THE MICROSOFT JOURNAL FOR DEVELOPERS

NOVEMBER 2014 VOL 29 NO 11

# msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

NOVEMBER 2014 VOL 29 NO 11

# msdn magazine

Unit Testing
Async Code.....16, 22



## COLUMNS

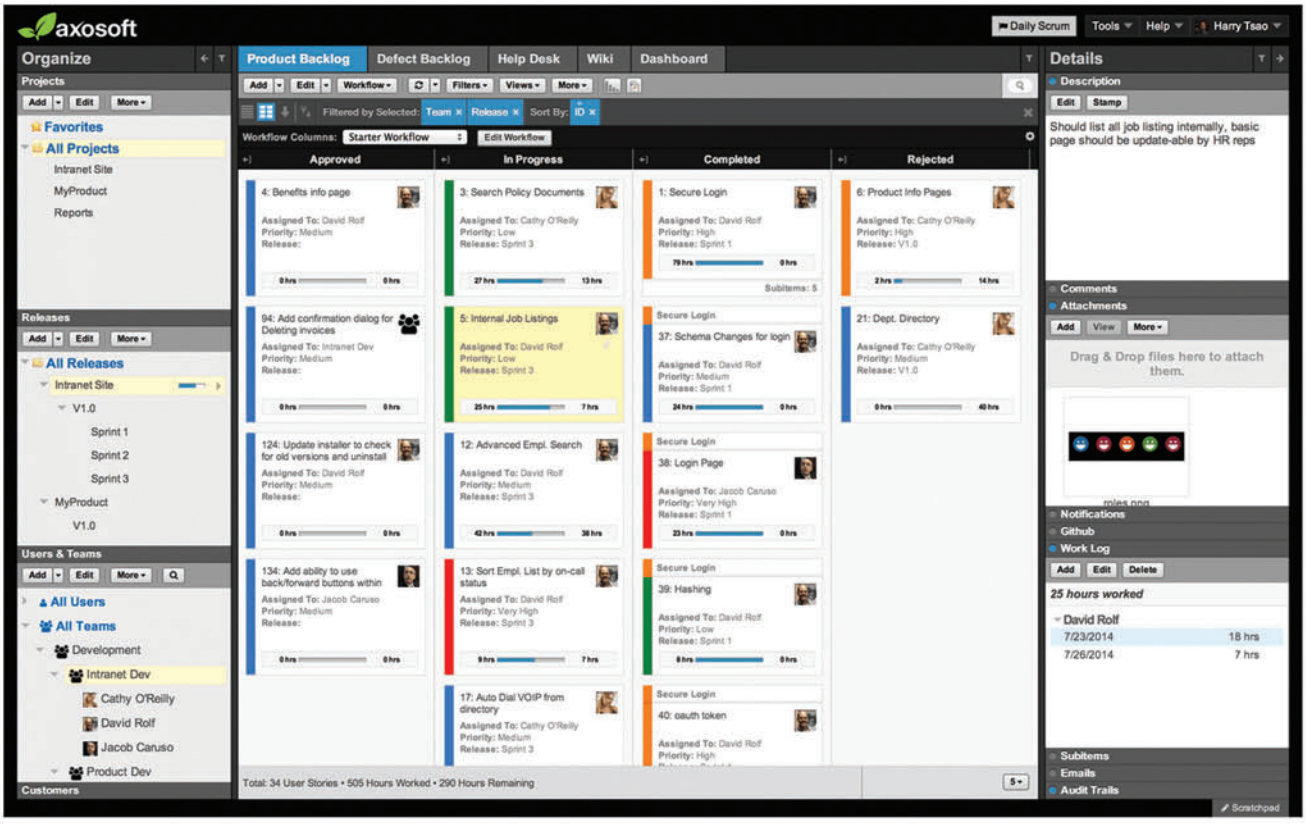


Microsoft

# Don't let your dev team get bogged down by complicated processes.

axosoft

# Transform your team into a lean, mean Scrum machine with Axosoft!




## Empower management & team members with total visibility into your development process.

**Axosoft Scrum** provides key project insights with a Kanban card view that clearly illustrates an item's workflow status, while burndown charts and custom reports help management identify further opportunities for improvement. Out-of-the-box the tool is configured with Scrum best practices in mind! Get started free at **Axosoft.com/MSDNscrum**.

# msdn magazine

**NOVEMBER 2014** VOLUME 29 NUMBER 11





*MSDN Magazine* (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: *MSDN Magazine*, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.





**Corporate Address:** 1105 Media, Inc., 9201 Oakdale Ave., Ste 101, Chatsworth, CA 91311, www.1105media.com

**Media Kits:** Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@1105media.com

**Reprints:** For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International, Phone: 212-221-9595, E-mail: 1105reprints@parsintl.com, www.magreprints.com/QuickQuote.asp

**List Rental:** This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jlong@meritdirect.com; Web: www.meritdirect.com/1105

***All customer service inquiries should be sent to MSDNmag@1105service.com or call 847-763-9560.***

Microsoft

BPA WORLDWIDE

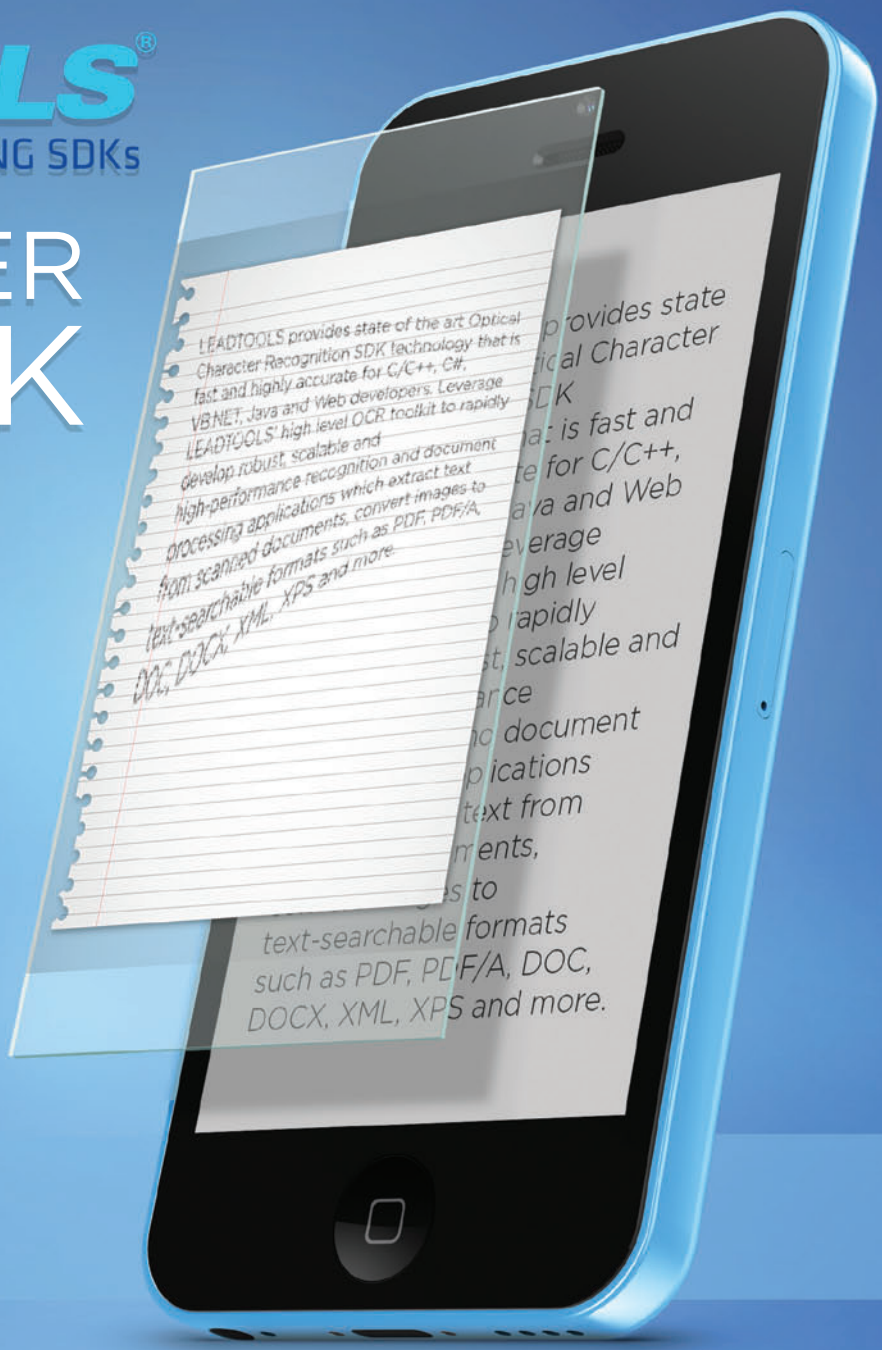
Printed in the USA

# LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

# THE PREMIER **OCR** SDK

Converting an Image to text has never been easier. For all your OCR needs, look for LEADTOOLS.




DESKTOP TABLET MOBILE

LEADTOOLS provides state of the art Optical Character Recognition SDK technology that is fast and highly accurate for multiple platforms. Leverage LEADTOOLS' high level OCR toolkit to rapidly develop robust, scalable and high-performance recognition and document processing applications which extract text from scanned documents and convert images to text-searchable formats such as PDF, PDF/A, DOC, DOCX, XML, XPS and more.

.NET Windows API WinRT Linux iOS OS X Android HTML5/JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
**WWW.LEADTOOLS.COM**

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
**800.637.1840**

# Editor's Note

MICHAEL DESMOND

# Putting Async to the Test

The lead articles in this issue of *MSDN Magazine* focus on a shared theme: Managing the task of unit testing asynchronous code projects.

Stephen Cleary has written extensively about async development over the past couple years here at *MSDN Magazine*, and his feature article, "Unit Testing Asynchronous Code," explores how unit testing must be adapted to prove out async projects. He looks at some of the current unit testing frameworks—such as MSTest, NUnit and xUnit—and offers strategies for ensuring best results. Sven Grand follows up with the second feature article in this issue, "Unit Testing Asynchronous Code: Three Solutions for Better Tests." He offers up three solutions for improving the design of tested async code with the goal of eliminating unit tests that are slow and fragile.

> "You don't need to write a test to ensure Microsoft Task.Run really works. Believe me, it works. Your tests get simpler and cleaner if you manage to test only your own code."
>
> *— Sven Grand, Quality Engineering Software Architect, Philips Healthcare*

Of course, async programming is a topic of ongoing concern here at *MSDN Magazine*. Back in the October 2011 issue, Microsoft experts Eric Lippert, Mads Torgersen and Stephen Toub wrote a trifecta of async-themed features that previewed the new async and await keywords in Visual Studio 2012 (msdn.microsoft.com/magazine/hh463583). Since then, we've published nearly a dozen feature articles focused on async development. One of those—Cleary's March 2013 article, "Best Practices in Asynchronous Programming"—is among the most-read articles in the past two years (msdn.microsoft.com/magazine/jj991977).

The point is, async is a hot topic—and for good reason. As Grand writes in his article this month, "Whether for CPU-based parallelism or IO-based concurrency, developers are employing asynchrony to help make the most of the resources available and ultimately do more with less. More responsive client applications and more scalable server applications are all within reach."

The benefits of async development are clear—particularly in an increasingly cloud-driven world—but async programming poses a challenge for developers who are new to the practice. As Cleary says in an interview: "A lot of async tutorials explain how to use async/await, but they don't prepare developers for the viral nature of async. So when they start out writing a single async method and this async/await stuff keeps growing through their code, the warnings go off and they search for verification that this is actually correct."

Which is why this month's issue of *MSDN Magazine* leads off with a pair of articles focused on unit testing asynchronous code. Testing is one of those things that can get pushed to the side when timelines get tight. Yet, a rigorous and disciplined test regime is critical to achieving consistent code quality. The two features this month help developers ensure their async development happens in a well-structured and supported environment.

"Most of the challenges for unit testing asynchronous code are the same as unit testing synchronous code. Code is easier to test when it's designed well," Cleary says. "For asynchronous code, a more functional approach can be helpful—in other words, return results rather than setting values as side effects."

Grand says it's important to focus testing efforts, as developers can end up testing library code that's being called from custom code. "You don't need to write a test to ensure Microsoft Task.Run really works. Believe me, it works. Your tests get simpler and cleaner if you manage to test only your own code," Grand says.

Is your dev shop writing asynchronous code and how have you adapted your testing to accommodate it? E-mail me at mmeditor@microsoft.com.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.



DINO ESPOSITO

# Store User Data in ASP.NET Identity

ASP.NET Identity in Visual Studio 2013 is a way to simplify the boring but essential tasks of managing user data and establishing a more effective membership system. Previously, I provided an overview of the ASP.NET Identity API (msdn.microsoft.com/magazine/dn605872) and examined its involvement with social networks and the OAuth protocol (msdn.microsoft.com/magazine/dn745860). In this article, I'll expand on the extensibility points of ASP.NET Identity starting with the user data representation and underlying data store.

## Lay the Groundwork

First, let's create a new blank ASP.NET MVC project in Visual Studio 2013. All of the defaults the wizard provides are OK, but be sure to select the single user authentication model. The scaffolded code stores user data in a local SQL Server file with an auto-generated name with the following convention: aspnet-[ProjectName]-[RandomNumber]. The code also uses Entity Framework to access the user database in reading and writing. The user data representation is in the class ApplicationUser:

```
public class ApplicationUser : IdentityUser
{
}
```

As you can see, the ApplicationUser class inherits from the system-provided class IdentityUser. To customize the user representation, let's start from here and add a new member to the class:

```
public class ApplicationUser : IdentityUser
{
  public String MagicCode { get; set; }
}
```

Figure 1 **Members Defined on the IdentityUser Base Class**

| Member | Description |
|---|---|
| Id | Unique auto-generated identifier (GUID) for the table. This field is the primary key. |
| UserName | Display name of the user. |
| PasswordHash | Hash resulting from the provided password. |
| SecurityStamp | A GUID automatically created at specific points in the UserManager object lifetime. Typically, it's created and updated when the password changes or a social login is added or removed. The security stamp generally takes a user information snapshot and automatically logs in users if nothing has changed. |
| Discriminator | This column is specific to the Entity Framework persistence model and determines the class to which the particular row belongs. You're going to have a unique discriminator value for each class in the hierarchy rooted in IdentityUser. |

The name of the class—ApplicationUser in this example—isn't mandatory and you can change it if you'd like. In this example, I've deliberately chosen the weird "magic-code" field to indicate the dual possibility. You can add fields that require a UI such as birth date or social security number or anything else you want to specify during registration. You can also add fields you need to have, but can calculate silently (for example, an app-specific "magic" code) when the user record is actually created. The ApplicationUser class includes by default the members listed in **Figure 1**.

> You have to pass the additional data down to the layer that will write it to the persistent store.

The fields listed in **Figure 1**, as well as any other fields you add programmatically in the ApplicationUser class definition, end up stored in a database table. The default table name is AspNetUsers. Also, the IdentityUser class exposes a few more properties such as Logins, Claims and Roles. These properties aren't stored in the AspNetUsers table, but find their place in other side tables in the same database—AspNetUserRoles, AspNetUserLogins and AspNetUserClaims (see **Figure 2**).

Modifying the ApplicationUser class doesn't ensure you'll have the additional fields immediately reflected in the UI and saved to the database. Luckily, updating the database doesn't take too much work.

## Changes to the Scaffolded Code

You'll need to edit the application views and models to reflect the new fields. In the application form where new users register with the site, you'll need to add some markup to show the UI for the magic code and any other extra fields. **Figure 3** shows amended code for the CSHTML Razor file behind the register form.

The CSHTML register form is based on a view model class, conventionally called RegisterViewModel. **Figure 4** shows the changes required to the RegisterViewModel class to plug it into the classic validation mechanism of most ASP.NET MVC applications based on data annotations.

These changes aren't enough, however. There's one more step required and it's the most critical. You have to pass the additional

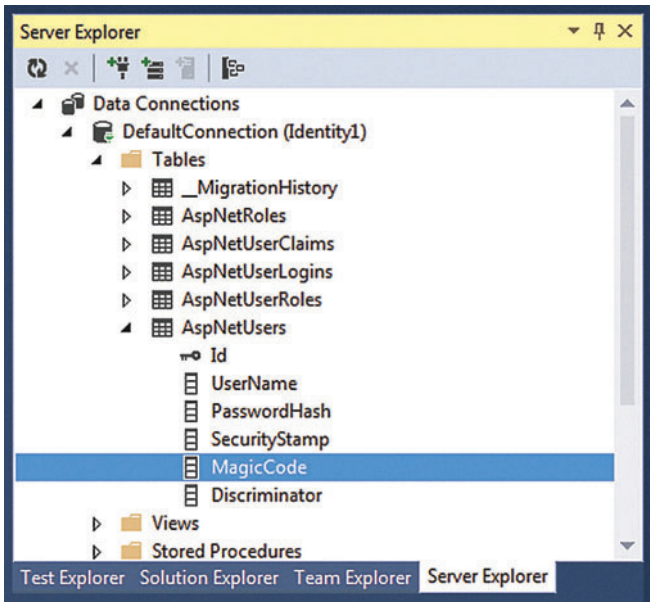



Figure 2 **Default Structure of the ASP.NET Identity User Database**

data down to the layer that will write it to the persistent store. You need to make further changes to the controller method that processes the POST action from the register form:

```
public async Task<ActionResult> Register(RegisterViewModel model)
{
  if (ModelState.IsValid) {
    var user = new ApplicationUser() { UserName = model.UserName,
      MagicCode = model.MagicCode };
    var result = await UserManager.CreateAsync(user, model.Password);
    if (result.Succeeded) {
      await SignInAsync(user, isPersistent: false);
      return RedirectToAction("Index", "Home");
    }
}
```

The key change is saving the posted data for the magic code (or whatever else you want to add to the user definition). This is saved into the ApplicationUser instance being passed to the CreateAsync method of the UserManager object.

## Looking into the Persistent Store

In the sample code generated by the ASP.NET MVC template, the AccountController class has a member defined here:

```
public UserManager<ApplicationUser> UserManager { get; private set; }
```

The UserManager class is instantiated passing the user store object. ASP.NET Identity comes with a default user store:

```
var defaultUserStore = new UserStore<ApplicationUser>(new ApplicationDbContext())
```

Figure 3 **Razor File to Present Users with Extra Fields**

```
@using (Html.BeginForm("Register", "Account",
  FormMethod.Post, new { @class = "form-horizontal", role = "form" }))
{
  @Html.AntiForgeryToken()
  <h4>Create a new account.</h4>
  <hr />
  @Html.ValidationSummary()
  <div class="form-group">
    @Html.LabelFor(m => m.MagicCode, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
      @Html.TextBoxFor(m => m.MagicCode, new { @class = "form-control" })
    </div>
  </div>
  ...
}
```

Much like ApplicationUser, the ApplicationDbContext class inherits from a system-defined class (named IdentityDbContext) and wraps Entity Framework to do the actual persistence job. The great news is you can unplug the default storage mechanism altogether and roll your own. You can base your custom storage engine on SQL Server and Entity Framework and just use a different schema. You could also take advantage of a completely different storage engine such as MySQL or a NoSQL solution. Let's see how to arrange a user store based on an embedded version of RavenDB (ravendb.net). The prototype of the class you'll need is shown here:

```
public class RavenDbUserStore<TUser> :
  IUserStore<TUser>, IUserPasswordStore<TUser>
  where TUser : TypicalUser
{
  ...
}
```

> You can base your custom storage engine on SQL Server and Entity Framework and just use a different schema.

