honored to have the opportunity to work with him.

> ***None of the technical authors I know make much or any money on books anymore.***

I'm also wrapping up the MVC 4 version of the MVC Music Store tutorial. It's not a printed book, but its 150 pages long so it's a lot of work to update.

DNC: That's very nicely put, seems to reflect the (Indian) adage, 'spreading knowledge increases it'. Coming to Windows 8; as a user, the more I use Windows 8 the more I like it, it still frustrates me at times, but I am eagerly looking forward to the commercial launch. As a regular Windows user and a developer, how amped are you about Win8? Do you think some of the flak it has got especially from entrenched desktop users has merit? Or do you think people will come to love it once they actually use it?

JG: I like it. I've been using internal builds since before the public release and have had some time to get used to it. There are a few things that bug me, but on the whole I like it a lot better than Windows 7, and that's saying something.

Back in the Windows Vista betas – long before I was at

Microsoft – I evaluated what others were saying and how I felt about it. I liked some things and disliked others. Then I figured, look, I'm going to be using this eventually, right? I mean, I'm not going to stay on XP forever. So why don't I just go with it and make a real effort at using it. If there are things I really don't like, I'll figure out how to work around them, but for the most part I'll try to get used to them and see how it all works once I get used to it. And it turned out that Vista worked just great for me once I got used to it. It was a huge improvement to XP. Also nice was that when Windows 7 hit, I was already familiar with a lot of it.

I've been trying the same approach with Windows 8. Most of the commentary you'll see on Windows 8 – both good and bad – doesn't seem to be based on any day to day experience with it, or in trying to learn how and why it's changed.

I find the "classic desktop" part to be a big improvement over Windows 7 – especially the Explorer, file management, and multi-monitor features. That's where I spend 99% of my time, and it's really improved. Jeff Atwood said "Windows 8 file copying is approximately one billion times better than Win7. It's really nice. Finally, some attention to detail in Redmond." I'd say a billion is on the high end, but it's a lot better, and I smile every time I use it.

I didn't really get the Start Screen idea until I read the blog post series on the Building Windows 8 blog explaining why they did it, and it makes sense for me. Back in XP, I got tired of mousing through the Start menu and started using launcher apps. When Vista came out, I was happy to be able to run everything from Start / Search exclusively, so I haven't been using the Start menu at all for about five years. It's really a pretty clumsy, slow way to get to things; it's so much easier to type a few letters (e.g. Win+"vis"+Enter to launch Visual Studio) and go. The traditional desktop icons are also kind of a mess – you either have to waste time organizing them or you end up with a big jumble of icons and files; the whole thing's pretty much a waste. So I'm ready to try out something that might be better than both the Start menu and the desktop.

So with Windows 8, I see the Start Screen as a full screen launcher that incorporates an actually useful desktop view. Regardless, after logging on, I generally only see it flash by as I use it to launch programs.

There are a few things that bug me here and there. The classic desktop and the new Start Screen seem to be more different

I am happy that it finally seems like .NET and Windows are made by the same company!

than they need to be. At times it can feel like you're flipping between two virtual machines. I don't really like the way the Start Search menu groups things into Apps, Settings and Files, but maybe that's just because I'm not completely used to the new shortcuts (e.g. Win+F to search for files).

Overall, though I think Windows 8 is a huge improvement on Windows 7. It's really nice when you have it installed on a few computers with one account and use the SkyDrive integration. Everything's in sync all the time, and setting up a new computer is incredibly fast: install Windows, all my settings and files show up, and all I need to do is install a few programs.

DNC: From a developer perspective, there was a lot of angst over the introduction of WinRT. I have used it now and it doesn't seem too much different from usual Windows development and the C# + XAML is a lot like WPF development. Though I feel it has a few rough edges. You mentioned you were co-authoring a Windows 8 book with Jessie Liberty so what's your take on the Windows 8 developer story?

JG: I've written a few apps. For the most part, I'm happy that it finally seems like .NET and Windows are made by the same company. There are a few things in WinRT that seem arbitrarily different from .NET, and that's a little frustrating. It's not just that it means I have to learn something new or change my code, but some common libraries can't be shared. I've seen some shims that help with this and I'm sure we'll see more of them, but I wish they'd given that higher priority.
But on the whole, the developer story is really nice. It's great to create a new project, use my XAML skills, pull in a few NuGet libraries, and things just work.

DNC: I think you have nicely summed up feelings of devs who are currently developing Win8 apps. Going on slight a tangent, let's do some 'crystal ball gazing' (and again just as a developer), let's consider the WinJS and JavaScript+HTML5 desktop development model of Windows 8. These apps run in an 'application container' that is practically IE10 under the hood. Do you think Windows 8 puts a foot in the door of 'boot to the browser' or 'boot to the internet'? Maybe Windows X (where X > 8) will eventually have a browser as it's 'desktop' and full screen modern apps will be hosted by the same application container that hosts desktop as well as internet applications?

JG: That seems farfetched to me. I think they tried that idea with the ActiveDesktop thing a long time ago and it didn't really catch on. Maybe the browser technology wasn't ready?
I think the important thing to keep in mind is that, at least in Windows 8, WinJS uses the IE10 rendering and JavaScript engines, but the app model is completely different. Just like a WinRT app, your application becomes an AppX package that's installed and executed by the modern Windows 8 desktop thingie (or whatever we're calling it now). So there's no client / server app model, and you're not using IE10 as a browser at all – it's just a scriptable rendering engine that happens to use HTML5, CSS3, and JavaScript.

I don't know anything about where they're heading with WinJS, and I wouldn't be surprised if some of that depends on the WinJS uptake by developers. I like your idea, but I wouldn't bet on it. We'll see…

DNC: Agree on the Active Desktop part, it was probably before its time, both with respect to browser technology as well as hardware capabilities. But as I said, it's wishful thinking, so we'll see :). Back to reality, when you are not evangelizing awesome web technologies (and officiating geek cage fights ;-)…) how do you like to spend your time? *(Editor's Note: For readers who have not been to, or seen recordings of NDC Oslo 2012, Jon officiated a 'cage fight' between Damien Edwards and Rob Connery showcasing SignalR vs. NodeJS)*

JG: I've got kind of a funny work-life balance. I work from home and my wife homeschools our three daughters (5, 9, and 11) so we spend a lot of time together. Sometimes it seems like I barely leave the house… and then I fly off for a week at a conference or a trip to Redmond. So I like to think I commute about as much as the average person, I just get it done all at once.

I'm usually on my laptop a good amount on nights and weekends, editing podcasts or working on a book or blog post. But I do try to force myself to unplug a bit. I like to jog and cook (though not both at the same time) when inspiration hits. I've been reading fiction lately – I recently finished 'Physics of the Future', 'Rainbow's End' and 'The Door Into Summer'. I am currently reading 'Cloud Atlas' and 'Trouble Is My Business'. I've been enjoying cold war spy films lately – 'The IPCRESS File' (and the rest of the Harry Palmer series), 'Three Days of the Condor', movie adaptations of John Le Carre's George Smiley. I love the pacing of those movies compared to the constant-stunts-as-music-video style in today's spy movies.

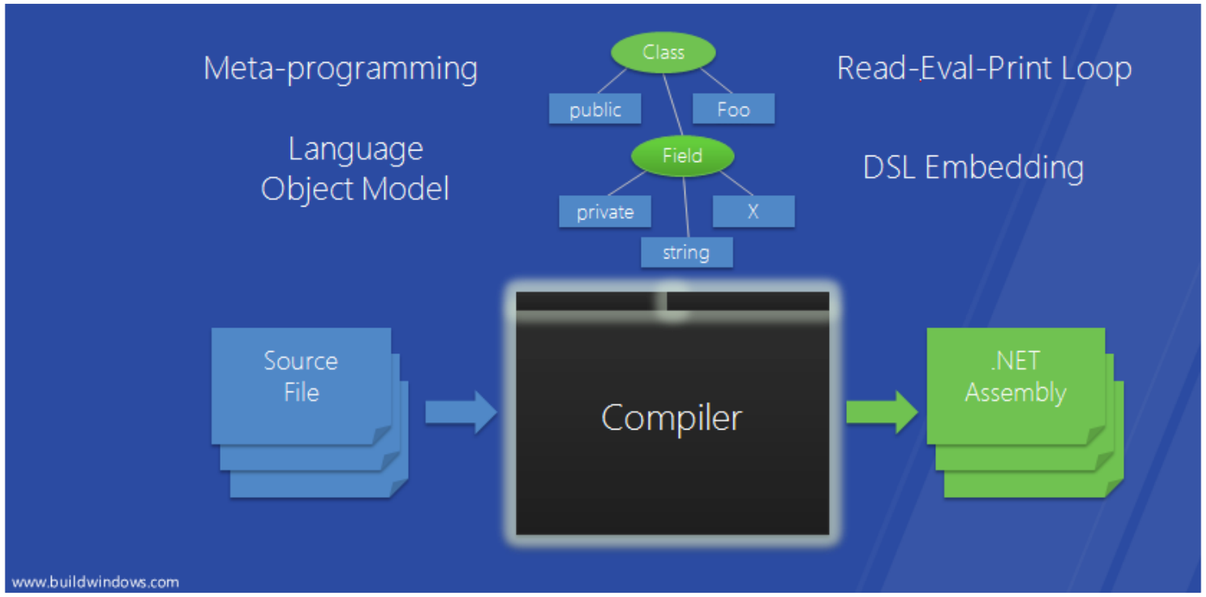
I like to play music, too. I've played bass guitar on and off over the past fifteen years in my big brother's band, Soul-Junk, which is fun. He's really innovative and hops between genres quite a bit (low-fi indie to math rock to free jazz to jungle rap and on and on) so he keeps me on my toes. I used to do a lot of digital audio editing and recording, and lately my wife has been getting into that as she's picked up the guitar, so we've been spending time with a recording program called REAPER. I've been getting back into playing keyboard a bit, too. I recently got to play a beautiful vintage Hammond organ on my father-in-law's album, due out soon. That was a blast.

DNC: Nice! That sounds like a sweet and musical family. We cannot help but ask this question to round off, did you get a chance to checkout the previous editions of DNC Magazine :-)… As an author, tech lover, any suggestions for us?

JG: I like that you've moved your code samples from images to text. I guess the only thing I'm interested to see in the future is easier ways to get each editions – maybe in a Windows 8 app, on my Kindle, etc. I think it's a great magazine and would recommend it to any .NET developer.

Yes, I've read every issue of the DNC Magazine cover to cover. I really like the technical depth and the article topics. I'm a really big fan!

Thanks for your kind words and suggestions Jon, this is what keeps us motivated. We are working on adding more distribution mediums for the magazine.■



MEET ROSLYN

MICROSOFT'S COMPILER AS A SERVICE PLATFORM

Microsoft MVP, Anoop Madhusudanan provides a first look at this exciting new technology in the .NET world. Roslyn provides the means to use C# and VB.NET as scripting languages and use the respective compilers as a service. Currently in CTP, we look at some of the new possibilities that Roslyn opens up for us.

Roslyn provides language services and APIs on top of .NET's compiler services. This enables .NET developers to do a lot more things – including

- Using C# and VB.NET as scripting languages
- Use compiler as a service in your own applications for code related tasks
- Develop better language and IDE extensions etc.
- Write code analysis and manipulation tools around Visual Studio IDE

Microsoft recently released Roslyn September CTP (CTP 3), which previews the upcoming features of C# and VB.NET. Roslyn CTP is a pretty exciting release, and it opens up lot of possibilities for C# and VB.NET programmers.

GETTING STARTED WITH ROSLYN CTP

There are a couple of ways to start with Roslyn. You can either fire up a Console project in Visual Studio, and install Roslyn via Nuget.

```
PM> install-package roslyn
```

Or, you can download and Install Roslyn CTP from the below link, that'll add the Roslyn Project types in Visual Studio.
http://www.microsoft.com/en-us/download/details.aspx?id=34685

Roslyn APIs will have full fidelity with C# and VB for syntax, semantic binding, code emission, etc. – and we hope to see a lot of pre-compilation language bending around C# and VB.NET, including new DSLs and may be few of Meta programming libraries (Not to mention the Roslyn dependent VS IDE extensions that's easy to develop).

HIGH LEVEL FEATURES

- Scripting APIs - Provides a runtime execution context for C# and VB.NET, Now you can use C#/VB.NET in your own applications as a scripting language
- Compiler APIs - For accessing the Syntax and Semantic model of your code.
- Workspace APIs - Provides an object model to aggregate the code model across projects in a solution. Mainly for code analysis and refactoring around IDEs like Visual Studio, though the APIs are not dependent on Visual Studio.
- Services APIs - Provides a layer on top of Visual Studio SDK (VSSDK) for features like Intellisense, code formatting etc.

WORKING WITH ROSLYN

In this article, we'll take a quick look at Roslyn, and will explore the Compiler APIs and Scripting APIs in detail.

SCRIPTING APIS

The Roslyn.Scripting.* namespace provides types for implementing your own scripting sessions, using C# and VB.NET. You can create a new Scripting session by creating a new instance of ScriptEngine class, and then by invoking the CreateSession method of your Script Engine.

Hello Roslyn Scripting Engine

Here is a pretty simple implementation that uses ScriptEngine to execute some code.

```
using Roslyn.Scripting;
using Roslyn.Scripting.CSharp;
public class Driver
{

        public static void Main()
        {
            var se = new ScriptEngine();
            var session = se.CreateSession();
            var result = session.Execute("20 + 30");
            Console.WriteLine(result);
        }

}
```

If you run the above code, as you expect, you'll see 50 as the answer.

