ISSUE 21 | NOV - DEC 2015

DNCMAGAZINE

www.dotnetcurry.com

Special Edition **AngularJS**

Angular 2 Developer Preview

Using AngularJS with Bower and RequireJS in Visual Studio

Creating Graphics using SVG and AngularJS

Create Your First Diagnostic Analyzer in VS 2015

DESKTOPS IN THE CLOUD

Universal Windows 10 Platform (UWP)
Music App for Windows Phone & Raspberry Pi2

Upgrading Existing C# Code to C# 6.0

TABLE OF CONTENTS

EDITORIAL

Suprotim Agarwal

Editor in Chief

Welcome to the 21st Edition of the DNC Magazine. This edition warms up with an excellent article by Damir on C# 6.0 where he categorizes some new C# features and shows us an upgrade path from previous versions of C# to the new one. For our client-side devs, we have three awesome articles by Gil, Mahesh and Ravi on AngularJS; mashing it up it with SVG, Require.js and Bower.

So far, we have been running Web apps in the Cloud. Kunal in his article shows us why and how to use Azure to run Desktop apps in the Cloud. Shoban follows up on his Windows 10 UWP series and deploys the same code base to Windows Mobile and Raspberry Pi. Damir follows up on his Visual Studio 2015 Diagnostic Analyzer article, by showing us how to actually create one for our needs.

Enjoy this edition and here's everybody in the DNC Team wishing you a very Happy Diwali and a joyous Holiday Season! We'll see you again in the New Year!

CREDITS

Editor In Chief
Suprotim Agarwal
suprotimagarwal@a2zknowledgevisuals.com

Art Director
Minal Agarwal
minalagarwal@a2zknowledgevisuals.com

Contributing Authors
Damir Arh
Gil Fink
Kunal Chandratre
Mahesh Sabnis
Ravi Kiran
Shoban Kumar

Technical Reviewers
Gil Fink
Ravi Kiran
Suprotim Agarwal

Next Edition
4th Jan 2016

www.dotnetcurry.com/magazine

POWERED BY
a2Z | Knowledge Visuals

.NET & JavaScript Tools

Shorten your Development time with this wide range of software and tools

CLICK HERE

ASP.NET MVC CONTROLS

WORK EFFORTLESSLY WITH ASP.NET MVC

Quickly create advanced, stylish, and high performing UIs for ASP.NET MVC with Ignite UI MVC. Leverage the full power of Infragistics' JavaScript-based jQuery UI/HTML5 control suite with easy-to-use ASP.NET MVC helpers and get a jump start on even the most demanding Web applications.

Download ASP.NET MVC Controls as part of the Ultimate Developer toolkit.

DOWNLOAD FREE TRIAL

INFRAGISTICS®

Upgrading Existing C# Code to C# 6.0

We are already used to the fact that a new version of Visual Studio also includes a new version of C#. Visual Studio 2015 is no exception to this: its new C# compiler supports version 6 of the language. Unlike previous language updates, this version is not focused on a single major feature. Instead, it includes several smaller improvements that did not make it into previous releases. Most of these features make the language more concise and less error prone.

However, ***does it make sense to change your existing C# code base to take advantage of the new C# 6.0 features?*** Let us find out.

What Has Changed in C# 6.0?

Changes in C# 6.0 make the language less verbose. Most of them affect declarations of class members and building of expressions. The only exceptions are improvements to `try-catch-finally` statements and the introduction of `using static` clause. Filip Ekberg already wrote an article about them when Visual Studio 2015 was still in preview, but a lot has changed between then and the final release. Instead of just writing an updated article on new language features, I will rather focus on the benefits that they can bring to the code base, when used appropriately.

Best Features of C# 6.0

I will start out with new features, which you should start using in your code base immediately, wherever applicable, because they can have a positive impact on the code maintainability and readability. Some of these new features are: `nameof` operator, `null-conditional` operator, and support for `await` in `catch` and `finally` blocks. I would strongly suggest you apply them even to your existing code once you have switched to Visual Studio 2015.

Nameof Operator

Base class libraries sometimes require you to put symbol names (such as property and parameter names) as string literals in your code, e.g. when your class implements `INotifyPropertyChanged` interface or when throwing an `ArgumentException`:

```
public int GetFibonacciNumber(int n)
{
  if (n < 0)
  {
    throw new
    ArgumentException("Negative qvalue
    not allowed", "n");
  }

  // TODO: calculate Fibonacci number
  return n;

}
```

When you rename the parameter, you will need to remember to change the string literal as well – the compiler will not warn you about it.

The `nameof` operator in C# 6.0 is a perfect solution to this problem: `nameof(n)` call will compile into a string literal while keeping strong typing at the source code level:

```
public int GetFibonacciNumber(int n)
{
  if (n < 0)
  {
    throw new
    ArgumentException("Negative value
    not allowed", nameof(n));
  }

  // TODO: calculate Fibonacci number
  return n;
}
```

Not only will the compiler now warn you if you forget to rename the parameter usage in `nameof` operator, rename refactoring in Visual Studio will automatically rename it for you.

Null-conditional Operator

Whenever you want to invoke a property or a method on a class, you need to check that the instance is not `null` beforehand, otherwise the dreaded `NullReferenceException` will be thrown at run time. This can result in a lot of trivial code that exists only to check for values not being `null`:

```
public int GetStringLength(string arg)
{
  return arg != null ? arg.Length : 0;
}
```

Null-conditional operator in C# 6.0 can replace all such `null` checking code:

```
public int GetStringLength(string arg)
{
  return arg?.Length ?? 0;
}
```

The `arg?.Length` in the above code will return `arg.Length` when `arg` is not `null`; otherwise, it will return `null`. You can even cascade

multiple calls one after another: `arg?.Length?.ToString()` will return null if arg or arg.Length are `null` and will never throw an exception.

Null-conditional operator is not limited to properties and methods. It also works with delegates, but instead of invoking the delegate directly, you will need to call its `Invoke` method:

```
public event PropertyChangedEventHandler
PropertyChanged;

private void
NotifyPropertyChanged(string
propertyName)
{
  PropertyChanged?.Invoke(this, new
  PropertyChangedEventArgs
  (propertyName));
}
```

Implementing events this way also ensures that they are thread-safe, unlike their naïve implementation in previous versions of C#, as shown here:

```
public event PropertyChangedEventHandler
PropertyChanged;

private void
NotifyPropertyChanged(string
propertyName)
{
  if (PropertyChanged != null)
  {
    PropertyChanged(this, new
    PropertyChangedEventArgs
    (propertyName));
  }
}
```

In multithreaded scenarios, the above code could throw a `NullReferenceException`, if another thread removed the last remaining event handler; after the first thread checked the event for being null, but before it called the delegate. To avoid this, `PropertyChanged` needed to be stored into a local variable in previous versions of C#, before checking it for null.

The null-conditional operator in C# 6.0 takes care of this automatically.

Support for Await in Catch and Finally Blocks

The `async` and `await` keywords, introduced in C# 5.0, made asynchronous programming much easier. Instead of creating a callback for each asynchronous method call, all the code could now be in a single `async` method, with the compiler creating the necessary plumbing instead of the developer. Unfortunately, `await` keyword was not supported in `catch` and `finally` blocks. This brought additional complexity to calling asynchronous methods from error handling blocks when they were also required to complete before continuing. Here's an example:

```
public async Task
HandleAsync(AsyncResource resource, bool
throwException)
{
  Exception exceptionToLog = null;
  try
  {
    await resource.
    OpenAsync(throwException);
  }
  catch (Exception exception)
  {
    exceptionToLog = exception;
  }
  if (exceptionToLog != null)
  {
    await resource.
    LogAsync(exceptionToLog);
  }
}
```

Of course, if you wanted to rethrow the exception and keep the stack trace, it would get even more complicated.

With C# 6.0, none of this is required any more – `async` methods can simply be awaited inside both `catch` and `finally` blocks:

```
public async Task
HandleAsync(AsyncResource resource, bool
throwException)
{
  try
  {
    await resource.
    OpenAsync(throwException);
  }
  catch (Exception exception)
  {
    await resource.LogAsync(exception);
  }
  finally
  {
    await resource.CloseAsync();
  }
}
```

The compiler will do all the necessary plumbing itself, and you can be sure that it will work correctly.

Recommended C# 6.0 Features for New Projects

Some additional new features in C # 6.0 can simplify writing new code, but do not add enough benefits to justify modifying existing code. All of them are just syntactic sugar, which will save you some typing, but will not make much difference otherwise.

String Interpolation

If C# is not the only programming language you work with, you might be familiar with this language construct called "String Interpolation" from other languages such as PowerShell, TypeScript, Python, etc. In most cases, you can use it instead of `String.Format` calls, making them easier to read and safer to use. Here is a simple example in C# 6.0:

```
var formattedPrice = $"{price:0.00} €";
```

It is equivalent to the following String.Format call:

```
var formattedPrice = String.
Format("{0:0.00} €", price);
```

Notice the following key aspects:

- Literals for string interpolation must start with $ character. This makes them of type `FormattableString`.
- You can directly use variable names and expressions in placeholders, instead of numerical values. They will be checked at compile time.
- Additional formatting strings can still be included in the placeholder after the colon, similar to `String.Format`.

Nevertheless, string interpolation is not a universal replacement for `String.Format`, e.g. you will still need to use `String.Format` for all localized strings and the translators will still have to deal with numeric placeholders.

Improvements to Member Declarations

You can take advantage of several new language constructs in your member declarations:

- Auto-properties can now be initialized the same way as fields:

```
public string FullName { get; set; } =
"Damir Arh";
```

- Read-only auto-properties are now supported as well. They create a `readonly` backing field. Their value can only be set with an auto-property initializer or inside the constructor:

```
public string FullName { get; } = "Damir
Arh";
```

- Simple methods and read-only properties can have their body defined with a lambda-like expression:

```
public int Add(int a, int b) => a + b;
public string FullName => _name + " " +
_surname;
```

Exception filters

The final feature in this category is new only in C# 6.0; both Visual Basic and F# have already supported it in previous versions. It can be used to filter the exception caught by a specific `catch` block not only by its type but by using any exception property:

```
try
{
  // ...
}
catch (ArgumentException exception) when
(exception.ParamName == "ignore")
{ }
```

Before exception filters were available in C# 6.0, you could write the same code in previous versions of C# in the following manner:

```
try
{
  // ...
}
catch (ArgumentException exception)
{
  if (exception.ParamName != "ignore")
  {
    throw;
  }
}
```

As you can see in the first example, the code using exception filters is easier to understand, and it keeps the stack untouched when the filter condition is not matched. Although catching and re-throwing the exception in the second example preserves the stack trace inside the exception, the top frame on the CLR stack will point to the place it was rethrown, instead to where it was originally thrown. This can make an important difference in the information available when analysing a crash dump resulting from such an exception.

The Exception filter feature can also be abused for unobtrusive logging:

```
try
{
  // ...
}
catch (Exception exception) when
(Log(exception))
{ }
```

This way the `Log` method can log all the exception details without affecting the CLR stack. Of course, it must return `false` to prevent the `catch` block from executing.

Remaining C# 6.0 Features

There are a couple of new language features in C# 6.0, which I have not mentioned yet. To be honest, in most cases I don't recommend using them, because although they can make the code less verbose, they can also make it more difficult to understand, especially in larger teams and in projects with longer time span.

For the sake of completeness, I am nevertheless listing them here:

- **Using static** clause can be used to import static class members into scope, so that they do not need to be prefixed with the class name any more. With the exception of a few well-known static classes, such as `Math` and `Console`, this can make it difficult to know where the methods in the code come from. In the following example both `System.Console` and `System.Math` are statically imported, which makes `WriteLine` and `Pow` methods directly available inside the scope:

```
using static System.Console;
using static System.Math;

class Program
{
  static void Main() {
    WriteLine(Pow(10, 2));
  }
}
```

- **Index initializers** expand on collection initializers and allow initialization of indexed values within a single expression. They can be used with any type having an indexer, but for most types, the same can be achieved using collection initializers. In the example below, index initializers are used to initialize a dictionary:

```
var properties = new Dictionary<string,
string>
{
  ["Name"] = "Damir",
  ["Surname"] = "Arh",
  ["Country"] = "Slovenia"
};
```

- **Collection initializers** are not limited to calling instance methods named `Add` anymore. Now extension methods with the same name can be called as well. The following example will work as long as there is an extension method named `Add` accepting a single parameter (`Dictionary` class does not have such a method). For a developer unaware of such an extension method, the initializer can be completely incomprehensible:

```
var typeNames = new Dictionary<Type,
string>
{
  typeof(string)
};
```

Before using any of the language constructs in this section, consider the following:

- Does this construct really make the code better?
- Is there no better way to achieve the same result?

How Can Visual Studio Help?

Compared to the previous releases, static code analysis tools are vastly improved in Visual Studio 2015. It introduces unified framework diagnostic analyzers, which are presented to the user the same way, whether they are built into Visual Studio or provided by third party extensions or NuGet packages.

Code fixes are a part of this framework and *could* prove useful in refactoring existing code to take advantage of new features in C# 6.0. The ones that are enabled for the project appear as information, warnings or errors in the code (depending on the configuration).

When activated in the editor window by clicking on the lightbulb icon or by pressing **Ctrl + .** ,the fix (i.e. the refactoring) can be applied only to a single occurrence, or to all occurrences in a document project or complete solution. This means that with an appropriate code fix, a specific new language feature *could* be applied to the whole solution in one single action.

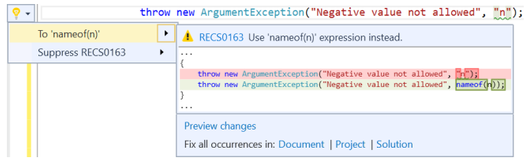


Figure 1: Visual Studio diagnostic analyzer in action

Unfortunately, there are no such code fixes built into Visual Studio. To my knowledge, even in the third party space, there is only a single code fix available that applies one of suggested refactorings explained in this article: "Use 'nameof(n)' expression instead.", distributed as a part of Refactoring Essentials Visual Studio extension at http://bit.ly/1HkOtnt .

This does not mean that you are on your own though. The renowned Visual Studio extension ReSharper by JetBrains has been updated for Visual Studio 2015 and C# 6.0 and in its latest release (9.2 at the time of writing), it includes so called inspections for many of the new language features.

- nameof operator,
- null-conditional operator,
- string interpolation,
- read-only auto-properties, and
- expression-bodied properties.

The suggested fixes are also displayed as warnings and can by default be applied via the lightbulb icon or by pressing Alt + Enter. There is no preview of the changes though.

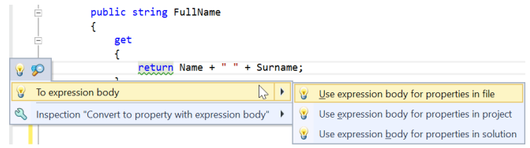


Figure 2: Applying ReSharper quick-fixes

Most of the fixes can be applied at the document, project or solution scope with a single action, as well. For those that can't be applied, invoking code inspection for the whole solution will create a list of all issues found, which can be used for navigating to them in code and fixing them one by one. It is still much quicker and reliable than doing it completely by hand.

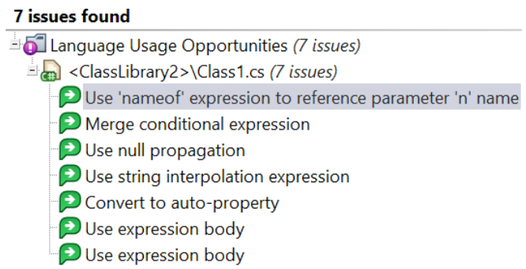


Figure 3: ReSharper code inspection results for solution

Unlike Refactoring Essentials, ReSharper is not a free extension and you will need to pay for it if you want to use it beyond the 30-day trial period. It does offer a lot more than just the refactorings, mentioned here though.

Conclusion:

C# 6.0 brings many small improvements to the language. The categorization of these into three different categories in this article is based solely on my opinion and experience. As such, it is very subjective and your opinion on individual new features may vary. Nevertheless, if all of your team members are using Visual Studio 2015 (which is a prerequisite for using C# 6.0 language features), I strongly suggest, you take a closer look at what is new and decide for yourself which features you want to use, and which not. Of course, I would be very glad if you base your final decision at least partly on my opinions expressed in this article ■

About the Author

damir arh

Microsoft® MVP Most Valuable Professional

Damir Arh *has many years of experience with Microsoft development tools; both in complex enterprise software projects and modern cross-platform mobile applications. In his drive towards better development processes, he is a proponent of test driven development, continuous integration and continuous deployment. He shares his knowledge by speaking at local user groups and conferences, blogging, and answering questions on Stack Overflow. He is an awarded Microsoft MVP for .NET since 2012.*

CREATING GRAPHICS USING SVG AND ANGULARJS

Introduction

A couple of months ago I started working with a startup called Genome Compiler that specializes in software platform to accelerate Genome and DNA design. Their main product was created using Flash and they wanted me to help them to create a new Genome viewer using plain web technologies. They wanted to visualize plasmids, which are small DNA molecules represented as a circle with annotations. They also wanted to visualize sequences, which are the primary structure of a biological molecule written in A, T, G and C characters. To create such visualization we needed the ability to create graphics inside the browser.

For more than four months I've been helping *Genome Compiler* to create their viewer using both Scalable Vector Graphics (SVG) and AngularJS. During the time, I learned how to combine SVG and AngularJS together to create biological models and other graphics.

In this article I'll explore what SVG is. Then, I'll explain how SVG can be used in AngularJS applications. Towards the end of the article, you will see a simple application that combines both SVG and AngularJS.

Note: *The article won't cover the project that I'm doing for Genome Compiler due to the fact that it's too huge to be covered in an article (or even in a book).*

Disclaimer: *This article assumes that you have basic knowledge of AngularJS. If you are not familiar with AngularJS, I encourage you to stop reading and start learning about this framework today.*

Editorial Note: *You can learn more about AngularJS using our tutorials at* http://www.dotnetcurry.com/tutorials/angularjs

SVG in a Nutshell

SVG is an XML-based graphics model that you can use in your front-end. As opposed to other new HTML5 graphics models (such as canvas and WebGL), SVG version 1.0 was made a W3C recommendation in 2001. The SVG developers' adoption was very small due to the popularity of plugins such as Flash, Java and Silverlight, and the lack of browser support. In 2011, W3C introduced the second edition of SVG, version 1.1, and SVG gained a lot of attention as an alternative graphics model, besides the Canvas pixel graphics model.

SVG is all about vector graphics. With SVG you can create and draw two-dimensional vector graphics using HTML elements. These HTML elements are specific to SVG, but they are part of the Document Object Model (DOM) and can be hosted in web pages. The vector graphics can be scaled without loss of image quality. This means that the graphics will look the same in different screens and resolutions. This makes SVG a very good candidate to develop graphics for applications that can be run on different screens (mobile, tablets, desktop or even wide screens). Some prominent areas where Vector graphics are used are in CAD programs, designing animations and presentations, designing graphics that are printed on high-res printers and so on.

The fact that SVG elements are HTML elements makes SVG a very interesting graphics model. You can get full support for DOM access on SVG elements. You can use scripts, CSS style and other web tools to shape your graphics and manipulate it. As opposed to Canvas, which doesn't include state, SVG is part of the DOM and therefore you have the elements and their state for your own usage. This makes SVG a very powerful model. There is one notable caveat though – drawing a lot of shapes can result in performance decrease.