If you intend to support logins, roles and claims, you'll need to implement more interfaces. For a minimal working solution, IUserStore and IUserPasswordStore are enough. The class TypicalUser is a custom class I created to remain decoupled from the ASP.NET Identity infrastructure as much as possible:

```
public class TypicalUser : IUser
{
  // IUser interface
  public String Id { get; set; }
  public String UserName { get; set; }

  // Other members
  public String Password { get; set; }
  public String MagicCode { get; set; }
}
```

At the very least, the user class must implement the IUser interface. The interface counts two members—Id and UserName. You'll

Figure 4 **Changes to the Register View Model Class**

```
public class RegisterViewModel
{
  [Required]
  [Display(Name = "User name")]
  public string UserName { get; set; }

  [Required]
  [StringLength(100, ErrorMessage = "The {0} must be at least {1} character long.",
    MinimumLength = 6)]
  [DataType(DataType.Password)]
  [Display(Name = "Password")]
  public string Password { get; set; }

  [DataType(DataType.Password)]
  [Display(Name = "Confirm password")]
  [Compare("Password", ErrorMessage = "Password and confirmation do not match.")]
  public string ConfirmPassword { get; set; }

  [Required]
  [Display(Name = "Internal magic code")]
  public string MagicCode { get; set; }
}
```

Figure 5 **A Minimally Working User Store Based on RavenDB**

```
public class RavenDbUserStore<TUser> :
  IUserStore<TUser>, IUserPasswordStore<TUser>
  where TUser : TypicalUser
{
  private IDocumentSession DocumentSession { get; set; }

  public RavenDbUserStore()
  {
    DocumentSession = RavenDbConfig.Instance.OpenAsyncSession();
  }

  public Task CreateAsync(TUser user)
  {
    if (user == null)
      throw new ArgumentNullException();

    DocumentSession.Store(user);
    return Task.FromResult<Object>(null);
  }
  public Task<TUser> FindByIdAsync(String id)
  {
    if (String.IsNullOrEmpty(id))
      throw new ArgumentException();

    var user = DocumentSession.Load<TUser>(id);
    return Task.FromResult<TUser>(user);
  }

  public Task<TUser> FindByNameAsync(String userName)
  {
    if (string.IsNullOrEmpty(userName))
      throw new ArgumentException("Missing user name");
    var user = DocumentSession.Query<TUser>()
      .FirstOrDefault(u => u.UserName == userName);
    return Task.FromResult<TUser>(user);
  }

  public Task UpdateAsync(TUser user)
  {
    if (user != null)
      DocumentSession.Store(user);
    return Task.FromResult<Object>(null);
  }

  public Task DeleteAsync(TUser user)
  {
    if (user != null)
      DocumentSession.Delete(user);
    return Task.FromResult<Object>(null);
  }

  public void Dispose()
  {
    if (DocumentSession == null)
      return;
    DocumentSession.SaveChanges();
    DocumentSession.Dispose();
  }

  public Task SetPasswordHashAsync(TUser user, String passwordHash)
  {
    user.Password = passwordHash;
    return Task.FromResult<Object>(null);
  }

  public Task<String> GetPasswordHashAsync(TUser user)
  {
    var passwordHash = user.Password;
    return Task.FromResult<string>(passwordHash);
  }

  public Task<Boolean> HasPasswordAsync(TUser user)
  {
    var hasPassword = String.IsNullOrEmpty(user.Password);
    return Task.FromResult<Boolean>(hasPassword);
  }
 }
}
```

likely want to add a Password member, as well. This is the user class being saved in the RavenDB archive.

Add RavenDB support to your project via the RavenDB.Embedded NuGet package. In the global.asax, you'll also need to initialize the database with the following code:

```
private static IDocumentStore _instance;
public static IDocumentStore Initialize()
{
  _instance = new EmbeddableDocumentStore { ConnectionStringName = "RavenDB" };
  _instance.Initialize();
  return _instance;
}
```

> Any interaction with RavenDB passes through the opening and closing of a document store session.

The connection string points to the path where you should create the database. In an ASP.NET Web application, the natural fit is a subfolder under App_Data:

```
<add name="RavenDB" connectionString="DataDir = ~\App_Data\Ravendb" />
```

The user store class contains code for the methods in the IUserStore and IUserPasswordStore interfaces. These let the application manage users and related passwords. **Figure 5** shows the store implementation.

Any interaction with RavenDB passes through the opening and closing of a document store session. In the constructor of the RavenDbUserStore, you open the session and dismiss it in the Dispose method of the store object. Before dismissing the session, though, call the SaveChanges method to persist all pending changes in accordance with the Unit-of-Work pattern:

```
public void Dispose()
{
  if (DocumentSession == null)
    return;
  DocumentSession.SaveChanges();
  DocumentSession.Dispose();
}
```

The API to work with the RavenDB database is fairly simple. Here's the code you'll need to create a new user:

```
public Task CreateAsync(TUser user)
{
  if (user == null)
    throw new ArgumentNullException();

  DocumentSession.Store(user);
  return Task.FromResult<Object>(null);
}
```

To retrieve a given user, use the Query method on the DocumentSession object:

```
var user = DocumentSession.Load<TUser>(id);
```

RavenDB assumes any class you persist has an Id property. If not, it will implicitly create such a property so you can always use the Load method to retrieve any object by Id, whether it's your own

ID or a system-generated ID. To retrieve a user by name, perform a classic query using a LINQ syntax:

```
var user = DocumentSession
             .Query<TUser>()
             .FirstOrDefault(u => u.UserName == userName);
```

Use ToList to select a variety of objects and store them into a manageable list. Dealing with passwords is equally simple. RavenDB stores passwords in a hashed format, but hashing is managed outside the RavenDB module. The SetPasswordHashAsync method, in fact, already receives the password hash the user provides.

**Dino Esposito** *is the co-author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Programming ASP.NET MVC 5" (Microsoft Press, 2014). A technical evangelist for the Microsoft .NET Framework and Android platforms at JetBrains and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents.wordpress.com and on Twitter at twitter.com/despos.*

**Thanks** to the following technical expert for reviewing this article: *Mauro Servienti*

ASP.NET Identity lets you completely unplug the Entity Framework-based storage infrastructure and use RavenDB, a document and schema-free database.

**Figure 5** is the full source code to set up a RavenDB user store compatible with ASP.NET Identity. It's sufficient to log users in and out when you have the embedded version of RavenDB installed. For more sophisticated features such as external logins and account management, you need to implement all ASP.NET Identity user-related interfaces or significantly rework the code in the AccountController you get from scaffolding.

## The Bottom Line

ASP.NET Identity lets you completely unplug the Entity Framework-based storage infrastructure and use RavenDB instead, a document and schema-free database. You can install RavenDB as a Windows service, an IIS application or embedded as I did here. Just write a class that implements a few ASP.NET Identity interfaces and inject this new store class into the UserManager infrastructure. Changing the user type schema is easy, even in the default configuration based on EF. If you use RavenDB, though, you get rid of any migration issues should the user format change. ■



JULIE LERMAN

# A Code First Migrations Mystery: Solved

A friend recently asked me a question about Entity Framework (EF) Code First Migrations that confused me almost as much as it confused him—but for different reasons.

The conversation went something like this:

"Hey Julie, why would my Azure Web site automatically migrate my database?"

"Are you sure you don't have automatic migrations set to true? It's false by default."

"Definitely not. I did not change that setting."

"What about the migrations initializer? That could do it, too."

Let me interrupt here and make sure you truly understand about that initializer. If migrations are explicit, something you create and execute in the Package Manager Console, why is there an initializer?

Migrations can run automatically, meaning that model changes will be discovered and migrations corresponding to changes will be created and executed on the database.

## Code First Initializers

Code First started out with three initializers:

**CreateDatabaseIfNotExists:** With this one, if the database isn't found it will be created, and that's the end of its job. If you modify the database, the database won't be recreated or migrated. You'll simply get an exception suggesting you use a different method to manage the database. Specifically, the suggestion will be to use migrations.

**DropCreateDatabaseAlways:** This one's ideal for integration testing. It will delete the existing database and recreate it from scratch, running any seed logic you've added in. For integration testing, it's a great way to always have a consistent database.

**DropCreateDatabaseIfModelChanges:** Before there were migrations, this was a useful option during development, if you didn't need to retain data or any other schema you might have changed in the database such as adding indexes or triggers.

For completeness, I'd like to also point out that in EF6, we got the NullDatabaseInitializer for those who find it hard to remember (or discover) the other syntax for disabling initialization: SetDatabaseInitialier<MyContext>(null).

Then migrations arrived in EF4.3. Migrations aren't active in your data layer until you explicitly enable them from the Package Manager Console Window by typing enable-migrations. All this really does is add a new folder with a Configuration class to your project. If the enable process discovers an existing database tied to your context, it creates an initial migration class, as well. This is to ensure that when you execute migrations, Code First doesn't attempt to create an already existing database object.

In fact, you'll see an informational message in the console after enabling migrations, as shown in **Figure 1**.

If you look in your database, in the new __MigrationHistory file, you'll see a row in the table reflecting that the initial migration has already been executed on the database.

It's important to understand there is nothing else—no additions to your application config or any other hidden pieces to the migration puzzle. The Configuration class is critical, though.

Here's the default implementation from when I enabled migrations on my data layer, where I have a DbContext class called MigratoryBirdsContext:

```
internal sealed class Configuration :
DbMigrationsConfiguration<MigratoryBirdsContext>
  {
    public Configuration()
    {
      AutomaticMigrationsEnabled = false;
      ContextKey = "DataLayer.MigratoryBirdsContext";
    }

    protected override void Seed(MigratoryBirdsContext context)
    {
    }
  }
```

Because I'll be using this class explicitly further on, I'll change its signature to public. For those of you simply scanning through the code, here's the new declaration:

```
public class Configuration :
DbMigrationsConfiguration<MigratoryBirdsContext>
```

Migrations can run automatically, meaning that model changes will be discovered and migrations corresponding to changes will be created and executed on the database. All of this happens at run time during database initialization. Automatic migrations are handy for simple apps, but you have very little control over them and I typically don't recommend enabling them. I was happy when Code First switched the default to false.

# Working with Files?

30 DAYS FREE TRIAL

## Try Aspose File APIs

## Save Time and Effort

## Creating, Converting, Combining Editing & Printing Files

### Aspose.Words
DOC, DOCX, RTF, HTML, PDF...

### Aspose.Email
MSG, EML, PST, EMLX...

### Aspose.Pdf
PDF, XML, XLS-FO, HTML, BMP...

### Aspose.BarCode
JPG, PNG, BMP, GIF, TIFF, WMF...

### Aspose.Cells
XLS, XLSX, XLSM, XLTX, CSV...

### Aspose.Imaging
PDF, BMP, JPG, GIF, TIFF, PNG..

### Aspose.Slides
PPT, PPTX, POT, POTX, XPS...

### Aspose.Tasks
XML, MPP, SVG, PDF, TIFF, PNG...

... and many more

100% Standalone - No Office Automation

Scan for 20% Savings!

ASPOSE
Your File Format APIs

US: +1 888 277 6734
sales@aspose.com

EU: +44 141 416 1112
sales.europe@aspose.com

AU: +61 2 8003 5926
sales.asiapacific@aspose.com

At run time, Code First looks for a DbMigrationsConfiguration class that's tied to a context when that context goes through its database initialization process. If that class is found, any explicit calls or settings to set database initialization to any of the original three initializers—whose job is to create or drop and create the database—will create the database using the migrations. This is a new pattern for EF6.1. Prior to that, those three initializers would use the initialization logic they had always used since EF4.3, by inferring the database schema from the model and configurations. This change to EF6.1 is convenient for developers who, like me, use the DropCreateDatabaseAlways initializer in their tests. Now it will drop the database but use my migrations to recreate it. If I've customized my migrations classes, I'll get that customization applied to my database, as well.

It's also important to keep in mind that the default initializer—CreateDatabaseIfNotExists—will also use migrations to create a database if the migrations exist but the database doesn't.

So, now I can look at the fourth initializer, MigrateDatabaseToLatestVersion, which is the one I asked my friend about when he was trying to get to the source of the mysterious migrations in his Azure Web site. This initializer has been around since migrations were introduced in EF4.3.

The name, MigrateDatabaseToLatestVersion, seems to suggest it would work much like automatic migrations, but there's a single, significant difference. Automatic migrations are triggered by a change in the model, whereas this initializer's task is triggered by existing migration classes.

> If I've customized my migrations classes, I'll get that customization applied to my database, as well.

To see how this works, let's start with the default state after you've enabled migrations. There's no MigrateDatabaseToLatestVersion initializer in play and AutomaticMigrations is set to false.

Here's a basic integration test that executes a simple query on MigratoryBirdsContext:

```
[TestMethod]
public void CanInitializeDatabase() {
  using (var context = new MigratoryBirdsContext())
  {
    context.MigratoryBirds.ToList();
  }
  Assert.Inconclusive("If we're here, it did not crash");
}
```

If I were to make a change to one of the classes in my model and run this test, an exception would be thrown, telling me, "The model backing the MigratoryBirdsContext context has changed since the database was created. Consider using Code First Migrations to update the database."

> Automatic migrations are handy for simple apps, but you have very little control over them and I typically don't recommend enabling them.

Even though migrations have been enabled and I do have the MigrationsConfiguration class, all that EF knows is that the current model doesn't match the latest version stored in the MigrationHistory table. At this point, I could use add-migration and update-database to fix the problem, but I want to show you how the initializer works, so I'll take a different path: I'll modify the test to enable the initializer. If you're not familiar with its constructor syntax, you have to point to the Configuration file along with the target DbContext, like so:

```
[TestMethod]
public void CanInitializeDatabase()
{
  Database.SetInitializer(
    new MigrateDatabaseToLatestVersion<MigratoryBirdsContext,
    DataLayer.Migrations.Configuration>());
  using (var context = new MigratoryBirdsContext())
  {
    context.MigratoryBirds.ToList();
  }
  Assert.Inconclusive("If we're here, it did not crash");
}
```

Running the test again throws another exception, and its message highlights the difference between this initializer and automatic migrations: "Unable to update database to match the current model because there are pending changes and automatic migration is disabled. Either write the pending model changes to a code-based migration or enable automatic migration."

AutomaticMigrations would note the difference, create the migration and update the database—all on-the-fly and quietly. However, AutomaticMigrations doesn't persist any migrations it creates as a class in your project. Although this initializer notes the change, it doesn't automatically create the migration for you. It can work only with existing migrations.

If you haven't performed migrations manually before, it's a two-step process:

First, you create the migration with the add-migration command. This will read your model and query the database for the latest entry in the migration history

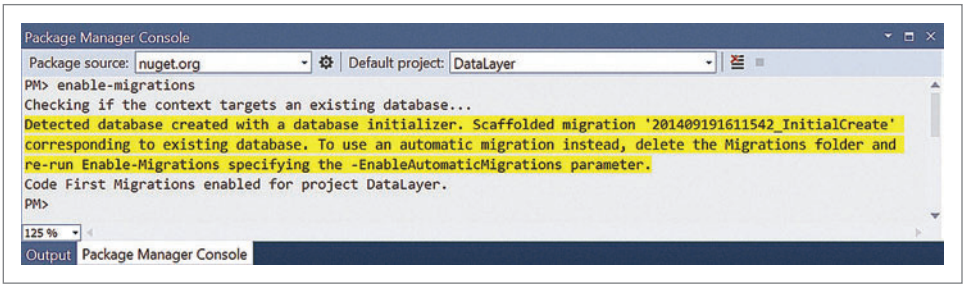
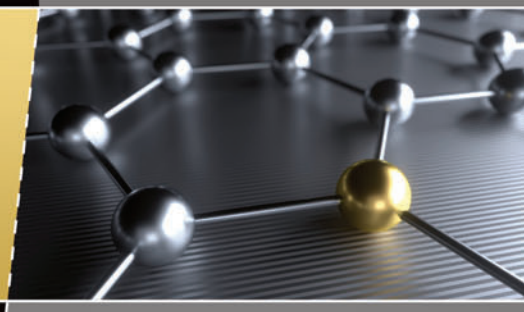



Figure 1: **The Enable-Migrations Response to Existing Database Created by an Initializer**

# Switch to Amyuni PDF



## Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .Net and ActiveX/COM
- New WinRT Component enables publishing C#, C++CX or Javascript apps to Windows Store
- New Postscript/EPS to PDF conversion module

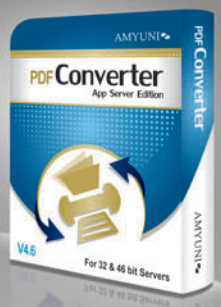

## Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as JPeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment


## Advanced HTML to PDF & XAML

- Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- Easy Integration and deployment within developer's applications
- WebkitPDF is based on the Webkit Open Source library and Amyuni PDF Creator


## High Performance PDF Printer For Desktops and Servers




- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2012 R2 and Windows 8.1
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language
- Easy licensing and deployment to fit system administrator's requirements

CERTIFIED FOR Windows Server 2008 · Windows 7 · Windows 8 · Windows Server 2012 Certified

AMYUNI

**All development tools available at**

# www.amyuni.com


**USA and Canada**
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

**Europe**
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

table, then compare the two. If your model has changed since it was last stored, it will create a new migration class with instructions for migrating the current schema of the database to align with the changes in your model. Next, you execute that migration (or some combination of migrations based on the command syntax) on the database. The base command you type into the Package Manager Console is update-database.

Not many developers realize you can perform the update database programmatically, as well. I knew it was possible, but hadn't done it in so long I actually had to go look for the details in my Code First Migrations course on Pluralsight. The migrations API has a DbMigrator class that lets you update the database with the same options you have access to from the update-database command in the console.

Not many developers realize you can perform the update database programmatically, as well.

The key steps are to instantiate your migration configuration file; instantiate the API DbMigrator class, passing in an instance of your migration Configuration class; and then call Update:

```
var migrator = new DbMigrator(new DbMigrationsConfiguration());
migrator.Update();
```

And this is what the MigrateDatabaseToLatestVersion initializer does on your behalf. The migrations need to already exist in your project (or in your compiled assembly) and the initializer will execute them on the database for you at run time.

With that change I've made to my domain class, I'll now go ahead and add a migration with the command add-migration 'JetStream.' Here's the class that gets generated for me:

```
public partial class JetStream : DbMigration
{
  public override void Up()
  {
    AddColumn("dbo.MigratoryBirds", "DependentOnJetStream", c =>
      c.Boolean(nullable: false));
  }

  public override void Down()
  {
    DropColumn("dbo.MigratoryBirds", "DependentOnJetStream");
  }
}
```

Now this class is part of my project. When I run the test again, where I've set up the migrate-database initializer, the test doesn't throw an exception. And my profiler shows me that prior to the SELECT being executed, the database table schema is altered to add in the new DependentOnJetStream property.