Deeper into the Scripting Engine

After the simple 'hello world' sample, let's get a little deeper into the Scripting Engine's capabilities

In Roslyn's scripting engine, you can use *AddReference* and *ImportNameSpace* methods of either the engine instance or the session instance to add references to other libraries and to import namespaces respectively, so that these types can be used in your Script. If you add references to the engine before creating the session, those references will be considered for the session as well.

You can pass an object instance as a context while creating a session, so that you may access the public instance members of the context object from your script as demonstrated below. In the following example, we are creating a more functional, **ScriptingHost** class that'll create a session by passing an object instance as the context, and will provide few wrapper methods to execute the code.

We'll also add some common references to our engine and will import some namespaces including a reference to the assembly where our Scripting Host class is residing as well– so that these types are known to our engine. Note that I'm adding a reference to the current assembly as well, along with assemblies for some common types.

```
/// <summary>
/// Let us create a quick Scripting Host class.
/// </summary>
public class ScriptingHost
{
        private readonly Session _session;
        private readonly ScriptEngine _engine;

        public ScriptingHost(dynamic context)
        {

                //Create the script engine
                _engine = new ScriptEngine();
```

```
        // Let us use engine's Addreference for
        // adding some common assemblies
        new[]
            {
                typeof (Type).Assembly,
                typeof (ICollection).Assembly,
                typeof (ListDictionary).Assembly,
                typeof (Console).Assembly,
                typeof (ScriptingHost).Assembly,
                typeof (IEnumerable<>).Assembly,
                typeof (IQueryable).Assembly,
                GetType().Assembly
            }.ToList().ForEach(asm => _engine.
                    AddReference(asm));

        //Import common namespaces
        new[]
            {
                "System", "System.Linq",
                "System.Collections",
                "System.Data.Entity",
                "System.Collections.Generic"
            }.ToList().ForEach(ns => _engine.
                    ImportNamespace(ns));

        _session = _engine.CreateSession(context);
    }

    public object Execute(string code)
    {
        return _session.Execute(code);
    }

    public void ImportNamespace(string ns)
    {
        _session.ImportNamespace(ns);
    }

    public void AddReference(Assembly asm)
    {
        _session.AddReference(asm);
    }

}
```

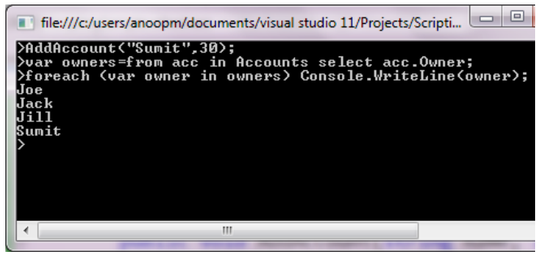
Now, let us do something useful with our ScriptingHost. Let us create a mock bank with few accounts. I'm having a DummyBank with few accounts in it. The following code should be self-explanatory.

```
//Our Bank
public class DummyBank
{
    public DummyBank()
        //Add some seed accounts for the demo
        Accounts = new List<DummyAccount>
        {
            new DummyAccount(2000, "Joe"),
            new DummyAccount(1020, "Jack"),
            new DummyAccount(3433, "Jill")
        };
    }

    public void AddAccount(string name, int balance)
    {
        Accounts.Add(new
        DummyAccount(balance,name));
    }

    public List<DummyAccount> Accounts { get; set; }
}

public class DummyAccount
{
    public DummyAccount(int balance,string owner)
    {
        Balance = balance;
        Owner = owner;
    }

    public int Balance { get; set; }
    public string Owner { get; set; }
}
```

Let us now create a simple Driver, so that we can initialize a session with our Scripting Host, by providing an instance of our Dummy Bank as the context.

```
public class Driver
{
public static void Main()
{
    //Initialize our bank
    var bank = new DummyBank();

    //Create an instance of our scripting host with
    // bank as the context that'll be used to create
    // the session
    var host = new ScriptingHost(bank);

    string codeLine;
    Console.Write(">");
    while ((codeLine = Console.ReadLine()) !=
     "Exit();")
    {
        try
        {
            //Execute the code
        var res = host.Execute(codeLine);
        //Write the result back to console if
        // not null
        if (res != null)
        Console.WriteLine(" = " + res.
                ToString());
        }
        catch (Exception e)
        {
            Console.WriteLine(" !! " + e.Message);
        }
        Console.Write(">");
}
}
}
```

Now, run the application. Do you observe that you can invoke the AddAccount method and Accounts property of the Bank directly from the script. And one more surprise – you can issue LINQ queries from your script as now LINQ support is available with the September CTP.



COMPILER APIS

Compiler APIs provide object models for accessing the syntax and semantic models of your code. Using compiler APIs, you can obtain and manipulate Syntax trees. The common Syntax APIs are found in the Roslyn.Compilers and the Roslyn.Compilers. Common namespace, while the language specific Syntax APIs are found in Roslyn.Compilers.CSharp and Roslyn.Compilers. VisualBasic.

'Syntax' is the grammatical structure whereas 'Semantics' refers to the meaning of the vocabulary symbols arranged with that structure. If you consider English, "Dogs Are Cats" is grammatically correct, but semantically it is nonsense. In this section, we'll explore how to parse and walk the syntax trees using Roslyn APIs.

The Syntax Tree

Roslyn provides APIs for building, parsing and modifying syntax trees. So let us write some code that'll parse a method and creates syntax tree. Later, we'll 'walk' the tree to dump it. Create a new Roslyn CTP Console project in Visual Studio, and try parsing some code. Note that in the new June 2012 CTP, Roslyn has added support for Query expressions, anonymous types, iterators, Indexers, switch statements etc.

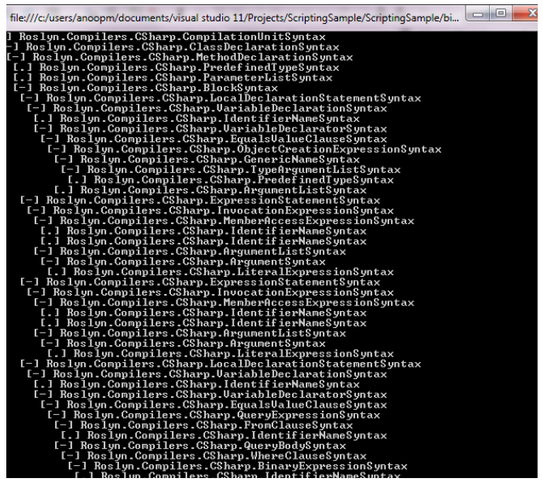
The following code basically shows

- Using SyntaxTree.ParseText to parse the syntax tree in the main method
- A simple 'Dump' extension method for dumping the syntax tree we just parsed.
- Using a custom syntax walker to 'Walk' the syntax tree

So, here we go.

```
using System;
using System.Linq;
using Roslyn.Compilers.CSharp;

namespace RoslynApp
{

public static class SyntaxTreeExtensions
{
public static void Dump(this SyntaxTree tree)
{
var writer = new ConsoleDumpWalker();
writer.Visit(tree.GetRoot());
}

class ConsoleDumpWalker : SyntaxWalker
{
public override void Visit(SyntaxNode node)
{
        int padding = node.Ancestors().Count();
        //To identify leaf nodes vs nodes with children
        string prepend = node.ChildNodes().Any() ? "[-]"
         : "[.]";
        //Get the type of the node
        string line = new String(' ', padding) + prepend
        + " " + node.GetType().ToString();
        //Write the line
        System.Console.WriteLine(line);
        base.Visit(node);
}
}
}

internal class Program
{
private static void Main(string[] args)
{

const string code = @"class SimpleClass {
public void SimpleMethod()
        {
            var list = new List<int>();
```

```
            list.Add(20);
            list.Add(40);
            var result = from item in list
                         where item > 20
                         select item;
        }
}";


//Parsing the syntax tree. Note that you've also got a
ParseFile method as well
var tree = SyntaxTree.ParseText(code);

//Dumping the syntax tree
tree.Dump();
}
}
```



WHAT'S NEW IN CTP 3

There have been some updates to the Roslyn September CTP (CTP3) APIs since the second CTP (CTP2) was released in June of 2012, and some of them are breaking changes. For example, there were some changes in Scripting APIs related to Scripting Engine and session creation, and also changes like the ones below in Compiler APIs

- Instead of SyntaxTree.ParseCompilationUnit you need to use ParseText or ParseFile instead, as I demonstrated above
- Also SyntaxNode.GetText and SyntaxNode.GetFullText have been replaced with ToString and ToFullString respectively

Along with that, a number of new language features got introduced in this September CTP, including Object Initializers, Expression trees, Collection Initializers etc. to mention a few.

CONCLUSION

We explored some interesting features of Roslyn from a very high level. Having a compiler available at runtime to parse and execute code can introduce multiple ways of abusing the languages ;-).

One practical use for it is in Business Applications, which have volatile validation requirements, or dynamic data structure. Instead of trying to store meta-information for everything, the administrator can use code snippets to define the validation that can be interpreted at runtime. This considerably simplifies meta-information storage for Business Applications.

Also, there are a number of interesting Roslyn based projects popping up, including the excellent Compilify.net project that allows you to execute C# from a browser compilify.net. And also, check out my SignalWire initial bits - which allows you to perform LINQ queries directly from JavaScript against your Entity Framework Model (https://github.com/amazedsaint/SignalWire). ■

Anoop Madhusudanan is a Microsoft MVP in C# for the last three years, and is presently working as a Solution Architect with Marlabs Inc. He is working with Microsoft .NET stack since the first version, and is experienced in multiple Microsoft Technologies, Cloud platforms and Mobile frameworks. He is also focused on DDD practices, with a sharp interest in highly scalable architectures like CQRS, and his interest and open source projects spans across areas like Neural networks, Quantum computing and Artificial intelligence. He blogs at http://amazedsaint.com and you can follow him on twitter @amazedsaint

Building an Enterprise Application

using IgniteUI™ jQuery Toolset

Traditionally enterprises often require focused apps, built quickly solving a particular business problem. This model favors aggregation of off-the-shelf components over 'roll-your-own' development efforts. With increasing 'consumerisation' of the Enterprise, applications catering to them have had to up their ante and move fast towards becoming more 'modern' or 'contemporary', if we may say. Today there are a plethora of jQuery/HTML5 based toolsets to create touch aware applications that work cross-browser, cross-platform and provide rich data visualizations. In an Enterprise, often choice of toolsets are dependent on feature sets and developer familiarity.

I was contacted by Infragistics to explore one such toolset to build a service oriented timesheet web application. Each commercial toolkit available in market extends base frameworks (like ASP.NET MVC) .

Today I am going to take a shot at building a timesheet application prototype using the Infragistics Ignite UI Suite of tools and pass on the things I encountered in our learning path, while exploring the toolset. Download toolset here.

Sumit helped me with this toolset and I thank him for his valuable insights.

DESIGNING THE APPLICATION

The Problem Definition

To setup our boundary conditions, let's create a problem definition, a laundry list of problems to solve

1. A cloud based Timesheet management product.
2. Multi-tenant architecture where multiple companies would have accounts for multiple users
3. Standard Forms Authentication support
4. Rich and Responsive UI
5. Mobile UI Support

As with all software solutions, we are assuming this application had to be ready by 'yesterday' :)

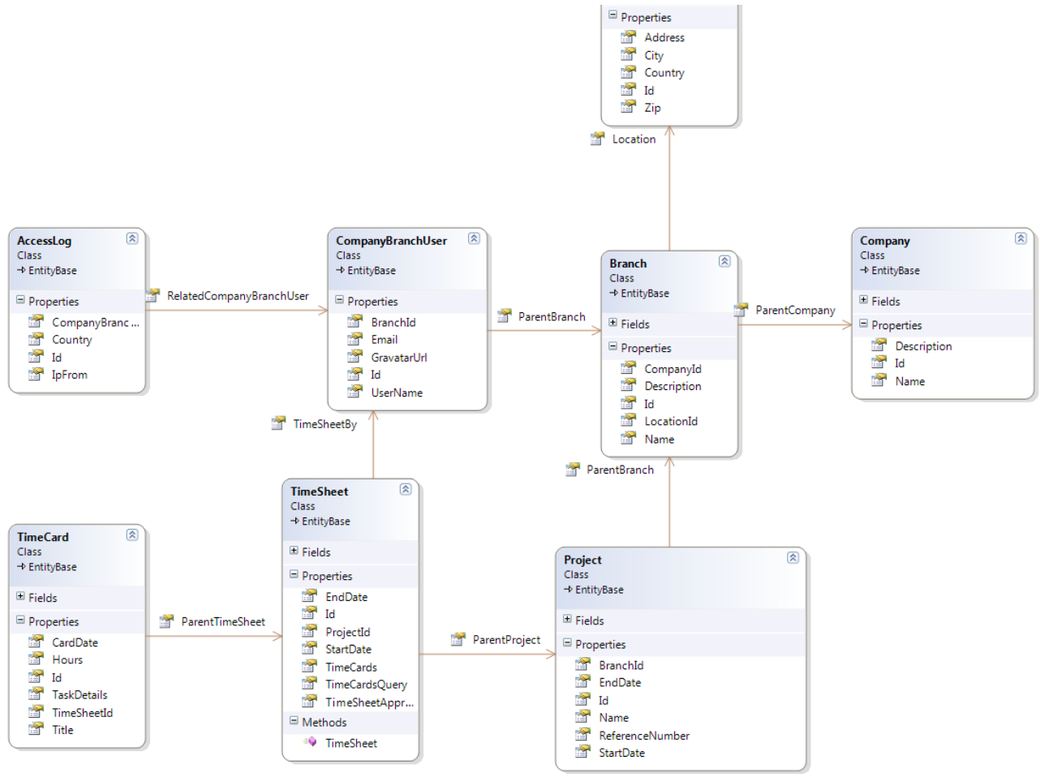
Design Choices

Given the above requirement, we made the following high level design choices:

1. Azure Website based application using ASP.NET MVC + SQL Server
2. For purposes of this article, we'll chose a Shared Database, Shared Schema tenancy which means each company information will be separated by the CompanyID and data from multiple companies will all reside in the same table. (In a large scale system you might want to explore independent schema or independent instance tenancy modes too).
3. The Infragistics Ignite suite was picked for prototyping and it provided,

   a. Rich and Responsive UI,
   b. Good cross-platform Mobile UI support,
   c. Very robust Grid Helpers and
   d. Built in Graphing and reporting capabilities over plain HTML5 + Roll Your Own options.