When you use SVG, you define the graphics within your HTML using the SVG tag. For example, the following code snippet declares an SVG element:

```
<svg version="1.1" xmlns="http://www.
w3.org/2000/svg">
</svg>
```

It is an HTML element and can be embedded inside your web page as any other HTML element. If not stated, the width of the SVG element will be 300 pixels and its height 150 pixels.

Note: *Pay attention to the SVG XML namespace which is different from regular HTML. We will use this information later on when we will use SVG with AngularJS.*

Using Shapes

SVG includes a lot of built-in shape elements that can be embedded inside the **SVG** tag. Each shape has its own set of attributes that helps to create the shapes appearance. For example, the following code snippet shows how to create two rectangles with different colors using the **RECT** element:

```
<svg width="400" height="200"
version="1.1" xmlns=
"http://www.w3.org/2000/svg">
  <rect fill="red" x="20" y="20"
  width="100" height="75" />
  <rect fill="blue" x="50" y="50"
  width="100" height="75" />
</svg>
```

The output of running this piece of SVG will be as shown here:

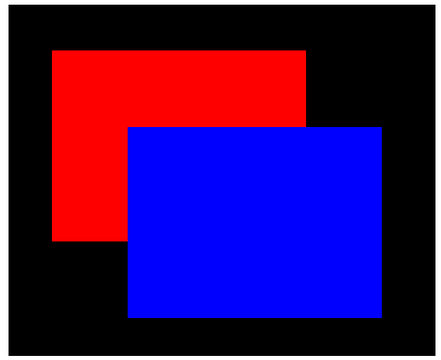


Figure 1: Two rectangles drawn by SVG

Here are a couple of points to note in the snippet:

1. The last rectangle will appear on top of the first rectangle. SVG behavior is to put the last declared elements on top of previously declared elements, if there are shapes that overlap.

2. In order to create a rectangle, you have to indicate its left-top point using the **x** and **y** attributes and its **width** and **height**. The default values for these attributes are all set to 0 and if you don't set them, the rectangle will not be drawn on the SVG surface.

There are other shapes that you can use such as circles, ellipsis, polygons, polylines, lines, paths, text and more. It is up to you to learn to draw these shapes and a good reference for the same can be found in the Mozilla Developer Network (MDN) - https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Basic_Shapes.

Other than attributes, shapes can also include styling using style attributes such as **stroke** or **fill**. Stroke accepts a color to create the shape border and Fill accepts a color to fill the entire shape. You can also set styles using regular CSS but not all CSS styles can be applied on SVG elements. For example, styles such as **display** or **visibility** can be used with SVG elements but margin or padding has no meaning in SVG. The following example shows a rectangle with some inline styles:

```
<rect x="200" y="100" width="600"
height="300" style="fill: yellow; stroke:
blue; stroke-width: 2"/>
```

As you can see in the code snippet, the rectangle will be filled with yellow color, will have a blue border, with a border width of 2 pixels.

You can group shapes using the **g** element. The **g** element is a container for other shapes. If you apply style to a **g** element, that style will be applied to all its child elements. When you create SVG, it is very common to group some elements inside a **g** container. For example the following snippet shows you the same two rectangles from Figure 1 but grouped inside a **g** element:

```
<svg width="400" height="200"
version="1.1" xmlns="http://www.
w3.org/2000/svg">
 <g>
  <rect fill="red" x="20" y="20"
        width="100" height="75" />
  <rect fill="blue" x="50" y="50"
        width="100" height="75" />
 </g>
</svg>
```

Using SVG Definitions

SVG includes a **defs** element which can be used to define special graphical SVG elements such as gradients, filters or patterns. When you want to use a special SVG element, you first define it inside the **defs** element and later on, you can use it in your SVG. Make sure you specify an id for your special element, so that you can use it later.

The next snippet shows how to define a linear gradient:

```
<svg version="1.1" xmlns="http://www.
w3.org/2000/svg">
 <defs>
  <linearGradient id="lg1">
   <stop offset="40%"
   stop-color="yellow"/>
   <stop offset="60%" stop-color="green"
   />
   <stop offset="80%" stop-color="blue"
   />
  </linearGradient>
 </defs>
 <rect fill="url(#lg1)" x="50" y="50"
 width="100" height="100"/>
</svg>
```

..and the output of running this SVG will be:

Figure 2: Gradient inside a rectangle

A couple of things to notice about the snippet:

1. I defined the gradient using the **linearGradient** element. There are other elements that you can use to define other graphical aspects. Each element has its own attributes and sub elements, so it's up to you to learn them.

2. The gradient has an **id** which is later on used in the rectangle using the url(#nameOfSVGElement) syntax.

Note: *The article doesn't cover all the possible SVG element definitions.*

Now that we are familiar with SVG it is time to move on and see how SVG and AngularJS work together.

SVG + AngularJS = ❤

You can combine SVG and AngularJS and it is very straight forward. Since SVG elements are part of the DOM, you can add them into view templates both as static graphics, and also as dynamic graphics. The first option is very simple and you just embed static SVG inside the HTML. The second option has a few caveats that you need to know in order to be on the safer side.

The first caveat is dynamic attributes and data binding. Since SVG has its own XML definition, it doesn't understand AngularJS expressions. That means that if you will try to use SVG attributes with data binding expressions (curly brackets), you will get an error. The work around is to prefix all the dynamic attributes with **ng-attr-** and then set the binding expression. You can find a reference about **ng-attr-** prefix in the AngularJS website under the topic "ngAttr attribute bindings" using the following link: https://docs.angularjs.org/guide/directive .The following example shows you how to use the ng-attr and define databinding expressions:

```
<rect ng-attr-x="{{xAxis}}" ng-
attr-y="{{yAxis}}" ng-attr-
height="{{rectHeight}}" ng-attr-
width="{{rectWidth}}"></rect>
```

In the example, you can see that all the rectangle attributes are set to some scope properties.

The second caveat is related to directives. Since the SVG XML definitions are different from HTML, directives that generate SVG elements need to declare that they generate SVG. That means that in the Directive Definition Object (DDO) that you return to define the directive, you will need to set the **templateNamespace** property to **'svg'**. For example, the following snippet shows a simple directive DDO that declares that it generates SVG:

```
(function () {
  'use strict';
  angular.
    module("svgDemo").
    directive("ngRect", ngRect);
  ngRect.$inject = [];
  function ngRect() {
    return {
      restrict: 'E',
      templateUrl: <rect x="50" y="50"
      width="100" height="100"></rect>',
      templateNamespace: 'svg'
    };
  }
}());
```

Now that we know how to combine SVG and AngularJS, it is time to see some code in action.

Building a Simple App with SVG and AngularJS

The application that we are going to build will generate a rectangle and you will be able to change its width and height, using data binding:

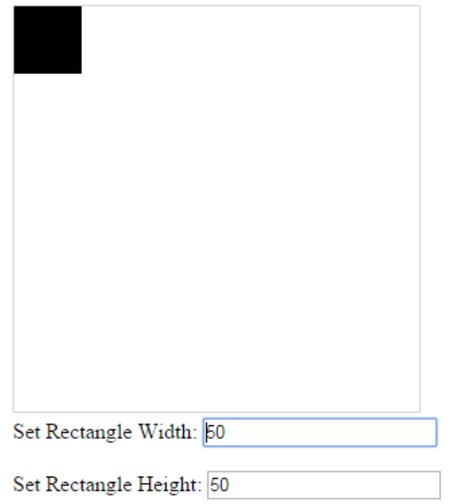


Figure 3: Application using SVG and Angular

We will first start by defining a rectangle directive that will resemble some of the code snippets that you saw earlier:

```
(function () {
  'use strict';
  angular.
    module("svgDemo").
    directive("ngRect", ngRect);
  ngRect.$inject = [];

  function ngRect() {
    return {
      restrict: 'E',
      replace: true,
      scope: {
        xAxis: '=',
        yAxis: '=',
        rectHeight: '=',
        rectWidth: '='
      },
      templateUrl: 'app/common/
```

```
      templates/rectTemplate.html',
      templateNamespace: 'svg'
    };
  }
}());
```

The directive will include an isolated scope that can accept the x, y, width and height of the rectangle. It also declares that it generates SVG and that it loads a template. Here is the template code:

```
<rect ng-attr-x="{{xAxis}}" ng-
attr-y="{{yAxis}}" ng-attr-
height="{{rectHeight}}" ng-attr-
width="{{rectWidth}}"></rect>
```

Now that we have our rectangle directive, we will define a directive that will hold our entire demo:

```
(function () {
  'use strict';
  angular.
    module("svgDemo").
    directive("ngDemo", ngSvgDemo);
  ngSvgDemo.$inject = [];

  function ngSvgDemo() {
    return {
      restrict: 'E',
      templateUrl: 'app/common/
      templates/demoTemplate.html',
      controller: 'demoController',
      controllerAs: 'demo'
    };
  }
}());
```

The main thing in the directive is the controller that it will use, and the template that it will load. So let's see both of them. We will start with the controller:

```
(function () {
  'use strict';
  angular.
    module("svgDemo").
    controller("demoController",
    demoController);
  demoController.$inject = [];

  function demoController() {
    var demo = this;
    function init() {
      demo.xAxis = 0;
      demo.yAxis = 0;
      demo.rectHeight = 50;
      demo.rectWidth = 50;
    }

    init();
  }
}());
```

In the controller, we define the properties that will be used later for data binding. Now we can look at the template itself:

```
<div>
 <div>
  <svg height="300" width="300">
   <ng-rect x-axis="demo.xAxis"
y-axis="demo.yAxis" rect-height="demo.
rectHeight" rect-width="demo.
rectWidth"></ng-rect>
   </svg>
 </div>

 <div>
    <label>Set Rectangle Width:
    </label>
    <input type="text"
        ng-model="demo.rectHeight" />
     </br></br>
    <label>Set Rectangle Height:
     </label>
     <input type="text"
        ng-model="demo.rectWidth" />
 </div>
</div>
```

Please observe the usage of **ng-rect** directive that we have created, and the binding of its attributes to the controller attributes. Also, we have bound the textboxes to the relevant properties using ng-model directive. That is all.

Now we can create the main web page and the svgDemo module. Here is how the main web page will look like:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>SVG & Angular</title>
  <link href="styles/main.css"
  rel="stylesheet"/>
</head>
<body>
  <div ng-app="svgDemo">
    <section class="body-content">
      <ng-demo></ng-demo>
    </section>
  </div>
  <script src="app/vendor/angular/
  angular.min.js"></script>
  <script src="app/app.js"></script>
  <script src="app/common/controllers/
  demoController.js"></script>
  <script src="app/common/directives/
  ngDemoDirective.js"></script>
  <script src="app/common/directives/
  ngRectDirective.js"></script>
</body>
</html>
```

And here is how the svgDemo module will be defined:

```
(function () {
  var app = angular.module('svgDemo',
[]);
}());
```

This is a very simple application but it shows you how to combine both SVG and AngularJS and to create dynamic graphics in your applications. You can also use common SVG generator libraries such as Raphael or d3.js inside your directives but the idea was to show you how to do raw SVG graphics before you jump into a library.

Summary

SVG is a very powerful graphics model that can be used in the browser. It generates graphics that looks good and scales well across different screens and resolutions. It also includes variety of elements that can help you shape your graphics easier. As you saw in the article, combining SVG and AngularJS to generate some sophisticated graphics is not so hard.

As I wrote in the introduction, I was able to generate very interesting biological models using SVG and AngularJS and this should encourage you to try and create your own models ■

Download the entire source code from GitHub at
bit.ly/dncm21-svg-angular

About the Author

Gil Fink is a web development expert, ASP.NET/IIS Microsoft MVP and the founder of sparXys. He is currently consulting for various enterprises and companies, where he helps to develop web and RIA-based solutions. He conducts lectures and workshops for individuals and enterprises who want to specialize in infrastructure and web development. He is also co-author of several Microsoft Official Courses (MOCs) and training kits, co-author of "Pro Single Page Application Development" book (Apress), the founder of Front-End.IL Meetup and co-organizer of GDG Rashlatz Meetup. You can get more information about Gil in his website http://www.gilfink.net.

DESKTOPS IN THE CLOUD

RUN DESKTOP APPLICATIONS IN THE CLOUD USING AZURE REMOTE APPS

Introduction

In this article we'll explore the following important aspects of Azure Remote Apps –

1. Azure Remote Apps - Why?
2. Concept
3. Architecture of Remote Apps deployment model
4. Demonstration

Let's get started!

Applicable Technology Stack

1. Valid Microsoft Azure subscription. A Free trial is available here.

2. Sample desktop application. You can use the one I have posted with source code that accompanies this article.

Azure Remote Apps is a fantastic feature to make your corporate desktop/ windows applications run in the Cloud, while ensuring that corporate policies and compliances are adhered to. Using this feature, users can experience true Bring-Your-Own-Device (BYOD) scenarios while using their corporate applications. This article explores various possibilities of Azure Remote Apps feature and demonstrates how to run a sample Windows application with SQL Azure DB as backend and using "cloud only collection" of Azure Remote apps, in Microsoft Azure.

Why do I need to run desktop applications in the Cloud?

Let us start with a very important question – **Why** do I need to run desktop apps in the Cloud? To understand this, we will spend some time understanding the use case of this feature.

Corporate Scenario

To understand a corporate scenario, first let me take a few steps back into the initial days of cloud computing. What was the important concept behind cloud computing evolution?

If you were an Application Development Company wanting to develop an internet application to solve a specific business problem, then to save you from the hassles of infrastructure set up and let you focus on your application, Cloud Computing came into existence. This is the basic essence of cloud computing. In Microsoft Azure, we have various features existing today that can take care of all your infrastructure needs of hosting an internet application. Some popular features are –

1. Cloud Services (Web and Worker Role) – PaaS offering

2. Azure Apps Service (Web Apps and WebJobs) – PaaS offering

3. Azure Virtual Machines – IaaS offering

These offerings take care of web application hosting. So the problem of Enterprise level *web* applications is dealt with, but what about Enterprise level *desktop* applications? Let's take an example to understand this a bit more.

Case 1 – Productivity and high sensitive data loss

I have observed a very common scenario in my country India where data entry work is outsourced from various global companies. Let's say a data entry company in India is having a couple of hundred workers performing data entry, related to health care claims information. These workers are allocated a desktop machine which has an application developed using C# or WPF. Now imagine if the hardware crashes for a desktop machine, then that unfortunate employee has to go to Support IT team of his/her organization and report the problem. The IT team will then need to provide a new desktop machine. They will need to configure the pre-requisites on the new desktop, then install the actual application. In big companies, this can take a couple of days, after which that employee can start his/her work. Additionally if there is any sensitive data on that machine, then it needs to be restored. This problem arose at the first place because a local desktop machine was being used for performing some sensitive work. In this scenario, **Productivity is affected and Loss of sensitive data is high.**

Case 2 – Device Portability

Today, the world is experiencing an explosion of devices. People want an application that can work on Laptop, should be able to work on their tablet, smartphone, mobile, desktop and so on. Additionally if an application can run on any device, users of the application don't necessarily have to be part of their corporate network and don't have to be present in their company premises. Rather they should be able to work remotely. However if an application is a legacy app, then it may not work on the latest hardware/ software. Here **device portability of desktop application and mobility is another challenge** corporates face.

Case 3 – Moving from Capex to Opex

Usually large organization take the help from vendors to perform many activities. Sometimes these corporates hire people on contract for a short duration to get their work done. In such cases, companies are not willing to spend on hardware cost as this might be a temporary affair for them. In other words, they don't want to spend money on Capex (Capital Expenditure) but they want to move to Opex (that is operational cost), so that they use the resources when in need and free them when not required. In traditional model of desktop application environments, this flexibility is quite challenging. So **moving from Capex to Opex which is recommended in today's IT world is another important challenge** corporates face.

Domestic Scenario

This section will help you understand the challenges individual desktop users may face to cope up with changing technologies, hardware and software requirements.

Let's say a user having a desktop PC is running Windows 7 basic edition with 4GB of RAM and 100GB of HDD. At a later date, he wishes to use modern software's like Office 2016, Visual Studio 2015, or assuming he is game freak; he would like to run Call Of Duty latest version on his existing desktop machine. Now all this with 4GB RAM will kill the machine if these software were to run on

it simultaneously. In fact some of the software's hardware prerequisites may not get full filled because of the existing desktop configuration. So what is the solution?

One solution is to increase the capacity of the existing machine. In that case, he will have to purchase 8GB or 16GB of RAM, a bigger capacity HDD and so on. In essence, he will have to shell out money from his pocket to make use of all the modern software's. Now even if he does invest, software gets outdated very soon. He may not need to use Office 2016 every day and after clearing all stages of the Call of duty game, he may want to get rid of it. So the additional investment done on this machine to run these modern software, will not be recoverable in this scenario and it may not get utilized to its fullest capacity. This is the problem.

High Level Solution

To overcome these challenges faced in corporate and domestic scenarios, companies like Microsoft offering cloud computing for Enterprises came up with the concept of **Desktop as a Service (DaaS)** commonly known as **Desktop Virtualization**.

So as a part of this concept, there is a desktop application installed on user's machine/ device. The user logs in to that desktop application from his machine / device. After successful login, all the modern software & services would be available for consumption. Although the services/ application run on remote machine (cloud environment), they will appear to the user as if they are running locally. Once done, the user closes the application and he would be charged only for what he has consumed.

This approach helps in –

1. Getting rid of hardware and software upgrades
2. Any modern device compatibility can be achieved easily
3. Applications scale without large capital expense
4. Provide access to corporate applications from anywhere
5. User specific shared storage helps in avoiding data loss

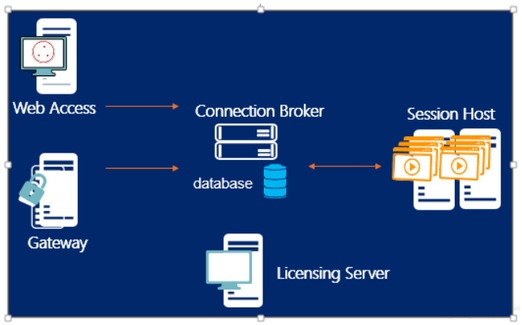
Looking at these benefits, it is quite clear running desktop applications in Cloud environment is a much better option.

Azure Remote Apps is an offering from Microsoft Azure that makes it easy to run desktop applications on the reliable platform of Azure.

The concept behind Azure Remote Apps

Running desktop application on a remote machine and accessing the app through a remote connection is the basic concept behind Azure Remote Apps. However this is not a new concept. This concept has been in practice since long and is commonly known as **Remote Desktop Service on Windows**. Basically anyone who is remoting from one computer to another computer using remote desktop connection (.rdp file) client from Windows PC, is using RDS technology behind the scene.

A typical high level architecture of RDS is as shown here –



Session Host (RDSH) - The first component of RDS is session host. This is where the actual code or application executes. You can run farms of these session host servers and you can run same application on multiple server machines.

Connection Broker - When we use multiple session host server to host the same application, then user connections should be distributed across them. This is where connection broker helps. Its job is similar to load balancer in Azure.

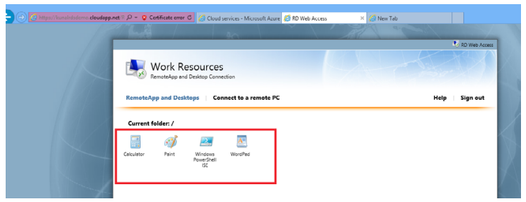
RD Web Access - RD Web Access is a simple web portal that helps users to consume applications from browsers.

Gateway - Helps you to make connection from public internet to your private network of multiple session host servers and farm.

License Server - Used for keeping a track of users using the application.

Creating RDS on Azure VM is out of scope for this article. Once you have configured RDS in Azure VM, you can publish the application from that VM and can access it using a web browser. A typical url of RDS web access is as follows: https://yourCloudServiceName.cloudapp.net/rdweb.

If you access the url of RDWeb, the browser will display a published application from session host server as shown here –



The red highlighted box displays the applications that are running on Azure VM (a remote machine) and can be accessed from the local machine (th'e RDP) from which RDWeb Access URL is opened.

And yes you guessed it correct, **Azure Remote Apps uses RDS behind the scene!**

What is Azure Remote Apps?

Azure Remote Apps is a combination of Windows applications and RDS capabilities on Azure platform that brings in scalability, agility, resiliency, high availability, global access and all cloud inherent benefits you can think of, to corporate applications.

Remote Apps Deployment Choices

Azure Remote Apps support two types of deployment.

1. *Cloud Only*

This is a very straight forward and easy deployment option. Microsoft has planned to make software readily available through this option. This is also called as "Cloud Collection". This collection resides completely in Azure.

Users can authenticate against cloud only collection based apps using Azure AD or individual Microsoft Account. As of today, with this deployment choice, you get a trial version of Office 2013.

When you provision Remote Apps cloud only namespace with Office 2013 trial version, internally Windows Server 2012 and RD session host gets configured for you. On these RD session host machine, the business application which in current case would be Office 2013 trial version, will get configured automatically and published as well. So internally they have prebuild image of Office 2013 trial version that gets configured on session host.

2. *Hybrid*

Hybrid deployment is where your application can work in Azure environment as well as on-premises environment. For Hybrid deployment, Site to Site VNET connectivity is mandatory. Also hybrid collection needs Azure AD and on premises AD setup to make corporate users authenticate against it. Additionally, in case of hybrid, you can bring your own custom app based image VM to run in Azure Remote Apps.

More deployment choices can be found out here - http://azure.microsoft.com/en-in/documentation/videos/azure-remoteapp-cloud-deployment-overview/.

Preparing custom image for Azure Remote Apps Cloud only collection

In the next section, we will see a step by step process to run our own custom desktop application developed in C# and with SQL Azure DB as backend; in Azure Remote Apps Cloud Only collection.

Earlier during the preview version of remote Apps, bringing custom applications based VM image was only possible with Hybrid collection. Starting Sep 2014, now Cloud only collection also supports custom application based VM image. Moreover the custom image can be prepared in Azure itself (by using Azure VM's) and have it configured in Azure Remote Apps cloud only collection.

Following are the high level steps we will perform going forward –

1. Create a cloud service and setup Azure storage account and upload sample DB .bacpac to storage for restore operation. (.bacpac file is present in the source code of this article)

2. Setup SQL Azure DB in Azure subscription

3. Create Azure VM based on RDS template

4. Configure application and create template image (in source code)

5. Upload template image in Azure remote Apps

6. Create cloud only collection and consume the application from Remote Apps.

The sample application we will be using was downloaded from CodeProject and it has been changed to suit our needs and demo of Azure remote Apps. The final version of the sample application download link is at the end of this article.

Note – **Steps 1 and 2 are not necessary** for Azure Remote Apps. These steps are very specific to our scenario. Instead of using SQL Azure DB, you may use SQL Server on Azure VM or on- premises SQL server depending on your application requirements. In that case, you may not need to perform these steps at all.

Steps 3 to 6 are generalized steps and any production Azure Remote Apps cloud collection deployment will need it.

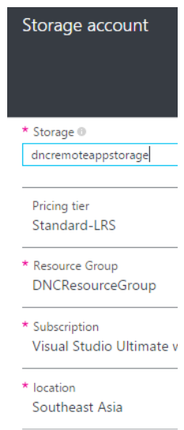
Setup Azure Storage account and Cloud Service

Login to Azure portal with your Microsoft account. Then select the subscription of your choice.
We need to have storage account created in our scenario because the SocialClub.bacpac file is present as of now on your local machine. To create a database in SQL Azure DB, we will be using Azure Portal. However we can create/ import the database from Azure Portal only if the .bacpac is present in Azure storage account.

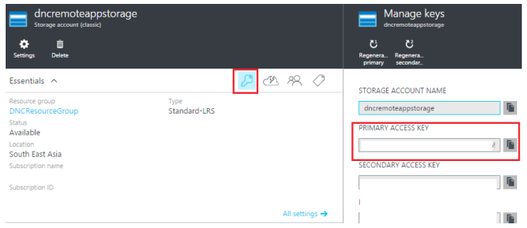
Click on Browse > Data + Storage > Storage Account > Classic Deployment Model > Create. Provide the values as shown here:

Name	Dncremoteappstorage
Pricing Tier	LRS
Resource Group	DNCResourceGroup [Select Create New option]
Subscription	<Your choice>.
Location	South East Asia
Diagnostics	Not Configured

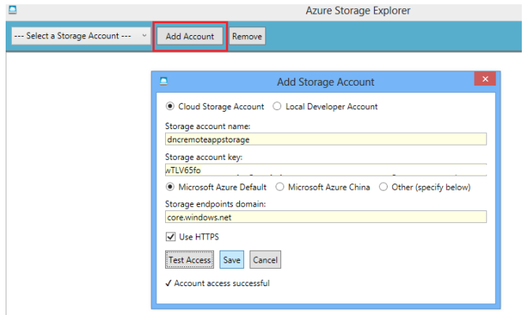
Click Create button to proceed with storage account creation.



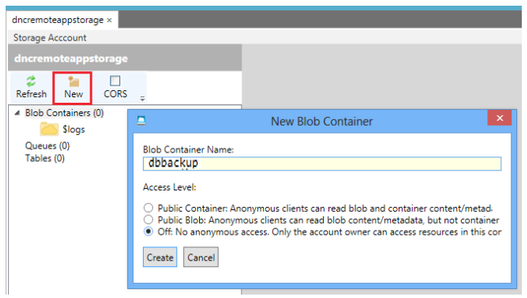
Click on the Key icon of storage account and note down the name and primary key which we will need to create container and blob in storage account.



Now we need to upload the SocialClub.bacpac file to blob within this storage account. I usually use the free storage explorer known as Azure Storage Explorer to perform most of the storage operations. It is free and satisfies all basic storage operations need. Alternative to this will be Visual Studio 2012 and above. Assuming you have installed Azure Storage Explorer, click on "Add Account" button. Provide the name of storage account and the primary key we noted in the previous step, and click on Save.

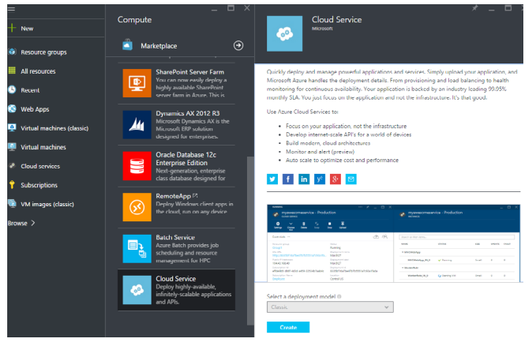


In Azure Storage Explorer, after successful connection, select "Blob Containers" -> "New". A pop up will appear wherein put the container name as "dbbackup" and Access Level as "Off:No anonymous access". Click create.



After container creation, select it and click on "Upload" button. A file selection window will appear. Select the Social.Club.bacpac file path from your local machine and click ok. This will create a blob named as "SocialClub.bacpac".

Similarly click on New > Compute > Cloud Service > Create.



Provide the values as depicted below –

DNS Name	DNCRemoteAppDemo
Subscription	<Your Choice>
Resource Group (select existing)	DNCResourceGroup
Location	South East Asia
Package and Certificate	<Keep default as it is>

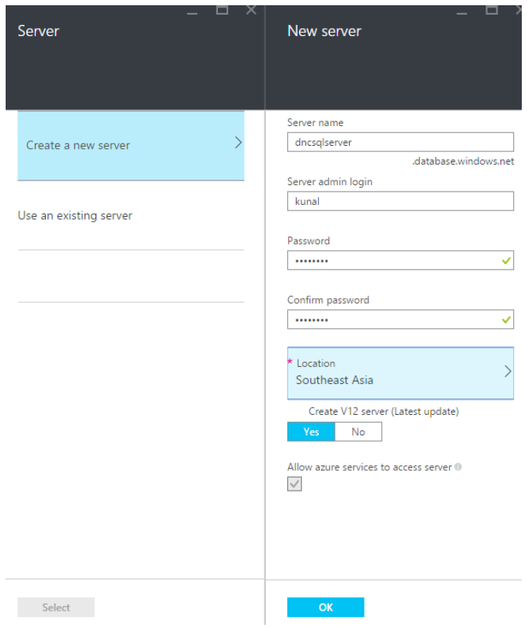
Click on Create button to finish cloud service wizard.

Setup SQL Azure Server and SQL Azure DB

In the Azure preview portal, we first need to setup SQL Server (PaaS). However standalone SQL server creation is not available as of today unless a single DB exists in it. Therefore we will create Test database along with server. After server creation, we will delete the Test database and create our own SocialClub database.

In Azure Management Portal click on New > Data + Storage > SQL Database. Provide the values of server configuration settings as shown here -

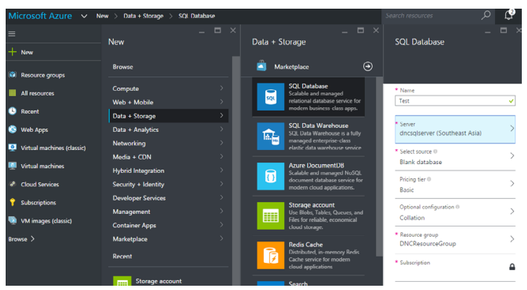
Database name	Test
Server Name	dncsqlserver
Server Location	South East Asia
Server Admin credentials	<Your choice>
Create V12	Yes
Allow Azure Services to access server	Yes [click the checkmark]



The rest of the values is as shown here –

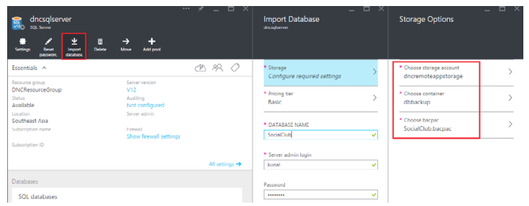
Source	Blank Database
Pricing Tier	Basic
Optional Configuration	<Keep as it is>
Resource Group	DNCResourceGroup
Subscription	<Your Choice>

Click on create to finish SQL server DB creation wizard.

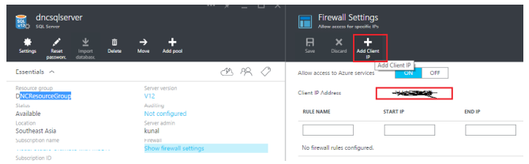


After SQL Server creation, open the database Test that we created in above steps and since we don't need it, delete it by clicking on Delete button in management portal.

To import the actual database, open Resource group "DNCResourceGroup" in Azure portal. Open the server "dncsqlserver". Select the values as shown below to import the SocialClub.bacpac –



Click on "Show Firewall Settings". The firewall settings will list current machine IP address automatically. Click on "Add Client IP" and then "Save" button to make sure that SQL server DB is accessible from the local machine.



Now open the app.config file of the project SocialClub.Desktop and replace the name of server, database, username and password in connection string. A sample connection string would be as follows –

```
connectionString="Server=
tcp:dncsqlserver.database.windows.net,
1433;Database=SocialClub;
User ID=kunal@dncsqlserver;
Password=<YourPasswordGoesHere>;
Trusted_Connection=False;
Encrypt=True;Connection Timeout=30;"
```

This completes the SQL server DB configuration for our scenario.

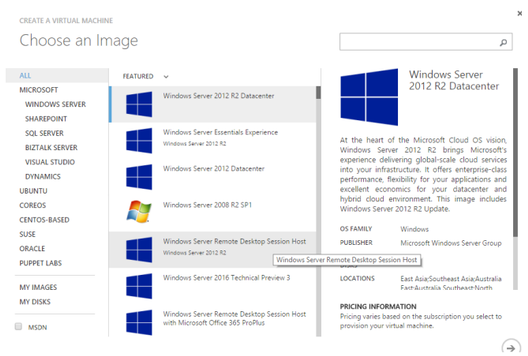
Create Azure VM based on Remote Desktop Session Host template

In this step, we will create a VM template image that can be used to run our sample desktop application of SocialClub. This step is nothing but creating Azure remote app image based on Azure VM. However if you look at the prerequisites and steps mentioned in this link, creating custom image based on Azure VM is complex. Therefore to relieve the user from this cumbersome job, Azure provides you ready base template VM that satisfies all prerequisites of Remote App image and on top of it, you can have it customized for your needs. Therefore we will use the same one as recommended which saves you from many complex steps.

Note: For creating VM, we should use the New Azure Preview portal (https://portal.azure.com). However RemoteApps and the template image of Remote Desktop Session Host as of this writing is not available on preview portal. In future it will be available on preview portal but for this article we have to fall back to full management portal (http://manage.windowsazure.com). So by the time you read this article and proceed, check to see if remote apps is available on preview portal. The steps would be similar and will not change to a great extent even if you are following new portal in the future.

Open the full management portal and login with Microsoft account and select the subscription of your choice.

Click on New > Compute > Virtual Machine > From Gallery option. This will open up a pop up. Select the VM image named as "Windows Server Remote desktop Session host" as shown here and click Next to continue –



On the Virtual Machine configuration page, provide values as given below and click on Next to continue –

Version Release Date	<keep the default as it is>
Virtual Machine Name	DNCRemoteAppVM
Tier	Standard
Size	A2
Username and Password	<your choice>

Now on the subsequent Virtual Machine Configuration page, provide the following values and click Next to continue –

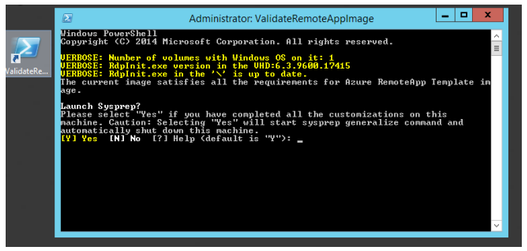
Cloud Service Dropdown	DNCRemoteAppDemo
Region/ Affinity Group/ Virtual Network	South East Asia
Storage Account	Dncremoteappstorage
Availability Sets and Endpoint	<keep the default as it is>

In the next screen, select the check box "Install the VM agent" and click on Complete to start the VM creation.

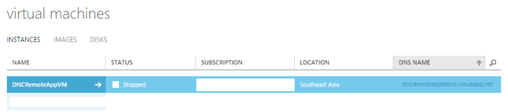
Once the VM is in running state, download the RDP file by clicking on "Connect" button in management portal and login with RDP credentials that was provided during provisioning. Now we need to copy our sample SocialClub application on this VM so that we can publish the application later when Azure Remote App is provisioned. Open Visual Studio 2013 and build the sample application in Release mode. After this, create a folder named SocialClub on 'C' drive of Azure VM and copy the contents of Release folder. Please make a note of the path of exe as we will need it later during publishing programs in Azure Remote Apps collection. If you have created a folder on C drive with the same name, then your complete path will be – **C:\SocialClub\SocialClub.Desktop.exe**

To verify everything is working, run the exe named as SocialClub.Desktop.exe on Azure VM and confirm if the application is working fine.

Now we are done with customization of our Azure Remote App VM. All we need to do is to check the VM against prerequisites and capture the template image using **Sysprep**. As we have provisioned our VM based on Remote Desktop Session Host image, we will have a readymade PowerShell script shortcut present on the desktop of VM named as "ValidateRemoteAppImage". Just run it and it will start the entire process of validating the current VM against remote app requirements. Once PowerShell is through with VM validation, a prompt will popup in the same window asking for Sysprep. Input "Y" to proceed ahead.

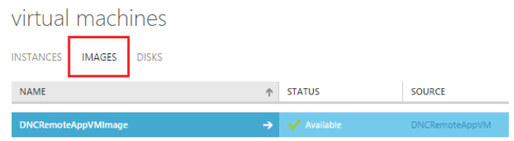


Sysprep is a utility used for generalization of VMs. After sysprep, you can capture the image of your VM. You can then upload it to Azure Remote App program to provision the required session host VM. Once the sysprep process completes, the RDP will get disconnected automatically and status of the VM will be shown as "Stopped" in the management portal as seen here –



Click on the Capture button present at the bottom pane of the portal and provide the values as shown here –

This process will capture the VM image and it will be available under the Images tab in management portal.



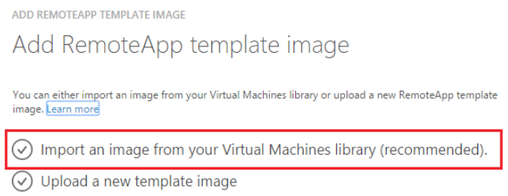
Now you must have observed that the original VM has been deleted. However the associated OS disk of original VM is still present. As we have captured the image from this VM, we do not need this disk. Moreover after a generalized process, it is not recommended to use the same disk for provisioning a new VM. Therefore instead of retaining this unnecessary disk, just delete it. So click on "Disks" tab and look out for the original VM OS disk. It must be having name of VM in it. So locate and delete the disk from management portal by selecting option of "Delete the associated vhd".

This completes the process of custom template VM image creation.

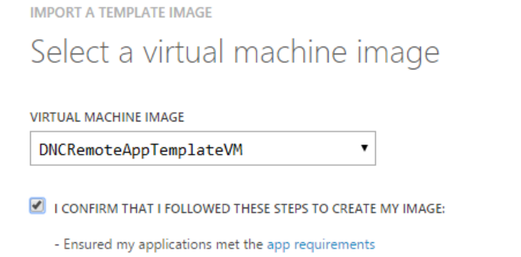
Upload VM image to Azure Remote Apps

To upload VM image to Remote App, click on Remote Apps option in management portal. Select the tab "Template Images" and select "Import or Upload a template image" or click on "Add" button as shown here –

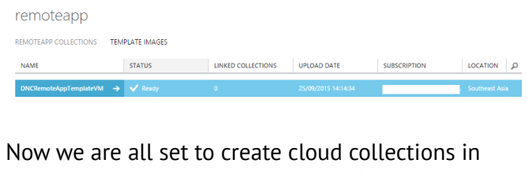
Select the option "Import an image from your Virtual Machine library (recommended)" and click on Next to continue –



The image we had created earlier will appear in the drop down. So click on the checkbox that has the confirm option as shown here and click on Next to continue.



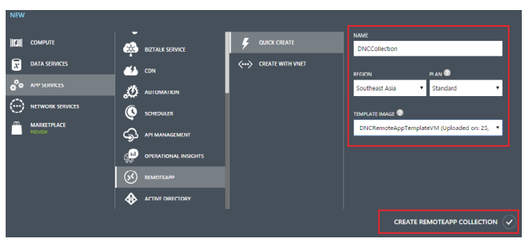
Provide the name of remote app template as "DNCRemoteAppTemplateVM" and location as "South East Asia" and click create. The image upload to the Azure Remote App is an asynchronous process and it will take a good amount of time depending on your bandwidth connection. After a successful upload operation, the image will appear as shown here –



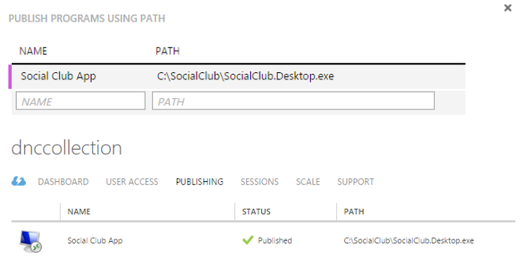
Now we are all set to create cloud collections in Azure Remote App and publish our SocialClub application.

Create Remote App Cloud collection

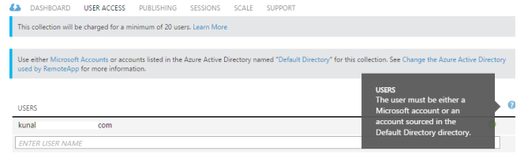
Select New > App Services > Remote App > Quick Create option. Provide the name as "DNCCollection", region as South East Asia, Plan as Standard and most important, the template image as "DNCREmoteAppTemplateVM". Then click on "Create Remote App Collection" option.



Creating remote app collection takes at least half an hour to get ready for use. Once the collection status becomes active, click on it and select the "Publishing" tab. Click on the "Publish" button present at the bottom and select the option "Publish Program using Path". A pop up will appear. Provide the name of program as "Social Club App" and path of the program which we had noted earlier when we created custom image – "**C:\SocialClub\SocialClub.Desktop.exe**". Now click on Complete.



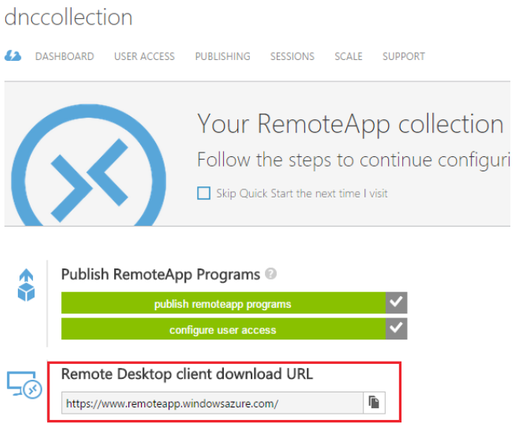
Now we need to make sure that proper user access is given. Therefore click on "User Access" tab and verify that your Microsoft account has access. In case you wish to add access to another Microsoft account, you can do so.



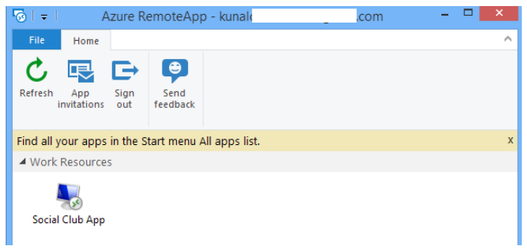
Now we are all set to use the application we published.

Using the Remote App published application

The beauty of remote apps is that you can run/access the published application on any platform by using Remote App client. As shown below, the url provides a download link from which you can download any version of the client app –



Open the link and click on the "download" menu. Download the client exe for Windows platform as I will be using it from my Windows 8 laptop. Alternatively you can have it downloaded as per your device platform. Once the client application has downloaded, signin with your Microsoft Account which has access to remote app collection. After successful login, the SocialClub application will appear as shown below –



Double click on Social Club app and a RDP session will get activated automatically and the login screen of our Social Club application will appear. Click on Search/ Manage Members menu which will retrieve data from SQL Azure DB and populate the existing members, if any. Any any new members and data should get inserted in SQL Azure DB.

This way you can download the client app on any device running any platform such as iOS, Android, Windows and access the application. You don't need any installation, configuration to be done on your device and this is greatest benefit of Azure Remote App. The application is running on remote computer in Azure and it appears as if it is running on your local machine. I hope the steps above must have given you good understanding of running desktop application in cloud using Azure remote apps. Now in next section we will particularly understand an advanced scenario of **Redirection** which is very common in enterprise corporate applications.

Advance topic - Redirection and User Profile Disk (UPD)

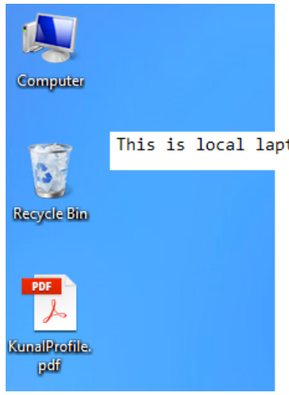
Azure remote app users get their own dedicated persistent storage space of 50GB and it is stored in a disk called as User Profile Disk (UPD). The user's configuration data, personal data and customization data across various devices is stored in UPD. The name of the disk is derived from the user name. When a user saves any data, it always gets saved in his Documents folder and it appears as if it is local to his device [which is not]. There is some excellent information already present in the documentation about UPD at http://bit.ly/206TMxm .

Our focus in this section is to determine how exactly can you use this feature in your application.

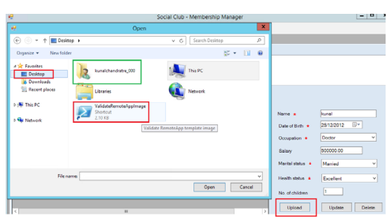
In the same sample application, I have added a button called as "Upload" under Manage tab. Click on it and a file dialog will open. The functionality of this button is simple. It allows a user to upload a file which is a very common scenario in many desktop based corporate or enterprise application. So in this case, the file should get uploaded to the user specific UPD. The code I have written on this button click is very straight forward and is shown here –