What I'm not getting with this pattern that I liked about the DropCreateDatabaseAlways initializer is a fresh version of the database for integration tests. To achieve this, I have to execute some raw SQL in the Seed method of the configuration file prior to seeding the database. Or, for the integration tests, I could simply disable initialization altogether and control the database creation explicitly. Eric Hexter wrote a great blog post about this, "Using SQL Compact for Integration Tests with Entity Framework" (bit.ly/1qgh9Ps).

## Where MigrateDatabaseToLatestVersion Shines

If you were to compile the project and give it to another developer who hasn't migrated her own development database, the initializer would trigger the app to look for a migration. The migration will be found because it's in the project. The initializer will check the database's migration history table and update the database schema, if necessary.

Considering this in production, not development, the migration class will be compiled into your assembly and deployed as part of the application. When the application is run, it will respond to a MigrateDatabaseToLatestVersion setting (in the appropriate place in your application) and update the database schema accordingly.

## So What Happened in the Azure Web Site?

Now that you have a good understanding of how MigrateDatabaseToLatestVersion works, let's get back to the conversation with my friend. We left off at: "What about the migrations initializer? That would do it, too."

He replied that he hadn't set the migrations initializer anywhere in the code. So now it was a big mystery. The behavior matched that of MigrateDatabaseToLatestVersion exactly. He deployed his app to Azure. When he ran it, the app noticed the change he had made in the model before redeploying and it updated the database schema.

With all of these questions asked and answered, leaving me a bit mystified, I went to my favorite search engine with some useful information in hand and made a discovery.

When you publish a Web app to Azure, there's a checkbox on the settings page that's unchecked by default. But that checkbox says "Execute code first migrations (runs on application start)." Checking this will add the MigrateDatabaseToLatestVersion setting declaratively in the config file.

I e-mailed my friend and showed him a screenshot of the page with that setting and asked, "Look familiar?" Mystery solved. He had created the migrations and updated his development database. So the migrations were compiled into his assembly when he deployed it. And he remembered ticking that checkbox.

So when he ran his Web site for the first time, the initializer called DbMigrator.Update, which compared the list of migrations in the assembly to the list in the MigrationsHistory table and ran any that weren't present in the table.

Now he recognizes that as a feature, not a mystery. ■

**Julie Lerman** *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010), as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Rowan Miller*


GROUPDOCS
Your Document Collaboration APIs

30 days free trial

# Your Document Collaboration APIs

Professional APIs that allow developers to empower their apps with document collaboration capabilities.

## GroupDocs.Viewer

Native-text, high-fidelity HTML5 document viewer with support for over 50 file formats.

## GroupDocs.Signature

Electronic signature API that allows users to place signatures on documents within your apps.

## GroupDocs.Annotation

A powerful API that lets developers annotate Microsoft Office, PDF and other documents within their apps.

## GroupDocs.Comparison

A diff view API that allows end users to quickly find differences between two revisions of a document.

## GroupDocs.Conversion

Universal document converter for fast conversion between more than 50 file formats.

## GroupDocs.Assembly

Merges data entered by users through online forms Into both Microsoft Office and PDF documents.

## 100% Standalone - No Office Automation

.NET Libraries | Java Libraries | Cloud APIs | Cloud Apps

SALES INQUIRIES
+1 214 329 9760
sales@groupdocs.com
groupdocs.com

# Unit Testing Asynchronous Code

Stephen Cleary

Unit testing is a cornerstone of modern development. The benefits of unit testing for a project are pretty well understood: Unit testing decreases the number of bugs, reduces time to market and discourages overly coupled design. Those are all nice benefits, but there are further advantages more directly relevant to developers. When I write unit tests, I can have much greater confidence in the code. It's easier to add features or fix bugs in tested code, because the unit tests act as a safety net while the code is changing.

Writing unit tests for asynchronous code brings a few unique challenges. Furthermore, the current state of async support in unit test and mocking frameworks varies and is still evolving. This article will consider MSTest, NUnit and xUnit, but the general principles apply to any unit testing framework. Most of the examples in this article will use MSTest syntax, but I'll point out any differences in behavior along the way. The code download contains examples for all three frameworks.

Before diving into the specifics, I'll briefly review a conceptual model of how the async and await keywords work.

This article discusses:

- How the async and await keywords work
- Unit tests that pass incorrectly
- Avoiding async void
- Async task unit tests
- Testing exceptions
- Async stubs and mocks

Technologies discussed:

MSTest, NUnit, xUnit, Moq

Code download available at:

msdn.microsoft.com/magazine/msdnmag1114

## Async and Await in a Nutshell

The async keyword does two things: it enables the await keyword within that method, and it transforms the method into a state machine (similar to how the yield keyword transforms iterator blocks into state machines). Async methods should return Task or Task<T> when possible. It's permissible for an async method to return void, but it's not recommended because it's very difficult to consume (or test) an async void method.

The task instance returned from an async method is managed by the state machine. The state machine will create the task instance to return, and will later complete that task.

An async method begins executing synchronously. It's only when the async method reaches an await operator that the method may become asynchronous. The await operator takes a single argument, an "awaitable" such as a Task instance. First, the await operator will check the awaitable to see if it has already completed; if it has, the method continues (synchronously). If the awaitable isn't yet complete, the await operator will "pause" the method and resume when the awaitable completes. The second thing the await operator does is retrieve any results from the awaitable, raising exceptions if the awaitable completed with an error.

The Task or Task<T> returned by the async method conceptually represents the execution of that method. The task will complete when the method completes. If the method returns a value, the task is completed with that value as its result. If the method throws an exception (and doesn't catch it), then the task is completed with that exception.

There are two immediate lessons to draw from this brief overview. First, when testing the results of an asynchronous method, the important bit is the Task it returns. The async method uses its Task to report completion, results and exceptions. The second lesson is that the await operator has special behavior when its awaitable is already complete. I'll discuss this later when considering asynchronous stubs.

Figure 1 **Testing Exceptions with Synchronous Test Methods**

```
// Old style; only works on desktop.
[TestMethod]
[ExpectedException(typeof(Exception))]
public void ExampleExpectedExceptionTest()
{
  SystemUnderTest.Fail();
}

// New style; only works on Windows Store.
[TestMethod]
public void ExampleThrowsExceptionTest()
{
  var ex = Assert.ThrowsException<Exception>(() => { SystemUnderTest.Fail(); });
}
```

## The Incorrectly Passing Unit Test

In free-market economics, losses are just as important as profits; it's the *failures* of companies that force them to produce what people will buy and encourage optimum resource allocation within the system as a whole. Similarly, the failures of unit tests are just as important as their successes. You must be sure the unit test will fail when it should, or its success won't mean anything.

A unit test that's supposed to fail will (incorrectly) succeed when it's testing the wrong thing. This is why test-driven development (TDD) makes heavy use of the red/green/refactor loop: the "red" part of the loop ensures the unit test will fail when the code is incorrect. At first, testing code that you know to be wrong sounds ludicrous, but it's actually quite important because you must be sure the tests will fail when they need to. The red part of the TDD loop is actually testing the tests.

With this in mind, consider the following asynchronous method to test:

```
public sealed class SystemUnderTest
{
  public static async Task SimpleAsync()
  {
    await Task.Delay(10);
  }
}
```

Newcomers to async unit testing will often make a test like this as a first attempt:

```
// Warning: bad code!
[TestMethod]
public void IncorrectlyPassingTest()
{
  SystemUnderTest.SimpleAsync();
}
```

Unfortunately, this unit test doesn't actually test the asynchronous method correctly. If I modify the code under test to fail, the unit test will still pass:

```
public sealed class SystemUnderTest
{
  public static async Task SimpleAsync()
  {
    await Task.Delay(10);
    throw new Exception("Should fail.");
  }
}
```

This illustrates the first lesson from the async/await conceptual model: To test an asynchronous method's behavior, you must observe the task it returns. The best way to do this is to await the task returned from the method under test. This example also illustrates the benefit of the red/green/refactor testing development cycle; you must ensure the tests will fail when the code under test fails.

Most modern unit test frameworks support Task-returning asynchronous unit tests. The IncorrectlyPassingTest method will cause compiler warning CS4014, which recommends using await to consume the task returned from SimpleAsync. When the unit test method is changed to await the task, the most natural approach is to change the test method to be an async Task method. This ensures the test method will (correctly) fail:

```
[TestMethod]
public async Task CorrectlyFailingTest()
{
  await SystemUnderTest.FailAsync();
}
```

## Avoiding Async Void Unit Tests

Experienced users of async know to avoid async void. I described the problems with async void in my March 2013 article, "Best Practices in Asynchronous Programming" (bit.ly/1ulDCil). Async void unit test methods don't provide an easy way for their unit test framework to retrieve the results of the test. In spite of this difficulty, some unit test frameworks do support async void unit tests by providing their own SynchronizationContext in which its unit tests are executed.

Providing a SynchronizationContext is somewhat controversial, because it does change the environment in which the tests run. In particular, when an async method awaits a task, by default it will resume that async method on the current SynchronizationContext. So the presence or absence of a SynchronizationContext will indirectly change the behavior of the system under test. If you're curious about the details of SynchronizationContext, see my *MSDN Magazine* article on the subject at bit.ly/1hlar1p.

MSTest doesn't provide a SynchronizationContext. In fact, when MSBuild is discovering tests in a project that uses async void unit tests, it will detect this and issue warning UTA007, notifying the user that the unit test method should return Task instead of void. MSBuild won't run async void unit tests.

NUnit does support async void unit tests, as of version 2.6.2. The next major update of NUnit, version 2.9.6, supports async void unit tests, but the developers have already decided to remove support in version 2.9.7. NUnit provides a SynchronizationContext only for async void unit tests.

As of this writing, xUnit is planning to add support for async void unit tests with version 2.0.0. Unlike NUnit, xUnit provides a SynchronizationContext for all of its test methods, even synchronous ones. However, with MSTest not supporting async void unit tests, and with NUnit reversing its earlier decision and removing

Figure 2 **Brittle NUnit Exception Testing**

```
[Test]
public void FailureTest_AssertThrows()
{
  // This works, though it actually implements a nested loop,
  // synchronously blocking the Assert.Throws call until the asynchronous
  // FailAsync call completes.
  Assert.Throws<Exception>(async () => await SystemUnderTest.FailAsync());
}

// Does NOT pass.
[Test]
public void BadFailureTest_AssertThrows()
{
  Assert.Throws<Exception>(() => SystemUnderTest.FailAsync());
}
```

Figure 3 **The AssertEx Class for Testing Exceptions Asynchronously**

```
using System;
using System.Threading.Tasks;

public static class AssertEx
{
  public static async Task<TException> ThrowsAsync<TException>(Func<Task> action,
    bool allowDerivedTypes = true) where TException : Exception
  {
    try
    {
      await action();
    }
    catch (Exception ex)
    {
      if (allowDerivedTypes && !(ex is TException))
        throw new Exception("Delegate threw exception of type " +
          ex.GetType().Name + ", but " + typeof(TException).Name +
          " or a derived type was expected.", ex);
      if (!allowDerivedTypes && ex.GetType() != typeof(TException))
        throw new Exception("Delegate threw exception of type " +
          ex.GetType().Name + ", but " + typeof(TException).Name +
          " was expected.", ex);
      return (TException)ex;
    }
    throw new Exception("Delegate did not throw expected exception " +
      typeof(TException).Name + ".");
  }

  public static Task<Exception> ThrowsAsync(Func<Task> action)
  {
    return ThrowsAsync<Exception>(action, true);
  }
}
```

support, I wouldn't be surprised if xUnit also chooses to drop async void unit test support before version 2 is released.

The bottom line is that async void unit tests are complicated for frameworks to support, require changes in the test execution environment, and bring no benefit over async Task unit tests. Moreover, support for async void unit tests varies across frameworks, and even framework versions. For these reasons, it's best to avoid async void unit tests.

## Async Task Unit Tests

Async unit tests that return Task have none of the problems of async unit tests that return void. Async unit tests that return Task enjoy wide support from almost all unit test frameworks. MSTest added support in Visual Studio 2012, NUnit in versions 2.6.2 and 2.9.6, and xUnit in version 1.9. So, as long as your unit testing framework is less than 3 years old, async task unit tests should just work.

Figure 4 **Using an Asynchronous Method from an Interface**

```
public interface IMyService
{
  Task<int> GetAsync();
}

public sealed class SystemUnderTest
{
  private readonly IMyService _service;

  public SystemUnderTest(IMyService service)
  {
    _service = service;
  }

  public async Task<int> RetrieveValueAsync()
  {
    return 42 + await _service.GetAsync();
  }
}
```

Unfortunately, outdated unit test frameworks don't understand async task unit tests. As of this writing, there's one major platform that doesn't support them: Xamarin. Xamarin uses a customized older version of NUnitLite, and it doesn't currently support async task unit tests. I expect that support will be added in the near future. In the meantime, I use a workaround that's inefficient but works: Execute the async test logic on a different thread pool thread, and then (synchronously) block the unit test method until the actual test completes. The workaround code uses GetAwaiter().GetResult() instead of Wait because Wait will wrap any exceptions inside an AggregateException:

```
[Test]
public void XamarinExampleTest()
{
  // This workaround is necessary on Xamarin,
  // which doesn't support async unit test methods.
  Task.Run(async () =>
  {
    // Actual test code here.
  }).GetAwaiter().GetResult();
}
```

## Testing Exceptions

When testing, it's natural to test the successful scenario; for example, a user can update his own profile. However, testing exceptions is also very important; for example, a user shouldn't be able to update someone else's profile. Exceptions are part of an API surface just as much as method parameters are. Therefore, it's important to have unit tests for code when it's expected to fail.

Originally, the ExpectedExceptionAttribute was placed on a unit test method to indicate that the unit test was expected to fail. However, there were a few problems with ExpectedExceptionAttribute. The first was that it could only expect a unit test to fail as a whole; there was no way to indicate that only a particular part of the test was expected to fail. This isn't a problem with very simple tests, but can have misleading results when the tests grow longer. The second problem with ExpectedExceptionAttribute is that it's limited to checking the type of the exception; there's no way to check other attributes, such as error codes or messages.

For these reasons, in recent years there has been a shift toward using something more like Assert.ThrowsException, which takes the important part of the code as a delegate and returns the exception that was thrown. This solves the shortcomings of ExpectedExceptionAttribute. The desktop MSTest framework supports only ExpectedExceptionAttribute, while the newer MSTest framework used for Windows Store unit test projects supports only Assert.ThrowsException. xUnit supports only Assert.Throws, and NUnit supports both approaches. **Figure 1** is an example of both kinds of tests, using MSTest syntax.

But what about asynchronous code? Async task unit tests work perfectly well with ExpectedExceptionAttribute on both MSTest and NUnit (xUnit doesn't support ExpectedExceptionAttribute at all). However, the support for an async-ready ThrowsException is less uniform. MSTest does support an async ThrowsException, but only for Windows Store unit test projects. xUnit has introduced an async ThrowsAsync in the prerelease builds of xUnit 2.0.0.

NUnit is more complex. As of this writing, NUnit supports asynchronous code in its verification methods such as Assert.Throws. However, in order to get this to work, NUnit provides a SynchronizationContext, which introduces the same problems


# **Desktop.** Web. Mobile. Your next great app starts here.

From interactive Desktop applications, to immersive Web and Mobile solutions, development tools built to meet your needs today and ensure your continued success tomorrow.

Download your free 30-day trial today and Experience the DevExpress Difference.

www.devexpress.com/try

DevExpress®

WIN ASP WPF SL VCL XAF CR

as async void unit tests. Also, the syntax is currently brittle, as the example in **Figure 2** shows. NUnit is already planning to drop support for async void unit tests, and I wouldn't be surprised if this support is dropped at the same time. In summary: I recommend you do not use this approach.

So, the current support for an async-ready ThrowsException/Throws isn't great. In my own unit testing code, I use a type very similar to the AssertEx in **Figure 3**. This type is rather simplistic in that it just throws bare Exception objects instead of doing assertions, but this same code works in all major unit testing frameworks.

This allows async task unit tests to use a more modern ThrowsAsync instead of the ExpectedExceptionAttribute, like this:

```
[TestMethod]
public async Task FailureTest_AssertEx()
{
  var ex = await AssertEx.ThrowsAsync(() => SystemUnderTest.FailAsync());
}
```

## Async Stubs and Mocks

In my opinion, only the simplest of code can be tested without some kind of stub, mock, fake or other such device. In this introductory article, I'll just refer to all of these testing assistants as mocks. When using mocks, it's helpful to program to interfaces rather than implementations. Asynchronous methods work perfectly well with interfaces; the code in **Figure 4** shows how code can consume an interface with an asynchronous method.

Figure 5 **Stub Implementations for Asynchronous Code**

```
[TestMethod]
public async Task RetrieveValue_SynchronousSuccess_Adds42()
{
  var service = new Mock<IMyService>();
  service.Setup(x => x.GetAsync()).Returns(() => Task.FromResult(5));
  // Or: service.Setup(x => x.GetAsync()).ReturnsAsync(5);
  var system = new SystemUnderTest(service.Object);

  var result = await system.RetrieveValueAsync();

  Assert.AreEqual(47, result);
}

[TestMethod]
public async Task RetrieveValue_AsynchronousSuccess_Adds42()
{
  var service = new Mock<IMyService>();
  service.Setup(x => x.GetAsync()).Returns(async () =>
  {
    await Task.Yield();
    return 5;
  });
  var system = new SystemUnderTest(service.Object);

  var result = await system.RetrieveValueAsync();

  Assert.AreEqual(47, result);
}

[TestMethod]
public async Task RetrieveValue_AsynchronousFailure_Throws()
{
  var service = new Mock<IMyService>();
  service.Setup(x => x.GetAsync()).Returns(async () =>
  {
    await Task.Yield();
    throw new Exception();
  });
  var system = new SystemUnderTest(service.Object);

  await AssertEx.ThrowsAsync(system.RetrieveValueAsync);
}
```

With this code, it's easy enough to create a test implementation of the interface and pass it to the system under test. **Figure 5** shows how to test the three major stub cases: asynchronous success, asynchronous failure and synchronous success. Asynchronous success and failure are the primary two scenarios for testing asynchronous code, but it's also important to test the synchronous case. This is because the await operator behaves differently if its awaitable is already completed. The code in **Figure 5** uses the Moq mocking framework to generate the stub implementations.

Speaking of mocking frameworks, there's a bit of support they can give to asynchronous unit testing, as well. Consider for a moment what the default behavior of a method should be, if no behavior was specified. Some mocking frameworks (such as Microsoft Stubs) will default to throwing an exception; others (such as Moq) will return a default value. When an asynchronous method returns a Task<T>, a naïve default behavior would be to return default(Task<T>), in other words, a null task, which will cause a NullReferenceException.