So with these basic choices, we started off.



Designing the Schema

The simplified application schema is captured in the Figure shown above.

Company: We start off with the company entity. For simplicity it has only a couple of bits of information i.e. Name and Description.

Branch: Branch is the entity against which actual time sheets are recorded and a Company can have multiple branches. As a result, all users are keyed to a particular branch.

CompanyBranchUser: This entity is a wrapper around the Authentication entity from ASP.NET Authentication provider. It adds the Branch information to a particular User Id and it does not store the password.

AccessLog: This is a log entity that records where a particular user is logged in from. This helps in reporting login reports.

Location: This is the address entity that stores a Company's Branch Address.

Project: The Project entity groups the Timesheets together and has other details like Name, ReferenceNumber, StartDate, EndDate and which branch the project is associated to. For simplicity, we are ignoring multiple branches working on same project.

Timesheet: The Timesheet object encapsulates all work done for a period like a week or 15 days.

TimeCard: Time Card is the finest granularity of information that is entered in Timesheets. It captures work done for a particular number of hours in a current Timesheet.

The Front End Design

Ideally we could design mocks here, but for sake of brevity, we'll skip the mocks and just have description of each Front End screen required.

Authentication

The Authentication Page will also be the landing page of the application where user will be requested to provide Username, Password and Company Name.

Access Control: Everyone has access to this page.

Administration

Admin Users: If Authenticated user belongs to the Admin User group, they will have access to the Administration module and the Administration landing page. This will allow users to create new Branches, Projects, and Users.

Super Users: Super Users belong to the service provider who is providing the solution as a hosted service. They will be able to Create Companies and other Super Users.

Access Control: Only Authenticated users who are members of the Super Users group have access.

Time Sheet and Time Cards

The Time Sheet and Time Card module will allow all users to create, Save and Submit Timesheets.

Access Control: Only Authenticated Users

THE UI LAYER

The UI has all the rendering logic. To test the application, we will use MVC tooling's default scaffolding mechanism to generate Views. These views however can get very verbose for a complex datatype like Timesheet spanning multiple CRUD screens. To consolidate these and provide a nice User Experience, we will hand roll custom views and use the Infragistics Ignite™ Suite to add polish to our Views.

Getting Started with Infragistics Ignite™ jQuery Suite

To include Infragistics, first install the jQuery Suite from here. The installer is about 167 MB and if you choose to install the help files, the installation takes about 600 MB. Without the help files (samples and scripts only) it's only about 25 MB.

Including the Scripts

For regular Web Apps

1. In your MVC project, add an "ig" folder under Scripts that comes by default when using the MVC 4 Internet application Template.
2. From the Infragistics installation folder (typically "C:\Program Files (x86)\Infragistics\IgniteUI 2012.2"), copy the "js" folder and paste it in the "ig/" folder created above.

For Mobile Web Apps

1. In your MVC project , add folder "igmobile" under Script folder.
2. From the Infragistics installation folder (typically "C:\Program Files (x86)\Infragistics\IgniteUI 2012.2\mobile"), copy the "js" folder and paste it in the "igmobile/" folder created above.

Including the CSS

For regular Web Apps

1. Under the Contents folder of the MVC project, add the folders "ig/css".
2. From the Infragistics installation folder, navigate to the css folder and copy structure and themes folder to the "ig/css" folder created above.

For Mobile Web Apps

1. Under the Contents folder of the MVC project, add the folder "igmobile".
2. From the Infragistics installation folder (for mobile) copy the css folder and paste it in the "igmobile" folder created above.

Adding MVC Dependencies

The Html helper implementations are in two dlls that we need to refer from the installation folder (C:\Program Files (x86)\ Infragistics\IgniteUI 2012.2\MVC\MVC4\Bin). These are:

```
Infragistics.Web.Mvc.dll
Infragistics.Web.Mvc.Mobile.dll
```

Once they have been added, select each of the dlls and navigate to their property panel (F4 by default). Set the 'Copy Local' property from False to True and do a clean build.

Including References in _Layout.cshtml

Now that the references have been included, let's use them. In the _Layout.cshtml, we add the following

```
<script src="../../Scripts/ig/js/infragistics.js"
type="text/javascript"></script>
<script src="../../Scripts/ig/js/infragistics.loader.js"
type="text/javascript"></script>
<script type="text/javascript">
        $.ig.loader({
            scriptPath: "../../Scripts/ig/js/",
            cssPath: "../../Content/ig/css/",
            resources: "igGrid.*",
            theme: "metro"
        });
</script>
```

The first two script references are obvious. The third script section uses a script and resource loader provided by Infragistics to load the specified references. In this case, we are specifying the javascript and css paths, the resources we want to load (here we are loading only the igGrid resources) and finally setting the theme to "metro". By default there are two themes out of the box, Infragistics and Metro.

Point to note here, you can use the loader in each page to load specific resources only. It is not mandatory to have it in _Layout. cshtml.

Including References in _Layout.mobile.cshtml

Now let's prepare the mobile layout base page so that we can develop the mobile pages quickly. First up, let's add a reference to jQuery Mobile using Nuget

```
PM> install-package jQuery.Mobile.Mvc
```

This installs all the jQuery Mobile dependencies and the special ViewSwitcher controller that takes control of switching between desktop and Mobile views.

The final Script for _Layout.mobile.cshtml looks as follows. The Infragistics specific inclusions are in blue color.

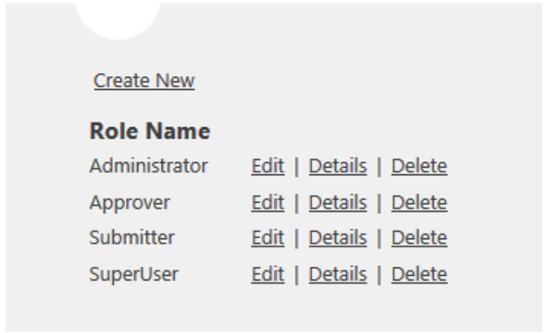
```
@using Infragistics.Web.Mvc.Mobile
<!DOCTYPE html>
<html>
<head>
    <title></title>
     <meta  name="viewport"  content="width=device-width,
initial-scale=1" />
    @Scripts.Render("~/bundles/modernizr")
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/jqueryui")
    @Scripts.Render("~/bundles/jquerymobile")
    @Styles.Render("~/Content/Mobile/css", "~/Content/
        jquerymobile/css")
    @RenderSection("scripts", required: false)
    <script type="text/javascript" src="@Url.
        Content("~/Scripts/igmobile/js/infragistics.
        mobile.js")"></script>
    <script type="text/javascript" src="@Url.
        Content("~/Scripts/igmobile/js/infragistics.
        mobile.loader.js")"></script>
    <script type="text/javascript">
        $(document).ready(function () {
            $.mobile.ajaxEnabled = false;
        });

    </script>
    <script>

    @(Html.InfragisticsMobile()
        .Loader()
        .ScriptPath(Url.Content("~/Scripts/igmobile/
        js/"))
        .CssPath(Url.Content("~/Content/igmobile/
        css/"))
        .Theme("iOS") // other options are
            // windowsphone/light, windowsphone/dark
            // android/hololight, android/holodark
            //.Resources("igmList")
        .Render()
    )
    </script>
</head>
<body>
    <div data-role="page" data-theme="a">
        @Html.Partial("_ViewSwitcher")
        @RenderSection("featured", false)
        @RenderBody()
    </div>
</body>
</html>
```

Replacing Default MVC View with Infragistics Grid

Once the resources are in place, we go ahead and replace the Default MVC Scaffolding with the Infragistics Grid.

We start off with the Roles View, which shows us the list of roles available in the System.

By default it looks rather bland as follows



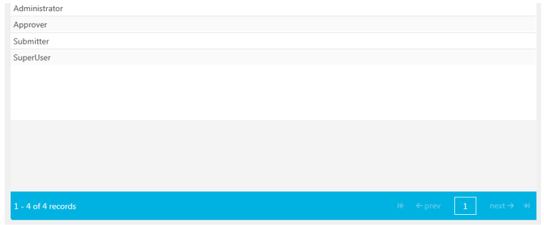
Now let's replace this with an Infragistics Grid in the Views/Role/Index.cshtml

To do this, we update the Index.cshtml as follows:

```
@using Infragistics.Web.Mvc;
@model IQueryable<TimeSheetGistics.Models.RoleModel>

@(Html.Infragistics().Grid(Model).ID("grid1")
    .Columns(column =>
        {
        column.For(x => x.RoleName).HeaderText("Available
                Roles");
        })
    .Features(features =>
    {
        features.Sorting().Type(OpType.Local);
        features.Paging().PageSize(30).Type(OpType.
        Local);
        features.Selection().Mode(SelectionMode.Row);
    })
    .DataBind()
    .Height("500px")
    .Width("100%")
    .Render()
)
```

- We add reference to the Infragistics.Web.Mvc namespace.
- Note we have changed the IEnumerable<T> to IQueryable<T>
- Next we use the HTML Helper to declare a grid with
    - o ID=grid1
    - o A single column that gets it's values from RoleName property of the model and whose header text is "Available Roles"
    - o We also add the following 'Features'
        - ☒ Enable Sorting (client side)
        - ☒ Paging
        - ☒ Set Selection mode to a full row select
    - o Next we call the DataBind method
    - o Set the Height to 500px
    - o Width to 100%
    - o Finally we call the Render method.

When we run our application now, we get the following view



Pretty neat for one helper method with a few lines of fluent configuration. But if you see, we have lost the Add/Edit/Delete functions. Let's see what it takes to add them back.

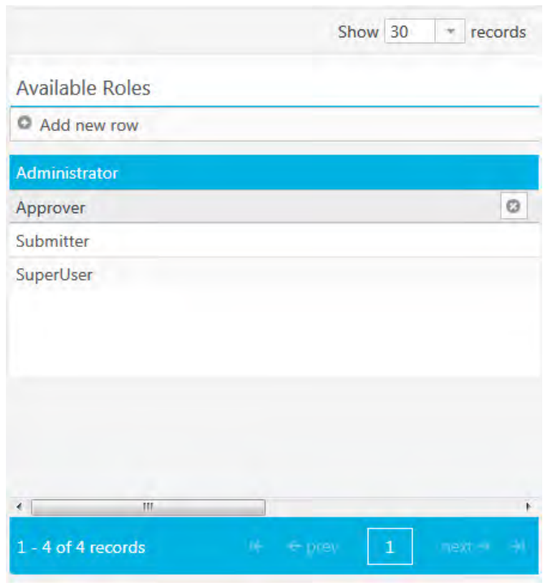
Inline Editing in Infragistics Grid

We can have two types of Editing in an Infragistics Grid.

First type is Inline Editing and second is Templated Editing. Since this particular Model has only one editable field, let's do Inline editing for this one. We'll take up Templated Editing for a more Complex Model.

The updated configuration for enabling Add/Edit/Delete in the UI is as follows. The new changes have been highlighted.

```
@(Html.Infragistics().Grid(Model).ID("grid1")
.Columns(column => { column.For(x => x.RoleName).
HeaderText ("Available Roles"); })
.Features(features =>{
features.Sorting().Type(OpType.Local);
features.Paging().PageSize(30).Type(OpType.Local);
features.Selection().Mode(SelectionMode.Row);
features.Updating().EnableAddRow(true).
EnableDeleteRow(true)
        .EditMode(GridEditMode.Row)
        .ShowReadonlyEditors(false)
        .ColumnSettings(settings =>
        {
                settings.ColumnSetting().
                ColumnKey("RoleName").ReadOnly(false);
                settings.ColumnSetting().
                ColumnKey("RoleName")
                        .EditorType(ColumnEditorType.
                Text).TextEditorOptions(options => {
                options.ValidatorOptions(option =>
                                {
                                option.
                KeepFocus(ValidatorKeepFocus.Never);
                option.BodyAsParent(false);
                option.Required(true);
                                });
                        });
        }); })
.DataBind()
.Height("500px")
.Width("100%")
.Render()
)
```



This results in the above UI.

As we can see, an 'Add new row' button has been added on top of the grid and when you hover over a row, it shows a delete button at the end of the row. Double clicking a row enables Editing. The "Add New" and Delete functionality is at the View Layer only. Data is not actually getting persisted.

So next, we'll see how to persist changes.

Persisting Changes

To persist changes, we need to configure the UpdateUrl by providing it the URL (Action Method) to Post the data back. So our fluent configuration now becomes as follows:

```
@(Html.Infragistics().Grid(Model).ID("grid1")
    .UpdateUrl("Create")
    .Columns(column =>
        {
            column.For(x => x.RoleName).
        HeaderText("Available Roles");
            column.For(x=> x.RoleId).Hidden(true);
        })
…
// rest of the code removed for brevity
)
```

To trigger the post back, we can do it in two ways:

1. Immediate Update: This updates our data as soon as we click 'Done' for new row or 'Delete' to delete a row. To do immediate updates, we add the following script.

```
<script>
    $.ig.loader(function () {
        $('#grid1').live('iggridupdatingrowadded',
         function () {
            $("#grid1").igGrid("saveChanges")
        });
        $('#grid1').live('iggridupdatingrowdeleted',
        function (e, args) {
            $("#grid1").igGrid("saveChanges")
        });
    });
</script>
```

As we can see above, we have used the ig.loader to associate two live methods for the grid. One is for the row added event and one is for the row delete event. In each case, it will call the 'saveChanges' on the igGrid which internally posts to the UpdateUrl.