```
OpenFileDialog fd = new
OpenFileDialog();
DialogResult result = fd.ShowDialog();
if (result == System.Windows.Forms.
DialogResult.OK)
{
  string file = fd.FileName;
  string fileName = Path.
  GetFileName(file);

  //get current logged in user
  MyDocuments folder
  string documentPath = Environment.
  GetFolderPath(Environment.
  SpecialFolder.MyDocuments);
  //MessageBox.Show(documentPath);
  File.Copy(file, documentPath + "\\" +
  fileName);

  MessageBox.Show("File uploaded
  successfully");
}
```

It simply copies the file selected to current logged in User's Document folder. So in this way, by using `Environment.SpecialFolder.MyDocuments` we are accessing UPD of current logged in user in Azure remote app based application. Let's say using the upload functionality of our Social Club application, we want to upload a PDF document from our "desktop" to Azure remote App UPD based Document folder. The screenshot of my desktop file is as shown here –



When we click on the Upload button in the application, the folders and files present on the desktop of local laptop should be listed. We click on Upload button and click on desktop and unfortunately we don't SEE LOCAL LAPTOP DESKTOP. Instead the desktop of Remote App RDSH VM is displayed, which is running Social Club application in Azure as shown below –



The reason of it showing Remote App VM desktop is quite obvious as the application is not running on our local laptop, rather it is running in Azure VM, hence desktop of Azure VM is displayed in file dialog box which is perfectly fine. However we want to **save the document from the desktop of our laptop to Document folders** (which is UPD) highlighted in green in the screenshot. In your case, the green folder will be displaying your name.

As the application is running in Azure VM, LOCAL FOLDER and FILEs of laptop are not getting listed through file dialog box and to make it happen we need **REDIRECTION**.

Redirection helps users to interact with Remote Apps using the devices attached to their local computer, laptop, tablet or phone. The devices can be camera, printers, speakers, monitors and so on. Of course by default, some of the redirection services are already enabled when you use remote app based application on your devices, however drive redirection is not enabled by default and that's the reason local computer drives are not shown in remote session. Once we enable the drive redirection, local computer drives get mapped to remote session and we would be able to access local folders and files in remote session.

To enable drive redirection, we need to use Azure PowerShell. Make sure that you have Azure PowerShell latest version downloaded and installed on your local machine from http://bit.ly/1PPnfr9 .

Once installed, open the Azure PowerShell window as administrator. Run the following commands to make a connection to your Azure subscription.

Add-AzureAccount

Select-AzureSubscription –SubscriptionId "YourSubscriptionId"

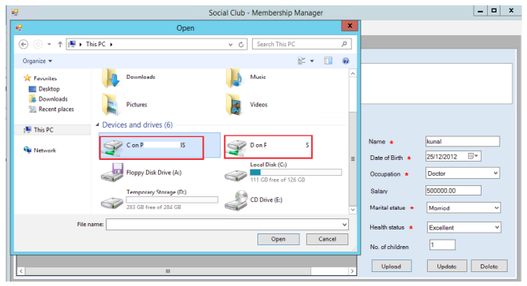
The azure subscription Id can be found out from the "Settings" option in management portal as shown below –

To enable drive redirection, run the following command –

Set-AzureRemoteAppCollection -CollectionName "dnccollection" -CustomRdpProperty "drivestoredirect:s:*"

In your case, the collection name can be different in the above command. This commands enables the drive redirection to all the published apps within Azure Remote App collection.

Now open the remote app client and open Social Club application again. Click on Upload and then you should be able to view the local drives of your computer as shown here –



Go ahead and select the path of local desktop as c:\Users\<YourLoginName>\Desktop\<YourFileName> and the selected file will get copied to Documents folder of Remote apps and "file uploaded successfully" message will appear in the application. This is how Redirection can be used to make remote app communicate with local devices.

Is Redirection a new concept?

Redirection is not new. It is already being used with RDP files. For example, get a .rdp file and right click on it and select "Open With Notepad".

Once the RDP file is open in notepad you will see something similar to the following –

```
File  Edit  Format  View  Help
prompt for credentials:i:1
screen mode id:i:2
desktopwidth:i:1440
desktopheight:i:900
session bpp:i:32
winposstr:s:0,3,0,0,800,600
compression:i:1
keyboardhook:i:2
audiocapturemode:i:0
videoplaybackmode:i:1
connection type:i:7
networkautodetect:i:1
bandwidthautodetect:i:1
displayconnectionbar:i:1
enableworkspacereconnect:i:0
disable wallpaper:i:0
allow font smoothing:i:0
allow desktop composition:i:0
disable full window drag:i:1
disable menu anims:i:1
disable themes:i:0
disable cursor setting:i:0
bitmapcachepersistenable:i:1
audiomode:i:0
redirectprinters:i:1
redirectcomports:i:0
redirectsmartcards:i:1
redirectclipboard:i:1
redirectposdevices:i:0
drivestoredirect:s:*
autoreconnection enabled:i:1
authentication level:i:2
negotiate security layer:i:1
remoteapplicationmode:i:0
alternate shell:s:
```

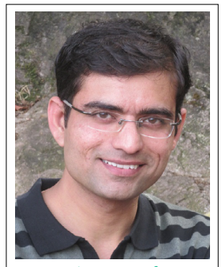
As you can see, redirection settings already exist in RDP file and you can change it from notepad and save it. In case of remote app, we are changing the same setting but through PowerShell. The highlighted block in the above screenshot shows drive redirection settings. Similarly there are many other settings that can be changed through redirection in Azure Remote apps. Learn more about redirection at http://bit.ly/1kH396F .

Conclusion

In this article, we saw how Microsoft Azure Remote Apps features can run desktop application. We also saw the benefits associated with running desktop apps in the Cloud. With the help of redirection, we saw how to make remote app based application more user interactive and device independent, thereby enabling true BYOD (Bring Your Own Device) environment for users ■

Download the entire source code from GitHub at bit.ly/dncm21-azuremoteapps

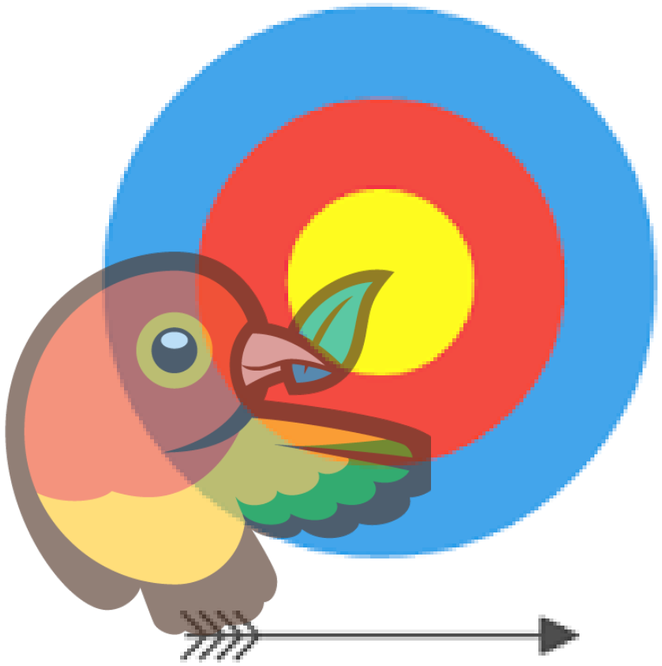
About the Author



kunal chandratre

Microsoft® Most Valuable Professional

Kunal Chandratre is a Microsoft Azure MVP and works as an Azure Architect in a leading software company in (Pune) India. He is also an Azure Consultant to various organizations across the globe for Azure support and provides quick start trainings on Azure to corporates and individuals on weekends. He regularly blogs about his Azure experience and is a very active member in various Microsoft Communities and also participates as a 'Speaker' in many events. You can follow him on Twitter at: @kunalchandratre or subscribe to his blog at http://sanganakauthority.blogspot.com



Using AngularJS with Bower and RequireJS in Visual Studio

Modern client-side application development demands modularity. Client-side modularity is required to achieve separation of logic from the UI. To implement this modularity, we usually use an MVC or MVVM framework to manage separate layers (in separate JavaScript files) for Models, Controllers, Modules, Services etc.

AngularJS is a great framework for implementing client-side modularity with MVC support.

However as the application grows, managing JavaScript dependencies on a page becomes a challenge, as you need to load your JavaScript by listing all <script> tags and their dependencies, in a particular order. Any JavaScript developer who has managed multiple <script> tags on a page knows how tedious and error prone the entire exercise is. In many cases, it also affects the performance of the app.

RequireJS is a JavaScript framework that enables asynchronous loading of files and modules and can come in very handy while managing dependencies across different AngularJS components and loading them asynchronously. An alternative to RequireJS is Almond.js, a stripped down version of RequireJS. RequireJS is more popular compared to it.

A simple overview of RequireJS

RequireJS is a JavaScript module and file loader. We start with each JavaScript file, listing the different files it depends on. RequireJS then recognizes these dependencies and loads them in the correct order. Since RequireJS manage dependencies on the client-side, we no longer need to manually refer to all script references on an html page. RequiresJS implements the AMD API (Asynchronous module definition API) and thus inherits all its advantages which includes asynchronous loading of dependencies and implicit resolution of dependencies, amongst other uses.

RequireJS can be used both in browsers as well as with server solutions like Node.js.

Since RequireJS loads files asynchronously, the browser starts its rendering process while waiting for RequireJS to do its work, without any blocking. This improves performance.

***Note:** If anybody is interested in looking at an implementation of KnockoutJS with RequireJS, please check here* http://www.dotnetcurry.com/aspnet-mvc/1180/html-data-grid-aspnet-mvc-using-knockoutjs-jquery-requirejs

In this article, we will be using **Bower** to install dependencies in this project; in our case - AngularJS and RequireJS. Bower is a package manager for the web. It manages frameworks, libraries etc. needed by the web application. Although I have implemented this application using Visual Studio 2015, you can also implement it using Visual Studio 2013. To learn more about Bower, check this article http://www.dotnetcurry.com/visualstudio/1096/using-grunt-gulp-bower-visual-studio-2013-2015 .

Pre-requisites for the implementation

Here are some pre-requisites before you get started:

- Free Community Edition of Visual Studio 2013 http://bit.ly/dncvscommunity OR Visual Studio 2015 http://bit.ly/dncvs15community

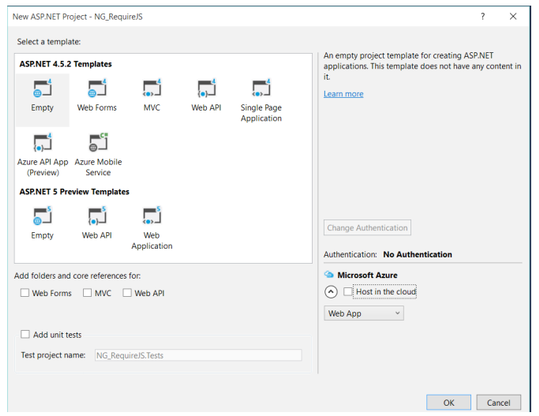
- Node for Visual Studio (NTVS) https://github.com/Microsoft/nodejstools#readme

- Git for Windows. This will be used for downloading dependencies using Bower https://git-for-windows.github.io/. This downloads an exe.

Note: Wherever you see Visual Studio in this article, it either means Visual Studio 2013 or Visual Studio 2015. You can choose either, as per availability.

Using Bower to Install Dependencies for the project

Step 1: Open Visual Studio and create an empty ASP.NET project with the name **NG_RequireJS** as shown in the following image:



Step 2: To install necessary dependencies for the project, open Node.js command prompt (Run as Administrator) and follow these steps.

- From the command prompt navigate to the project folder.

- Install Bower for the project

```
E:\Mahesh_New\Articles\Oct15\NG_RequireJS\NG_RequireJS>npm install bower
bower@1.6.3 node_modules\bower
```

- Initialize Bower. This will add **bower.json** file to the project. By default, the file is added in the project path which will not be displayed in the Solution explorer. So to view the file, click on **Show all files** on the toolbar of our Solution Explorer and you will find the bower.json file. Right-click on the file and select **Include in project**. This needs to be done for all dependencies which will be added in the next steps.