This behavior is undesirable. A more reasonable default behavior for asynchronous methods would be to return Task.FromResult-(default(T))—that is, a task that's completed with the default value of T. This enables the system under test to use the returned task. Moq implemented this style of default behavior for asynchronous methods in Moq version 4.2. To my knowledge, as of this writing, it's the only mocking library that uses async-friendly defaults like that.

## Wrapping Up

Async and await have been around since the introduction of Visual Studio 2012, long enough for some best practices to emerge. Unit test frameworks and helper components such as mocking libraries are converging toward consistent async-friendly support. Asynchronous unit testing today is already a reality, and it will get even better in the future. If you haven't done so recently, now is a good time to update your unit test frameworks and mocking libraries to ensure you have the best async support.

Unit test frameworks are converging away from async void unit tests and toward async task unit tests. If you have any async void unit tests, I recommend you change them today to async task unit tests.

I expect over the next couple years you'll see much better support for testing failure cases in async unit tests. Until your unit test framework has good support, I suggest you use the AssertEx type mentioned in this article, or something similar that's more tailored to your particular unit test framework.

Proper asynchronous unit testing is an important part of the async story, and I'm excited to see these frameworks and libraries adopt async. One of my first lightning talks was about async unit testing a few years ago when async was still in community technology preview, and it's so much easier to do these days! ■

**Stephen Cleary** *is a husband, father and programmer living in northern Michigan. He has worked with multithreading and asynchronous programming for 16 years and has used async support in the Microsoft .NET Framework since the first community technology preview. He is the author of "Concurrency in C# Cookbook" (O'Reilly Media, 2014). His homepage, including his blog, is at stephencleary.com.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *James McCaffrey*

# Bring your XML development projects to light with the complete set of tools from Altova®

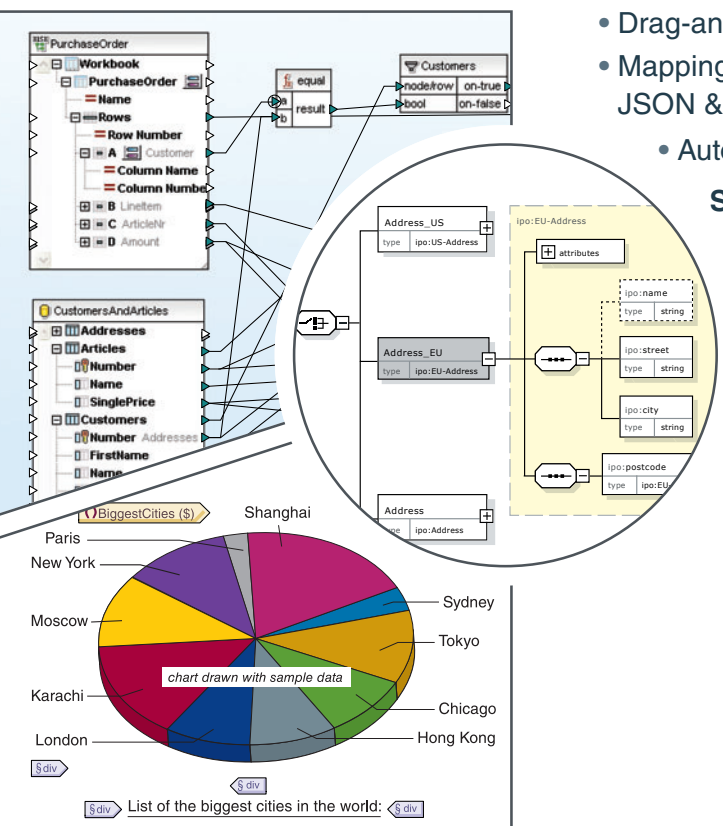
ALTOVA® MissionKit®

**Experience how Altova MissionKit®, a software development suite of industrial-strength XML, SQL, and data integration tools, can simplify even the most advanced XML development projects.**

POWERED BY RAPTORXML®

## NEW in Version 2015:

- Sophisticated XML file modifications via XQuery Update Facility in XMLSpy
- New tools for XBRL Table and XBRL Formula editing in XMLSpy
- JSON data mapping in MapForce
- Option to generate HTML fragments in StyleVision
- And much more




## Altova MissionKit includes multiple, tightly-integrated XML tools:

**XMLSpy®** – industry-leading XML editor

- Support for XML Schema 1.1 and XPath/XSLT/XQuery 3.0
- Industry's strongest validation engine with Smart Fix
- Powered by RaptorXML® for lightning-fast validation & processing
- Graphical editing views, powerful debuggers, code generation, & more

**MapForce®** – any-to-any data mapping & integration tool

- Drag-and-drop data conversion
- Mapping of XML, DBs, EDI, XBRL, flat files, Excel®, JSON & Web services
- Automation via MapForce Server

**StyleVision®** – visual XSLT stylesheet & report designer

- Graphical XSLT stylesheet & report design for XML, XBRL, & SQL databases
- Output to HTML, PDF, Word & more
- Automation via StyleVision Server

Download a 30 day free trial!

Try before you buy with a free, fully functional trial from **www.altova.com**

# Unit Testing Asynchronous Code: Three Solutions for Better Tests

Sven Grand

Asynchronous programming has become more and more important during the last decade. Whether for CPU-based parallelism or IO-based concurrency, developers are employing asynchrony to help make the most of the resources available and, ultimately, do more with less. More responsive client applications and more scalable server applications are all within reach.

Software developers have learned a lot of design patterns for effectively building synchronous functionality, but best practices for designing asynchronous software are relatively new, though the support provided by programming languages and libraries for parallel and concurrent programming has dramatically improved with the release of the Microsoft .NET Framework 4 and 4.5. While there's already quite a lot of good advice for using the new techniques (see "Best Practices in Asynchronous Programming" at bit.ly/1ulDCil and "Talk: Async Best Practices" at bit.ly/1DsFuMi), best practices for designing the internal and external APIs for applications and libraries with language features like async and await and the Task Parallel Library (TPL) are still unknown to many developers.

This article discusses:

- Separating the functionality from the asynchronous aspects of a program via a "Humble Object"
- Synchronizing the test with the completion of the tested thread
- Executing operations on the same thread as the test itself

Technologies discussed:

Microsoft .NET Framework 4 and 4.5, Task Parallel Library, Async and Await

Code download available at:

msdn.microsoft.com/magazine/msdnmag1114

This gap affects not just the performance and reliability of the applications and libraries such developers are building, but also the testability of their solutions, as many of the best practices that enable the creation of robust asynchronous designs enable easier unit testing, as well.

With such best practices in mind, this article will present ways to design and refactor code for better testability, and demonstrate how this will influence the tests. The solutions are applicable to code that takes advantage of async and await, as well as code based on lower-level multithreading mechanisms from earlier frameworks and libraries. And, in the process, the solutions will not only be

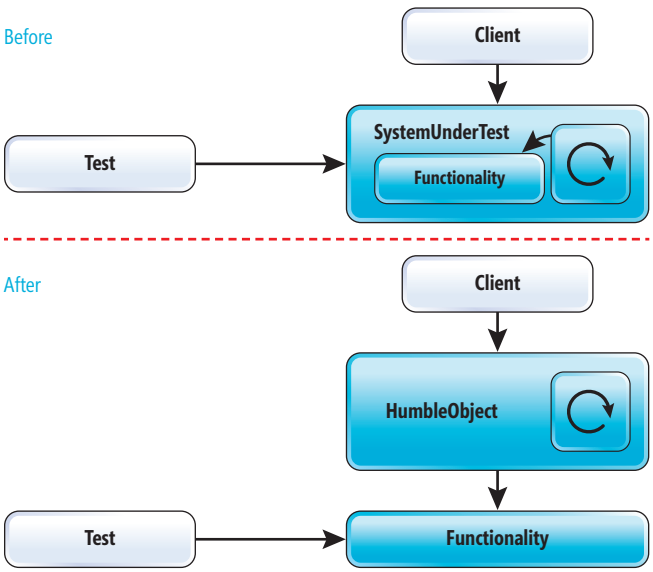



Figure 1 **The Humble Object Pattern**

better factored for testing, they'll be more easily and more efficiently consumable by users of the developed code.

The team I work with is developing software for medical X-ray devices. In this domain it's critical our unit-test coverage is always at a high level. Recently, a developer asked me, "You're always pushing us to write unit tests for all our code. But how can I write reasonable unit tests when my code starts another thread or is using a timer that later starts a thread and runs it several times?"

That is a meaningful question. Suppose I have this code to be tested:

```
public void StartAsynchronousOperation()
{
  Message = "Init";
  Task.Run(() =>
    {
      Thread.Sleep(1900);
      Message += " Work";
    });
}
public string Message { get; private set; }
```

My first attempt to write a test for that code was not very promising:

```
[Test]
public void FragileAndSlowTest()
{
  var sut = new SystemUnderTest();
  // Operation to be tested will run in another thread.
  sut. StartAsynchronousOperation();
  // Bad idea: Hoping that the other thread finishes execution after 2 seconds.
  Thread.Sleep(2000);
  // Assert outcome of the thread.
  Assert.AreEqual("Init Work", sut.Message);
}
```

I expect unit tests to run fast and deliver predictable results, but the test I wrote was fragile and slow. The StartAsynchronousOperation method kicks off the operation to be tested in another thread, and the test should check the outcome of the operation. The time it takes to start a new thread or to initialize an existing thread sitting in the thread pool and to execute the operation isn't predictable, because it depends on other processes running on the test machine. The test may fail from time to time when the sleep is too short and the asynchronous operation hasn't yet finished. I'm between the devil and the deep blue sea: Either I try to keep the waiting time as short as possible, with the risk of a fragile test, or I increase the sleep time to make the test more robust, but slow down the test even more.

The problem is similar when I want to test code that uses a timer:

```
private System.Threading.Timer timer;
private readonly Object lockObject = new Object();

public void StartRecurring()
{
  Message = "Init";
  // Set up timer with 1 sec delay and 1 sec interval.
  timer = new Timer(o => { lock(lockObject){ Message += " Poll";} }, null,
    new TimeSpan(0, 0, 0, 1), new TimeSpan(0, 0, 0, 1));
}
public string Message { get; private set; }
```

And this test is likely to have the same problems:

```
[Test]
public void FragileAndSlowTestWithTimer()
{
  var sut = new SystemUnderTest();
  // Execute code to set up timer with 1 sec delay and interval.
  sut.StartRecurring();
  // Bad idea: Wait for timer to trigger three times.
  Thread.Sleep(3100);
  // Assert outcome.
  Assert.AreEqual("Init Poll Poll Poll", sut.Message);
}
```

When I test a reasonably complex operation with several different code branches, I end up with a huge number of separate tests. My test suites get slower and slower with each new test. The maintenance costs for these test suites increase, because I have to spend time investigating the sporadic failures. Moreover, slow test suites tend to be executed only infrequently and therefore have fewer benefits. At some point, I'd probably stop executing these slow and intermittently failing tests altogether.

The two kinds of tests shown earlier are also unable to detect when the operation throws an exception. Because the operation runs in a different thread, exceptions aren't propagated to the test runner thread. This limits the ability of tests to check the correct error behavior of the code under test.

I'm going to present three general solutions for avoiding slow, fragile unit tests by improving the design of the tested code, and I'll show how this enables unit tests to check exceptions. Each solution has advantages, as well as disadvantages or limitations. At the end, I'll give some recommendations regarding which solution to choose for different situations.

This article is about unit testing. Typically, unit tests are testing code in isolation from other parts of the system. One of those other parts of the system is the OS multithreading capability. Standard library classes and methods are used to schedule asynchronous work, but the multithreading aspect should be excluded in unit tests, which should concentrate on the functionality that runs asynchronously.

Unit tests for asynchronous code make sense when the code contains chunks of functionality running in one thread, and the unit tests should verify that the chunks are working as expected. When the unit tests have shown that the functionality is correct, it makes sense to use additional test strategies to uncover concurrency problems. There are several approaches for testing and analyzing multithreaded code to find these kinds of problems (see, for example, "Tools and Techniques to Identify Concurrency Issues," at bit.ly/1tVjpII). Stress tests, for example, can put a whole system or a large part of the system under load. These strategies are sensible to complement unit tests, but are out of scope for this article. The solutions in this article will show how to exclude the multithreading parts while testing the functionality in isolation with unit tests.

## Solution 1: Separate Functionality from Multithreading

The simplest solution for unit testing the functionality involved in asynchronous operations is to separate that functionality from multithreading. Gerard Mezaros has described this approach in the Humble Object pattern in his book, "xUnit Test Patterns" (Addison-Wesley, 2007). The functionality to be tested is extracted into a separate new class and the multithreading part stays within the Humble Object, which calls the new class (see **Figure 1**).

The following code shows the extracted functionality after refactoring, which is purely synchronous code:

```
public class Functionality
{
  public void Init()
  {
    Message = "Init";
  }
  public void Do()
  {
    Message += " Work";
  }
  public string Message { get; private set; }
}
```

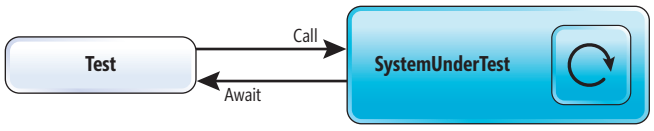



Figure 2 **Synchronization via Async and Await**

Before refactoring, the functionality was mixed with asynchronous code, but I've moved it to the Functionality class. This class can now be tested via simple unit tests because it doesn't contain any multithreading anymore. Note that such refactoring is important in general, not just for unit testing: components shouldn't expose asynchronous wrappers for inherently synchronous operations and should instead leave it up to the caller to determine whether to offload the invocation of that operation. In the case of a unit test, I choose not to, but the consuming application might decide to do so, for reasons of either responsiveness or parallel execution. For more information, see Stephen Toub's blog post, "Should I Expose Asynchronous Wrappers for Synchronous Methods?" (bit.ly/1shQPfn).

In a lightweight variation of this pattern, certain private methods of the SystemUnderTest class can be made public to allow tests to call these methods directly. In this case, no additional class needs to be created for testing the functionality without multithreading.

Separating the functionality via the Humble Object pattern is simple and can be done not only for code that immediately schedules asynchronous work once, but also for code that uses timers. In that case the timer handling is kept in the Humble Object and the recurring operation is moved to the Functionality class or a public method. An advantage of this solution is that tests can directly check exceptions thrown by the code under test. The Humble Object pattern can be applied regardless of the techniques used to schedule asynchronous work. The drawbacks of this solution are that the code in the Humble Object itself isn't tested and that the code under test has to be modified.

## Solution 2: Synchronize Tests

If the test is able to detect the completion of the operation under test, which runs asynchronously, it can avoid the two disadvantages, fragility and slowness. Though the test runs multithreaded code, it can be reliable and fast when the test synchronizes with the operation scheduled by the code under test. The test can concentrate on the functionality while the negative effects of the asynchronous execution are minimized.

In the best case, the method under test returns an instance of a type that will be signaled when the operation has completed. The

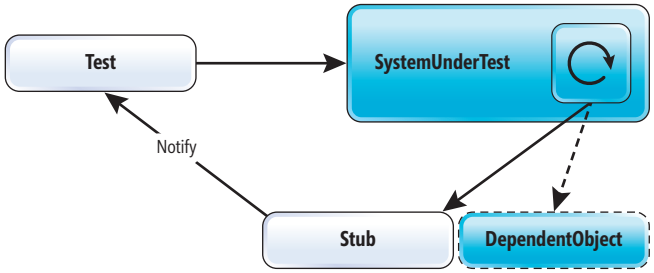



Figure 3 **Synchronization via a Dependent Object Stub**

Task type, which has been available in the .NET Framework since version 4, meets this need nicely, and the async/await feature available since the .NET Framework 4.5 makes it easy to compose Tasks:

```
public async Task DoWorkAsync()
{
  Message = "Init";
  await Task.Run( async() =>
  {
    await Task.Delay(1900);
    Message += " Work";
  });
}
public string Message { get; private set; }
```

This refactoring represents a general best practice that helps in both the unit testing case and in general consumption of the exposed asynchronous functionality. By returning a Task that represents the asynchronous operation, a consumer of the code is able to easily determine when the asynchronous operation has completed, whether it failed with an exception, and whether it returned a result.

This makes unit testing an asynchronous method as simple as unit testing a synchronous method. It's now easy for the test to synchronize with the code under test simply by invoking the target method and waiting for the returned Task to complete. This waiting can be done synchronously (blocking the calling thread) via the Task Wait methods, or it can be done asynchronously (using continuations to avoid blocking the calling thread) with the await keyword, before checking the outcome of the asynchronous operation (see **Figure 2**).

In order to use await in a unit test method, the test itself has to be declared with async in its signature. No sleep statement is needed anymore:

```
[Test]
public async Task SynchronizeTestWithCodeViaAwait()
{
  var sut = new SystemUnderTest();
  // Schedule operation to run asynchronously and wait until it is finished.
  await sut.StartAsync();
  // Assert outcome of the operation.
  Assert.AreEqual("Init Work", sut.Message);
}
```

Luckily, the latest versions of the major unit test frameworks—MSTest, xUnit.net and NUnit—support the async and await tests (see Stephen Cleary's blog at bit.ly/1x18mta). Their test runners can cope with async Task tests and await the completion of the thread before they start to evaluate the assert statements. If the test runner of the unit testing framework can't cope with async Task test method signatures, the test can at least call the Wait method on the Task returned from the system under test.

In addition, the timer-based functionality can be enhanced with the help of the TaskCompletionSource class (see details in the code download). The test can then await the completion of specific recurring operations:

```
[Test]
public async Task SynchronizeTestWithRecurringOperationViaAwait()
{
  var sut = new SystemUnderTest();
  // Execute code to set up timer with 1 sec delay and interval.
  var firstNotification = sut.StartRecurring();
  // Wait that operation has finished two times.
  var secondNotification = await firstNotification.GetNext();
  await secondNotification.GetNext();
  // Assert outcome.
  Assert.AreEqual("Init Poll Poll", sut.Message);
}
```

BEST SELLER

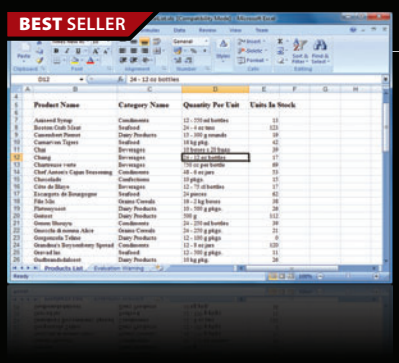
## Help & Manual Professional | from $583.10

ec software

**Easily create documentation for Windows, the Web and iPad.**

- Powerful features in an easy accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to HTML, WebHelp, CHM, PDF, ePUB, RTF, e-book or print
- Styles and Templates give you full design control

BEST SELLER



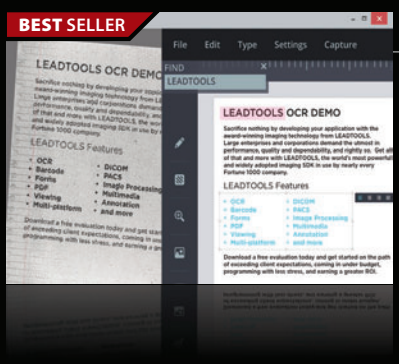
## Aspose.Total for .NET | from $2,449.02

ASPOSE
Your File Format APIs

**Every Aspose .NET component in one package.**

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR and OneNote files alongside many more document management features in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types

BEST SELLER




## LEADTOOLS Document Imaging SDKs V18 | from $2,695.50

LEAD TECHNOLOGIES

**Add powerful document imaging functionality to desktop, tablet, mobile & web applications.**

- Comprehensive document image cleanup, preprocessing, viewer controls and annotations
- Fast and accurate OCR, OMR, ICR and Forms Recognition with multi-threading support
- PDF & PDF/A Read / Write / Extract / View / Edit
- Barcode Detect / Read / Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-Footprint HTML5/JavaScript Controls & Native WinRT Libraries for Windows Store

BEST SELLER


## ComponentOne Studio Enterprise 2014 v2 | from $1,315.60

ComponentOne
a division of GrapeCity

**.NET Tools for the Professional Developer: Windows, HTML5/Web, and XAML.**

- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Includes 40+ UI widgets built with HTML5, jQuery, CSS3 and SVG
- New Sparkline control for HTML5 and Web Forms and options for exporting Data views
- 40+ Windows 8.1 & Windows Phone 8.1 (beta) controls and Universal Windows app support
- All Microsoft platforms supported, Visual Studio 2013, ASP.NET, WinForms, WPF & more



We accept purchase orders.
Contact us to apply for a credit account.