2. Batch Updates: We can batch our updates too. The Infragistics Grid provides visual cues for Edit and Delete actions. Edited rows are italicized and Deleted rows are struck out by default. Now to save all changes in one go, we can put a button control for Saving all changes and handle it's click in JavaScript as follows

```
<script type="text/javascript">
    function saveChanges2() {
        $("#grid1").igGrid("saveChanges");
        return false;
    }
</script>
<input type="button" onclick="saveChanges2(); return
false;" value="Save Changes" />
```

For both the cases, our corresponding Action method on the Controller is as follows:

```
public ActionResult Create()
try
{
    GridModel m = new GridModel();
    List<Transaction<RoleModel>> transactions =
        m.LoadTransactions<RoleModel>(HttpContext.
        Request.Form["ig_transactions"]);

        foreach (Transaction<RoleModel> t in
        transactions)
        {
             if (t.type == "newrow")
            {
                Roles.CreateRole(t.row.RoleName);
            }
            else if (t.type == "deleterow")
            {
                Roles.DeleteRole(t.rowID);
            }
         }
         JsonResult result = new JsonResult();
         Dictionary<string, bool> response = new
        Dictionary<string, bool>();
         response.Add("Success", true);
         result.Data = response;
         return result;
    }
    catch
    {
        return View();
    }
}
```

GridModel is wrapper provided by Infragistics to wrap all the transactions done on the client. So it can be used for batch updates as well. As you can see, you get a nice enumerable list of entities that have been modified.

In our case, we cannot really 'edit' a group name as the default RoleProvider doesn't have a 'Edit' role group. So we have just handled Add and Delete.

Finally we are returning a JsonResult with value 'Success'.
This was a rather simple example of how to use the Infragistics Grid.

Modal Dialogs and Combo Boxes in Grid

By default ASP.NET provides us with the 'Register' UI infrastructure; however we don't want anyone to 'Register' because 'Registration' is controlled. So we will add Index, Create, Edit, Details and Delete views for User creation. Next we will see how we implement the Index page for the existing Users. In the process, we'll learn some more involved techniques for manipulating data in a Grid.

Editing Data in Modal Dialogs

Instead of inline editing of the User Data, we will use jQuery templates to define a row template that will be used by Infragistics MVC Helpers to build a modal dialog for editing.

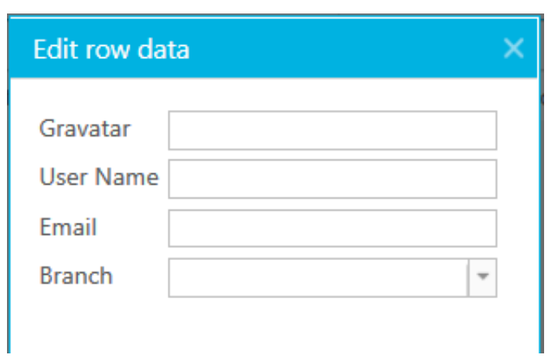
To enable Row Templates, we need to set two properties for the Grid's Feature collection, these are

```
…
feature => feature
.EditMode(GridEditMode.RowEditTemplate)
.RowEditDialogRowTemplateID("rowEditDialogRowTemplate1")
…
```

Here the string in the RowEditDialogRowTemplateID method refers to the jQuery template. In our case the template is defined as follows:

```
<script id="rowEditDialogRowTemplate1" type="text/x-
jquery-tmpl">
        <tr>
            <td> ${headerText}
            </td>
            <td data-key='${dataKey}'>
                <input />
            </td>
        </tr>
</script>
```

As we can make out, it's one row with two columns - one for the label that's picked up from the header text configuration and the other is the value that's retrieved using the columnKey configuration. So for each column, the above <tr> is repeated thus pivoting our column data into a nice modal dialog that looks as follows



Combo Box Editor in Grid

Note that the Branch is setup as a ComboBox. To do this, we use the settings collection and for the particular column set it up as a ComboBox. Data for the values in the Combo List is added to the ViewData store by the Controller. The configuration looks as follows

```
.ColumnSettings(settings =>
{
…//code removed for brevity
settings.ColumnSetting().ColumnKey("ParentBranch")
        .EditorType(ColumnEditorType.Combo).
        ComboEditorOptions(
                x => x.DataSource(ViewData["BranchId"])
                        .TextKey("Name").ValueKey("Id")
        );

});
… //code removed for brevity
```

The above setting helps us bring up the combo-box in the row template. However, when we select the value and click 'Done' in the grid ,we will see Id value because ParentBranch has the foregin key setup as it's Id. To work around this, we use the column formatter extension and manipulate the display value using JavaScript. This involves three steps:

1. In the column settings, setup the FormatterFunction

```
.Columns(column =>
        {
            … // code removed for brevity
            column.For(x => x.ParentBranch).
HeaderText("Branch").FormatterFunction("lookupBranchNa
me");
        })
… // code removed for brevity
```

2. Next we define the JavaScript function lookupBranchName as declared above. It uses a JS object called lookupBranchList that will hold the JSON representation of the Branch object in ViewData

```
var lookupBranchList = {};
function lookupBranchName(productNumber) {
        return lookupBranchList[productNumber];
}
```

3. Finally we have the method fillBranchNameLookup that is called from the IG Loader function

```
function fillBranchNameLookup(objectToFill) {
var colSettings = $("#indexGrid").
igGridUpdating("option", "columnSettings");
var colSetting;
for (var i = 0; i < colSettings.length; i++) {
colSetting = colSettings[i];
if (colSetting.columnKey === "ParentBranch") {
        if (colSetting.editorType && colSetting.editor
        Type === "combo") {
                var ds = colSetting.editorOptions.data
                Source;
                var textKey = colSetting.editorOptions.
                textKey;
                var valueKey = colSetting.editorOp
                tions.valueKey;
                var item;
                for (var j = 0; j < ds.length; j++) {
                        item = ds[j];
        objectToFill[item[valueKey]] = item[textKey];
        }
        }
        break;
} }
```

This function retrieves the columnSettings, then narrows it down to the ParentBranch setting and adds the text value to the dictionary, keyed in by the value key/id. Thus now when the combo returns the ID, we find the correct value and display it in the grid.

Mobile Views

Now that we have seen how to use the Grid UI, let's move on to the Mobile Views. We already have an Index view for all Roles, let's introduce a mobile view for it.

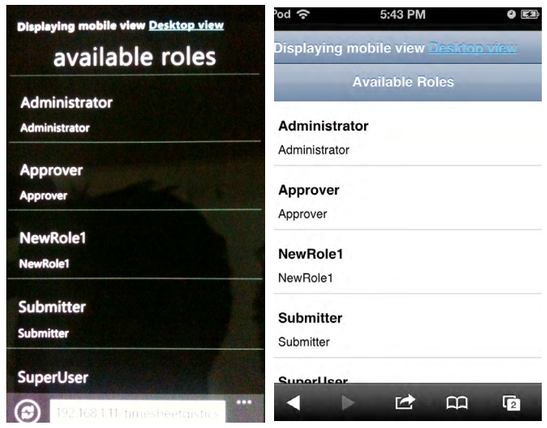
The plumbing is already in place for Mobile UI so we'll simply add an empty view called Index.mobile.cshtml

The Html helper for Mobile view can be configured as follows

```
@using Infragistics.Web.Mvc.Mobile
@model IQueryable<TimeSheetGistics.Models.RoleModel>
@(Html.InfragisticsMobile()
      .Loader()
      .ScriptPath(Url.Content("~/Scripts/igmobile/js/"))
      .CssPath(Url.Content("~/Content/igmobile/css/"))
              .Theme("iOS") // other options are
        // windowsphone/light, windowsphone/dark
        // android/hololight, android/holodark
        //.Resources("igmList")
        .Render()
    )
```

```
<div data-role=”header”>
    <h1>Available Roles</h1>
</div>
<div data-role=”content”>
@( Html
       .InfragisticsMobile()
       .ListView(Model as IQueryable<TimeSheetGistics.
       Models.RoleModel>)
       .ID(“serverSideListView”)
       .ImageMode(ImageMode.None)
       .Bindings(b =>
       {
           b.HeaderKeyFor(e => e.RoleName)
           .PrimaryKeyFor(e => e.RoleId)
           .TextKeyFor(e => e.RoleName);
       })
       .DataBind()
       .Render()
)
</div>
```

As we can see in the comments, out-of-the-box, the Infragistics suite comes with four themes. A screen grab from an iPod and a camera shot from a WP7 is shown below



We are using the Model passed to the view and picking assigning the Role Name to the heading and the text of the list item. We can optionally configure it with an image too. We can also configure a custom template if required.

Master Details View of Time Sheet Management

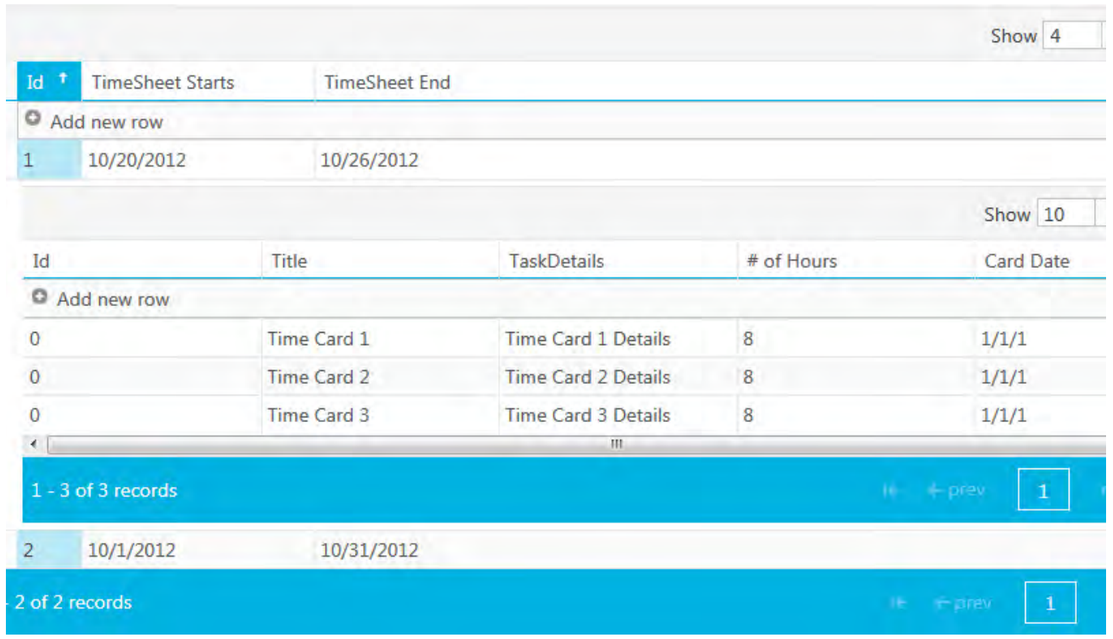
The Timesheet-TimeCard entities form a master-detail relationship that can be represented in a hierarchical grid. The Infragistics Grid control can render master detail data in a hierarchy. Below is the fluent API setup for binding Timesheet and TimeCard information.

Features that we have enabled are as follows:

1. DatePicker for selecting date. This is done using the ColumnSettings.
2. Enabled Add New row, Delete Row, column resizing and paging via the Features collection.
3. Child Grid via the ColumnLayout functionality.
4. The Child Grid’s target data type is setup using the x.TimeCards lambda expression and their relationship is specified using the PrimaryKey and ForeignKey definitions as highlighted below.

```
@(Html.Infragistics()
.Grid(Model)
.ID(“grid1”)
.Features(features =>
{
features.Sorting().Type(OpType.Local);
… // code removed for brevity
        .ColumnSettings(cs =>
        {
                cs
                  .ColumnSetting()
                  .ColumnKey(“StartDate”)
   .EditorType(ColumnEditorType.
                              DatePicker);
                cs
                  .ColumnSetting()
                  .ColumnKey(“EndDate”)
             .EditorType(ColumnEditorType.
                              DatePicker);
        });
… // code removed for brevity;

.AutoGenerateLayouts(true)
.ColumnLayouts(layouts =>
{
layouts.For(x => x.TimeCards)
.Features(features =>
{
        features.Sorting().Type(OpType.Local);
        features.Updating()
                .EnableDeleteRow(true)
        .EditMode(GridEditMode.Row);
        features.Paging()
                .PageSize(10)
                .Type(OpType.Local)
                .PageSizeList(new List<int>() { 2, 3,
                5, 10, 20 });
})
.DataSourceUrl(Url.Action(“BindInventories”))
.PrimaryKey(“Id”)
.ForeignKey(“TimeSheetId”)
.AutoGenerateColumns(false)
.AutoGenerateLayouts(true)
.Width(“100%”)
.Columns(childcols1 =>
{
        childcols1.For(x => x.Id)
        .HeaderText(“Id”).Width(“100px”);
        childcols1.For(x => x.Title)
        .HeaderText(“Title”).Width(“100px”);
        childcols1.For(x => x.Title)
        .HeaderText(“TaskDetails”).Width(“100px”);
        childcols1.For(x => x.CardDate)
        .HeaderText(“CardDate”).Width(“100px”);
        childcols1.For(x => x.CardDate)
        .HeaderText(“CardDate”).Width(“100px”);
});
})
.Width(“100%”)
.DataBind()
.Render())
```

The final view looks as follows:



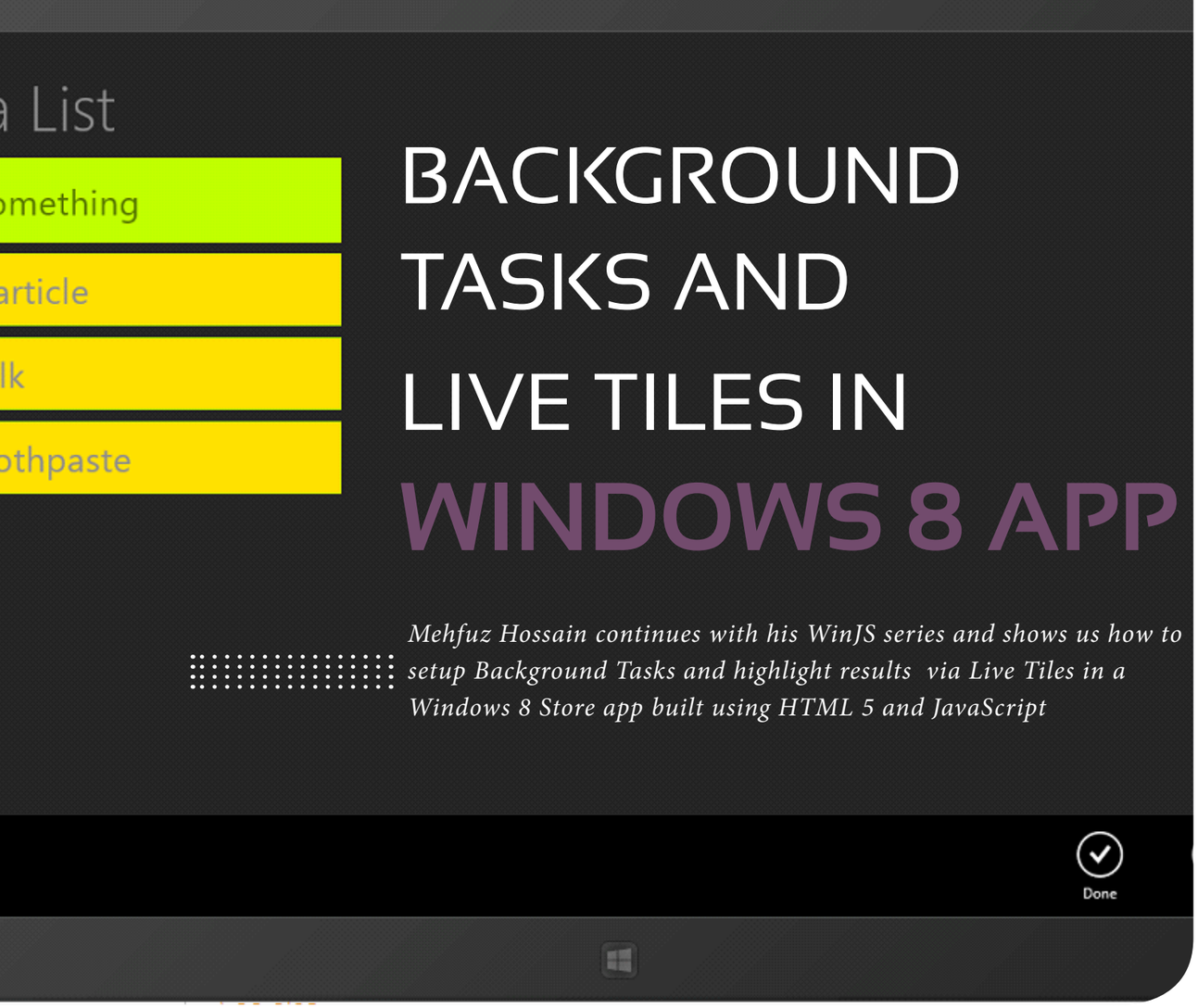
That covers all the UI components we used for the application. As a prototype, most of the data layer interaction is directly via Entity Framework code scaffolded by the MVC controller however the end application will have concerns well segregated into UI, Domain and Data Layers.

CONCLUSION

In conclusion, we saw how the Infragistics Ignite UI jQuery toolkit was applied to a Business scenario for rapidly prototyping a relatively complex application with rich functionality like hierarchical data, cross platform Mobile UI and Html helper based APIs.

Along the way, I discovered that Ignite UI has Html helper based extensions built on jQuery and jQuery UI. However if there are any missing wrappers, one can dig into Ignite’s jQuery APIs and take the work to conclusion. ■

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



BACKGROUND TASKS AND LIVE TILES IN WINDOWS 8 APP

Mehfuz Hossain continues with his WinJS series and shows us how to setup Background Tasks and highlight results via Live Tiles in a Windows 8 Store app built using HTML 5 and JavaScript

This article is based on a simple to-do list (Tada List) app that lets you add or remove tasks through gesture and showed some usage of built-in animation library.

For details on the gesture handling part, you can refer to the original article in the September-October 2012 issue of the DNC Magazine.

In this article, we will use the same premise of the to-do list application and we will add reminder support that will push pending tasks to the Windows 8 Live Tile which is triggered by a periodic Background Task.

We will first see some basics behind background task and live tile animation and then move forward to show how we can do it in the app itself.

BACKGROUND TASK

Background tasks are implemented in two steps:

Step 1: We have to define an entry-point module that will be executed as the background task becomes active. This can be a simple JavaScript file that might look like this:

```
(function () {
    "use strict";
    // get current instance of the background task.
    var backgroundTaskInstance = Windows.UI.WebUI.
        WebUIBackgroundTaskInstance.current;
    // This function will do the work of your
    // background task.
        function doWork() {
         var key = null,
             settings = Windows.Storage.
                ApplicationData.current. localSettings;
        // Write JavaScript code to do work in the
        // background.
        // Record information in LocalSettings to
        // communicate with the app.
          key = backgroundTaskInstance.task.taskId.
                toString();
          settings.values[key] = true;

          // A JavaScript background task must call
          // close when it is done.
          close();
    }
    doWork();
})()
```

Here note that all background task module should call the global close() to signal that the task is completed.

Step 2: During app initialization, we have to register the background task

```
var builder = new Windows.ApplicationModel.Background.
BackgroundTaskBuilder();
var trigger = new Windows.ApplicationModel.Background.
SystemTrigger(   Windows.ApplicationModel.Background.
SystemTriggerType.userAway, false);

builder.name = "reminderTask";
builder.taskEntryPoint = "js\\background.js";
builder.setTrigger(trigger);

var task = builder.register();
task.addEventListener("completed", function (args)
{
 var settings = Windows.Storage.ApplicationData.current.
localSettings;
 status = settings.values[task.taskId];
});
```

Here we are doing the following:

- Creating the task builder.
- Setting the entry-point module.
- Setting trigger.
- Adding a task completed event

In addition, we need to check if the task is already registered and in that case, need to skip the registration to make sure that our app is not using more than one task at a time.

```
var taskRegistered = false;
var background = Windows.ApplicationModel.Background;
var iter = background.BackgroundTaskRegistration.allTas-
ks.first();
while (iter.hasCurrent) {
   var task = iter.current.value;
   if (task.name === taskName) {
      taskRegistered = true;
      break;
   }
   iter.moveNext();
}
```

TILE NOTIFICATIONS

To implement live tiles in a Windows 8 App, first it is required to create tile xml. Tile xml is predefined and is retrieved through an enumeration:

```
// get a XML DOM version of a specific template by using
getTemplateContent
var tileXml = Windows.UI.Notifications.TileUpdateManager.
getTemplateContent(Windows.UI.Notifications.
TileTemplateType.tileWideSmallImageAndText04);
```

Here, we set the template type for a wide tile that will have an image on the left and text on the right. The details of the tile enumeration can be found here:

http://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.notifications.tiletemplatetype

Next step is to populate the template with content:

```
var img = tileXml.getElementsByTagName("image");

img[0].setAttribute("src", feed.imageUrl);
img[0].setAttribute("alt", feed.title);

var text = tileXml.getElementsByTagName("text");

text[0].appendChild(tileXml.createTextNode(feed.title));
text[1].appendChild(tileXml.createTextNode(feed.posts.
getAt(0).title));
```

Finally, we need to create the notification and send it to the tile updater.

```
// create the notification from the XML
var tileNotification = new Windows.UI.Notifications.
TileNotification(tileXml);

// send the notification to the app's application tile
```

```
Windows.UI.Notifications.TileUpdateManager.createTileUp-
daterForApplication().update(tileNotification);
```

If we want to queue multiple notifications, then we need to explicitly enable notification queue. It can be done during app initialization and in the following way:

```
Windows.UI.Notifications.TileUpdateManager.createTileUp-
daterForApplication().enableNotificationQueue(true);
```

It is also possible to schedule a notification. The basic tile
creation process is the same. However, we need to create an instance of ScheduleTileNotification class and then add it to the updater:

```
// create the notification from the XML
var tileNotification = new Windows.UI.Notifications.
ScheduledTileNotification(tileXml, dueDate);
tileNotification.id = item.id;
notification.TileUpdateManager.createTileUpdater-
ForApplication().addToSchedule(tileNotification);
```

Removing a scheduled notification

Now that we have created a scheduled notification, we might need to remove it once the parent item is removed. We can do it in this way:

```
var notifier = Windows.UI.Notifications.TileUpdateManager.
createTileUpdaterForApplication();

var scheduled = notifier.getScheduledTileNotifications();

for (var i = 0, len = scheduled.length; i < len; i++)
{

   if (scheduled[i].id === itemId) {
       notifier.removeFromSchedule(scheduled[i]);
   }
}
```

However, it is also possible to set an expiration date that will invalidate the tile automatically.

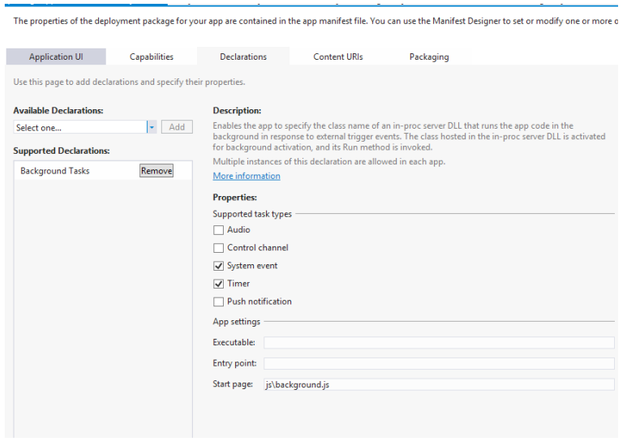
This was an outline of the steps required to implement background task and live tile animation in Windows 8 Store apps being developed in WinJS.

PREPARING THE APP

Now that we have seen the basics of Background Tasks and Tile notifications let's see how we can add these functionalities to our app.

Declaring Intent

Before an app can use background task, it should be declared in the app manifest. For a background task that is triggered in a timely manner and uses background.js, the app manifest looks something like the following:



As seen above, we have Added the Declaration called – Background Tasks. Since we'll be using a Timer, we have added Timer in addition to System event (that we'll use for testing).

Updating the UI

In our previous magazine issue article, we created a custom list to demonstrate animation and gesture support. In this article, we will move that over to WinJS.UI.ListView where most of the add or remove item animation will be out of the box and we will only deal with data in bindings level.

Creating listview and binding it to a data source is relatively easy. First we declared a div with data-win-control="WinJS.UI.Lisview" as follows:

```
<div id="todoList" data-win-control="WinJS.UI.ListView">
</div>
```

We also defined the UI template rather creating layout dynamically

```
<div   class="item-template"   data-win-control="WinJS.
Binding.Template">
     <div  class="item-container"  data-win-bind="style.
background:completed">
                  <h4   class="item-title"   data-win-
bind="textContent:title"></h4>
   </div>
</div>
```

Next on page ready event (todo.js), we add the following lines:

```
listView = todoList.winControl;
ui.setOptions(listView, {
        itemTemplate:  element.querySelector(".item-tem-
plate"),
      itemDataSource: data.items.dataSource
});
```

Here, data.item is WinJS.Bindings.List and can be created around a normal array in this way:

```
items: new WinJS.Binding.List(todos)
```

The final app would look like this:

In the app, we also moved the action items to app bar which is again pretty easy to implement and done in the following way:

```
<div   id="horizontalAppBar"   data-win-control="WinJS.
UI.AppBar" >
        <button data-win-control="WinJS.
       UI.AppBarCommand" data-win-options="{id:'cmdDon
       e',label:'Done',icon:'accept',tooltip:'Done',
       selection : 'global'  }">
        </button>
        <button data-win-control="WinJS.
        UI.AppBarCommand" data-win-options="{id:
       'cmdRemove',label:'Delete',icon:'delete',tooltip
       :'Delete', selection : 'global'  }">
        </button>
        <button data-win-control="WinJS.
       UI.AppBarCommand" data-win-options="{id:'cmdRemi
       nd',label:'Remind',icon:'clock',tooltip:'Remi
       nd', selection : 'global'  }">
        </button>
    </div>
```

To hook the app bar commands, we have to do this (todo.js):

```
document.getElementById("cmdRemove").
addEventListener("click", removeTodoHandler);
document.getElementById("cmdRemind").
addEventListener("click", showFlyoutHandler);
document.getElementById("cmdDone").
addEventListener("click", markAsCompleteHandler);
```

Now for setting reminder for each task, I have used flyout menu which is WinJS.UI.Flyout control:

```
<div   id="remiderFlyout"   data-win-control="WinJS.
UI.Flyout"
</div>
```

This wraps the content that will be the UI we want to show when the menu is shown:

```
remiderFlyout.winControl.
show(this, "auto");
```

Here 'auto' defines the placement of the Flyout. Other possible options are top, bottom, left or right.