```
E:\Mahesh_New\Articles\Oct15\NG_RequireJS\NG_RequireJS>bower init
? May bower anonymously report usage statistics to improve the tool over time? y
y/n)

? May bower anonymously report usage statistics to improve the tool over time? Y
es
? name ng-requirejs
? description The use of RequireJS with AngularJS
? main file
? what types of modules does this package expose? amd, globals, node
? keywords
? authors MS
? license MIT
? homepage
? set currently installed components as dependencies? Yes
? add commonly ignored files to ignore list? Yes
? would you like to mark this package as private which prevents it from being ac
? would you like to mark this package as private which prevents it from being ac
cidentally published to the registry? Yes
```

The above image shows **bower init** command which will ask for options to be added in the bower.json file. Please select options as per your application's requirements. For this article, we have selected module options for **amd, globals and node**. Once these options are selected and added, the bower.json file will take the following shape:

```
{
  name: 'ng-requirejs',
  description: 'The use of RequireJS with AngularJS',
  main: '',
  moduleType: [
    'amd',
    'globals',
    'node'
  ],
  authors: [
    'MS'
  ],
  license: 'MIT',
  homepage: '',
  private: true,
  ignore: [
    '**/.*',
    'node_modules',
    'bower_components',
    'test',
    'tests'
  ]
}
```

The project will be added with bower.json.

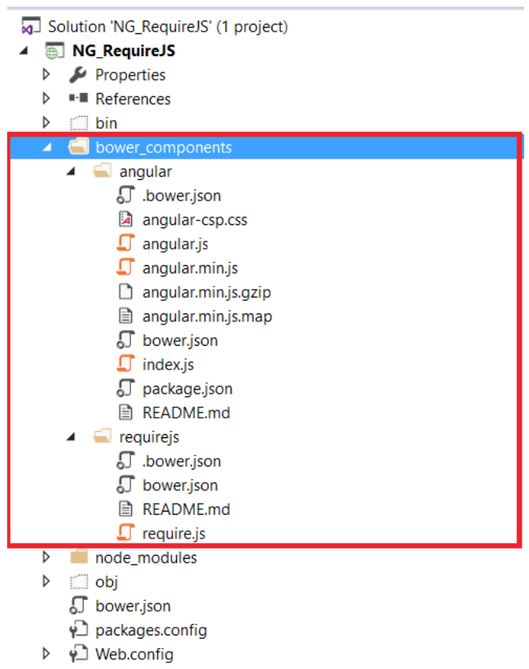
- Install AngularJS with the following command:
`Bower install angularjs –save`

```
E:\Mahesh_New\Articles\Oct15\NG_RequireJS\NG_RequireJS>bower install angularjs -
-save
bower not-cached    git://github.com/angular/bower-angular.git#*
bower resolve       git://github.com/angular/bower-angular.git#*
bower download      https://github.com/angular/bower-angular/archive/v1.4.7.tar.
gz
bower extract       angularjs#* archive.tar.gz
bower resolved      git://github.com/angular/bower-angular.git#1.4.7
bower install       angular#1.4.7

angular#1.4.7 bower_components\angular

E:\Mahesh_New\Articles\Oct15\NG_RequireJS\NG_RequireJS>
```

- Now install RequireJS with the following command: `Bower install requirejs –save`

```
E:\Mahesh_New\Articles\Oct15\NG_RequireJS\NG_RequireJS>bower install requirejs -
-save
bower not-cached    git://github.com/jrburke/requirejs-bower.git#*
bower resolve       git://github.com/jrburke/requirejs-bower.git#*
bower download      https://github.com/jrburke/requirejs-bower/archive/2.1.20.ta
r.gz
bower extract       requirejs#* archive.tar.gz
bower resolved      git://github.com/jrburke/requirejs-bower.git#2.1.20
bower install       requirejs#2.1.20

requirejs#2.1.20 bower_components\requirejs

E:\Mahesh_New\Articles\Oct15\NG_RequireJS\NG_RequireJS>
```

Step 3: Go back to the project in Visual Studio, select **Show All Files** from the Solution Explorer tool bar and you should now see **bower_components** include in the project. The project will be displayed as shown in the following image.



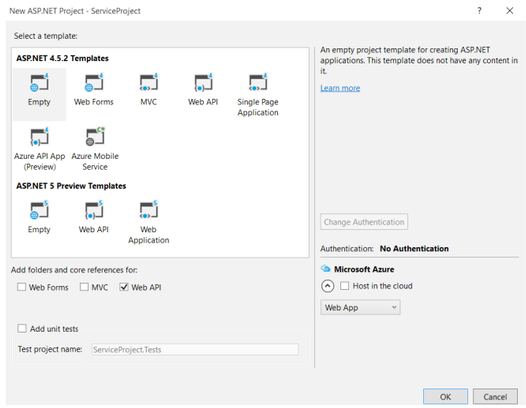
bower.js will show dependencies as shown below:

```
"dependencies": {
  "angular": "angularjs#~1.4.7",
  "requirejs": "~2.1.20"
}
```

So far so good, we have added the necessary infrastructure for the project. Now we will add the necessary AngularJS components to the project.

Adding AngularJS components

Step 4: In the solution, add a new empty Web API project of the name **ServiceProject**. We will be using this API as a service provider project for making http calls using Angular's $http service.



Step 5: In this project, in the Models folder, add a new class file with the following code:

```
using System.Collections.Generic;

namespace ServiceProject.Models
{
  public class Product
  {
    public int ProdId { get; set; }
    public string ProdName { get; set; }
  }

  public class ProductDatabase :
  List<Product>
  {
    public ProductDatabase()
    {
      Add(new Product()
      {ProdId=1,ProdName="Laptop" });
      Add(new Product() { ProdId = 2,
      ProdName = "Desktop" });
    }
  }
}
```

Step 6: In the Controllers folder, add a new empty Web API Controller of the name ProductInfoAPIController.

In this controller, add a Get() method with the following code:

```
using ServiceProject.Models;
using System.Collections.Generic;
using System.Web.Http;

namespace ServiceProject.Controllers
{
  public class ProductInfoAPIController
  : ApiController
  {
    public IEnumerable<Product> Get()
    {
      return new ProductDatabase();
    }
  }
}
```

Using RequireJS

Step 7: In the NG_RequireJS project, add a new folder of the name app. In this folder, add the following JavaScript files:

Main.js

```
require.config({
  paths: {
    angular: '../bower_components/
    angular/angular'
  },
  shim: {
    'angular': {
      exports:'angular'
    }
  }
});

require(['app'], function () {
  require(['serv', 'ctrl'], function ()
  {
    angular.bootstrap(document,
    ['app']);
  });
});
```

The above script bootstraps RequireJS by calling `require.config()` function, passing in a configuration object which contains path for the AngularJS framework. Since AngularJS is an external dependency here that is being managed using Bower, we are specifying a *relative* path to our bower_components directory. To make sure that AngularJS is embedded correctly, we are using

RequireJS to assign AngularJS to the global variable **angular**. This is done using the attribute **shim**. We are using **exports** which defines global access to the **angular** object, so that all JavaScript modules/files can make use of this object.

The above file also defines `require()` used for loading the necessary modules asynchronously along with the loading of the **document** (DOM). `require()` loads the modules - app, serv and ctrl asynchronously.

The following line:

```
angular.bootstrap(document, ['app']);
```

..performs the same operation as the **ng-app** directive does. The reason we are using `angular.bootstrap()` instead of `ng-app` is because the files are loaded *asynchronously*. If we use `ng-app`, there could be a possibility that when AngularJS loads a module in the ng-app directive, RequireJS may have not yet loaded the module. Hence we use angular.bootstrap() to initialize our app *only* when RequireJS has finished loading its modules. The first parameter to angular.bootstrap() is the DOM element and the second parameter is the application module that is to be loaded. Here it simply means that the **app** module will be loaded at **document** level.

App.js

```
define(['angular'],function (angular) {
  var app = angular.module('app', []);
  return app;
});
```

The above file defines angular module of name 'app'. The *angular* object declared through *shim* is passed to it.

serv.js

```
define(['app'], function (app) {
 app.service('serv', ['$http',function
 ($http) {
  this.getdata = function () {
   var resp = $http.get('http://
   localhost:4737/api/ProductInfoAPI');
   return resp;
  }
 }]);
})
```

This code defines angular service of name `serv`, which makes call to the Web API service and receives a response.

ctrl.js

```
define(['app'], function (app) {
  app.controller('ctrl',
  ['$scope','serv', function ($scope,
  serv) {
    loaddata();
    function loaddata() {
    var promise = serv.getdata();
    promise.then(function (resp) {
      $scope.Products = resp.data;
      $scope.Message = "Operation
      Completed Successfully...";
    }, function (err) {
      $scope.Message = "Error " + err.
      status;
    })
   };
 }]);
});
```

The above file contains code for declaring angular controller. This file has dependency on the angular service. The above controller makes a call to `getData()` function of the angular service. Once the data is received from the service, it is stored in the Products `$scope` object.

Step 8: On the project root, add an index.html with the following markup and code:

```
<!DOCTYPE html>
<html>
<head>
 <title>Using RequireJS with
 Angular.js</title>
  <meta charset="utf-8" />
 <style type="text/css">
  table, td,th {
   border:double;
  }
 </style>
</head>
<body ng-controller="ctrl">
 <h1>Using RequireJS with Angular.js
 </h1>

 <table>
  <thead>
   <tr>
    <th>Product Id</th>
    <th>Product Name</th>
   </tr>
  </thead>
 <tbody>
   <tr ng-repeat="p in Products">
    <td>{{p.ProdId}}</td>
    <td>{{p.ProdName}}</td>
   </tr>
  </tbody>
 </table>
 <div>{{Message}}</div>
<script src="bower_components/requirejs/
require.js" data-main="app/Main.js">
</script>
</body>
</html>
```

Here the following line

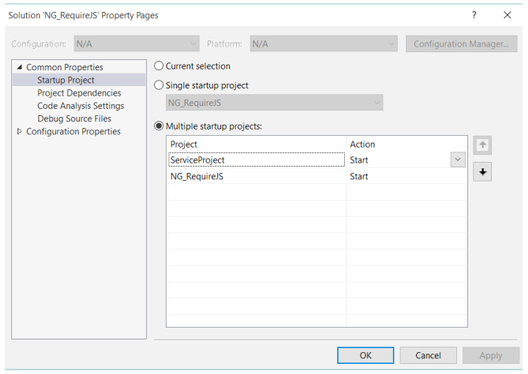
```
<script src="bower_components/requirejs/
require.js" data-main="app/Main.js">
</script>
```

..is used to initialize and load RequireJS. The **data-main** attribute is used to load Main.js, the starting point of our application. This attribute tells require.js to load app/Main.js *after* require.js has loaded. Main.js configures Angular and other module dependencies.

This is how we eliminate the need to refer to all JavaScript files in an HTML page using RequireJS.

Running the Application

Run the application by setting multiple startup projects in Visual Studio, as shown in the following image:



Run the application and the Index.html loads in the browser with the following result:

Using RequireJS with Angular.js

Product Id	Product Name
1	Laptop
2	Desktop

Operation Completed Successfully...

Right click on the page > View source and you will see all modules listed in order.

Conclusion

RequireJS is an awesome library for managing asynchronous module loading when the logic is segregated in separate JavaScript files. It loads modules and the relevant dependencies in their right order. Make sure you learn this library and use it in a project containing multiple JavaScript files! ■

Download the entire source code from GitHub at bit.ly/dncm21-require-angularjs

About the Author

Mahesh Sabnis is a Microsoft MVP in .NET. He is also a Microsoft Certified Trainer (MCT) since 2005 and has conducted various Corporate Training programs for .NET Technologies (all versions). Follow him on twitter @maheshdotnet. Mahesh blogs regularly on .NET Server-side & other client-side Technologies at bit.ly/HsS2on

Microsoft® Most Valuable Professional

Create Your First

Diagnostic Analyzer in Visual Studio 2015

This article talks about creating custom Diagnostic Analyzers in Visual Studio 2015. If you are new to the concept of Diagnostic Analyzers, read the "Introduction to Diagnostic Analyzers in Visual Studio 2015" article over here http://www.dotnetcurry.com/visualstudio/1197/diagnostic-analyzers-visual-studio-2015

Diagnostic analyzers are a great new extensibility feature in Visual Studio 2015 for performing static code analysis. Most developers will probably settle with using the ones provided by Microsoft and third party vendors. Nevertheless, there are scenarios that warrant development of custom diagnostic analyzers, such as enforcing coding guidelines in a company. Keeping this customization in mind, Microsoft has put a lot of effort in making the initial experience as pleasant as possible. This article will walk you through the process of creating a simple diagnostic analyzer on your own.

Software Prerequisites

To use diagnostic analyzers, Visual Studio 2015 is required: Community, Professional or Enterprise edition to be exact. Although this might suffice for development, the experience can get much better by installing the .NET Compiler Platform SDK. It includes two components that are important for our task:

- Visual Studio project template for a diagnostic analyzer, and
- *Roslyn Syntax Visualizer* for interactive syntax tree exploration.

To follow the examples in this article, you will need to have this extension installed in your copy of Visual Studio 2015.

The template we are going to use will also create a Visual Studio extension. Therefore your Visual Studio installation will need to include the *Visual Studio Extensibility Tools*.

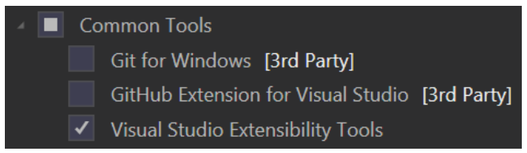


Figure 1: Visual Studio Extensibility Tools feature in Visual Studio setup

You do not have to worry if you have already installed Visual Studio 2015 without this feature. On creating a project, Visual Studio detects and will offer to install the missing feature for you.

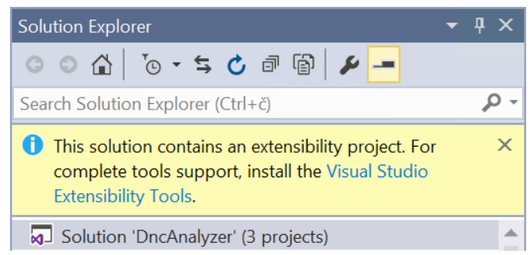


Figure 2: Install missing feature directly from Visual Studio

Trying Out the Default Diagnostic

We will start out by creating a new Visual Studio project from the template and giving it a test run.

Run Visual Studio 2015 and create a new project based on the *Diagnostic with Code Fix (NuGet + VSIX)* template (you can find it in *Templates* > *Visual C#* > *Extensibility* node). It will create a solution with three projects for you (*DncAnalyzer* part of each name will be replaced with whatever name you choose for your project):

- Portable class library project *DncAnalyzer* with the actual analyzer code.
- Unit test project *DncAnalyzer.Test* with unit tests for the analyzer.
- VSIX project *DncAnalyzer.VSIX* for packaging the analyzer as a Visual Studio extension.

Visual Studio will conveniently preselect the last one as a startup project. When you run the solution from within Visual Studio, it will perform the following steps for you:

- Compile the analyzer.
- Create a Visual Studio extension from it.
- Run a new instance of Visual Studio.
- Install the created extension in it.

To try out the analyzer, you can create a new class

library project inside this Visual Studio instance. It should immediately report a warning and offer you a fix for it.

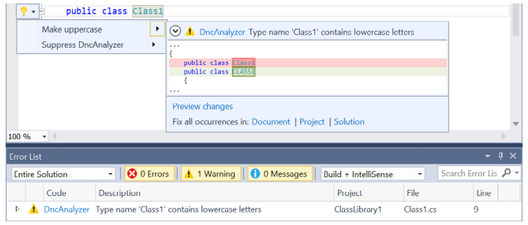


Figure 3: Default diagnostic analyzer in action

Even debugging works flawlessly in this setup. Switch back to your first Visual Studio instance and open *DiagnosticAnalyzer.cs* from *DncAnalyzer* project. Set a break point in the first line of *AnalyzeSymbol* method. Now switch back to the second instance and change the class name. Code execution should stop at the line with the breakpoint. This makes it easy to troubleshoot a diagnostic during development.

The only downside is a relatively long Visual Studio start-up time, which makes the typical cycle of finding a problem, fixing it and restarting the project, slower than one would wish. This is where the unit test project can prove helpful. To try it, resume the execution from the breakpoint and close the second Visual Studio instance. Run the test in the first Visual Studio instance by navigating to *Test > Run > All* Tests in the main menu. *Test Explorer* will show up with test results.

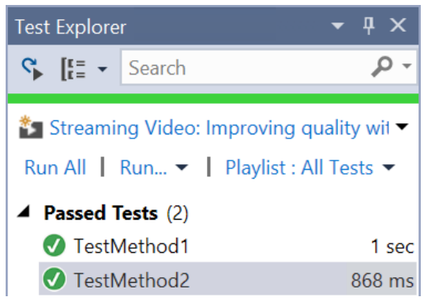


Figure 4: Unit test results in Test Explorer

If you right click *TestMethod2* and select *Debug Selected Tests* from the context menu, the execution should again stop at the breakpoint, assuming you have not removed it yet. You can save a lot of time by testing your code with unit tests for the majority of development time and only run it in Visual Studio towards the end to make sure it behaves as expected.

Validation of Regular Expressions

Although the template has already provided us with a working diagnostic analyzer, this is a rather simple example. To make it more interesting, let us set a more ambitious and useful final goal: validating regular expression patterns appearing in source code as string literals. In order to have it working by the end of this article, we will actually have to limit the scope a little bit. Instead of covering all possible cases, we will settle with inspecting only the following one:

```
using System.Text.RegularExpressions;

namespace RegexSample
{
  public class Class1
  {
    public void Foo()
    {
      Regex.Match("", "[");
    }
  }
}
```

To be exact, we will inspect all calls to *Regex.Match* method, having the second argument (i.e. the regular expression pattern) as a string literal.

Instead of jumping directly into writing code, we will first take advantage of a very useful tool, bundled with .NET Compiler Platform SDK, the *Roslyn Syntax Visualizer*. To see it in action, create a new class library project, and add the above code into it. Now, navigate to *View > Other Windows > Syntax Visualizer* to open it. If you try moving the cursor to different spots in the code editor window, you will notice that the syntax visualizer tree view is keeping in sync, always focusing on the currently selected token. It works the other way around, as well: whenever you select a node in the tree view, the corresponding part of source code will be selected in the code editor.

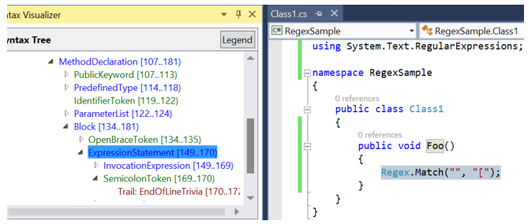


Figure 5: Regex.Match method call in syntax visualizer

Select the Regex.Match method call as seen in Figure 5 and fully expand it in the tree view to get a better sense of the code that we will be inspecting. You will notice that the nodes in the tree view are of three different colors:

- Blue color represents intermediate nodes in the syntax tree, i.e. higher level language constructs
- Green color represents tokens, e.g. keywords, names, and punctuation
- Red color represents trivia, i.e. formatting characters that do not affect semantics.

To get an even better view of the sub tree, right click on the *InvocationExpression* node in the tree view and click on *View Directed Syntax Graph*. This will render it nicely in a separate window.

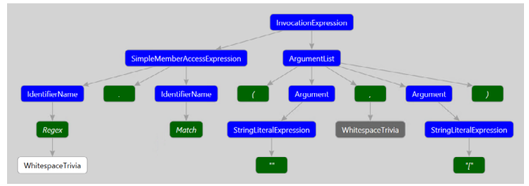


Figure 6: InvocationExpression syntax tree

Looking at this image, we can formulate the basic idea of how to implement the analyzer. The first step is to recognize the language construct of interest, i.e. the one corresponding to this syntax tree:

- We will need to inspect all *InvocationExpression* nodes, because they are representing the method calls.
- Out of these, the ones that interest us are only those which are invoking *Regex.Match* method with two arguments.
- Even that is not specific enough: we will only analyze calls with a string literal as its second argument.

The second step will be much easier. We will retrieve the value of the literal and verify that it is a valid regular expression pattern. This can easily be done using .NET's *Regex* class.

Implementing the Idea

With the plan in place, we can get to work. Switch back to the Visual Studio instance with the default diagnostic analyzer project open (as created at the beginning of this article). We are going to use it as our starting point.

Before we start implementing the analysis algorithm, we should modify the identity of the analyzer to match our intentions. The relevant code can be found at the beginning of the *DncAnalyzerAnalyzer* class:

```
public const string DiagnosticId = "DncAnalyzer";