**US Headquarters**
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

**European Headquarters**
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

**Asia / Pacific Headquarters**
ComponentSource
3F Kojimachi Square Bldg
3-3 Kojimachi Chiyoda-ku
Tokyo
Japan
102-0083

Sales Hotline - US & Canada:
(888) 850-9911
www.componentsource.com

VISA DISCOVER

GSA Schedule
Contract GS-35F-0188R

Unfortunately, sometimes the code under test can't use async and await, such as when you're testing code that has already shipped and for breaking-change reasons the signature of the method being tested can't be changed. In situations like this, the synchronization has to be implemented with other techniques. Synchronization can be realized if the class under test invokes an event or calls a dependent object when the operation is completed. The following example demonstrates how to implement the test when a dependent object is called:

```
private readonly ISomeInterface dependent;
public void StartAsynchronousOperation()
{
  Task.Run(()=>
  {
    Message += " Work";
    // The asynchronous operation is finished.
    dependent.DoMore()
  });
}
```

An additional example for event-based synchronization is in the code download.

The test can now be synchronized with the asynchronous operation when the dependent object is replaced by a stub while testing (see **Figure 3**).

The test has to equip the stub with a thread-safe notification mechanism, because the stub code runs in another thread. In the following test code example, a ManualResetEventSlim is used and the stub is generated with the RhinoMocks mocking framework:

```
// Setup
var finishedEvent = new ManualResetEventSlim();
var dependentStub = MockRepository.GenerateStub<ISomeInterface>();
dependentStub.Stub(x => x.DoMore()).
  WhenCalled(x => finishedEvent.Set());
var sut = new SystemUnderTest(dependentStub);
```

The test can now execute the asynchronous operation and can wait for the notification:

```
// Schedule operation to run asynchronously.
sut.StartAsynchronousOperation();
// Wait for operation to be finished.
finishedEvent.Wait();
// Assert outcome of operation.
Assert.AreEqual("Init Work", sut.Message);
```

This solution to synchronize the test with the tested threads can be applied to code with certain characteristics: The code under test has a notification mechanism like async and await or a plain event, or the code calls a dependent object.

A big advantage of the async and await synchronization is that it's able to propagate any kind of result back to the calling client. A special kind of result is an exception. So the test can handle exceptions explicitly. The other synchronization mechanisms can only recognize failures indirectly via the defective outcome.

Code with timer-based functionality can utilize async/await, events or calls to dependent objects to allow tests to synchronize with the timer operations. Every time the recurring operation finishes, the test is notified and can check the outcome (see examples in the code download).

Unfortunately, timers make unit tests slow even when you use a notification. The recurring operation you want to test generally starts only after a certain delay. The test will be slowed down and take at least the time of the delay. This is an additional disadvantage on top of the notification prerequisite.

Now I'll look at a solution that avoids some of the limitations of the two previous ones.

## Solution 3: Test in One Thread

For this solution, the code under test has to be prepared in a way that the test can later directly trigger the execution of the operations on the same thread as test itself. This is a translation of the approach of the jMock team for Java (see "Testing Multithreaded Code" at jmock.org/threads.html).

The system under test in the following example uses an injected task scheduler object to schedule asynchronous work. To demonstrate the capabilities of the third solution, I added a second operation that will be started when the first operation is finished:

```
private readonly TaskScheduler taskScheduler;
public void StartAsynchronousOperation()
{
  Message = "Init";
  Task task1 = Task.Factory.StartNew(()=>{Message += " Work1";},
                                     CancellationToken.None,
                                     TaskCreationOptions.None,
                                     taskScheduler);
  task1.ContinueWith(((t)=>{Message += " Work2";}, taskScheduler);
}
```

The system under test is modified to use a separate TaskScheduler. During the test, the "normal" TaskScheduler is replaced by a DeterministicTaskScheduler, which allows starting the asynchronous operations synchronously (see **Figure 4**).

The following test can execute the scheduled operations in the same thread as the test itself. The test injects the DeterministicTaskScheduler into the code under test. The DeterministicTaskScheduler doesn't immediately spawn a new thread, but only queues scheduled tasks. In the next statement the RunTasksUntilIdle method synchronously executes the two operations:

```
[Test]
public void TestCodeSynchronously()
{
  var dts = new DeterministicTaskScheduler();
  var sut = new SystemUnderTest(dts);
  // Execute code to schedule first operation and return immediately.
  sut.StartAsynchronousOperation();
  // Execute all operations on the current thread.
  dts.RunTasksUntilIdle();
  // Assert outcome of the two operations.
  Assert.AreEqual("Init Work1 Work2", sut.Message);
}
```

The DeterministicTaskScheduler overrides TaskScheduler methods to provide the scheduling functionality and adds among others the RunTasksUntilIdle method specifically for testing (see the code download for DeterministicTaskScheduler implementation details). As in synchronous unit testing, stubs can be used to concentrate on just a single unit of functionality at a time.

Code that uses timers is problematic not only because tests are fragile and slow. Unit tests get more complicated when the code uses a timer that's not running on a worker thread. In the .NET

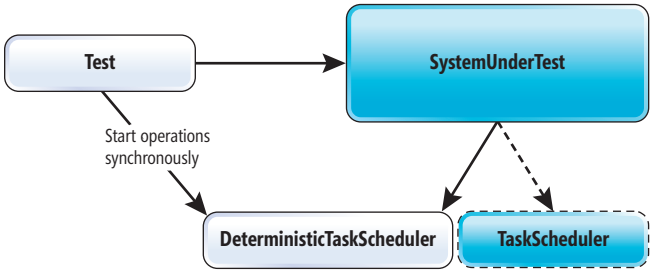



Figure 4 **Using a Separate TaskScheduler in SystemUnderTest**

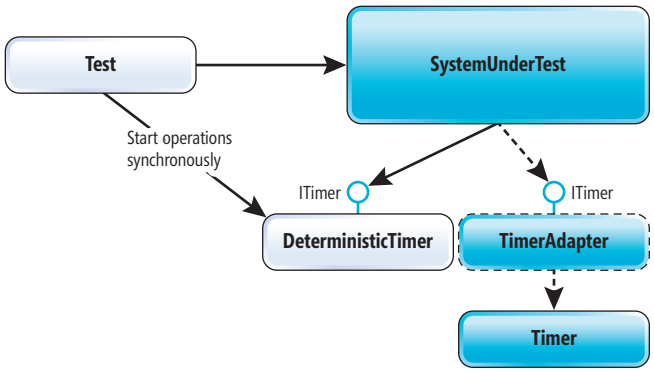



Figure 5 **Use ITimer in SystemUnderTest**

framework class library, there are timers specifically designed to be used in UI applications, like System.Windows.Forms.Timer for Windows Forms or System.Windows.Threading.DispatcherTimer for Windows Presentation Foundation (WPF) applications (see "Comparing the Timer Classes in the .NET Framework Class Library" at bit.ly/1r0SVic). These use the UI message queue, which isn't directly available during unit testing. The test shown at the beginning of this article will not work for these timers. The test has to spin up the message pump, for example by using WPF DispatcherFrame (see the example in the code download). To keep the unit tests simple and clear when you're deploying UI-based timers, you have to replace these timers during testing. I'm introducing an interface for timers to enable replacing the "real" timers with an implementation specifically for testing. I do this also for "thread"-based timers like System.Timers.Timer or System.Threading.Timer, because I can then improve the unit tests in all cases. The system under test has to be modified to use this ITimer interface:

```
private readonly ITimer timer;
private readonly Object lockObject = new Object();

public void StartRecurring()
{
  Message = "Init";
  // Set up timer with 1 sec delay and 1 sec interval.
  timer.StartTimer(() => { lock(lockObject){Message += " Poll";} }, new
TimeSpan(0,0,0,1));
}
```

By introducing the ITimer interface, I can replace the timer behavior during testing, as shown in **Figure 5**.

The extra effort of defining the interface ITimer pays off because a unit test checking the outcome of the initialization and the recurring operation can now run very quickly and reliably within milliseconds:

```
[Test]
public void VeryFastAndReliableTestWithTimer()
{
  var dt = new DeterministicTimer();
  var sut = new SystemUnderTest(dt);
  // Execute code that sets up a timer 1 sec delay and 1 sec interval.
  sut.StartRecurring();
  // Tell timer that some time has elapsed.
  dt.ElapseSeconds(3);
  // Assert that outcome of three executions of the recurring operation is OK.
  Assert.AreEqual("Init Poll Poll Poll", sut.Message);
}
```

The DeterministicTimer is specifically written for testing purposes. It allows the test to control the point in time when the timer action is executed, without waiting. The action is executed in the same thread as the test itself (see the code download for DeterministicTimer implementation details). For the execution of the tested code in a "non-testing" context, I have to implement an ITimer adapter for an existing timer. The code download contains examples of adapters for several of the framework class library timers. The ITimer interface can be tailored to the needs of the concrete situation and may only contain a subset of the full functionality of specific timers.

Testing asynchronous code with a DeterministicTaskScheduler or a DeterministicTimer allows you to easily switch off multithreading during testing. The functionality is executed on the same thread as the test itself. The interoperation of initialization code and asynchronous code is kept and can be tested. A test of this kind, for example, can check the correct time values used to initialize a timer. Exceptions are forwarded to tests, so they can directly check the error behavior of the code.

## Wrapping Up

Effective unit testing of asynchronous code has three main benefits: Maintenance costs for tests are reduced; tests run faster; and the risk of not executing the tests anymore is minimized. The solutions presented in this article can help you reach this goal.

The first solution, separating the functionality from the asynchronous aspects of a program via a Humble Object is the most generic. It's applicable for all situations, regardless of how threads are started. I recommend using this solution for very complex asynchronous scenarios, complex functionality or a combination of both. It's a good example of the separation of concerns design principle (see bit.ly/1lB8iHD).

The second solution, which synchronizes the test with the completion of the tested thread, can be applied when the code under test provides a synchronization mechanism such as async and await. This solution makes sense when the notification mechanism prerequisites are fulfilled anyhow. If possible, use the elegant async and await synchronization when non-timer threads are started, because exceptions are propagated to the test. Tests with timers can use await, events or calls to dependent objects. These tests can be slow when timers have long delays or intervals.

The third solution uses the DeterministicTaskScheduler and the DeterministicTimer, thus avoiding most limitations and disadvantages of the other solutions. It requires some effort to prepare the code under test, but high unit test code coverage can be reached. The tests for code with timers can be executed very fast without the wait for delay and interval times. Also, exceptions are propagated to the tests. So this solution will lead to robust, fast and elegant unit test suites combined with high code coverage.

These three solutions can help software developers avoid the pitfalls of unit testing asynchronous code. They can be used to create fast and robust unit test suites and cover a wide range of parallel programming techniques. ■

**Sven Grand** *is a quality engineering software architect for the diagnostic X-ray business unit of Philips Healthcare. He got "test infected" years ago, when he heard for the first time about test-driven development at a Microsoft software conference in 2001. Reach him at sven.grand@philips.com.*

**Thanks** to the following technical experts for reviewing this article: *Stephen Cleary, James McCaffrey, Henning Pohl and Stephen Toub*


## Work With Relational Data, Not Complex APIs or Services

Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with. If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!

## Give RSSBus a try today and see what mean:

**visit us online at www.rssbus.com to learn more or download a free trial.**

rssbus

INTEGRATION YOUR WAY

# Mapping in Windows Phone 8.1

Keith Pijanowski

Each generation of Windows Phone brings improved mapping capabilities, and Windows Phone 8.1, with the new Map control introduced with Visual Studio 2013 Update 3, is no exception. If all you need is a simple map, centered on a specified location, you can accomplish this with the new Map control with just a few lines of code. But if you need to respond to user interaction, customize the look of the control, or calculate routes, you can also use the new Map control and the mapping services API for these more complex scenarios.

This article will cover the new Windows Phone 8.1 Map control and the mapping services APIs. I'll start by showing basic map display features, then take a look at more advanced features that show how to place push pins on the map at a specified address. I'll also show you how to place XAML controls on the Map control, and how to get the GPS location (and address) that corresponds to a specific point on the Map control. This is useful if you want to allow the user to touch a location of interest and then determine the GPS location and street address. Finally, I'll show how to calculate driving and walking directions.

This article discusses:

- Authenticating your application
- Displaying maps
- Geocoding and reverse geocoding
- Calculating driving and walking routes

Technologies discussed:

Windows Phone 8.1, Visual Studio 2013 Update 3

Code download available at:

msdn.microsoft.com/magazine/msdnmag1114

A good companion piece to the material I'll cover is the September 2014 article, "Creating a Location-Aware App with Geofencing," by Tony Champion. A geofence is a specified region around a GPS location that can be registered with Windows. Once the region is registered, an application can receive notifications when the device enters or leaves the region. Geofencing is also new to Windows Phone 8.1. If you're going to build an application that uses geofencing, consider using the Map control to allow users to specify geofences and show their existing geofences.

As of this writing, there's a different Map control for every flavor of mobile and tablet platform:

- Windows Phone Silverlight 8.0/8.1 applications: Microsoft.Phone.Maps.Controls.Map
- Windows Store 8.x applications: Bing.Maps.Map
- Windows Phone 8.1 Runtime applications: Windows.UI.Xaml.Controls.Maps

This article focuses on the new Map control and mapping services that can be used within applications written for the Windows Phone 8.1 Runtime.

## Getting Started

**Authenticating Your Application** Your Windows Phone application needs to be authenticated every time it uses the Map control or the mapping services found in the Windows.Services.Maps namespace. To authenticate your application, you need a Map service ApplicationID and a Map service Authentication Token, both of which can be retrieved from your Developer Dashboard in the Windows Phone Dev Center. You'll find step-by-step instructions for getting an ApplicationID and AuthenticationToken from the MSDN Library article at bit.ly/1y78M2F.

Once you have them, you need to place the ApplicationID within your application's Package.appxmanifest file. Instructions for doing

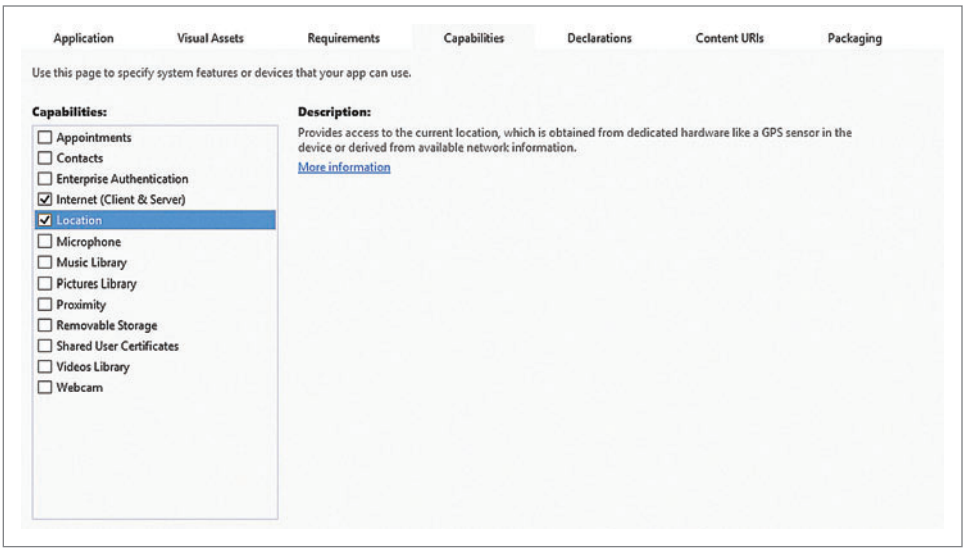



Figure 1 **Declaring Internet and Location Capabilities**

this are also in the MSDN Library article. You'll use the Map Service Authentication Token in your code, as you'll see later in this article.

**Capabilities** Many of the samples in this article use the user's current location, and the Map control may need the Internet. Therefore, the Internet and Location capabilities should be declared in the application manifest file as shown in **Figure 1**. However, mapping services also work without Internet connectivity when maps are downloaded for offline use. For more information on downloading maps, see "How to Download and Update Offline Maps (XAML)" at bit.ly/1nXBOrS.

## Displaying Maps

**Displaying a Simple Map** Adding a Map control to a page is as easy as adding any other control. You can drag the control from the Visual Studio Toolbox onto your page or you can enter the XAML as follows:

```
<Maps:MapControl
  x:Name="myMapControl"
  MapServiceToken="Place your token here."
  Height="200" Width="300"
  HorizontalAlignment="Center"
  Margin="0,0,0,12"
  />
```

Notice the MapServiceToken property. This property must be given the value of the Map Service Authentication Token that was assigned to your application in the Developer Dashboard of the Windows Phone Dev Center. Also, notice that you can position the Map control like any other control. The Map control doesn't have to dominate the entire screen. A small map zoomed into a specific location can make an effective visual on a page that contains location-specific information. On the other hand, maps that are zoomed out (or may be zoomed out by the user) will need additional real estate. For such maps, consider using the full page.

The preceding XAML will produce uninteresting results if placed within an application and run. This control will show a map completely zoomed out. Therefore, the next step in using the Map control is to identify a specific location as the center point of the map, as well as the zoom level.

The following code first calculates the user's current position and then sets that position into the Center property of the map. It also sets the zoom level to 15. The minimum value for the zoom level is 1, which shows the map fully zoomed out—and displays half the globe in your Map control. The maximum value for the zoom level is 20, which can zoom the Map control into a specific street address:

```
Geolocator geolocator = new Geolocator();
Geoposition geoposition = null;
try
{
  geoposition = await geolocator.GetGeopositionAsync();
}
catch (Exception ex)
{
  // Handle errors like unauthorized access to location
  // services or no Internet access.
}
myMapControl.Center = geoposition.Coordinate.Point;
myMapControl.ZoomLevel = 15;
```

**Adding Images to the Map Control** If you want to mark a specific location on your map, you can add images with a title to the Map control. The code in **Figure 2** will point to the device's current location by adding an image from the project's Assets folder to the Map control.