Creating the background task handler

We have seen above how to setup a background task in general. For the app, when someone selects a task sets reminder. It basically updates the due date of the task and saves it back to the local settings.

```
forEachSelection(function (item) {
var jSon = [];
 for (var index = 0; index < jSon.length; index++) {
    if (jSon[index].id == data.id && jSon[index].dueDate
< datePicker.winControl.current) {
                    matched = true;      }
}
if (!matched) {
   jSon.push({ id: data.id, title: data.title, dueDate:
               datePicker.winControl.current });
  settings.values.reminders = JSON.stringify(jSon);}});
```

Here note that we are not only setting reminders for future tasks as schedule notifications can only be set for future events.

Background task for this app is triggered every 15 minutes through TimeTrigger class:

```
builder.setTrigger(new     Windows.ApplicationModel.
Background.TimeTrigger(15, false));
```

Note: 15 minutes is the smallest TimeTrigger you can set. Anything smaller and the app will throw an exception. While testing, if you don't want to wait for 15 minutes, you could use a system event like TimeZoneChanged to trigger the background task. For time zone event trigger, you can use the following code, instead of the above.

```
builder.setTrigger(new     Windows.ApplicationModel.
Background.SystemTrigger(Windows.ApplicationModel.
Background.SystemTriggerType.timeZoneChange, false));
```

Inside the background task (defined in background.js), we loop through the reminders and push them to the local settings for background process Id

```
if (settings.values.reminders) {
   var jSon = JSON.parse(settings.values.reminders);
   var currentDate = new Date();
   var remindersToSet = [];
  for (var index = 0; index < jSon.length; index++) {
       var dataTime = new Date(jSon[index].dueDate.
        toString("r")).getMilliseconds();
       if (currentDate.getMilliseconds() >= dataTime) {
                   remindersToSet.push(jSon[index]);
       }
  }
}

key = backgroundTaskInstance.task.taskId.toString();
settings.values[key] = JSON.stringify(remindersToSet)
```

As the task is completed (or calls close()) background task builder raises a "completed" event and due tasks are sent to the notification queue to schedule.

```
this.onCompleted = function (arg) {
 var settings = Windows.Storage.ApplicationData.
       current.localSettings;
  var  reminders    =  JSON.parse(settings.values[task.
taskId]);
 for (var index = 0 ; index < reminders.length; index++)
{
   _this.scheduleTile(reminders[index]);
 }
}
```

Note that while creating the schedule tasks, we set the data Id as notification Id:

```
var tileNotification = new Windows.UI.Notifications.
     ScheduledTileNotification(tileXml, data.dueDate);
tileNotification.id = data.id;
```

This is used to remove tasks from the queue as the task is deleted using the method describe in the "Removing a scheduled notification" section.

Watching it in Action

Now that the App is all set, run the app, select any of the todo items (tap or right click with mouse) and set the reminder from the App Bar. If you are using the code for 15 minute reminder, than wait for 15 minutes to see the Fly-out notification. If you are using the TimeZone code, just change the time zone of your computer to watch the reminder update on the live tile.

CONCLUSION

In this article, we took the sample app created in the previous issue of DNC magazine and added active task reminders support using scheduled notifications and background task handler. Background Tasks and Notfication flyouts add to the 'fast and fluid' user experience that Windows 8 Store apps aim to achieve and as we saw they are not too hard to implement using WinJS. ■

Download Code
bit.ly/dncmag-winjs2

Mehfuz works as the Team Lead at Telerik focusing on JustMock. He is passionate playing around with the latest bits. He has been a Microsoft MVP, author of OS projects like LinqExtender and LINQ to flickr. Prior to working at Telerik , Mehfuz worked as a core member in many high volume web applications including Pageflakes that is acquired by Live Universe in 2008. He is a frequent blogger and was also a contributor and site developer at dotnetslackers.com. Follow him at @mehfuzh

EXTENDING VISUAL STUDIO 2012

SUMIT MAITRA DEMONSTRATES HOW TO DEVELOP A PACKAGE TO CREATE COMPRESSED ARCHIVES OF CURRENT SOLUTION

Image Courtesy: mohamedradwan.wordpress.com

Visual Studio, as we all know, is a superb IDE for software development in multiple languages. It is my IDE of choice for most my work. However, for all the features that Visual Studio provides, we always crave for more and this is where Visual Studio extensions like JustCode by Telerik and Resharper by JetBrains come in. Though these are few of the prominent ones, there are thousands of extensions around (checkout the Microsoft Visual Studio extensions gallery). While using extensions, we don't really pause to wonder how a third party could have developed such well ingrained and seamless functionality into Visual Studio itself. In addition to the excellent developmental capabilities of these organizations, the answer to this lies in Visual Studio's excellent extensibility framework.

Today we will explore a small corner of Visual Studio's extensibility framework. We will build a small extension that integrates with Visual Studio's Tools > Options and helps us create an archive of the currently opened solution. This is pretty useful for sharing code samples. Essence of the article is to try and understand the various pieces of creating an extension rather than the end functionality of the plugin, which can no doubt be improved as you need.

We also use a .NET 4.5 specific feature that is the Zip library. Apart from this dependency, the rest of the steps are same if you intend to build an extension with backward compatibility with VS 2010.

Packages are files that end in .vsix extension, are essentially zip files themselves with a package manifest for configuring the extension The manifest itself is an xml file.
If you have created code snippets, project templates or used T4 before you've had a peek at the power of Visual Studio extensibility. Time to roll up sleeves and get in deep.

ROLLING UP YOUR SLEEVES AND DIGGING IN DEEP

THE VISUAL STUDIO 2012 SDK

Before we start, we have to get the Visual Studio SDK. It's around 12 MB download for the VS 2012 version bit.ly/vs12sdk. If you are using VS 2010, check your version before downloading.

Important: To develop Visual Studio 2012 packages you need a Professional version or above. The free Express editions do not support creation of Visual Studio extensions.

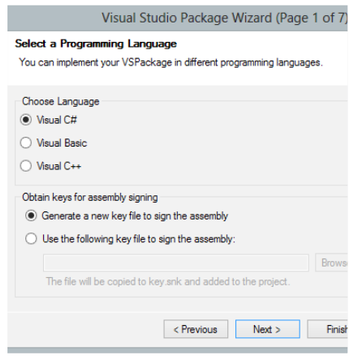
Close all instances of Visual Studio and install the SDK. Once you have installed it, you have all the project templates required and also the required dependencies.

Extensions are entry points into Visual Studio and if they go wrong during development, they can affect stability of VS. To prevent messing up your default Visual Studio settings, extensions are, by default, installed in a special instance of Visual Studio called the "Experimental Instance of Visual Studio". This sandboxes the extension so if anything goes wrong, the working instance of Visual Studio is not affected.

CREATE A SKELETON EXTENSION

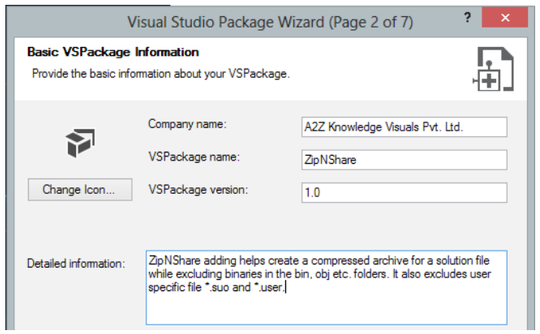
The SDK installs two more VS project templates in the "Other Project Types > Extensibility" category. These are "Visual Studio Shell Isolated" and "Visual Studio Package". We will start with the "Visual Studio Package".

Once we provide the Plugin Name and click Ok, Visual Studio launches a seven step wizard, starting with the "Choose Language" page.



> *"Visual Studio is a massively extensible IDE and one could write multiple books on how to do all the things possible with it "*

Here after you provide the company name, package name, description and version, you can also pick a Package icon.



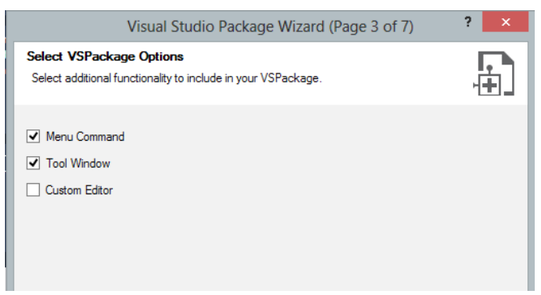
Next we select the additional functionality we need. We have the option of selecting Menu Command, Tool Window and Custom Editor. These need a little explanation.

Menu Command – This implies you will have a menu item, by default, this is added as a new menu group in the Visual Studio 'Tools' menu item.

Tool Window – Now this doesn't imply a property page in the Visual Studio 'Tools > Options'. Instead it implies a window (tab) that can be launched from the View > Other Windows > [Your Extension]

Custom Editor – Implies addition of a new editor like maybe something to replace the XAML Editor ;-). In our case, we'll leave this un-checked.

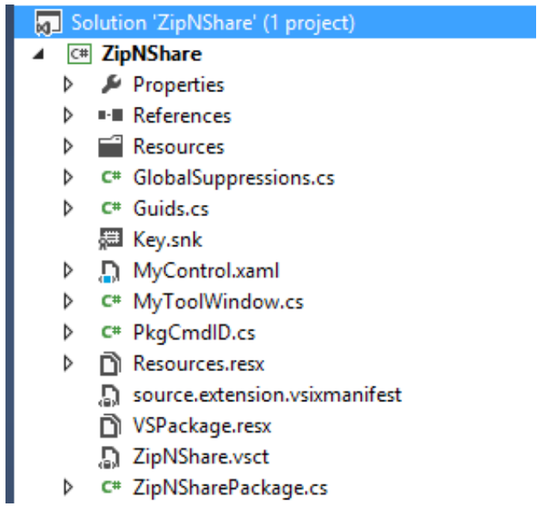
We will not use the Tool Window also, we still leave it checked to see the scaffolding it generates. Also, eventually we will end up not using the Tools > [Your Extension Menu] also, but it will generate the scaffolding we need for another menu we will use.



In the next two pages of the wizard, we will provide the Command Name and its CommandId and Tool Window Name and its corresponding CommandId.

In the last page, we get to select if we want to generate the Integration and Unit Test projects.

Once the Wizard completes code generation, we will have a Solution with code structure as follows:



Let's look at the generated files, starting at the bottom.

ZipNSharePackage.cs

This class is the entry point into the extensions and inherits from the Microsoft.VisualStudio.Shell.Package class.

The Initialize() method is overridden and it is responsible for wiring up the Commands that we named in the wizard. By default, it will wire up one command for the Menu click (MenuItemCallback) and one for the new Tools Window (ShowToolWindow).

Essentially this is where things get wired between Visual Studio Extensibility framework and your custom code.

ZipNShare.vsct

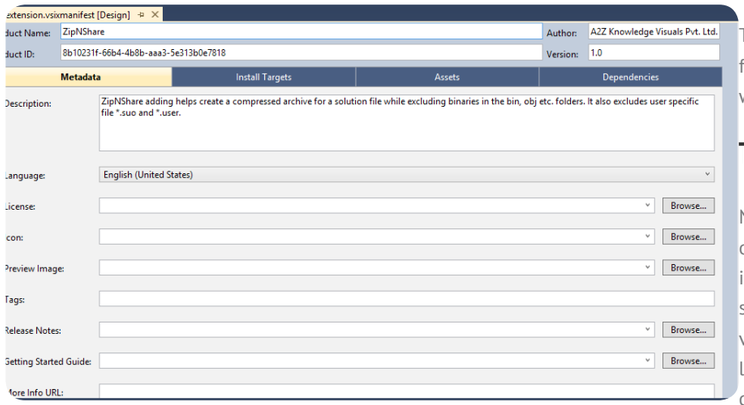
This is an xml file that has configurations for the new Menu items. It is also responsible for declaratively wiring up the Menus with the respective commands. We will re-visit this page in details a little later. As of now, this is where were we declare new Menu Items and wire them up.

VSPackage.resx

Standard Resource file storing resource strings.

source.extension.vsixmanifest

This the VSIX package manifest that includes more details about the package. It is an XML file in the end, but has a special editor for itself as seen below.



Resources.resx

Second set of resource files.

PkgCmdID.cs

This is a specially formatted file that contains the list of command IDs in the package

```
// PkgCmdID.cs
// MUST match PkgCmdID.h
using System;

namespace A2ZKnowledgeVisualsPvtLtd.ZipNShare
{
    static class PkgCmdIDList
    {
        public const uint cmdidZipAndShareMenuCommand =
        public const uint cmdidZipNShareTool =    0x101;

    };
}
```

It's used internally for the wiring up. If we add a new command, we have to add the identifier here.

MyToolWindow.cs

This class inherits from ToolWindowPane and acts as the container for controls to be shown for the particular tool window. Its Content is set to an instance of the next class MyControl.

MyControl.xaml.cs

This is the control that is hosted by MyToolWindow. By default it just has a button that on being clicked triggers a command.

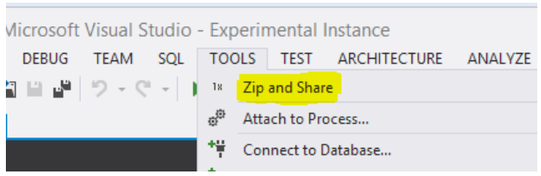
Guids.cs

This is a static class that just holds Guids for the package, commands set, and tool window.