// You can change these strings in
the Resources.resx file. If you do not
want your analyzer to be localize-able,
you can use regular strings for Title
and MessageFormat.

private static readonly
LocalizableString Title = new
LocalizableResourceString
(nameof(Resources.AnalyzerTitle),
Resources.ResourceManager,
typeof(Resources));

private static readonly
LocalizableString MessageFormat = new
LocalizableResourceString
(nameof(Resources.
AnalyzerMessageFormat), Resources.
ResourceManager,
typeof(Resources));
```

```
private static readonly
LocalizableString
Description = new
LocalizableResourceString
(nameof(Resources.AnalyzerDescription),
Resources.ResourceManager,
typeof(Resources));

private const string Category =
"Naming";

private static DiagnosticDescriptor Rule
= new DiagnosticDescriptor(DiagnosticId,
Title, MessageFormat, Category,
DiagnosticSeverity.Warning,
isEnabledByDefault: true, description:
Description);
```

We will do the following changes:

- Set *DiagnosticId* to *"DNC01"*. This is the unique identifier of our analyzer.

- Set Category to *"Usage"*. There is no predefined list of supported categories. It is best you check existing analyzers and choose one from there.

- Set the default severity to error by changing the fifth argument of the *DiagnosticDescriptor* constructor call to *DiagnosticSeverity.Error*. As you can see, all values describing the analyzer are passed to this constructor.

We still have not changed all the texts, describing our analyzer. Since they are localizable, we need to open the *Resources.resx* file and change them there:

- *AnalyzerTitle* is a short name for the error. We will set it to: *"Regular expression is invalid"*

- *AnalyzerDescription* is a longer description of what is being analyzed. We will set it to: *"Pattern must be a valid regular expression."*

- *AnalyzerMessageFormat* is the template for the actual error message. It will be passed to S*tring.Format* along with your additional data, when you report the error. We will set it to: *"Regular expression is invalid: {0}"*

Once we are through with the formalities, we can continue by specifying when Visual Studio should execute our custom analyzer code. This is how we will change the *Initialize* method:

```
public override void
Initialize(AnalysisContext context)
{
  context.
RegisterSyntaxNodeAction(AnalyzeSyntax,
SyntaxKind.InvocationExpression);
}
```

The previously used *RegisterSymbolAction* would have allowed us to do analysis at the level of a symbol, which was fine for the default analyzer, checking the casing of a class name. We are interested in a full node in the syntax tree; hence, we will call *RegisterSyntaxNodeAction* and specify the type of node that interests us by passing it *SyntaxKind.InvocationExpression*.

The first argument is the name of the callback method that will be called. This is where we will put all of our logic. We can delete the existing *AnalyzeSymbol* method and create a new *AnalyzeSyntax* method with the following signature:

```
private void
AnalyzeSyntax(SyntaxNodeAnalysisContext
context)
{
}
```

Since in our call to RegisterSyntaxNodeAction, we have specified that we are only interested in InvocationExpression, we can safely assume that our method will only be called for this type of nodes:

```
var invocationExpression =
(InvocationExpressionSyntax)context.
Node;
```

We can expect Visual Studio to call our method a lot; therefore, it is important to make it as efficient as possible and to avoid any unnecessary processing. Very likely, only a small part of method calls will actually be calls to Regex.Match. To stop processing those as soon as possible, we will check which method is being called in two phases:

```
var memberExpresion =
invocationExpression.Expression as
MemberAccessExpressionSyntax;
if (memberExpresion?.Name?.ToString() !=
"Match") return;

var memberSymbol =
context.SemanticModel.
GetSymbolInfo(memberExpresion).Symbol;
if (memberSymbol?.ToString() !=
"System.Text.RegularExpressions.Regex.
Match(string, string)") return;
```

First, we only check the name of the method. In the unlikely case that it is actually named Match, we do a lookup into the symbol table and check the full method signature. Of course, symbol table lookup is much slower than a simple string comparison. If any of the two checks fails, we quietly exit the method and stop further analysis.

Being sure that the right method is called, we continue by checking whether the second argument is really a literal:

```
var argumentList = invocationExpression.
ArgumentList as ArgumentListSyntax;
if ((argumentList?.Arguments.Count ?? 0)
!= 2) return;

var regexLiteral = argumentList.
Arguments[1].Expression as
LiteralExpressionSyntax;
if (regexLiteral == null) return;
```

Having already checked the full method signature makes counting the number of arguments redundant in this case. Keep in mind though, that we really do not want the analyzers to throw any exceptions. Therefore, if there was even the slightest doubt that retrieving the second argument from the array could throw one, it is better to do a second check, especially if it is as efficient as this one.

At this point in code, we can be sure that the method call exactly matches what we are interested in. We still have to retrieve the value of the literal:

```
var regexOpt = context.SemanticModel.
GetConstantValue(regexLiteral);
var regex = regexOpt.Value as string;
```

We will leave the validation of the regular expression pattern to the Regex class. This is how it will be called in the compiled code as well:

```
try
{
  System.Text.RegularExpressions.Regex.
  Match("", regex);
}
catch (ArgumentException e)
{
  var diag = Diagnostic.Create(Rule,
  regexLiteral.GetLocation(),
  e.Message);
  context.ReportDiagnostic(diag);
}
```

Notice how the error is reported when the regular expression is invalid. We create a report by passing all the relevant data to its factory method: *Rule* with full analyzer description, location in the code corresponding to the erroneous node, and the arguments for our error message format.

Our diagnostic analyzer is complete. Before trying it out, delete the *CodeFixProvider.cs* file from the project to prevent the default code fix from popping up in Visual Studio. You can now run the solution. Once a new instance of Visual Studio starts, open the regular expression sample which we have been analyzing using the Roslyn Syntax Visualizer. The invalid regular expression should be marked as error.

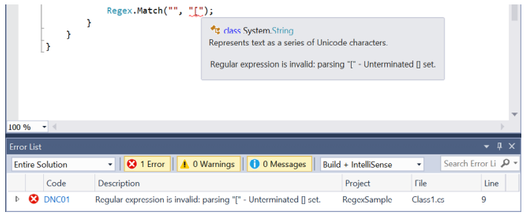


Figure 7: Invalid regular expression detected

Conclusion

As you can see, thanks to Roslyn, static code analysis became more accessible than it was ever before. Admittedly, our example is still simple, and extending it to cover all possible cases would be a lot of work. Nevertheless, such a low barrier to entry should encourage many developers to try diagnostic analyzer development themselves.

I am sure some of them will end up developing diagnostic analyzers, which we will be using every day. Dear reader, maybe you will be one of them! ■

Download the entire source code from GitHub at
bit.ly/dncm21-vs-diag-analyzer

About the Author

damir arh

Damir Arh has many years of experience with Microsoft development tools; both in complex enterprise software projects and modern cross-platform mobile applications. In his drive towards better development processes, he is a proponent of test driven development, continuous integration and continuous deployment. He shares his knowledge by speaking at local user groups and conferences, blogging, and answering questions on Stack Overflow. He is an awarded Microsoft MVP for .NET since 2012.

Microsoft® Most Valuable Professional

Universal Windows 10 Platform (UWP) Music App for

Windows Phone & Raspberry Pi2

Windows 10 introduces an entirely new way of developing apps targeting multiple devices and it is called UWP apps or Universal Windows Platform apps. Instead of creating separate apps for different platforms, UWP apps run across all major platforms with minor code changes.

This article covers part 2 of our journey to develop a Windows 10 Universal App (UWP) that will run on most of the devices without any code change. In Part 1 http://bit.ly/dnc-win10-soundcloud-1 , I wrote how the new Windows 10 controls allow us to target a wide variety of Windows devices with minimal code. With Adaptive Triggers and Custom App shell, we developed a simple Sound Cloud player that adapts the UI based on the size of the screen.

In this article, we will improve our app by adding more features and changing existing code to support Background Audio and Media Transport Controls. We will also deploy our app to both a Windows Phone and Raspberry Pi2 without any code changes.

Disclaimer: This article contains a lot of code which contains comments wherever needed explaining what the code does.

Project Setup

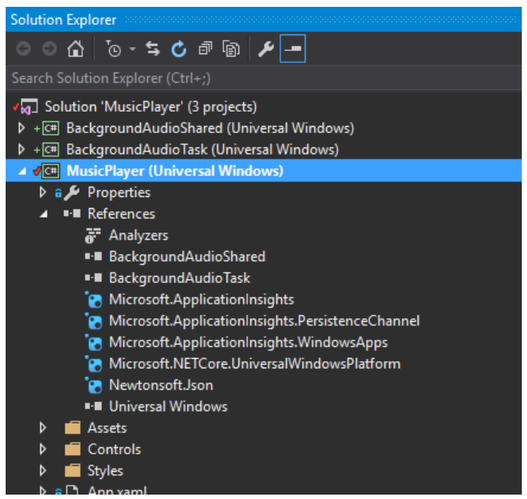
The previous sample played the first track from the "likes" playlist from SoundCloud but it did not support Background Audio and had the following limitations:

1. Minimizing App stopped the play back
2. Default Media Controls (Keyboard, Screen etc) were hidden
3. Media Controls did not show Song, Album art etc

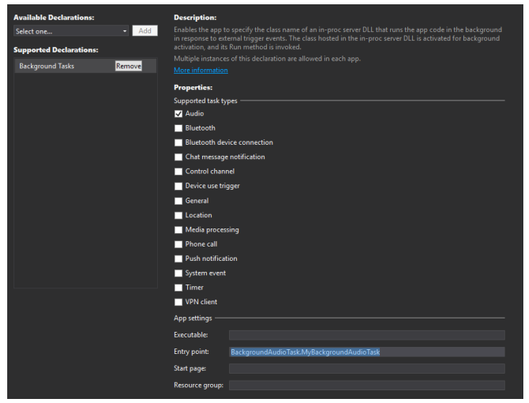
We will add Background Audio support to our App and fix the above limitations. For Windows 10 development, it's much easier to enable Background Audio and Manage playlist. To read more about the details of the implementation, read Background Audio (http://bit.ly/1H9mf9Y) MSDN documentation. To make it simpler, we will use the sample Background Audio App provided by Microsoft which can be downloaded from GitHub (http://bit.ly/1NzKtPv) .

Let us perform the following steps. I am assuming you have downloaded the source code of the previous article & Microsoft's Github project.

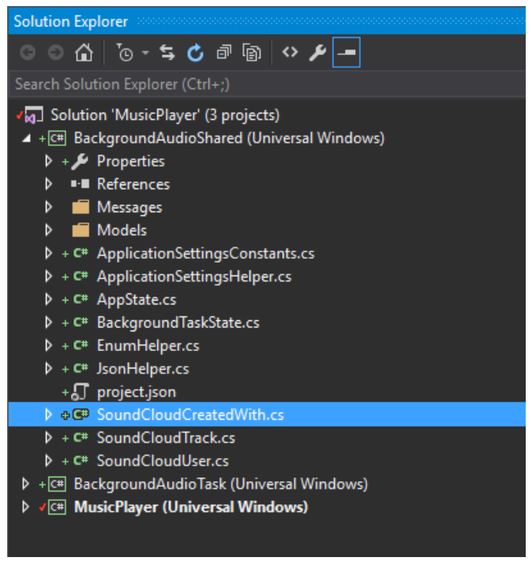
- Add the **BackgroundAudioShared** and **BackgroundAudioTask** projects from the sample project to our solution, and add references to these projects in **MusicPlayer** App. Your Solution Explorer should look like the following:



- Open the Package Manifest file for MusicPlayer project and Navigate to declarations tab. Add the Background Tasks declaration and set *BackgroundAudioTask.MyBackgroundAudioTask* as the Entry Point value. Make sure only "Audio" is checked under *Supported Task Types*.



- Move *SoundCloudUser.cs* , *SoundCloudTrack.cs* and *SoundCloudCreatedWith.cs* from MusicPlayer project to BackgroundAudioShared project and update the namespace names. Your solution explorer should now look like the following:



MusicPlayer

- Make the following changes to *App.xaml.cs* to declare a new SoundCloudUser object which we

will use later to display profile details.

```
public static string SoundCloudClientId
= "<YOUR CLIENT ID>";
public static int SCUserID = 0;
public static string SoundCloudLink =
"http://api.soundcloud.com/";
public static string SoundCloudAPIUsers
= "users/";
public static List<SoundCloudTrack>
likes = new List<SoundCloudTrack>();
public static int nowplayingTrackId = 0;
public static SoundCloudUser SCUser {
get; set; }
```

- Make the following code changes to *AppShell.xaml.cs* to change menu items

```
// Declare the top level nav items
private List<NavMenuItem> navlist = new
List<NavMenuItem>(new[]
{
  new NavMenuItem()
  {
    Symbol = Symbol.Play,
    Label ="Now Playing",
    DestPage = typeof(NowPlaying)
  },
  new NavMenuItem()
  {
    Symbol = Symbol.Emoji2,
    Label = "Likes",
    DestPage = typeof(Likes)
  },
  new NavMenuItem()
  {
    Symbol = Symbol.Contact,
    Label = "Me",
    DestPage = typeof(Me)
  }
});
```

- Add a new *Blank Page* to the project and name it *Likes.xaml*

- Add the following code to *Likes.xaml*

```
<Page
  x:Class="MusicPlayer.Likes"
  xmlns="http://schemas.microsoft.com/
  winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.
  com/winfx/2006/xaml"
  xmlns:local="using:MusicPlayer"
  xmlns:d="http://schemas.microsoft.com/
  expression/blend/2008"
  xmlns:mc="http://schemas
  .openxmlformats.org/
  markup-compatibility/2006"
  mc:Ignorable="d"
  xmlns:data="using:
  BackgroundAudioShared">
  <Page.Resources>
    <DataTemplate x:Key=
    "ImageOverlayTemplate"

x:DataType="data:SoundCloudTrack">
      <StackPanel Height="130"
      Width="190" Margin="4,4,4,8">
        <TextBlock Text="{x:Bind title}"
        Margin="8,4" Width="186"
        Style="{StaticResource
        BaseTextBlockStyle}"
        HorizontalAlignment="Left"/>
        <Image Source="{x:Bind
        AlbumArtUri}"
        Margin="8,0,8,8"
        Stretch="UniformToFill"/>
      </StackPanel>
    </DataTemplate>
  </Page.Resources>
  <Grid Background="{ThemeResource
  ApplicationPageBackgroundThemeBrush}">
    <GridView x:Name="grdLikes"
    ItemTemplate="{StaticResource
    ImageOverlayTemplate }"
    IsItemClickEnabled="True"
    IsSwipeEnabled="False"
    CanDragItems="False"
    SelectionMode="None"
    ItemClick="grdLikes_ItemClick" />
  </Grid>
</Page>
```

In the above code, we have added a GridView control and a Data template which will be used to display a list of all the Tracks added to our "Likes" playlist.

- Add the following code to likes.xaml.cs.

I have deliberately removed the namespaces to save space but you can always look up the source code of this article for the same.

```
namespace MusicPlayer
{
  /// <summary>
  /// An empty page that can be used on
  its own or navigated to within a Frame.
  /// </summary>
  public sealed partial class Likes :
  Page
  {
public Likes()
{
  this.InitializeComponent();
  this.Loaded += Likes_Loaded;
}

private void Likes_Loaded(object
sender, RoutedEventArgs e)
{
  grdLikes.ItemsSource = App.likes;
}

private void grdLikes_ItemClick
(object sender, ItemClickEventArgs e)
{
  var song =  e.ClickedItem as
  SoundCloudTrack;
  MessageService.
  SendMessageToBackground
  (new TrackChangedMessage(new
  Uri(song.stream_url)));

}

private void grdLikes_
SelectionChanged(object sender,
SelectionChangedEventArgs e)
{

}
  }
}
```

In this code, we have done the following:

1. Set the ItemsSource property to global App.likes property. This will populate the Gridview with the list of tracks.

2. When an Item (Track) is clicked in the GridView, we also load the Track details and Send a Message to our Background Task which will control the media playback.

- Make the following code changes to *MainPage.xaml.cs* to populate the SoundCloudUser object which is declared in App.xaml.cs

```
// The Blank Page item template is
documented at http://go.microsoft.com/
fwlink/?LinkId=402352&clcid=0x409

namespace MusicPlayer
{
 /// <summary>
/// An empty page that can be used on
its own or navigated to within a Frame.
/// </summary>
public sealed partial class MainPage :
Page
{
 public MainPage()
 {
  this.InitializeComponent();
  this.Loaded += MainPage_Loaded;
 }

 private void MainPage_Loaded(object
 sender, RoutedEventArgs e)
 {
  //Get User
  GetUserDetails();
 }

 private async void GetUserDetails()
 {
  try
  {
   string responseText = await
   GetjsonStream(App.SoundCloudLink
   + App.SoundCloudAPIUsers
   + "shoban-kumar" + ".json?client_id="
   + App.SoundCloudClientId);
   App.SCUser = JsonConvert.
   DeserializeObject<SoundCloudUser>
   (responseText);

   App.SCUserID = App.SCUser.id;

   //Get Likes
   GetLikes();
  }
  catch (Exception ex)
  {
   ...
  }
 }

 private async void GetLikes()
 {
  try
  {
   string responseText = await
   GetjsonStream(App.SoundCloudLink
   + App.SoundCloudAPIUsers
   + App.SCUserID +
   "/favorites.json?client_id="
   + App.SoundCloudClientId);
   App.likes = JsonConvert.
   DeserializeObject<List
   <SoundCloudTrack>>(responseText);

   //remove songs which do not have
   stream url
```

```
  App.likes = App.likes.
  Where(t => t.stream_url != null).
  ToList<SoundCloudTrack>();

  //add "?client_id=" + App.
  SoundCloudClientId to stream url
  App.likes = App.likes.Select(t
  => { t.stream_url += "?client_id="
  + App.SoundCloudClientId; return t;
  }).ToList<SoundCloudTrack>();
  loginProgress.IsActive = false;

  AppShell shell = Window.Current.
  Content as AppShell;
  shell.AppFrame.Navigate
  (typeof(NowPlaying));
 }
 catch (Exception ex)
 {
  ...
 }
}

public async Task<string>
GetjsonStream(string url) //Function
to read from given url
{
 HttpClient client = new
 HttpClient();
    HttpResponseMessage response = await
    client.GetAsync(url);
    HttpResponseMessage v = new
    HttpResponseMessage();
    return await response.Content.
    ReadAsStringAsync();
   }
  }
}
```

- Add a new Blank Page named *Me.xaml* and add the following code to the xaml file:

```
<Page
 x:Class="MusicPlayer.Me"
 xmlns="http://schemas.microsoft.com/
 winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/
 winfx/2006/xaml"
 xmlns:local="using:MusicPlayer"
 xmlns:d="http://schemas.microsoft.com/
 expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats
 .org/markup-compatibility/2006"
 mc:Ignorable="d">

 <Grid Background="{ThemeResource
 ApplicationPageBackgroundThemeBrush}">
  <StackPanel HorizontalAlignment=
  "Center">
   <Ellipse Height="250" Grid.Row="0"
   Width="250" x:Name="Albumart" >
    <Ellipse.Fill>
     <ImageBrush x:Name="profilePhoto"
     ImageSource="Assets\Albumart.png"
     Stretch="UniformToFill" />
   </Ellipse.Fill>
  </Ellipse>

  ...
  <TextBlock x:Name="txtFollowing"
       Text="" Style="{StaticResource
            HeaderTextBlockStyle}" />

  <TextBlock Text="Followers"
  Style="{StaticResource
            TitleTextBlockStyle}"
Foreground="Gray"        />

  <TextBlock x:Name="txtFollowers"
       Text="" Style="{StaticResource
            HeaderTextBlockStyle}" />

  </StackPanel>
 </Grid>
</Page>
```

In the above code, we added few controls to display the Profile Photo, Name, Website and some other details from SoundCloud.

Add the following code to *Me.xaml.cs* to populate profile details from the new SoundCloudUser object.