To add an image to your Map control, first create a MapIcon object and set its Location property to a Geopoint, which represents the location of interest on your map. The code in **Figure 2** uses the user's current location. The Title property is the text that will be displayed above the image that's added to the Map control.

To specify the image, set the MapIcon's Image property using the RandomAccessStreamReference.CreateFromUri method. This is a static method found in the Windows.Storage.Streams namespace. The image should be approximately 50 pixels by 50 pixels; if it's larger, it will be scaled down.

When the purpose of your image is to point to a specific location, it's a best practice to use the MapIcon's NormalizedAnchorPoint property. This property allows you to indicate the point within your image that will be placed over the specified Geopoint. If your image

Figure 2 **Adding a Custom Image to the Map Control**

```
Geolocator geolocator = new Geolocator();
Geoposition geoposition = null;
try
{
  geoposition = await geolocator.GetGeopositionAsync();
}
catch (Exception ex)
{
  // Handle errors like unauthorized access to location services or no
Internet access.
}
MapIcon mapIcon = new MapIcon();
mapIcon.Image = RandomAccessStreamReference.CreateFromUri(
  new Uri("ms-appx:///Assets/PinkPushPin.png"));
mapIcon.NormalizedAnchorPoint = new Point(0.25, 0.9);
mapIcon.Location = geoposition.Coordinate.Point;
mapIcon.Title = "You are here";
myMapControl.MapElements.Add(mapIcon);
```

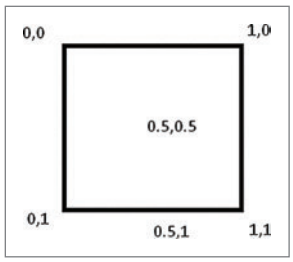



Figure 3 **Coordinate System Used for the Normalized-AnchorPoint Property**

is a push pin, this would be the tip of the pin. If you're using an image of an arrow, this point would be the tip of the arrow. The default value for the NormalizedAnchorPoint property is a Point object created with a value of (0, 0), which represents the upper-left corner of the image. The bottom-right corner of the image is (1,1). This coordinate system is depicted in **Figure 3** with a few sample points.

If the image used as the map icon contains a pointer and you don't set the NormalizedAnchorPoint property correctly, the map icon will point to an incorrect position on your map. If your image is merely a dot or small circle, you'd use a value of (0.5,0.5), which is the center of the image.

If you don't specify an image, the MapControl has a default image, shown in **Figure 4**, that will be used.

As you can see, the default MapIcon image has a pointer located at the bottom center of the image. Therefore, the Normalized-AnchorPoint property needs to be set to (0.5, 1.0).

The final detail to notice about the code is that the MapIcon object is added to the Map's MapElements collection, so you can add more than one MapIcon to your map if you wish.

There's no guarantee that the map icon and the title will be shown. It might be hidden if it will obscure other elements or labels on the map. This is especially true as the map is zoomed out (zoom level decreases) and more information has to fit within the Map control.

The code in **Figure 2** displays the map in **Figure 5**, assuming you're watching a baseball game in Fenway Park.

**Adding Controls to the Map Control** Adding a XAML control to the Map control is similar to adding an image. The following code creates an ellipse, sizes it to a circle, adds it to the Map control, and centers it on a geographic coordinate:

```
// Create a circle
Windows.UI.Xaml.Shapes.Ellipse fence = new Windows.UI.Xaml.Shapes.Ellipse();
fence.Width = 30;
fence.Height = 30;
fence.Stroke = new SolidColorBrush(Colors.DarkOrange);
fence.StrokeThickness = 2;

MapControl.SetLocation(fence, geoposition.Coordinate.Point);
MapControl.SetNormalizedAnchorPoint(fence, new Point(0.5, 0.5));

myMapControl.Children.Add(fence);
```

Notice that the Map control's static function SetLocation must be used to attach the geographic coordinate to the ellipse object. Similarly, the SetNormalizedAnchorPoint function is used to attach the Normalized Anchor Point property.

Any control that inherits from UIElement or MapItemsControl can be added to the Map control using these techniques.

## Getting the Location

The heart of the Map control is the BasicGeoposition object, which can represent every location on Earth using just three numbers: latitude, longitude and altitude. Not only is this is an efficient way to divide the globe, it's also precise. Every location has its own unique combination of latitude, longitude and altitude. No two locations will have the same combination.

Unfortunately, your application is going to interface with human beings, and human beings do not like to represent locations as pure numeric values. Consequently, the globe is divided into continents, countries within continents, regions (or states) within countries, towns within regions, and streets within towns. While continent names, country names, and postal codes are unique, regions can sometimes contain two or more towns with the same name. Additionally, a town may have two or more streets with the same name. The bottom line is that it's possible for an address to be confused with two or more locations. This is especially true when dealing with an incomplete representation of an address (leaving off the state name, postal code or town).

Therefore, if your application is going to show more than the user's current location, which can be retrieved directly from the device, you'll need to convert between the numeric and named systems previously described. The process of converting an address to a geographic location is known as geocoding. The process of converting a geographic location to a human-readable address is known as reverse geocoding.

**Converting an Address to a Geographic Location (Geocoding)** Geocoding an address requires the use of the mapping services APIs—the collection of classes found in the Windows.Services.Maps namespace. To use these APIs, you need to set your Authentication Token in the static property ServiceToken of the MapService class. The following line of code shows how to do this (the remainder of this article will assume that this line of code is in the OnLaunched event of the application):

```
MapService.ServiceToken = "Place your token here";
```

Once the Authentication Token has been set, you can use the FindLocationsAsync static function,

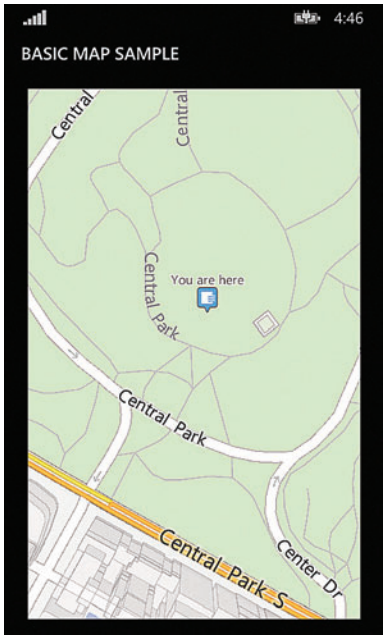



Figure 4 **The Default Map Icon**

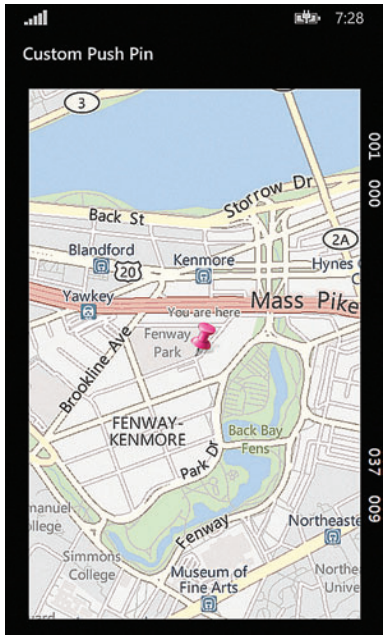



Figure 5 **Map Control Showing Fenway Park in Boston, Mass.**

# GdPicture.NET 10


- Document viewing, processing, printing and scanning (TWAIN & WIA).
- Reading, writing and converting vector and raster images in more than 90 formats, PDF included.
- OMR, OCR, barcode reading and writing (linear & 2D).
- Annotations for image and PDF within Windows and Web applications.
- Color detection engine for image and PDF compression.

And much more ...

# All-In-One Document Imaging SDK

## Royalty-Free Document Imaging Toolkits for .NET and COM/ActiveX

## GdPicture.NET 10 Plugins

- NEW Color detection
- NEW DICOM image reader
- NEW MICR reader
- Full managed PDF support
- Full annotations support for PDF and images
- OCR
- Forms processing
- JBIG2 encoding
- 1D and 2D barcode reading and writing


**Try GdPicture.NET 10 FREE for 30 days**

www.gdpicture.com

which is found in the MapLocationFinder class. Here's how to use the FindLocationsAsync function:

```
Geolocator geolocator = new Geolocator();
Geoposition currentPosition = await geolocator.GetGeopositionAsync();

MapLocationFinderResult result = await MapLocationFinder.
FindLocationsAsync(
  address, currentPosition.Coordinate.Point, 5);
```

The first parameter of the FindLocationsAsync function is a string representation of an address. Any string may be passed to this function. The back-end mapping services, which the FindLocationsAsync function communicates, will do the best it can to find a matching location. The second parameter is known as a reference point and represents the geographic location where the search should start. This parameter is sometimes referred to as a hint. If used properly it can greatly increase the relevance of the returned locations and the speed with which they're calculated. For example, if you know the user is doing a local search, then pass in the user's current location as the reference point. However, if you know the user is planning her vacation, pass in the location of her hotel. The final parameter is the maximum number of locations that will be returned.

The return value of the FindLocationsAsync function is an object of type MapLocationFinderResult. It will tell you if the search was successful or not. If the search was successful, the MapLocationFinderResult object will contain a collection of MapLocation objects. The code in **Figure 6** checks the success of the search, and if it's successful, places the collection into a ListView control so that all matching locations can be shown to the user. The MapLocation object contains both human-readable address information and geographic information that can be used by the MapControl. The code in **Figure 6** could easily be converted to show push pins on a Map control.

**Converting a Geographic Position into an Address (Reverse Geocoding)** Another useful feature a mapping application can provide is the ability to get a human-readable address from a touched area of the Map control. To code this feature, you first need a way to get the position of the control that was touched by the user. This can be accomplished using the MapTapped event of the Map control. The Map control also has a Tapped event, which is fired when a XAML child of the Map control is tapped. Any direct tap of the Map control goes through the MapTapped event. The code in **Figure 7** shows an implementation of the MapTapped event that displays the address of the tapped location in a TextBlock control.

There are a few details worth calling out in this implementation of the MapTapped event. First, the geopoint that's associated with the tapped location is passed to this event via args.Location. The FindLocationsAtAsync function can use this geopoint to get a human-readable address. This function is a static function in the MapLocationFinder class, which is part of the mapping services API (Windows.Services.Maps namespace). This function returns a MapLocationFinderResult object that contains a Status property and a collection of MapLocation objects. Even though the return value allows for a collection of MapLocation objects, usually there will be only one MapLocation object. However, there might be a high-rise building somewhere in the world that has two or more street addresses associated with it. If a user clicks in the middle of a big field or lake, a value will still be returned. If there's no actual street address nearby, the MapLocation object will contain only the Region, Country and Postal Code.

## Routes

The mapping services can calculate routes from a given starting point to a given ending point. The Map control itself is capable of displaying the route, which can be calculated as driving routes or walking routes.

Figure 6 **Retrieving the Results of a Location Search**

```
if (result.Status == MapLocationFinderStatus.Success)
{
  List<string> locations = new List<string>();
  foreach (MapLocation mapLocation in result.Locations)
  {
    // create a display string of the map location
    string display = mapLocation.Address.StreetNumber + " " +
      mapLocation.Address.Street + Environment.NewLine +
      mapLocation.Address.Town + ", " +
      mapLocation.Address.RegionCode + "  " +
      mapLocation.Address.PostCode + Environment.NewLine +
      mapLocation.Address.CountryCode;

    // Add the display string to the location list.
    locations.Add(display);
  }

  // Bind the location list to the ListView control.
  lvLocations.ItemsSource = locations;

}
else
{
  // Tell the user to try again.
}
```

Figure 7 **Finding the Address of a Tapped Location on the Map Control**

```
private async void myMapControl_MapTapped(MapControl sender, MapInputEventArgs args)
{
  // Find the address of the tapped location.
  MapLocationFinderResult result =
    await MapLocationFinder.FindLocationsAtAsync(args.Location);
  if (result.Status == MapLocationFinderStatus.Success)
  {
    if (result.Locations.Count > 0)
    {
      string display = result.Locations[0].Address.StreetNumber + " " +
        result.Locations[0].Address.Street;

      tbAddress.Text = display;
    }
  }

}
```

Figure 8 **Calculating a Driving Route**

```
// Start at Grand Central Station.
BasicGeoposition startLocation = new BasicGeoposition();
startLocation.Latitude = 40.7517;
startLocation.Longitude = -073.9766;
Geopoint startPoint = new Geopoint(startLocation);

// End at Central Park.
BasicGeoposition endLocation = new BasicGeoposition();
endLocation.Latitude = 40.7669;
endLocation.Longitude = -073.9790;
Geopoint endPoint = new Geopoint(endLocation);

// Get the route between the points.
MapRouteFinderResult routeResult =
await MapRouteFinder.GetDrivingRouteAsync(
  startPoint,
  endPoint,
  MapRouteOptimization.Time,
  MapRouteRestrictions.None,
  290);
```

Figure 9 **Looping Through Route Legs and Route Maneuvers**

```
// Display summary info about the route.
tbTurnByTurn.Inlines.Add(new Run()
{
  Text = "Total estimated time (minutes) = "
    + routeResult.Route.EstimatedDuration.TotalMinutes.ToString("F1")
});
tbTurnByTurn.Inlines.Add(new LineBreak());
tbTurnByTurn.Inlines.Add(new Run()
{
  Text = "Total length (kilometers) = "
    + (routeResult.Route.LengthInMeters / 1000).ToString("F1")
});
tbTurnByTurn.Inlines.Add(new LineBreak());

// Display the directions.
tbTurnByTurn.Inlines.Add(new Run()
{
  Text = "DIRECTIONS"
});
tbTurnByTurn.Inlines.Add(new LineBreak());

// Loop through the legs and maneuvers.
int legCount = 0;
foreach (MapRouteLeg leg in routeResult.Route.Legs)
{
  foreach (MapRouteManeuver maneuver in leg.Maneuvers)
  {
    tbTurnByTurn.Inlines.Add(new Run()
    {
      Text = maneuver.InstructionText
    });
    tbTurnByTurn.Inlines.Add(new LineBreak());
  }
}
```

**Driving Routes** As the name implies, driving routes are intended to be driven in some vehicle that won't travel the wrong way down a one-way street and must obey all other rules of the road. The code in **Figure 8** calculates the driving directions from New York City's Grand Central Station to Central Park.

The code is fairly straightforward. The static function MapRouteFinder.GetDrivingRouteAsync is passed a starting point, an ending point, an optimization parameter (described later) and a parameter that specifies restrictions (also described later). The fifth and final parameter is an optional heading that specifies the current direction of the user in compass degrees (0 degrees represents North, 180 degrees represents South and so on). This is useful when routes are calculated while the user is driving. When the heading parameter is used, the route is calculated based on the user's current direction of travel. The first few legs of the route will contain maneuvers that get the user to the best possible route, which is especially useful within cities where the user might be on a one-way road heading in the wrong direction. Changing the heading to 135 in the code in **Figure 8** causes the route to be calculated such that it contains a legal way to change directions.

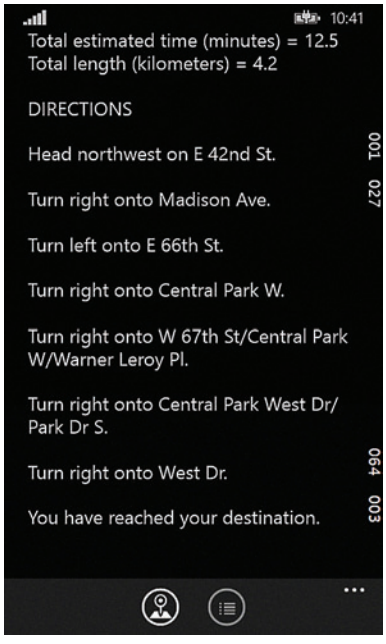



Figure 10 **Turn-by-Turn Directions for a Driving Route**

Figure 11 **Showing a Route Visually in a Map Control**

```
if (routeResult.Status == MapRouteFinderStatus.Success)
{
  // Use the route to initialize a MapRouteView.
  MapRouteView viewOfRoute = new MapRouteView(routeResult.Route);
  viewOfRoute.RouteColor = Colors.Blue;
  viewOfRoute.OutlineColor = Colors.Blue;

  // Add the new MapRouteView to the Routes collection
  // of the MapControl.
  MapwithDrivingRoute.Routes.Add(viewOfRoute);

  // Fit the MapControl to the route.
  await MapwithDrivingRoute.TrySetViewBoundsAsync(
    routeResult.Route.BoundingBox,
    null,
    Windows.UI.Xaml.Controls.Maps.MapAnimationKind.Bow);
}
```

Once you have a result from the MapRouteFinder.GetDrivingRouteAsync function, you need to check it. If a route was successfully retrieved, it can then be displayed as turn-by-turn instructions. The route can also be visually displayed in a Map control. The code in **Figure 9** displays the route from Grand Central Station to Central Park as turn-by-turn instructions.

The MapRoute object contains summary information that indicates how long it should take to traverse the route and the length of the route in meters. The MapRoute object organizes route information by using MapRouteLeg objects and MapRouteManeuver objects. A MapRoute contains one or more MapRouteLeg objects and each MapRouteLeg contains one or more MapRouteManeuver objects. The information that will be shown to the user is in the MapRouteManeuver object. The MapRouteManeuver objects contain navigational instructions, exit information (if applicable), length of the maneuver, and maneuver notices (if applicable). The Kind property is also very useful. Sample values include left turn, right turn, and so forth. It can be used to associate an image with each maneuver. A full list of possible values for the Kind property can be found at bit.ly/1nX0yi7. **Figure 10** shows what these directions would look like in an application.


Figure 12 **A Driving Route in a Map Control**

Figure 13 **The MapRouteOptimization Enumeration**

| | |
|---|---|
| Time | The route will be optimized such that it can be covered in the least amount of time given the distance and speed limit for each maneuver of the route. |
| Distance | The route will optimized such that it can be covered in the shortest distance given the distance of each maneuver. |
| TimeWithTraffic | The route will be optimized such that it can be covered in the least amount of time given the distance, speed limit and current traffic conditions for each maneuver of the route. |

Figure 14 **The MapRouteRestrictions Enumeration**

| | |
|---|---|
| None | No restrictions are applied to the calculation of a route. The route can contain maneuvers that include highways, tunnels, ferries, toll roads and dirt roads. |
| Highways | The route will not contain any maneuvers that travel over a highway. |
| TollRoads | The route will not contain any maneuvers that travel over a toll road. |
| Ferries | The route will not contain any maneuvers that require the use of a ferry. |
| Tunnels | The route will not contain any maneuvers that travel through a tunnel. |
| DirtRoads | The route will not contain any maneuvers that travel over a dirt road. |

Routes can also be displayed graphically within a Map control (see **Figure 11**).