TAKING IT FOR A SPIN

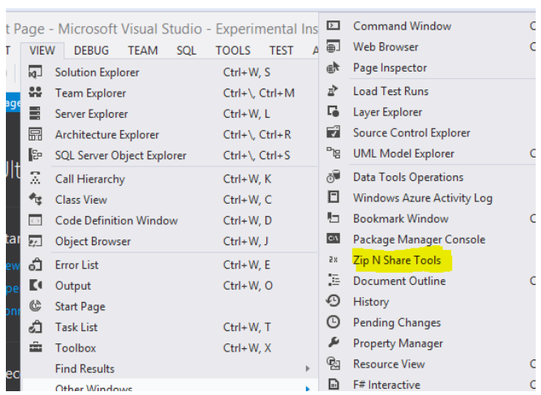
Now if we simply hit F5, current instance of VS will try to spin up the Experimental instance. Since this instance has never started before you will get the 'preparing visual studio' for launch dialog. Select your language and be sure to say no help documentation required.

Once done, the VS Sandbox will load. Check out the menu items added by your package. Under Tools menu, we'll see the 'Zip and Share' menu group first thing under Tools

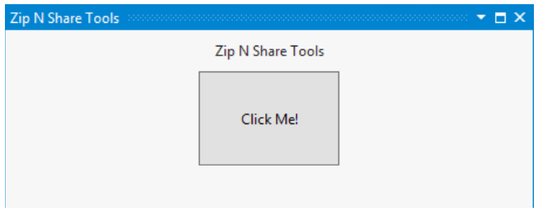


Clicking the menu shows a Message Dialog that's invoked from the command in ZipNSharePackage's MenuItemCallback method.

In the View > Other Windows menu, we see 'Zip N Share Tools' menu item



Clicking on the menu brings up the default Tools Window that you can dock to any of the panes in Visual Studio



ADDING A CUSTOM PROPERTY PAGE IN TOOLS > OPTIONS

With our skeleton plugin in place, it's time to implement a property page in Visual Studio >Tools dialog. To do this, we first add a class ZipNShareTools to our project.

Defining the Property Page

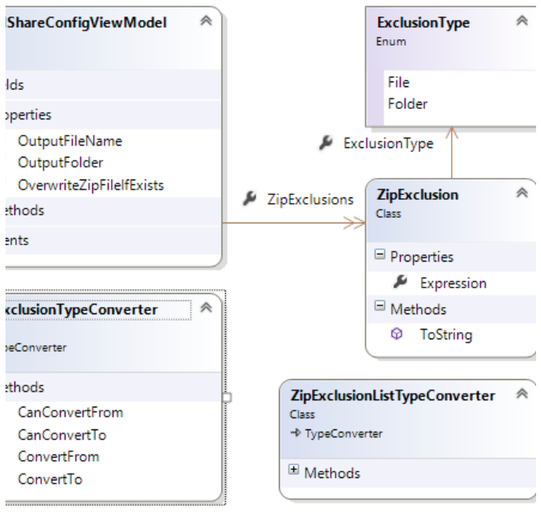
To integrate with the Option dialog, we need to ensure the following:

a. The Options Dialog page must inherit from DialogPage class from the Microsoft.VisualStudio.Shell namespace.
b. It must have GUID set for itself using the [GUID('...')] attribute at the class level. Use Visual Studio's GUID Generator to generate one for yourself.
c. All values must be exposed as properties. As a result our Dialog Page code ends up as follows

```
[ClassInterface(ClassInterfaceType.AutoDual)]
    [Guid("C15BE071-55B9-40B8-B2E3-D49E00C75575")]
    public class ZipNShareTools : DialogPage
    {
        private string _outputFolder;
        private List<ZipExclusion> _exclusions;
        private bool _overwriteIfExists;
        private string _outputFileName;

        public ZipNShareTools()
        {
            _exclusions = new List<ZipExclusion>();
        }

        [TypeConverter(typeof(
               ZipExclusionListTypeConverter))]
        public List<ZipExclusion> Exclusions
        {
            get { return _exclusions; }
            set { _exclusions = value; }
        }

[EditorAttribute(typeof(System.Windows.Forms.Design.
FolderNameEditor),
            typeof(System.Drawing.Design.
UITypeEditor))]
        public string OutputFolder
        {
            get { return _outputFolder; }
            set { _outputFolder = value; }
        }

        [Description("Use %SOLUTION_NAME% to use the
               respective solution name")]
        public string OutputFileName
        {
            get { return _outputFileName; }
            set { _outputFileName = value; }
        }

        public bool OverwriteZipFileIfExists
        {
            get { return _overwriteIfExists; }
            set { _overwriteIfExists = value; }
        }
    }
```

Essentially, we have defined the 'ViewModel' for the dialog. We have a string to save the 'OutputFolder' and 'OutputFileName'. We also have a boolean to indicate if we should overwrite the file if it exists. Finally we have a List of Exclusions.

The ViewModel



We have already seen ZipExclusion being used above. As we can see in the diagram, ExclusionType is an Enum with two values File and Folder.

The ZipNShareConfigViewModel is used in another UI that we will see shortly. It acts as the DataContext for a WPF form.
Now we come to the two TypeConverters. TypeConverters are mechanisms to define additional information that a 'PropertyGrid' control can use to render itself. Back in our custom Tools Dialog, if you remember we have a TypeConverter specified for the List<ZipExclusion> property. The TypeConverter attribute tells the property grid how to interpret the data that comes to it. It also provides the mechanism that PropertyGrid hooks into for 'saving' the data.

In case of Tools Option Dialog, property values are stored as string. So we have to ensure that anything that cannot be saved as, or created from a string, has an appropriate TypeConverter specified. The PropertyGrid at the time of rendering the property calls the CanConvertFrom and CanConvertTo methods to check if the property can be retrieved or saved. If it can, it calls the ConvertFrom or ConvertTo as required. We override these two methods to create a string representation of the ZipExclusion object as well as it's translation from the saved string into the object. The Convert methods are as follows

```
public override object ConvertFrom(ITypeDescriptorConte
xt context,  System.Globalization.CultureInfo culture,
object value)
{
   if (value.GetType() == typeof(string))
   {
      List<ZipExclusion> values = new
              List<ZipExclusion>();
       string[] vals = ((string)value).Split(new
                              char[] { ',' });
      for (int i = 0; i < vals.Length; i++)
      {
           values.Add((ZipExclusion)(new
                ZipExclusionTypeConverter()).
                    ConvertFromString(vals[i]));
      }
      return values;
   }
return base.ConvertFrom(context, culture, value);
}

public override object ConvertTo(ITypeDescriptorConte
xt context, System.Globalization.CultureInfo culture,
object value, Type destinationType)
{
       List<ZipExclusion> values = value as
        List<ZipExclusion>;
       if (values != null)
   {
      if (context == null)
      {
         string content = string.Empty;
         foreach (var item in values)
         {
            content += string.Concat(item.
            ToString(), ",");
         }
          content = content.TrimEnd(
                      new char[] { ',' });
          return content;
      }
      else
      {
          return string.Format("({0} Items in
             collection)",
             ((List<ZipExclusion>)value).Count);
      }
   }
   return "0 Items in collection";
}
```

As we can see we use the ConvertTo method to convert the objects into a comma separated string. The ConvertFrom method uses the ZipSolutionTypeConverter to convert the incoming string into a ZipSolution object.

HOOKING UP THE CUSTOM PROPERTY PAGE TO THE TOOLS MENU

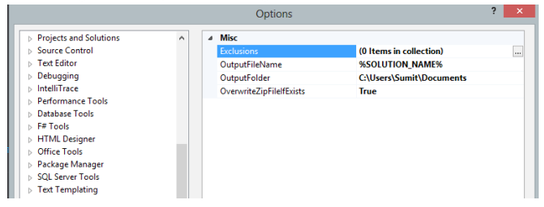
Now that we have defined the properties we need, let's hook up the Page to our Extension. It's a single line of code. In the ZipNSharePackage.cs file, we need to add the following attribute at the class level

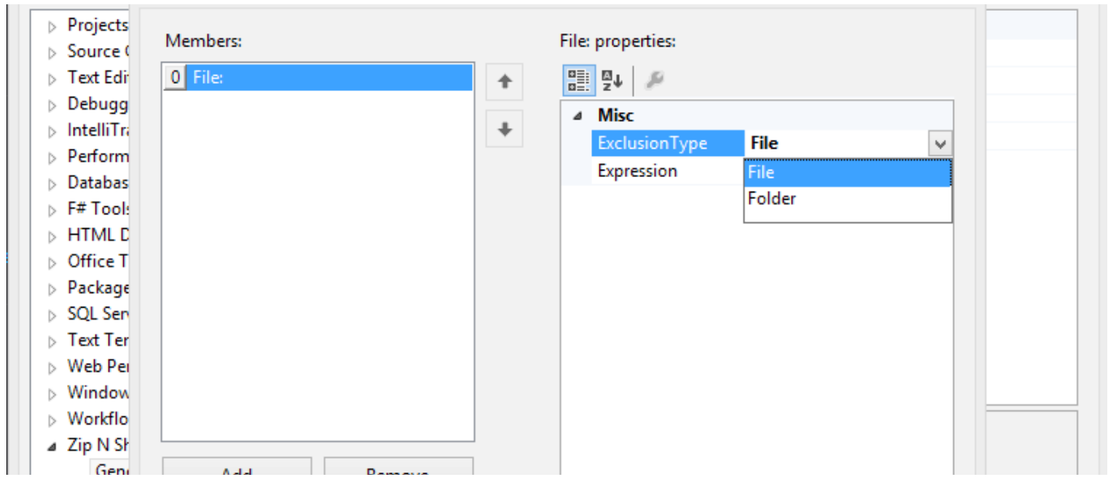
```
[ProvideOptionPage(typeof(ZipNShareTools), "Zip N
Share", "General", 101, 106, true)]
```

The Parameters for the attribute are

- Type definition of class inheriting from DialogPage
- Name of the Category to which the Property page should be added. In our case it will add a new Category
- Name of Sub-Category under the given category
- Category Resource Id
- Page Name Resource Id
- Support Automation

With this, if we run the plug-in in the sandbox, we will see the following in the Options dialog





If you click on the Exclusions' ellipsis button, property grid will inspect the backing store and when it sees a generic list, it will use the generic Collection property editor. If the type specified in the generic list has a default constructor, then the Add and Remove buttons will work by default.

As we can see here, it correctly adds new instance of the ZipExclusion object and automatically binds to the Enum and presents its values for selection in a drop down.

This completes the Tools > Options Property Dialog setup.

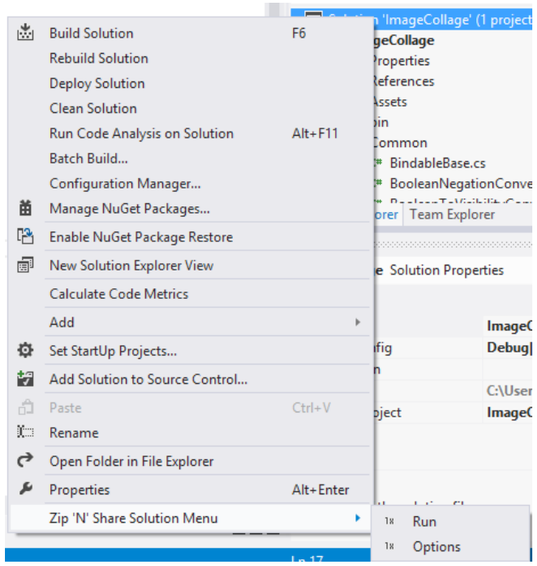
ADDING CONTEXT MENU TO THE SOLUTION IN SOLUTION EXPLORER

Now that our package is setup, let us see if we can improve its accessibility. We want to create an archive of any open solution.

So the natural tendency would be to right click on the Solution node in Solution Explorer and find a menu to trigger the archival process. We have two options, either fire off the process using the default setting (as set via the Tools > Options dialog that we just created) or let users update the settings before they create the archive.

It may become cumbersome to have a dialog pop up every time we run the archive command. Instead if we have a menu group and two sub-menu items, one for running the archive command and one for modifying settings before running the archive command then it will suit everyone!

The menu would look as follows:



The idea is when user clicks on 'Run' option, it should go ahead and create the archive with current settings. If user selects the Option, show them a custom dialog allowing them to modify the settings before launching the archival process.

Specifying Commands

First step towards creating the new Menu items is deciding on the new commands required (if any). In our case the Wizard already generated one command for us. We can reuse it for the 'Run' option. However we will need a new command for the new 'Options' menu. To do this, we add the following line in the

PkgCmdID.cs

```
public const uint cmdidZipNShareOptions =   0x103;
```

Next we create a CommandID object and register it with the MenuCommandService in the ZipNSharePackage.cs. To do this, we update the Initialize() method. The newly added code is highlighted below. We are essentially using the current package's command set and adding a new command to it. Then we are creating a MenuCommand and registering it with Visual Studio's MenuCommandService.

```
protected override void Initialize()
{
Debug.WriteLine(string.Format(CultureInfo.
CurrentCulture, "Entering Initialize() of: {0}", this.
ToString()));
base.Initialize();

// Add our command handlers for menu (commands must
exist in the .vsct file)
OleMenuCommandService mcs = GetService(typeof(IMenuComm
andService)) as OleMenuCommandService;
if (null != mcs)
{
        // Create the command for the menu item.
        CommandID menuCommandID = new
        CommandID(GuidList.guidZipNShareCmdSet,
        (int)PkgCmdIDList.cmdidZipAndShareMenuCommand);
        MenuCommand menuItem = new
                MenuCommand(MenuItemCallback,
                menuCommandID);
        mcs.AddCommand(menuItem);
        // Create the command for Options Menu Item
        CommandID menuOptionsCommandID = new
        CommandID(GuidList.guidZipNShareCmdSet,
                (int)PkgCmdIDList.
                cmdidZipNShareOptions);
        MenuCommand menuOptionsItem = new MenuComm
                and(OptionsMenuItemCallback,
                menuOptionsCommandID);
        mcs.AddCommand(menuOptionsItem);
        // Create the command for the tool window
        CommandID toolwndCommandID = new
                CommandID(GuidList.guidZipNShareCmdSet,
                (int)PkgCmdIDList.cmdidZipNShareTool);
        MenuCommand menuToolWin = new
        MenuCommand(ShowToolWindow,
        toolwndCommandID);
        mcs.AddCommand(menuToolWin);
} }
```

In the above code, we don't have the OptionsMenuItemCallback defined yet. So we define a skeleton method in the ZipNSharePackage.cs with the following method signature

```
private void OptionsMenuItemCallback(object sender,
EventArgs e) {    //TODO: Implement Options Menu }
```

Updating the VSCT file

The VSCT file contains the Menu items definitions as well as the co-relations between the menus and the commands that need to be invoked. If you use an id that is not defined, the compiler will flag an error.