```
...

  private void Me_Loaded(object sender,
       RoutedEventArgs e)
  {
   if (App.SCUser != null)
   {
    txtFirstname.Text = Convert.
          ToString(App.SCUser.first_name);
    txtlastname.Text = Convert.
          ToString(App.SCUser.last_name);
    txtWebsite.Text = Convert.
          ToString(App.SCUser.website);
    txtCity.Text = Convert.ToString(App.
          SCUser.city);
    txtCountry.Text = Convert.
          ToString(App.SCUser.country);
    txtFollowers.Text = Convert.
          ToString(App.SCUser.followers_
    count);
    txtFollowing.Text =
Convert.ToString(App.SCUser.followings_
count);
    profilePhoto.ImageSource = new
    BitmapImage(new Uri(
      App.SCUser.avatar_url));

   }
  }
 }
}
```

- Now add the following code changes to NowPlaying.xaml.cs. ***Please check the source code as the code is too big to fit here.***

```
// The Blank Page item template is
documented at http://go.microsoft.com/
fwlink/?LinkId=234238

namespace MusicPlayer
{
 public sealed partial class NowPlaying
 : Page
 {
   private bool
      isMyBackgroundTaskRunning = false;
   AppShell shell =
      Window.Current.Contentas AppShell;
private AutoResetEvent
backgroundAudioTaskStarted;

private bool IsMyBackgroundTaskRunning
{
 get
 {
  if (isMyBackgroundTaskRunning)
   return true;

   string value =
   ApplicationSettingsHelper.
   ReadResetSettingsValue
   (ApplicationSettingsConstants.
   BackgroundTaskState) as string;

   if (value == null)
   {
    return false;
   }
   else
   {
    ...
    return isMyBackgroundTaskRunning;
   }
  }
 }

 public NowPlaying()
 {
  this.InitializeComponent();
  this.Loaded += NowPlaying_Loaded;
 }

private void NowPlaying_Loaded
(object sender, RoutedEventArgs e)
{
 backgroundAudioTaskStarted
  = new AutoResetEvent(false);
 if (!IsMyBackgroundTaskRunning)
 {
  StartBackgroundAudioTask();
 }
 else
 {
  //Start playback if Paused.
  ...
 }
}
private void
    StartBackgroundAudioTask()
{
   //check source code
   ...
}

private void
   AddMediaPlayerEventHandlers()
{
   ...
  }

  protected override void
    OnNavigatedTo(NavigationEventArgs e)
  {
    ...
  }

  async void BackgroundMediaPlayer_
  MessageReceivedFromBackground(object
sender,MediaPlayerDataReceivedEventArgs e)
  {
    ...
    backgroundAudioTaskStarted.Set();
    return;
   }
  }

  public int GetSongIndexById(Uri id)
  {
   return App.likes.FindIndex(s => new
        Uri(s.stream_url) == id);
  }

  protected override void
    OnNavigatedFrom(NavigationEventArgs
  e)
  {
   if (isMyBackgroundTaskRunning)
```

```
 {
  ApplicationSettingsHelper.
  SaveSettingsValue(
  ApplicationSettingsConstants.
  BackgroundTaskState,
  BackgroundTaskState.Running.
  ToString());
 }
 base.OnNavigatedFrom(e);
}

private Uri GetCurrentTrackId()
{
 ...
}

private async void LoadTrack(
SoundCloudTrack currentTrack)
{
 ...
}

private void btnPlay_Click(object
sender, RoutedEventArgs e)
{
 if (IsMyBackgroundTaskRunning)
 {
  if (MediaPlayerState.Playing
  == BackgroundMediaPlayer.Current.
  CurrentState)
  {
   BackgroundMediaPlayer.Current.
   Pause();
  }
  else if (MediaPlayerState.Paused
  == BackgroundMediaPlayer.Current.
  CurrentState)
  {
   BackgroundMediaPlayer.Current.
   Play();
  }
  else if (MediaPlayerState.Closed
  == BackgroundMediaPlayer.Current.
  CurrentState)
  {
   StartBackgroundAudioTask();
  }
 }
 else
 {
  StartBackgroundAudioTask();
 }
}

private void btnNext_Click(object
sender, RoutedEventArgs e)
{
 //Send message to background task
 MessageService.
  SendMessageToBackground(new
  SkipNextMessage());
 }

 private void btnPrev_Click(object
 sender, RoutedEventArgs e)
 {
  //Send message to background task
  MessageService.
  SendMessageToBackground(new
  SkipPreviousMessage());
 }
 }
}
```

In the above code, we have done the following:

1. As soon as the Page is loaded, we check if the BackgroundTask is currently running and toggle playback.

2. If this is the first time or the Background task is not running, we start a new BackgroundTask and enable App to Background Task messaging.

3. This allows us to control the Media Player from our UI as well as get notified if Media controls (Keyboard, Phone Hardware buttons etc) are pressed, which can be used to display correct Track details if our App is in the foreground.

4. We also send respective Next/Previous Message to Background Task to move between songs in Playlist.

BackgroundAudioTask

- Please check the source code to observe the code changes to *MyBackgroundAudioTask.cs* file.

You will observe in the source code that we updated the song model with **SoundCloudTrack** custom object, handled messages from Foreground App, saved current Track Id and changed tracks.

BackgroundAudioShared

- Update *SoundCloudUser.cs* file with the following code to add new properties which are used in *Me.xaml* file in MusicPlayer project.

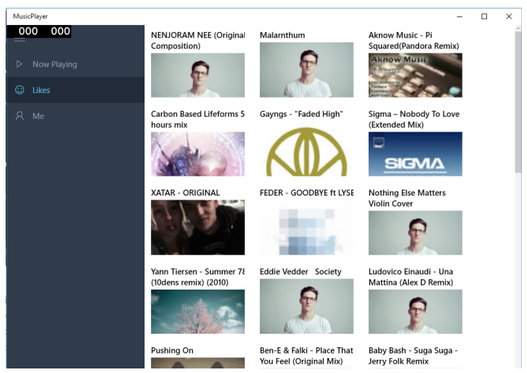
```
namespace BackgroundAudioShared
{
 public class SoundCloudUser
 {
  public int id { get; set; }
  public string permalink { get; set; }
  public string username { get; set; }
  public string avatar_url { get; set; }
  ...
  public int followers_count
  { get; set; }
  public int followings_count
  { get; set; }
 }
}
```

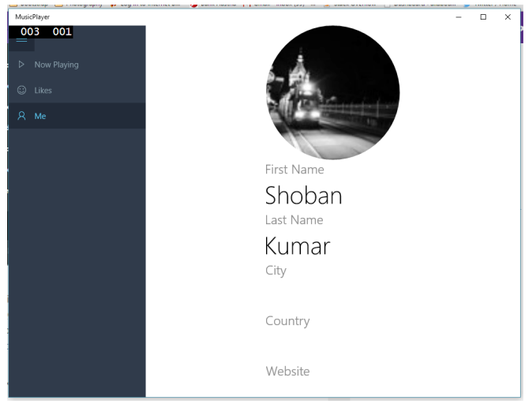
- Change UpdatePlaylistMessage.cs to use SoundCloudTrack object.

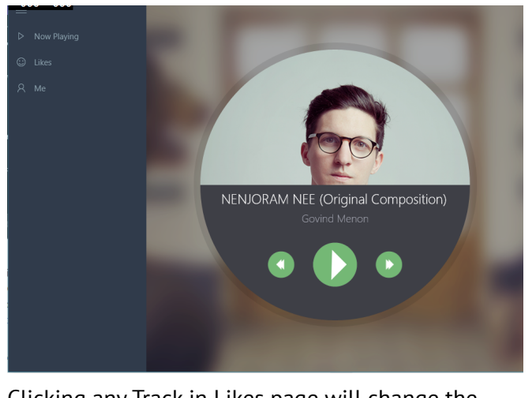
```
namespace BackgroundAudioShared.Messages
{
 [DataContract]
 public class UpdatePlaylistMessage
 {
  public UpdatePlaylistMessage
  (List<SoundCloudTrack> songs)
  {
   this.Songs = songs;
  }

  [DataMember]
  public List<SoundCloudTrack> Songs;
 }
}
```

Testing our Windows 10 Sound Cloud Music Player app

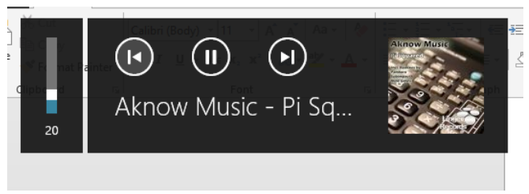
Press debug and test the changes and if there are no compile errors, you should see the changes in Profile, Likes and Now Playing Pages as shown here.







Clicking any Track in Likes page will change the "now playing" Track to the selected track. Press any of the hardware buttons to show Media Controls.



Minimize the Music Player app and enjoy the music.

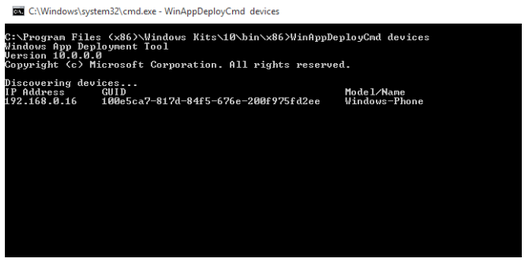
Windows 10 Application Deployment tool

Windows 10 Application Deployment tool (WinAppDeployCmd.exe) is a command line utility that can be used to remotely deploy Apps to Windows Mobile Devices.

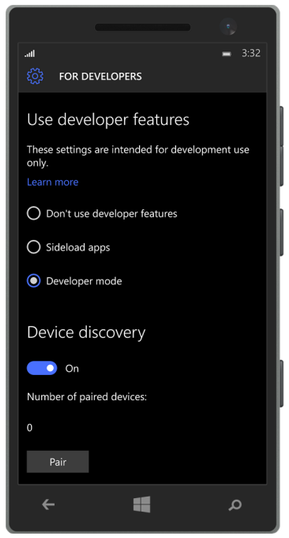
Let us try to deploy Music Player App to a **Windows Mobile**.

- Open Command Prompt and navigate to *C:\Program Files (x86)\Windows Kits\10\bin\x86*

- Type *WinAppDeployCmd* devices and press enter to display the list of available remote devices.



Make sure you enable developer features in your device and pair it with your development PC.

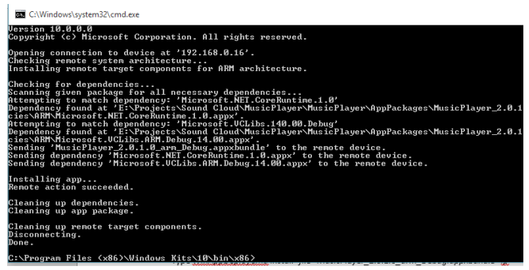


- Create App Package for MusicPlayer by Right click MusicPlayer project -> Store -> Create App Packages...

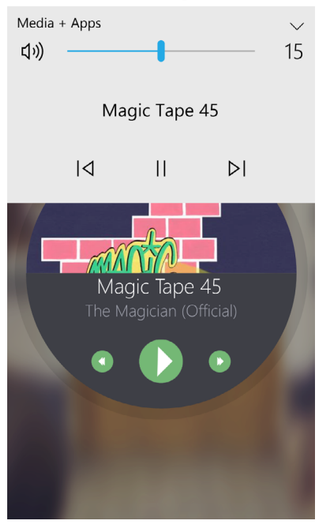
- In the popup message. Select No for "Do you want to build packages to upload to the Windows Store" and click Next.

- Choose a folder and click create to create App Packages (Make sure ARM Architecture is selected)

- Type *WinAppDeployCmd install -file "MusicPlayer_2.0.1.0_arm_Debug.appxbundle" ip [YOUR REMOTE IP HERE]* and press enter to install the App Package.



Open the App in Windows Mobile device and see how our App interacts with Media Controls and plays Audio in the background.



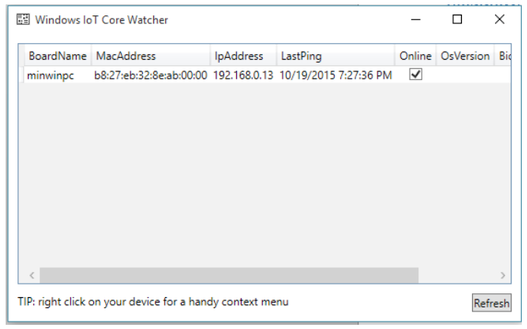
Raspberry Pi 2

Let us deploy our Music Player app to a Raspberry Pi 2 which is running Windows 10 IoT Core and see how the App works in a complete new device family ***without any change***.

Setup your PC by following the Getting Started Instructions here http://bit.ly/1DQVtr3 . Then download the latest preview version of Windows 10 IoT Core from downloads Page http://bit.ly/1GBq9XR

Setup your Raspberry Pi 2 by following the instructions here http://bit.ly/1dLQTOp

If everything is setup properly, you will see your Raspberry Pi 2 listed in Windows IoT Core Watcher.

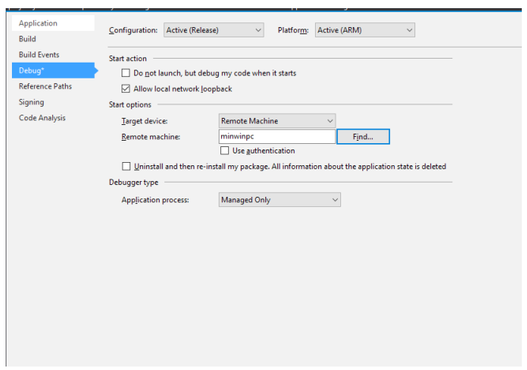


Open the Music Player solution in Visual Studio and follow these steps to deploy the App.

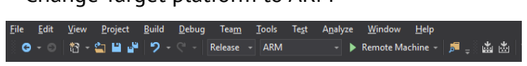
- Right click Music Player project and select Properties

- Click Debug tab and Select Target Device to Remote Machine

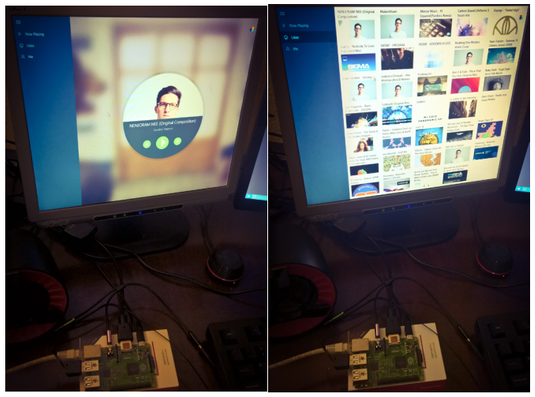
- Enter the ip address of your Raspberry Pi2 or click Find to scan for multiple devices and select the Target device.



- Change Target platform to ARM



- Press F5 to deploy and debug the App or simply Right click Music Player Project -> Deploy to *deploy the App* and start enjoying your Sound Cloud Music player in Raspberry Pi 2. Here's a screenshot:



Windows 10 IoT core can run only one App in the foreground. To increase the performance, the raspberry Pi version of the Music Player app can be simplified by removing the background Task and using MediaElement or playback as shown in the first part of this series.

Conclusion

The Universal Windows Platform Apps aims at changing the way we create apps. Instead of creating separate apps for different platforms, UWP apps would run across all major platforms with minor code changes. With Windows 10, Microsoft aims at bringing different platforms - PC, tablets, phones, XBox, Internet of Things (IoT) together and we sincerely hope that this platform and the benefits it brings, will create an attractive ecosystem for Windows 10 devices ■

Download the entire source code from GitHub at bit.ly/dncm21-win10uwp

About the Author

Shoban Kumar *is an ex-Microsoft MVP in SharePoint who currently works as a SharePoint Consultant. You can read more about his projects at* http://shobankumar.com. *You can also follow him in Twitter @ shobankr*

Angular 2 Developer Preview

These days, front-end JavaScript frameworks are a trending topic across the developer community. Heavy usage of these frameworks has made the JavaScript language designers think of making the language better, and fit it to the scale at which it is being used these days. The ECMAScript 6 specification has brought in a significant number of changes to the language and the language designers have decided to make these updates more frequent to keep the language relevant to the changing landscape of web development. Browser vendors are also keeping up by progressively implementing ECMAScript 6 changes over time. [Recommended reading - http://www.dotnetcurry.com/javascript/1090/ecmascript6-es6-new-features]

These changes to the language and the web as a whole, makes the ecosystem better. The frameworks built on this new ecosystem will be able to provide better ways of solving problems that the developers are actually facing. This is one of the reasons why the Angular team has decided to write the next version, Angular 2 from scratch leveraging the latest enhancements of the web.

Though Angular 1.x has proved to be a good framework for building rich internet applications (RIA), it was built with older browsers in mind. The framework had to build its own module system and a set of abstractions to make its usage easier. By leveraging the new features of the web, the framework won't have to define certain abstractions, and will instead focus on making the experience of working on it better in many other ways. Angular 2 still adopts some of the good things from Angular 1, but at the same time, it abandons features that complicate things while writing large applications or, the features that are already covered by the language.

In case you are wondering about a migration strategy from Angular 1 to Angular 2, please refer to this link - http://angularjs.blogspot.in/2015/08/angular-1-and-angular-2-coexistence.html

In case you haven't heard of it earlier, Microsoft and Google are working together to help make Angular 2 better. Angular has been using AtScript, a superset of Microsoft's TypeScript and both these languages have been merged now into TypeScript. Going forward, you will be writing Angular 2 applications using TypeScript.

In this article, we will take a look at the core concepts and the basic features of Angular 2.

Disclaimer: *Angular 2 is still in alpha. The code snippets shown in this article and in the supporting sample code may not work in the future releases.*

Angular 2 Core Concepts

Like Angular 1, Angular 2 (currently in alpha) is built on a set of concepts that are used throughout the framework and they would be used directly or indirectly, while writing applications. Let's take a look at them.

Change Detection

At the heart of any front-end web framework is the technique used for change detection. Angular 2 adds a powerful and much flexible technique to detect changes on the objects used in the application. In Angular 1, the only way the framework detects changes, is through dirty checking. Whenever digest cycle runs in Angular 1, the framework checks for changes on all objects bound to the view and it applies the changes wherever they are needed. The same technique is used for any kind of objects. So here we don't have a chance to leverage the powers available in objects - like observables and immutables. Angular 2 opens this channel by providing a change detection system that understands the type of object being used.

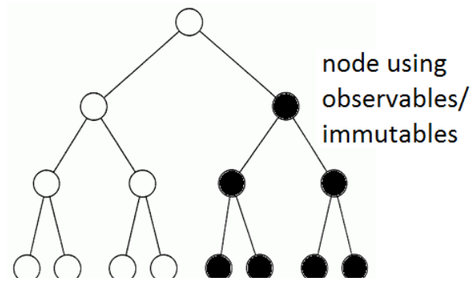
In addition, the change detectors in Angular 2 follow a tree structure to detect changes. This makes the system predictable and it reduces the time taken to detect changes.

If plain JavaScript objects are used to bind data on the views, Angular has to go through each node and check for changes on the nodes, on each browser event. Though it sounds similar to the technique in Angular 1, the checks happen very fast as the system has to parse a tree in a known order.
If we use Observables or, Immutable objects instead of the plain mutable objects, the framework understands them and provides better change detection. Let's delve further into the details:

- Immutable Objects: As the name itself says, an immutable object cannot be modified as it is created. A change made in the object re-creates the object itself. The re-creation kicks in an event to notify user about the change. So the bindings under the subtree using immutable objects, are not parsed until such an event occurs. When the event is raised, the subtree is checked and the changes are applied on it. The checks will not happen on browser events following this check, unless the event is raised again.

- Observable Objects: The observable objects implement a reactive mechanism of notifying the changes. They emit events when the object changes. Subscribers to these events get notified whenever the object changes and they can use the new value to update UI or, perform the next action upon receiving the notification.

Angular 2 understands observable objects. When an object sends a change notification, the framework checks for changes in the subtree under the node containing the component depending on the changed object. This subtree won't be checked for any changes until another change event on the same object is raised.



Both immutable objects and observable objects reduce the number of comparisons to be made while detecting changes. This improves the performance of the application.

There is no restriction on usage of libraries for the two types of objects discussed above. We can use any library according to our convenience. It is also possible to use both immutable objects and observable objects together in the same application.

We need to tell the framework that the component uses one of these objects. It can be done using a property in the component annotation.