To show a route in a Map control you first need to create a view of the route using the MapRouteView class. Once you have a view you can add it to the Routes collection of the Map control. Finally, the Map control's TrySetViewBoundsAsync function will size the Map control such that the entire route is visible. This function even comes with an animation parameter that provides a visual effect as the Map control is being redrawn to accommodate the route. **Figure 12** shows what this would look like in an application.

**Route Optimizations** Driving routes can be optimized with respect to time, distance and traffic. The MapRouteOptimization enumeration provides the values in **Figure 13** for optimizing the calculation of a route.

**Route Restrictions** Driving routes can also be calculated with restrictions, such as "no highways" or "no toll roads." The MapRouteRestrictions enumeration provides the values shown in **Figure 14** for restricting the calculation of a driving route.

These values can be logically combined. For example, the following code is a restriction that prevents the use of highways and toll roads:

```
MapRouteFinderResult routeResult =
  await MapRouteFinder.GetDrivingRouteAsync(
  startPoint,
  endPoint,
  MapRouteOptimization.Time,
  MapRouteRestrictions.Highways || MapRouteRestrictions.TollRoads);
```

**Walking Routes** Calculating walking routes is very similar to calculating driving routes. The following code calculates a walking route based on the same end points as the driving route example:

```
// Get the route between the points.
MapRouteFinderResult routeResult =
  await MapRouteFinder.GetWalkingRouteAsync(
  startPoint,
  endPoint);
```

The static function MapRouteFinder.GetWalkingRouteAsync is used to perform this calculation. Unlike the corresponding function for calculating driving routes, this function has no overloads that allow for optimizations, restrictions or current heading. These attributes don't impact walking route calculations because speed limits and traffic conditions aren't relevant. Also, the current heading can be easily changed. Walking routes are calculated for the shortest distance. The images in **Figure 15** and in **Figure 16** show the walking route from Grand Central Station to Central Park.


Figure 15 **Turn-by-Turn Directions for a Walking Route**


Figure 16 **Walking Route Shown in a Map Control**

## Wrapping Up

In this article I showed how to use the new Map control for Windows Phone 8.1, including basic map display, adding images and adding controls. I also introduced the underlying mapping services API and took a look at geocoding, reverse geocoding and route calculations.

A number of other topics related to mapping should be considered: overlaying tiled images, data binding and managing offline maps. You can find information about these topics on the Windows Dev Center (bit.ly/X7S7ei). ■

**Keith Pijanowski** *has more than 20 years of experience in the software industry. He has worked for startups and large companies in roles that ranged from writing code to business development. Currently, he's working for himself as an independent consultant. Pijanowski lives in the New York City area, but has never cheered for a New York sports team. Reach him at keithpij@msn.com or twitter.com/keithpij.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Mike O'Malley*

# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

**ORLANDO**
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

**NOV 17-21**


***"HELP US VISUAL STUDIO LIVE! – YOU'RE OUR ONLY HOPE!"***

# THE CODE IS STRONG WITH THIS ONE!

**While your development challenges may not be as dire as the Rebellion's, we know you're searching for real-world knowledge and education on the Microsoft Platform. And for 21 years and counting, Visual Studio Live! has been the conference more developers trust for independent, practical training.**

EVENT PARTERS
Magenic Microsoft

PLATINUM SPONSOR
esri

GOLD SPONSORS
New Relic. pluralsight hardcore dev and IT training sitecore

# LIVE! 360

TECH EVENTS WITH PERSPECTIVE

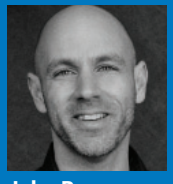
## FEATURED SPEAKERS:


Andrew Brust


Rockford Lhotka


Miguel Castro


Deborah Kurata


John Papa


Rachel Appel


Brian Randell


Leonard Lobel


Brian Noyes

## REGISTER TODAY
## SESSIONS ARE FILLING UP QUICKLY!

Use promo code ORLNOV4

## 5 GREAT CONFERENCES. 1 GREAT PRICE.

Visual Studio Live! Orlando is part of Live! 360, the ultimate IT and Developer line-up. This means you'll have access to 5 co-located conferences for 1 low price!

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

SharePoint LIVE!
TRAINING FOR COLLABORATION

Modern Apps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

live360events.com

TURN THE PAGE FOR MORE EVENT DETAILS

SUPPORTED BY msdn magazine, Visual Studio, Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

vslive.com/orlando

# AGENDAS AT-A-GLANCE Visual Studio Live! & Modern Apps Live!

| Visual Studio / .NET Framework | JavaScript / HTML5 Client | ASP.NET | Cross-Platform Mobile Development | Cloud Computing | Windows Client | Windows Phone | Data Management | Modern Apps LIVE! |
|---|---|---|---|---|---|---|---|---|

| START TIME | END TIME | Visual Studio Live! & Modern Apps Live! Pre-Conference: Sunday, November 16, 2014 |
|---|---|---|
| 4:00 PM | 9:00 PM | Pre-Conference Registration - Royal Pacific Resort Conference Center |
| 6:00 PM | 9:00 PM | Dine-A-Round Dinner @ Universal CityWalk |

| START TIME | END TIME | Visual Studio Live! & Modern Apps Live! Pre-Conference Workshops: Monday, November 17, 2014 (Separate entry fee required) | | | | |
|---|---|---|---|---|---|---|
| 6:30 AM | 8:00 AM | Pre-Conference Workshop Registration - Coffee and Morning Pastries | | | | |
| 8:00 AM | 5:00 PM | **VSM01** - Workshop: Native Mobile App Development for iOS, Android, and Windows using C# - *Marcel de Vries & Rockford Lhotka* | | **VSM02** - Workshop: AngularJS in 0 to 60 - *John Papa* | **VSM03** - Workshop: From Code to Release: DevOps for Developers - *Brian Randell* | **MAM01** - Workshop: Modern App Technology Overview - Android, iOS, Cloud, and Mobile Web - *Nick Landry, Kevin Ford, & Steve Hughes* |
| 5:00 PM | 6:00 PM | EXPO Preview | | | | |
| 6:00 PM | 7:00 PM | Live! 360 Keynote: To Be Announced | | | | |

| START TIME | END TIME | Visual Studio Live! & Modern Apps Live! Day 1: Tuesday, November 18, 2014 | | | | |
|---|---|---|---|---|---|---|
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | | |
| 8:00 AM | 9:00 AM | Visual Studio Live! & Modern Apps Live! Keynote: To Be Announced | | | | |
| 9:00 AM | 9:30 AM | Networking Break • Visit the EXPO | | | | |
| 9:30 AM | 10:45 AM | **VST01** - Getting Started with Xamarin - *Walt Ritscher* | **VST02** - Great User Experiences with CSS 3 - *Robert Boedigheimer* | **VST03** - What's New in MVC 5 - *Miguel Castro* | **VST04** - New IDE and Editor Features in Visual Studio 2013 - *Deborah Kurata* | **MAT01** - Defining Modern App Development - *Rockford Lhotka* |
| 11:00 AM | 12:15 PM | **VST05** - Building Your First Universal Application for Windows Phone and Windows Store - *Brian Peek* | **VST06** - HTML5 and Modernizr for Better Web Sites - *Robert Boedigheimer* | **VST07** - What's New in Web API 2 - *Miguel Castro* | **VST08** - Visual Studio Data Tools for Developers - *Deborah Kurata* | **MAT02** - Modern App Architecture - *Rockford Lhotka* |
| 12:15 PM | 2:00 PM | Lunch • Visit the EXPO | | | | |
| 2:00 PM | 3:15 PM | **VST09** - Introduction to Developing Mobile Apps for the Web Developer - *Ben Hoelting* | **VST10** - Build an Angular and Bootstrap Web Application in Visual Studio from the Ground Up - *Deborah Kurata* | **VST11** - ASP.NET MVC for Mobile Devices - *Rob Daigneau* | **VST12** - The Road to Continuous Delivery, Automated UI Testing for Web, WPF and Windows Forms - *Marcel de Vries* | **MAT03** - ALM with Visual Studio Online (TFS) and Git - *Brian Randell* |
| 3:15 PM | 4:15 PM | Networking Break • Visit the EXPO | | | | |
| 4:15 PM | 5:30 PM | **VST13** - Creating Responsive Cross-Platform Native/Web Apps with JavaScript and Bootstrap - *Ben Dewey* | **VST14** - Hate JavaScript? Try TypeScript - *Ben Hoelting* | **VST15** - What's New in WPF 4.5 - *Walt Ritscher* | **VST16** - Understanding Dependency Injection & Writing Testable Software - *Miguel Castro* | **MAT04** - Reusing Business and Data Access Logic Across Platforms - *Kevin Ford* |
| 5:30 PM | 7:30 PM | Exhibitor Reception | | | | |

| START TIME | END TIME | Visual Studio Live! & Modern Apps Live! Day 2: Wednesday, November 19, 2014 | | | | |
|---|---|---|---|---|---|---|
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | | |
| 8:00 AM | 9:00 AM | Live! 360 Keynote: To Be Announced | | | | |
| 9:15 AM | 10:30 AM | **VSW01** - Build Cross-Platform Apps with Shared Projects, CSLA .NET, and C# - *Rockford Lhotka* | **VSW02** - AngularJS Jump-Start - *John Papa* | **VSW03** - Best Practices for Self Hosting Web API Based Services - *Rob Daigneau* | **VSW04** - The Busy Developers Guide to Virtualization with Hyper-V - *Brian Randell* | **MAW01** - Coding for Quality and Maintainability - *Jason Bock* |
| 10:30 AM | 11:00 AM | Networking Break • Visit the EXPO | | | | |
| 11:00 AM | 12:15 PM | **VSW05** - Building Mobile Cross-Platform Apps with HTML5 & Cordova in Visual Studio - *Nick Landry* | **VSW06** - AngularJS Anywhere with Node.js - *John Papa* | **VSW07** - Slice Development Time With ASP.NET MVC, Visual Studio, and Razor - *Philip Japikse* | **VSW08** - Build It and Ship It with TFS and Release Management - *Brian Randell* | **MAW02** - UX Design for Modern Apps - *Anthony Handley* |
| 12:15 PM | 1:45 PM | Birds-of-a-Feather Lunch • Visit the EXPO | | | | |
| 1:45 PM | 3:00 PM | **VSW09** - What's New in Azure for Developers - *Vishwas Lele* | **VSW10** - I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(me) Maybe? - *Rachel Appel* | **VSW11** - Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI - *Marcel de Vries* | **VSW12** - Readable Code - *John Papa* | **MAW03** - Applied UX: iOS, Android, Windows - *Anthony Handley* |
| 3:00 PM | 4:00 PM | Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. | | | | |
| 4:00 PM | 5:15 PM | **VSW13** - API Management - *Vishwas Lele* | **VSW14** - Creating HTML5 and Angular Websites Using Visual Studio LightSwitch - *Michael Washington* | **VSW15** - Build Real-Time Websites and Apps with SignalR - *Rachel Appel* | **VSW16** - Visual Studio Online: An Update Every Three Weeks - *Brian Randell* | **MAW04** - Leveraging Azure Services - *Kevin Ford* |
| 8:00 PM | 10:00 PM | Live! 360 Evening Event | | | | |

| START TIME | END TIME | Visual Studio Live! & Modern Apps Live! Day 3: Thursday, November 20, 2014 | | | | |
|---|---|---|---|---|---|---|
| 7:30 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | | |
| 8:00 AM | 9:15 AM | **VSH01** - Creating Visual Studio Cloud Business Applications for Office 365 / SharePoint 2013 - *Michael Washington* | **VSH02** - To Be Announced | **VSH03** - Games Development with Unity and Other Frameworks for Windows and Windows Phone - *Brian Peek* | **VSH04** - Managing the .NET Compiler - *Jason Bock* | **MAH01** - Analyzing Results with Power BI - *Steve Hughes* |
| 9:30 AM | 10:45 AM | **VSH05** - Microsoft Azure Web Sites for the Web Developer - *Eric D. Boyd* | **VSH06** - Building Rich Data Input Windows 8 Applications - *Brian Noyes* | **VSH07** - Build Your First Mobile App in 1 Hour with Microsoft App Studio - *Nick Landry* | **VSH08** - Writing Asynchronous Code Using .NET 4.5 and C# 5.0 - *Brian Peek* | **MAH02** - Building a Native iOS App |
| 11:00 AM | 12:15 PM | **VSH09** - Moving Web Apps to the Cloud - *Eric D. Boyd* | **VSH10** - XAML Antipatterns - *Ben Dewey* | **VSH11** - Creating Map Centric Web Applications with HTML5 and JavaScript - *Ben Ramseth* | **VSH12** - Asynchronous Debugging in .NET - *Jason Bock* | **MAH03** - Building an Android App with Xamarin - *Nick Landry* |
| 12:15 PM | 1:30 PM | Lunch on the Lanai | | | | |
| 1:30 PM | 2:45 PM | **VSH13** - Node.js for .NET Developers - *Jason Bock* | **VSH14** - Building Maintainable and Extensible MVVM WPF Apps with Prism 5 - *Brian Noyes* | **VSH15** - Learning Entity Framework 6 - *Leonard Lobel* | **VSH16** - Rock Your Code Using Code Contracts - *David McCarter* | **MAH04** - Building a Windows App - *Brent Edwards* |
| 3:00 PM | 4:15 PM | **VSH17** - Building Mobile Cross-Platform Apps in C# with Azure Mobile Services - *Nick Landry* | **VSH18** - XAML for the WinForms Developer - *Philip Japikse* | **VSH19** - Database Development with SQL Server Data Tools - *Leonard Lobel* | **VSH20** - Rock Your Code With Visual Studio Add-ins - *David McCarter* | **MAH05** - Building a Responsive Single Page App - *Allen Conway* |
| 4:30 PM | 5:45 PM | Live! 360 Conference Wrap-up | | | | |

| START TIME | END TIME | Visual Studio Live! & Modern Apps Live! Post-Conference Workshops: Friday, November 21, 2014 (Separate entry fee required) | | | | |
|---|---|---|---|---|---|---|
| 7:30 AM | 8:00 AM | Post-Conference Workshop Registration - Coffee and Morning Pastries | | | | |
| 8:00 AM | 5:00 PM | **VSF01** - Workshop: Service Orientation Technologies: Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro* | | **VSF02** - Workshop: Build a Windows 8.1 Application in a Day - *Philip Japikse* | | **MAF01** - Workshop: Modern App Development In-Depth: iOS, Android, Windows, and Web - *Brent Edwards, Anthony Handley, & Allen Conway* |

*Speakers and sessions subject to change*

# Security and Identity Management with Azure Mobile Services

Bruno Terkaly and Greg Oliver

Satya Nadella has clearly defined the new Microsoft vision statement. He has stated we live in a world that is "Mobile-First/Cloud-First." This is no surprise. Microsoft has been pursuing mobile and cloud technologies for several years now. The industry is changing and the pressure for IT leaders to embrace the bring-your-own-device (BYOD) revolution has never been greater.

Supporting the brave new world of BYOD in the enterprise is a top priority. In this article, we'll help you understand how the cloud can help line-of-business (LOB) application developers create and support Apple iOS, Google Android, and Windows Phone apps. We'll address the device-management challenges facing corporations of all sizes. We'll also cover some of the low-level native Objective-C code you'll need in order to properly manage OAuth 2.0 tokens from Azure Mobile Services. The key capability is saving a token securely on a mobile device so users aren't forced to log in every time they want to use an application.

## Key Characteristics of LOB Apps

There are many issues to consider when creating LOB apps. One of the most important is identity management (see **Figure 1**). We're not just talking about user identity, but also device identity. Those two factors combined is often called compound identity. This makes sense because LOB apps typically have access to confidential corporate resources.

This article discusses:

- Developing apps to support multiple mobile devices
- Managing user and device identity
- Applying layered security measures

Technologies discussed:

Microsoft Azure, Azure Mobile Services, Windows Phone, Android, iOS, Objective-C

## Device Management

One thing is clear about centralized IT: It needs to enforce some type of control over personal devices. In an ideal world, IT can force users to fully domain join their devices. A less-extreme version of control is called a workplace join, which can be thought of as a relaxed version of a fully domain-joined device.

Any IT administrator's goal should be to ensure secure, encrypted communication between corporate resources and the device itself. Installing a device certificate as part of the provisioning process is one way for a corporation to safely and securely manage the device.

## Workplace Provisioning

The first goal of this provisioning process is to authenticate a user against a trusted directory service. Successful device provisioning (or registration) results in a JSON-based token on the device. This token helps ensure secure communication between users and the corporate network (see **Figure 2**).

## Layered Identity

Thinking about a spectrum of security levels is prudent (see **Figure 3**). Some corporate information is simply more confidential. Extra measures are often needed to protect that extra-sensitive data.

Most of us are already familiar with user authentication. At that most basic level of security, a user logs in with his username and password. The second level is device authentication, possibly enforcing a multi-factor authentication approach. By combining the user identity and the device identity (using an SMS challenge, for example), you can then form a compound identity.

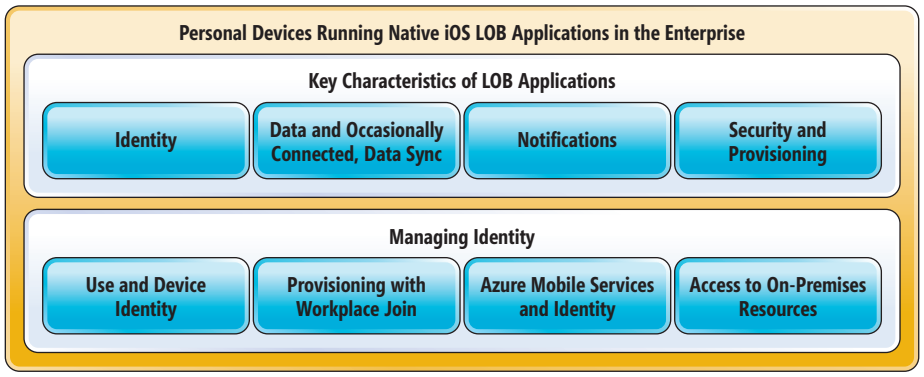



Figure 1 **Overview of iOS Line-of-Business Applications**

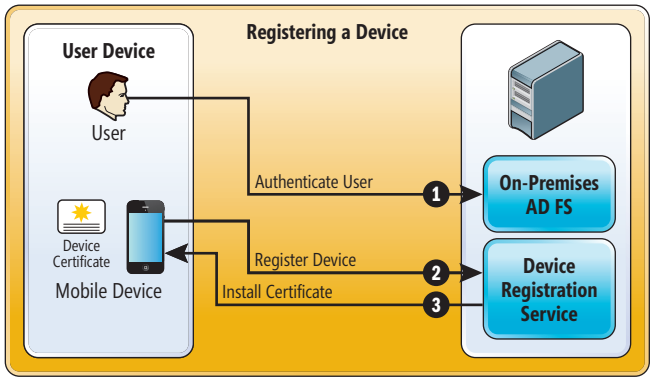



Figure 2 **Workplace Provisioning Helps Authenticate Users**

## Network Location Awareness

Another important concern when running applications on personal devices is Network Location Awareness (NLA). This means when a request comes in for a protected network resource, you can determine whether that request originated from outside the corporate network. NLA provides an extra layer of protection because it helps enforce additional rules, such as multi-factor authentication for requests generated outside the corporate network.