We start with the <GuidSymbol> group and add the following IDs

```
<IDSymbol name="SolutionMenuGroup" value="0x2020" />
<IDSymbol name="SolutionSubMenu" value="0x2100" />
<IDSymbol name="SolutionSubMenuGroup" value="0x2021" />
<IDSymbol name="cmdidZipNShareOptions" value="0x0103" />
```

- The SolutionMenuGroup refers to the menu container marked in Red.
- The SolutionSubMenu refers to the actual menu marked in Green
- The SolutionSubMenuGroup refers to the child menu container marked in Blue.

The cmdidZipNShareOptions symbol is used for identifying the command of the Options button. For the Run button, we will use the existing cmdidZipAndShareMenuCommand id. Essentially the Hierarchy of Menus is as follows

[Solution Context Menu (existing)]
  [Custom Menu Group]
    [Custom Sub Menu]
      [Custom Sub Menu Group]

Each of the Groups may contain a Menu or multiple menu buttons. Now we have to define the Custom Menu and link the Sub Menu to it. The XML markup for it is as follows. This will add the region highlighted in red.

```
<Menus>
      <Menu guid="guidZipNShareCmdSet"
                id="SolutionSubMenu"
          priority="0x0100" type="Menu" >
        <Parent guid="guidZipNShareCmdSet"
        id="SolutionMenuGroup"/>
        <CommandFlag>DynamicVisibility</CommandFlag>
        <Strings>
          <ButtonText>Zip 'N' Share Solution Menu
```

```
            </ButtonText>
          </Strings>
        </Menu>
      </Menus>
```

Note: We have not yet extracted the 'SolutionMenuGroup'. This is the next step.

We have to extract the [Solution Context Menu] from Visual Studio. To do this we use the existing ID value 'IDM_VS_CTXT_SOLNNODE'. This is defined in the 'Groups' section as shown below.

```
<Group guid="guidZipNShareCmdSet"
         id="SolutionMenuGroup" priority="0xF000">
        <Parent guid="guidSHLMainMenu"
              id="IDM_VS_CTXT_ SOLNNODE"/>
</Group>
```

Next we add a SubMenuGroup to our newly defined Menu (above). This is the blue section in the image.

```
<Group guid="guidZipNShareCmdSet"
id="SolutionSubMenuGroup" priority="0x0610">
        <Parent guid="guidZipNShareCmdSet"
       id="SolutionSubMenu"/>
</Group>
```

Finally we add the Sub Menu buttons in the <Buttons> section. As we can see, both the buttons have their parents set to the SubMenuGroup.

```
<Button guid="guidZipNShareCmdSet" id=" cmdidZipAnd-
ShareMenuCommand" priority="0x0102" type="Button">
  <Parent guid="guidZipNShareCmdSet"
        id="SolutionSubMenuGroup" />
  <Icon guid="guidImages" id="bmpPic1" />
  <CommandFlag>DynamicVisibility</CommandFlag>
  <Strings>
    <ButtonText>Run</ButtonText>
  </Strings>
</Button>

<Button          guid="guidZipNShareCmdSet"
id="cmdidZipNShareOptions"      priority="0x0103"
type="Button">
  <Parent guid="guidZipNShareCmdSet"
        id="SolutionSubMenuGroup" />
  <Icon guid="guidImages" id="bmpPic1" />
  <CommandFlag>DynamicVisibility</CommandFlag>
  <Strings>
    <ButtonText>Options</ButtonText>
  </Strings>
</Button>
```

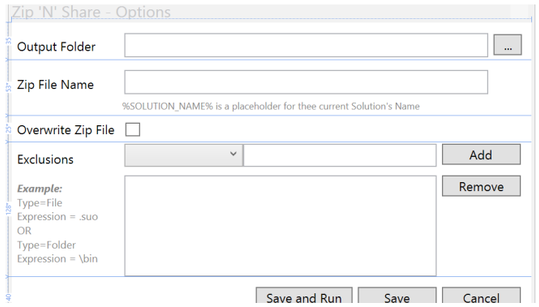
Adding menu items seem a little complex but once you get the hang of how the hierarchy is interpreted, it becomes simple. The Final Hierarchy is as follows

[Solution Context Menu (existing)]
  [Custom Menu Group]
    [Custom Sub Menu]
      [Custom Sub Menu Group]
        [Run Menu Button]
        [Configure Menu Button]

With this configuration, if you build your solution and run it, the Experimental Visual Studio will start up. Now if you open a Solution in it and check the context menu, you can see the new menu items.

BUILDING A CUSTOM DIALOG FOR PROPERTY UPDATES

Now that we have got the menus in place. Let's setup the custom dialog that will provide the same properties that come up from the Options dialog, but will also allow initiation of the Archival process. We will add a simple XAML Dialog and set it up as follows:



We refer back to the ZipNShareConfigView model class in our ViewModel diagram from earlier. This is a standard ViewModel class implementing INotifyPropertyChanged
interface and is data bound to the UI. We have simple code behind for the Add and Remove buttons that add or remove ZipExclusion objects from the ZipExclusions collection.

Data for the dialog is loaded using the ZipNShareTools instance that is passed at the time of Command invocation. When user clicks save, we just invoke the SaveSettingsToStorage() method on the instance of ZipNShareTools.

```
private    void    SaveButtonClick(object    sender,
RoutedEventArgs e) {
            toolsPage.SaveSettingsToStorage();
            this.TriggerRun = false;
            this.DialogResult = true;
}
```

The custom dialog has a Boolean property TriggerRun indicating, if the package should initiate the archival process. The property is set to true when user clicks on Save and Run button. Otherwise it's set to false. To hookup the dialog to the menu, we update the Command call back in ZipNSharePackage class as follows

```
private void OptionsMenuItemCallback(object sender,
EventArgs e)
{
            IVsUIShell uiShell = (IVsUIShell)
               GetService(typeof(SVsUIShell));
            ZipNShareTools toolsPage =
               (ZipNShareTools)
               GetDialogPage(typeof(ZipNShareTools));
            ZipNShareConfigDialog dialog = new
               ZipNShareConfigDialog(toolsPage);
            bool dialogResult = (bool)dialog.
               ShowDialog();
            if (dialogResult && dialog.TriggerRun)
            {    ArchiveFiles();            }
}
```

We retrieve instance of the VsUIShell interface and using this retrieve the ZipNShareTools toolsPage instance. We pass this instance to the config dialog via the constructor and initiate it. Once the dialog returns, we check if User clicked Save or 'Save and Run' (dialog.TriggerRun == true in that case). If they did, we initiate the Archive process. In case user clicks on the 'Run' context menu directly, it invokes the ArchiveFiles() method directly.

```
private void MenuItemCallback(object sender, EventArgs
e) { ArchiveFiles(); }
```

IMPLEMENTING THE ARCHIVING FUNCTIONALITY

The Archiving functionality uses the .NET 4.5's System.IO.Compression.ZipArchive class to compress the files. To use it, you have to add reference to two dlls – System.IO.Compression and System.IO.Compression.FileSystem. The algorithm is simple

- Check if the target zip file already exists. If it does check if user has allowed overwrites. If yes continue, if not, return.
- Start at the Folder where the solution file is e.g. if Solution is in C:\Sample\sample.sln we start at C:\ and start iterating through the Sample folder.
- Use the Directory.EnumerateFiles helper to get an enumerator for all files and folders.
- For every item found, check if it falls under any of the excluded conditions, if not add it to the Zip stream. Increment file counter.
- Once all files have been enumerated, close Zipfile stream, copy to the destination folder and return the total number of files processed.

We implement this in the Archiving\ZipSolution.cs class and declare it as a public static. We hook it up with the ZipNSharePackage class in the 'CreateArchive()' method, where we again use the ToolPage object to retrieve the settings and pass on to the ZipSolution class. Now if you run the project, you can load a solution in the experimental instance of Visual Studio, right click on the solution and select Run to create a zip file for the solution. That's it. You have just completed your first Visual Studio Package. Download the entire source code from the Git Repo here bit.ly/dncmag-zns

Final Cleanup

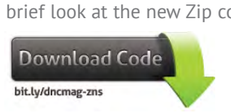
Since we no longer need the Menu item 'Tools > Zip n Share' that was created by default, we can remove it from the VSCT file.

Installing it for yourself

To use your extension in the regular Visual Studio, just navigate to the bin folder and locate the .vsix file. Double click on it and let Visual Studio Install it. Restart Visual Studio and you are done. You can uninstall it from the Tools > Extensions and Updates menu.

CONCLUSION

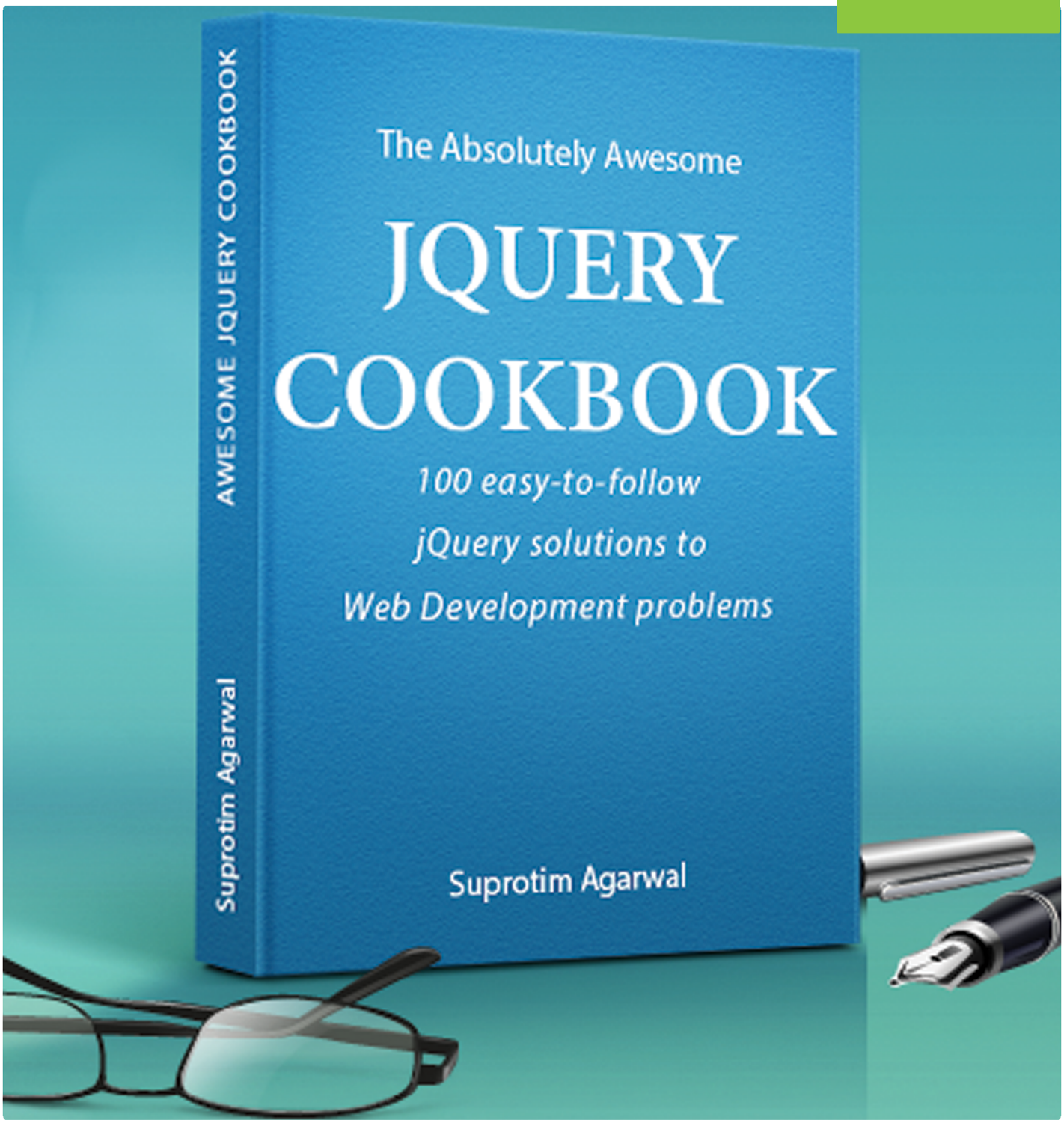
Visual Studio is a massively extensible IDE and one could write multiple books on how to do all the things possible with it. Today we saw a neat way of extending it via a custom vsix package. There are more things that we can do like integrate with the Errors window or the Output window to show the process output etc. We keep those for another day. We also had a brief look at the new Zip compression library in .NET 4.5. ■



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

The Absolutely Awesome jQuery Cookbook

NEW EBOOK



100 Easy-to-follow jQuery solutions

With scores of practical jQuery recipes you can use in your projects right away, this cookbook will help you gain hands-on experience with the jQuery API!

Please click below to learn more.

Click Here www.jquerycookbook.com