```
@Component({
  selector:'my-component',
  changeDetection: ON_PUSH
})
```

Dependency Injection

Dependency injection (DI) has been a part of typed object-oriented languages for quite some time and the pattern continues to play an important role there. It makes the code testable and maintainable as usage of the pattern reduces dependency on concrete objects in components of the application. Many JavaScript developers, including me, felt happy when we saw DI implemented in the Angular framework. Without any doubt, it is one of the reasons behind wide usage of the framework.

Angular 2 continues to use this pattern. It now provides us with a simple and yet powerful DI container. The DI container in Angular 2 has a single API for injecting dependencies and it also provides ways to control lifecycle of dependencies and specify the type of dependency to be injected when a type is requested. The DI works in the same hierarchy in which the components are built. Dependencies have to be declared along with annotations of the components. A dependency declared on a component can be used either by the same component or by any of the children of the component. Children components can override the way a dependency is declared in the parent as well.

Zones

After the application starts, we need a process to keep running behind the scenes that kicks in the process of change detection so that the model and the UI are always in sync. Angular 2 includes a library called Zone.js, which helps us in keeping things always in sync. A zone is an execution context that persists across async tasks. All operations performed in Angular 2 are carried under this context. We don't need to invoke them or attach operations to them. Zones know what kind of operations to be handled in the context. Because of zones, we don't have to take any pain while including non-angular UI components in Angular 2.

Directives

Like in Angular 1.x, directives in Angular 2 are used to extend HTML. The directives in Angular 2 are more similar to the way DOM appears and they provide ways to get into life cycle events to take better control over the way a directive works. Architecture of the directives in Angular 2 embraces bindings a lot and hence reduces the need of direct DOM manipulation. Also, we don't need to deal with different restrict types as well as with the camel case notation while naming directives; both of these properties are handled by a single selector property.

There are three different types of directives in Angular 2:

- **Component Directives:** Everything in an Angular2 application is a component. The application starts by bootstrapping a component and other components are rendered as children of this component. So hypothetically if we take an aerial view, an Angular 2 application looks like a tree of components. The components are used to define a new HTML element. This feature is built using the techniques introduced in HTML5 web components. If the browser doesn't support any of the features, the framework emulates them.

Code of a component includes a class and two annotations. The following snippet shows structure of a component written in TypeScript (if you are not familiar with TypeScript, you may want to check a basic tutorial on the topic: Hello TypeScript – Getting Started):

```
@Component({
  selector: 'my-component',  //Name of
  the component, it will be used in HTML
  bindings: [MyService]  //List of
  services to be used in the component
  or, its children
})

@View({
  template: '<div>{{sampleText}}
  </div>',  //Template of the component
  directives: []  //List of directives
  to be used in the component
})

class MyComponentClass{
  public sampleText: string;
  constructor(private svc: MyService){
    this.sampleText = this.svc.message;
  }
}
```

As we can see, the component is a class decorated with the annotations *Component* and *View*. An object of this class is the view model for the component. Any public property defined in this class can be used to bind data in the component's template.

- **Decorator Directives:** A decorator is the simplest kind of directive. It is used to extend behavior of an HTML element or an existing component. This directive doesn't make a lot of impact on the view, but helps in achieving certain small yet important functionalities on the page. A decorator directive is a class with the *Directive* applied on it. The following snippet shows an example of a decorator directive:

```
@Directive({
  selector:'[my-decorator]',
  host: {
    '(mouseover)': 'myFunction()'
  }
})
class MyDecoratorClass{
  //class definition
}
```

If you observe this snippet, the directive interacts with the host element through configuration. And if we use this feature effectively, we can avoid performing direct DOM manipulations in the component. Such behavior makes the code easy to port to non-browser based applications.

- **Directives with ViewContainer:** The directives that use a *ViewContainerRef* object can take control over the content of the element on which they are applied. These directives can be used when template of the view has to be shown/hidden based on a condition, to iterate over, or to tweak the content based on a scenario. The directives like ng-for and ng-if fall into this category.

Bindings and Templates

Like every front-end framework, Angular 2 has a very good support for binding. As already mentioned, view-model for any component is an object of its class. Public properties and methods of the classes can be used to show data, handle events and handle properties of the HTML elements. The framework introduces a different syntax for data binding. Public properties of the component class can be bound to any element on the view. For example, following snippet binds the property *heading* to an *h1* element:

```
<h1>{{heading}}</h1>
```

The public properties can be assigned as value to any of the properties as well. The properties can be bound in two ways:

```
<span [title]=”myTitle”> Some text in
the span </span>
```

Or,

```
<span bind-title=”myTitle”> Some text in
the span </span>
```

The *bind-title* attribute in the above snippet is not a decorator directive, the prefix *bind* is a convention and it is an alternative syntax to the square bracket [attribname] syntax used earlier. The *bind-* prefix can be used with any valid HTML attribute to bind the property with a value on the view model.

Similarly, public methods defined in the component class can be used to handle events.

```
<button (click)=”clicked()”>Click
Here!</button>
```

Or,

```
<button on-click=”clicked()”>Click
Here!</button>
```

Angular 2 doesn't have direct support for two-way binding. All property binders are one-way. It has a work around to support two-way binding. Following is the syntax:

```
<input type=”text” [(ng-model)]=”name”
/>
```

The two-way binding in Angular 2 is a combination of property binding and event binding. Value of the property is bound to text of the textbox and any change in the text raises an event to update the model. In fact, the above snippet is the shorter form

of the following snippet:

```
<input type=”text” [ng-model]=”name”
(ng-model)=”name=$event” />
```

Services

Angular 2 doesn't have the concept of services. Anything other than directives are simple classes. Good thing is, we can apply dependency injection on these classes and these classes can be injected into any directives. We can consider any plain simple class as a service. The following snippet shows a service and its usage in a component using TypeScript:

```
class Message{
  greeting: string;
  constructor(){
    this.greeting = "A sample message";
  }
}

@Component({
  selector: 'show-message',
  bindings: [Message]
})
@View({
  template: '<div>{{message}}</div>'
})
class ShowMessage{
  message: string;
  constructor(messageSvc: Message){
    this.message = messageSvc.greeting;
  }
}
```

Let's build a component

Now that we had a tour around some new features of Angular 2, let's build a simple component. An Angular 2 application can be written using ECMAScript 5, ECMAScript 6 (ECMAScript 2015) or, TypeScript. The code written in ES6 and TypeScript is almost similar except that TypeScript would have support for types. We will build a simple music player component using ES5 and then using TypeScript.

Note: *Instead of writing the component in both TypeScript and ES5, I could have used a TypeScript compiler to generate ES5 code. However my intention in this article is to show how Angular 2 code can be written using both TypeScript and ES5.*

Music player using EcmaScript 5 (ES5)

The Angular 2 framework has two different versions of its JavaScript file, one to be used with ES6 or, TypeScript and the second for ES5. If you visit Angular's code library for current version (alpha 37), you will find multiple versions available for each file. The Angular 2 core framework's ES5 version is angular2.sfx.dev.js and ES6 version is angular2.dev.js. For the ES5 example, we will use angular2.sfx.dev.js.

After including this file on the page, we can start writing some JavaScript. At first, let's create a service to store data. It can be any constructor function with some public fields. In the component, we will be showing the current song being played and next and previous buttons with tooltips showing the corresponding songs. We just need three fields assigned in the constructor function.

```
function MusicPlayerInfo() {
  this.nowPlaying = "Yaad kiya dil ne";
  this.next = "Hosh walo ko khabar";
  this.prev = "Hotho se chulo tum";
}
```

To show the next and previous songs, we need to create a directive that uses data on the above service and displays name of the song in a tooltip. The Angular 2 library exposes an API that makes the code of directive look very close to the ES6 syntax. As already discussed, we need to define a class and apply the directive decorator to make it a directive. ES5 version defines the methods *Directive* and *Class* to make this task easier. The following snippet shows code of the directive:

```
var SongTip = ng.Directive({
  selector: '[song-tip]',
  properties: [
    'label: song-tip'
  ],
  host: {
    '(mouseover)': 'show()',
    '[title]': 'title'
  }
})
.Class({
  constructor: [MusicPlayerInfo,
  function (info) {
    this.info = info;
  }],
  show: function () {
    this.title = this.info[this.label];
  }
});
```

In the above snippet:

- The properties option in the directive decorator is used to add properties to instance of the directive class with value assigned to a field on the element. Here, a field named *label* is added to the class that would hold value assigned to song-tip

- The host option is used to interact with the host element of the directive. Here we are handling the mouseover event and setting *title* to the element using a method and a property on the directive object

- Constructor defined in the class depends on the service *MusicPlayerInfo*. So the dependency is specified in the annotation. We haven't bound this dependency yet, we will do that in the next component

We need to have a component element to show play, current and previous buttons. We will call this component *MusicPlayer*. The previous and next buttons will have the *SongTip* directive applied. It would use the field *nowPlaying* from the *MusicPlayerInfo* service to show the currently playing track. The following snippet shows code of the component:

```
var MusicPlayer = ng.Component({
  selector: 'music-player',
  bindings: [MusicPlayerInfo]
})
.View({
  templateUrl: 'templates/musicPlayer.
  html',directives: [SongTip]
})
.Class({
  constructor: [MusicPlayerInfo,
  function (musicInfo) {
  this.nowPlaying = musicInfo.
  nowPlaying;
 }] });
```

Following is the mark up in the musicPlayer.html file:

```
<div class="col-sm-6">
 <div class="col-sm-3 text-center"><span
 class="glyphicon glyphicon-step-
 backward" song-tip="prev"></span></div>

 <div class="col-sm-6 text-center"><span
 class="glyphicon glyphicon-play">
 </span><br/>{{nowPlaying}}</div>

 <div class="col-sm-3 text-center"><span
 class="glyphicon glyphicon-step-
 forward" song-tip="next"></span>
 </div></div>
 <div class="col-sm-6"></div>
```

A few things to observe in the above snippet are:

- The bindings option in component decoration says the component depends on *MusicPlayerInfo*. That means it can be injected in the component and any of its children components or, directives

- The directives option on the view says the component will use the directive *SongTip*

- The field *nowPlaying* created in the constructor is used in template of the component

Let's create another component MusicPlayerHost to host the above component and this component will be used for bootstrapping the application. The following is the snippet showing the code of the component:

```
var MusicPlayerHost = ng.
Component({
  selector: 'music-player-host'
})
.View({
  template: '<h2>{{message}}
  </h2><music-player></music-player>',
  directives: [MusicPlayer]
})
.Class({
  constructor: function () {
    this.message = "My music player";
  }
});
```

As the Angular 2 application has to start with a component, let's bootstrap the application using the

MusicPlayerHost component. The bootstrap process has to begin after the DOM content is loaded, so we need to add a listener to the DOMContentLoaded event to bootstrap the application.

```
document.
addEventListener('DOMContentLoaded',
function () {
  ng.bootstrap(MusicPlayerHost);
});
```

Music Player using TypeScript

Setting up the environment with TypeScript needs some efforts. The TypeScript code has to be converted to JavaScript before the application is loaded in the browser. The process of converting TypeScript to JavaScript is called **transpilation**. The word transpilation is derived from the words translation and compilation. In addition, we should have a module loader to load the transpiled modules along with the modules exposed by Angular 2. The demo code contains the following NPM packages to fill these gaps:

- typescript, gulp and gulp-typescript to transpile TypeScript
- systemjs to load modules
- traceur to make the browser understand Angular 2's annotations

Note: We are using Gulp to perform the task of transpilation. If you are not familiar with Gulp, you may read the section on Gulp in the article Using Grunt, Gulp and Bower in Visual Studio 2013 and 2015

The typescript task defined in gulpfile.js transpiles the files and generates sourcemaps. This is shown below:

```
var gulp = require('gulp'),
  gts = require('gulp-typescript'),
  ts = require('typescript'),
  sourcemaps = require('gulp-
  sourcemaps');

  gulp.task('typescript', function(){
  var tsProject = gts.
  createProject('tsconfig.json', {
    typescript: ts
});

return gulp.src(['typings/**/**.ts',
'typescript/*.ts'])
  .pipe(sourcemaps.init())
  .pipe(gts(tsProject))
  .pipe(sourcemaps.write('../maps', {
  includeContent: false, sourceRoot: '/
  typescript' })
  .pipe(gulp.dest('js'));
});
```

This task copies all of the transpiled files into the js folder. The HTML file rendering the application loads the helper libraries and then loads the main module of the application using SystemJS, which in turn loads all other modules.

```
<script src="\node_modules\traceur\bin\
traceur-runtime.js"></script>

<script src="\node_modules\systemjs\
dist\system.src.js"></script>

<script>
  System.config({
    defaultJSExtensions: true
  });
</script>

<script src="\node_modules\angular2\
bundles\angular2.dev.js"></script>

<script>
  System.import('/js/bootstrap');
</script>
```

All of the code that we wrote in the previous section will take a different form with TypeScript classes. The following is the class for *MusicPlayerInfo* service:

```
export class MusicPlayerInfo{
  nowPlaying: string;
  next: string;
  prev: string;

  constructor(){
    this.nowPlaying = "Yaad kiya dil
    ne";
    this.next = "Hosh walo ko khabar";
    this.prev = "Hotho se chulo tum";
  }
}
```


The *SongTip* directive needs the *Directive* annotation and the service *MusicPlayerInfo* to be imported into the module and then use them in the code. Annotations in Angular 2 are decorators (a feature to be released in ES7, and is already implemented by TypeScript) with metadata information. As TypeScript understands decorators but not annotations, as they are not a part of the standard language, we need to use the *traceur* compiler (which is already included) to parse them on the browser. Following snippet shows the code of this directive:

```
import {Directive} from 'angular2/
angular2';

import {MusicPlayerInfo} from './
MusicPlayerInfo';

@Directive({
  selector:'[song-tip]',
  properties: [
    'label: song-tip'
  ],
  host: {
    '(mouseover)': 'show()',
    '[title]':'title'
  }
})

export class SongTip{
  title: string;
  label: string;
  constructor(private musicPlayerInfo:
MusicPlayerInfo){}

  show(){
    this.title = this.
musicPlayerInfo[this.label];
  }
}
```

Notice the constructor in the above class. It accepts a parameter of type *MusicPlayerInfo* and we don't need to add anything in addition for dependency injection, as the type specification in the constructor itself is used for injecting the dependency.

Similarly, the *MusicPlayer* component needs the annotations *Component* and *View* from Angular's core modules and *MusicPlayerInfo* from the current folder. The annotations *Component* and *View* have the same responsibilities as they had in previous section; they just appear a bit different.

```
import {Component, Directive, View} from
'angular2/angular2';
import {MusicPlayerInfo} from './
MusicPlayerInfo';
import {SongTip} from './SongTip';

@Component({
  selector:'music-player'
})
@View({
  templateUrl:'templates/musicPlayer.
  html',
  directives:[SongTip]
})
export class MusicPlayer{
  nowPlaying: string;

  constructor(info: MusicPlayerInfo){
    this.nowPlaying = info.nowPlaying;
  }
}
```

The *MusicPlayerHost* component is straightforward. It uses the above component, so it has to be specified in directives.

```
import {bootstrap, Component, View} from
'angular2/angular2';
import {MusicPlayerInfo} from './
MusicPlayerInfo';
import {SongTip} from './SongTip';
import {MusicPlayer} from './
MusicPlayer';

@Component({
  selector:'music-player-host',
  bindings: [MusicPlayerInfo]
})

@View({
  template: '<h2>{{message}}</h2><music-
  player></music-player>',
  directives:[MusicPlayer]
})

class MusicPlayerHost{
  message: string;
  constructor(){
    this.message = "My music player";
  }
}
```

To bootstrap the application, the above component can be passed to the bootstrap function.

```
bootstrap(MusicPlayerHost);
```

Conclusion

Angular 2 is written from ground-up using latest features available in the web ecosystem and it brings several significant improvements over the framework's older version. While it retires a number of Angular 1 features, it also adopts a number of core concepts and principles from older version of the framework. After collaborating with the TypeScript team at Microsoft, both the teams are working really hard to create a great framework and they are also working with TC39 team to make JavaScript a better language. Let's try it, explore and provide feedback to the core team on making it better! ■

Download the entire source code from GitHub at
bit.ly/dncm21-angular2features

About the Author

ravi kiran

Microsoft® Most Valuable Professional

Rabi Kiran (a.k.a. Ravi Kiran) is a developer working on Microsoft Technologies at Hyderabad. These days, he is spending his time on JavaScript frameworks like AngularJS, latest updates to JavaScript in ES6 and ES7, Web Components, Node.js and also on several Microsoft technologies including ASP.NET 5, SignalR and C#. He is an active blogger, an author at SitePoint and at DotNetCurry. He is rewarded with Microsoft MVP (ASP.NET/IIS) and DZone MVB awards for his contribution to the community

THANK YOU

FOR THE 21ST EDITION

@damirarh

@sravi_kiran

@gilfink

@kunalchandratre

@maheshdotnet

@shobankr

@suprotimagarwal

@saffronstroke

JOIN OUR TEAM

A MAGAZINE FOR .NET AND JAVASCRIPT DEVS

- ASP.NET
- SHAREPOINT
- JAVASCRIPT
- PATTERNS
- AZURE
- VISUAL STUDIO
- .NET
- C#, WPF

We've got it all!

80K PLUS READERS

200 PLUS AWESOME ARTICLES

21 EDITIONS

FREE SUBSCRIPTION USING YOUR EMAIL

EVERY ISSUE DELIVERED

RIGHT TO YOUR INBOX

NO SPAM POLICY

SUBSCRIBE TODAY!