To implement network location transparency typically means you create some sort of proxy Web service in a DMZ. A DMZ is a network that exposes an organization's external-facing services to a larger and untrusted network, like the Internet. You can use these proxies to trigger additional rules and insulate private resources on a network from outside access. Microsoft has specific technologies that enable these scenarios. Learn more about them at bit.ly/1v50JPq.

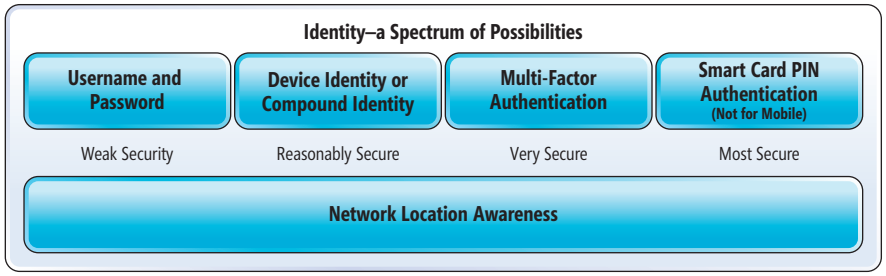



Figure 3 **The Spectrum of Security Levels**

## Support Identity with Native iOS Code

We will illustrate some low-level techniques to leverage native iOS capabilities. Before diving into that, however, we'll provide some content around Azure Mobile Services. Azure Mobile Services provides several commonly required capabilities to client programs. Client programs can be native iOS, Android, Windows Phone, browser-based HTML/JavaScript or Windows.

Any client platform that can call a REST API can leverage Azure Mobile Services. It provides varying levels of library support across those supported clients. Other capabilities include federated identity, access to storage, offline data sync and push notifications. Some capabilities are implemented in the back end, others in the front end. And you can use either the Microsoft .NET Framework or Node.js as the back end.

One of the more challenging capabilities to implement is storing a security token on an iOS device. The token is initially provided by Azure Mobile Services. The OS needs to protect the token and it needs to be encrypted. Luckily, the keychain mechanism built into the iOS SDK provides this capability. It lets you securely store a token on an employee's device. This is a purely native approach, meaning the code isn't really reusable.

There are advantages to storing a security token locally on the device. The most important is that it lets users access the application without logging in every time. Azure Mobile Services applications may leverage any or all of several security token providers or identity providers, such as Active Directory, Azure Active Directory, Facebook, Twitter and others. Each one provides basic authentication, but they're unique in other ways.

One way they're different is the ability to expire tokens. Active Directory and Azure Active Directory allow for this, but Twitter does not. This ability to expire a token is important for authentication that grants access to corporate resources, so Active Directory and Azure Active Directory are ultimately better choices for this level of access than Twitter.

In the spirit of "start simple," we'll describe the process using Twitter authentication in this article, then cover Azure Active Directory in a future article. Xcode is the IDE used to develop iOS applications. It's somewhat analogous to Visual Studio, but less sophisticated and capable. The traditional language to code iOS applications is Objective-C.

To illustrate these techniques, start with the tutorial at bit.ly/1vcxHMQ. Add federated authentication leveraging Twitter identity provider services. The tutorial includes both back-end and front-end modules. You can select either back end (.NET or JavaScript) as you follow along. Take the Objective-C pathway for the front end. We'll borrow some code from another sample project (iOS-LensRocket) on GitHub to support the KeyChain feature.

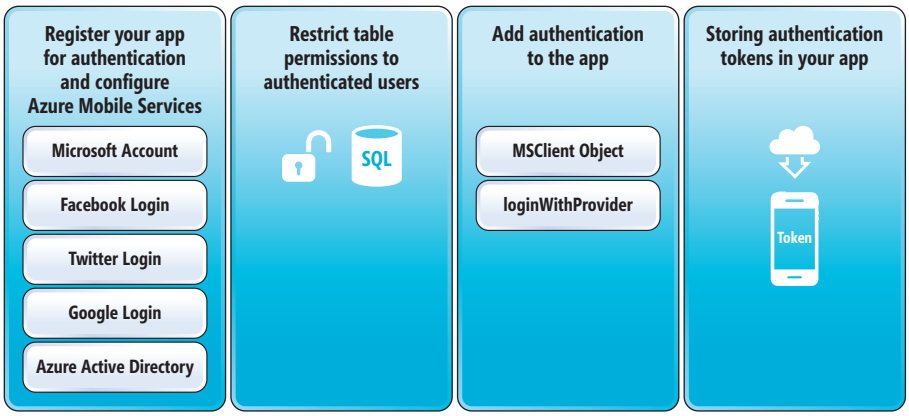



Figure 4 **High-Level Steps for Storing Authentication Tokens on an iOS Device**

Not many changes were needed to the front-end Xcode project provided by Azure Mobile Services. All we need to do is add the KeyChain code and security features from the security.framework library. When a user starts the application, basic token management logic is applied. The application starts by checking for a token by making a call into the KeyChain API. If the token is present, it's retrieved and used. The user won't be asked to log in to Twitter again.

Here's a summary of the code changes that we'll make in the Xcode IDE:

- Link to an additional library (find more guidance at bit.ly/1x7Ajzz)
- Add a few files from iOS-LensRocket to support KeyChain functionality
- Modify the controller code

**Figure 4** demonstrates how we'll use Azure Mobile Services and native iOS code to control access to a SQL database using a token cached on the device.

The tutorial will get you to the point of running a native iOS application that can read and write to a SQL database hosted in the cloud. You'll have a working Xcode project written in Objective-C. However, it's missing authentication—we'll explain that later. The existing implementation uses an application key provided by Azure Mobile Services:

```
// Initialize the Mobile Services client with your URL and key.
MSClient *client =
  [MSClient clientWithApplicationURLString:@"https://
msdnmagazine.azure-mobile.net/"
applicationKey:@"cctyYudYfxuczeRDMPcDBRCxDrOUIM73"];
```

The concern for this approach is obvious. Anytime you embed secrets into your application code, you expose yourself to security risks. What we really want to do is force the user to log in to perform database updates. However, we don't want the user to have to log in constantly, so we'd like to store the authentication token on the device locally. We'll configure the iOS app to retrieve the token when the application starts.

Most enterprise scenarios won't embrace socially based identity providers, such as Facebook or Twitter. Nevertheless, it's a useful place to start because we can demonstrate some of the core concepts for later leveraging on-premises identities with Active Directory. Also, many business applications leverage relational databases and Azure Mobile Services offers pre-built functionality to support them.

To get things started, you'll have to do some basic tasks at the Azure Management Portal and the third-party identity provider (which in this case is Twitter). Create application metadata in the Twitter portal and the related application metadata in the Azure Mobile Services portal. Then get an application URL. Copy the application URL to Twitter and copy the Twitter API keys to Azure Mobile Services. This is how you bind an identity provider to an Azure Mobile Services application (see **Figure 5**).

This essentially involves creating mobile application metadata in the portal following these steps:

1. Get URL for mobile app you created from Azure Management Portal
2. Sign in and go to dev.twitter.com/apps/new
3. Create a new Twitter app
4. Get the API Key and API Secret for your Twitter app
5. Return to Azure Management Portal
6. Paste in API Key and API Secret from Twitter

## Portal Work

As mentioned earlier, there are a couple of approaches to building your back end. If you select the .NET Framework, you'll have great control in Web API over use case authorization for all users, whether

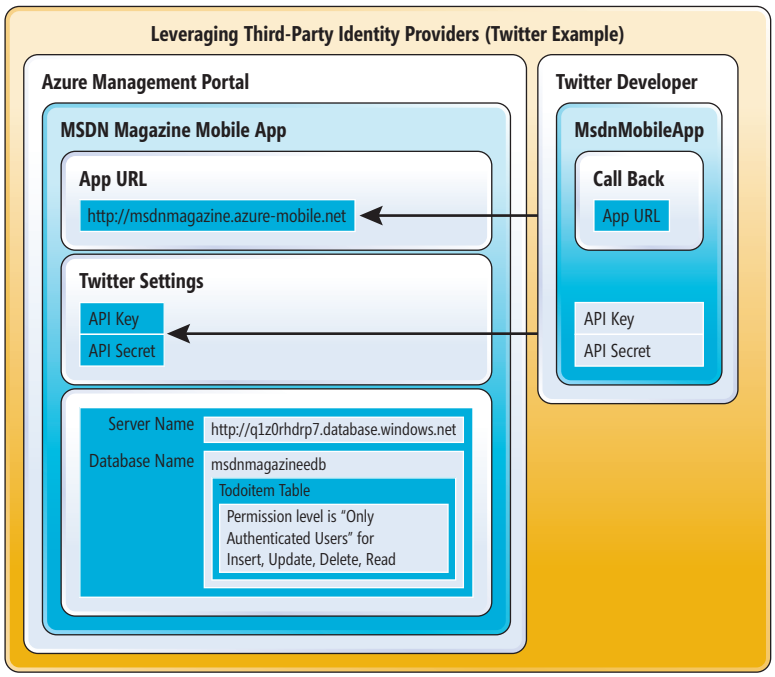



Figure 5 **The Relationship Between Azure Mobile Services and Twitter**

they're authenticated or not. If you selected Node.js, the portal provides declarative and scripted approaches.

What we really need to do is go back to the Xcode IDE running on the Mac. We'll use it to edit and compile the source code of the LOB iOS app that will run on the mobile device.

We'll make the following code changes in the Xcode IDE:

- Add a library reference to security.framework.
- Copy these four files from iOS-LensRocket at bit.ly/1pqvQ25:
  - KeyChainWrapper.h
  - KeyChainWrapper.m
  - LensRocketConstants.h
  - LensRocketConstants.m

Figure 6 **Changes Made to Native iOS Code**

```
#import "KeychainWrapper.h"  // Add this line.

- (void)viewDidLoad
{
  // Code omitted for brevity.
  [self loadAuthInfo];  // Add this line.
}

// Code omitted for brevity.

////////////////////////////////////////////////////////////////////////////
//   Added viewDidAppear(), loadAuthInfo() below.
////////////////////////////////////////////////////////////////////////////

// ----------------------------------------------------------------------
// Event is viewDidAppear. When app becomes active, check credentials.
//
- (void)viewDidAppear:(BOOL)animated
{
  ////////////////////////////////////////////////////////////////////////
  //  Check if current user is logged in. If not, log in to Twitter.

  MSClient *client = self.todoService.client;
  if(client.currentUser != nil)   {
    return;
  }
  [client loginWithProvider:@"twitter" controller:self animated:YES
    completion:^(MSUser *user, NSError *error) {
    [self saveAuthInfo];
    [self refresh];
  }];

}
// ----------------------------------------------------------------------
// Save token locally.
// Leverage built-in iOS features called Keychain for user ID and tokenl storage
//
- (void) saveAuthInfo{
  // Save the userId
  [KeychainWrapper createKeychainValue:self.todoService.client.currentUser.userId
    forIdentifier:@"userid"];
  // Save the token itself
  [KeychainWrapper
    createKeychainValue:self.todoService.client.currentUser.
      mobileServiceAuthenticationToken
  forIdentifier:@"token"];
}

// ----------------------------------------------------------------------
// Load the user ID and Token from the keychain. Avoid forcing user to log in again.
//
- (void)loadAuthInfo {
  NSString *userid =
    [KeychainWrapper keychainStringFromMatchingIdentifier:@"userid"];
  if (userid) {
    // Fetch the user ID and token from the Keychain.
    NSLog(@"userid: %@", userid);
    self.todoService.client.currentUser = [[MSUser alloc] initWithUserId:userid];
    self.todoService.client.currentUser.mobileServiceAuthenticationToken =
      [KeychainWrapper keychainStringFromMatchingIdentifier:@"token"];
  }
}
```

There are three key functions/methods involved. The first is viewDidLoad, where we added a call to loadAuthInfo. This loads into memory a previously saved token from an earlier successful login. Here is where we'll have to load the token from disk instead of forcing the user to log in again (see **Figure 6**).

We also added the viewDidAppear function, which gets called when the application becomes visible or active. We use the MSClient object to check to see if the user is logged in. If the user isn't logged in, we force a Twitter login and then save the credentials (token) to disk using the Keychain capabilities. This would more properly be implemented in viewDidLoad.

The saveAuthInfo and loadAuthInfo methods simply use the Keychain code to save and load the token to and from disk on the mobile device. You can read more about Keychain services at bit.ly/1qqcNFm.

You can follow a second tutorial on implementing authentication at bit.ly/1B3AvOk. You'd follow a similar process to allow for federation to Azure Active Directory, as opposed to Twitter:

1. Get URL for mobile app you created from the Azure Management Portal
2. Go to your Active Directory pages (also in the Azure Management Portal)
3. Create a new application
4. Get the Client ID and key
5. Return to mobile app page in Azure Management Portal
6. Paste in Client ID and key from Active Directory

We'll go into more detail around federating to both Active Directory on-premises and Azure Active Directory in the cloud in a future article. In the meantime, you can see some great code samples at bit.ly/1slQMz2 and bit.ly/1mK1LQF.

## Wrapping Up

As the pressure to support personal devices at the workplace continues to grow, it's important for developers to better understand how to secure those devices and work with tokens. Implementing a username/password approach to authentication is often insufficient for truly confidential data.

To fully leverage all security capabilities on a device, you'll often need to access native capabilities, as seen with iOS Keychain services. In upcoming articles, we'll continue to explore the challenges facing IT leaders around mobile devices in the enterprise, focusing on identity, data synchronization and push notifications. ■

**Bruno Terkaly** *is a developer evangelist for Microsoft. His depth of knowledge comes from years of experience in the field, writing code using a multitude of platforms, languages, frameworks, SDKs, libraries and APIs. He spends time writing code, blogging and giving live presentations on building cloud-based applications, specifically using the Microsoft Azure platform. You can read his blog at blogs.msdn.com/b/brunoterkaly.*

**Greg Oliver** *joined the Microsoft Azure ISV organization in 2010. He spends most of his time helping companies with their migration plans and implementations. Most recently, he has been working with a popular consumer games company with their migration from a competing cloud provider. Prior to joining Microsoft, he was a technology partner in a startup company.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Matthew Henderson*

# Application Analysis with Pin

Hadi Brais

Program analysis is a fundamental step in the development process. It involves analyzing a program to determine how it will behave at run time. There are two types of program analysis: static and dynamic.

You'd perform a static analysis without running the target program, usually during source code compilation. Visual Studio provides a number of excellent tools for static analysis. Most modern compilers automatically perform static analysis to ensure the program honors the language's semantic rules and to safely optimize the code. Although static analysis isn't always accurate, its main benefit is pointing out potential problems with code before you run it, reducing the number of debugging sessions and saving precious time.

You'd perform a dynamic program analysis while running the target program. When the program ends, the dynamic analyzer produces a profile with behavioral information. In the Microsoft .NET Framework, the just-in-time (JIT) compiler performs dynamic analysis at run time to further optimize the code and ensures it won't do anything that violates the type system.

The primary advantage of static analysis over dynamic analysis is it ensures 100 percent code coverage. To ensure such high code coverage with dynamic analysis, you usually need to run the program many times, each time with different input so the analysis takes different paths. The primary advantage of dynamic analysis is it can produce detailed and accurate information. When you develop and run a .NET application or secure C++ application, both kinds of analysis will be automatically performed under the hood to ensure that the code honors the rules of the framework.

The focus in this article will be on dynamic program analysis, also known as profiling. There are many ways to profile a program, such as using framework events, OS hooks and dynamic instrumentation. While Visual Studio provides a profiling framework, its dynamic instrumentation capabilities are currently limited. For all but the simplest dynamic instrumentation scenarios, you'll need a more advanced framework. That's where Pin comes into play.

This article discusses:

- Performing dynamic application analysis
- Developing a Pintool to analyze applications
- Properly configuring a Pintool and integrating with Visual Studio 2012

Technologies discussed:

Intel Pin, Visual Studio 2012

## What Is Pin?

Pin is a dynamic binary instrumentation framework developed by Intel Corp. that lets you build program analysis tools called Pintools for Windows and Linux platforms. You can use these tools to monitor and record the behavior of a program while it's running. Then you can effectively evaluate many important aspects of the program such as its correctness, performance and security.

You can integrate the Pin framework with Microsoft Visual Studio to easily build and debug Pintools. In this article, I'll show how to use Pin with Visual Studio to develop and debug a simple yet useful Pintool. The Pintool will detect critical memory issues such as memory leaking and double freeing allocated memory in a C/C++ program.

To better understand the nature of Pin, look at the complete definition term by term:

- A framework is a collection of code upon which you write a program. It typically includes a runtime component that partially controls program execution (such as startup and termination).
- Instrumentation is the process of analyzing a program by adding or modifying code—or both.
- Binary indicates the code being added or modified is machine code in binary form.
- Dynamic indicates the instrumentation processes are performed at run time, while the program is executing.

The complete phrase "dynamic binary instrumentation" is a mouthful, so people usually use the acronym DBI. Pin is a DBI framework.

You can use Pin on Windows (IA32 and Intel64), Linux (IA32 and Intel64), Mac OS X (IA32 and Intel64) and Android (IA32). Pin also supports the Intel Xeon Phi microprocessor for supercomputers. It not only supports Windows, but also seamlessly integrates with Visual Studio. You can write Pintools in Visual Studio and debug them with the Visual Studio Debugger. You can even develop debugging extensions for Pin to use seamlessly from Visual Studio.

## Get Started with Pin

Although Pin is proprietary software, you can download and use it free of charge for non-commercial use. Pin doesn't yet support Visual Studio 2013, so I'll use Visual Studio 2012. If you've installed both Visual Studio 2012 and 2013, you can create and open Visual Studio 2012 projects from 2013 and use the C++ libraries and tools of Visual Studio 2012 from 2013.

There are many ways to profile a program, such as using framework events, OS hooks and dynamic instrumentation.

Download Pin from intel.ly/1ysiBs4. Besides the documentation and the binaries, Pin includes source code for a large collection of sample Pintools you'll find in source/tools. From the MyPinTool folder, open the MyPinTool solution in Visual Studio.

Examine the project properties in detail to determine the proper Pintool configuration. All Pintools are DLL files. Therefore, the project Configuration Type should be set to Dynamic Library (.dll). You'll also have to specify all headers, files, libraries and a number of preprocessor symbols required by the Pin header files. Set the entry point to Ptrace_DllMainCRTStartup%4012 to properly initialize the C runtime. Specify the /export:main switch to import the main function.

You can either use the properly configured MyPinTool project or create a new project and configure it yourself. You can also create a property sheet containing the required configuration details and import that into your Pintool project.

## Pin Granularity

Pin lets you insert code into specific places in the program you're instrumenting—typically just before or after executing a particular instruction or function. For example, you might want to record all dynamic memory allocations to detect memory leaks.

Pin is a dynamic binary instrumentation framework developed by Intel Corp.

There are three main levels of granularity to Pin: routine, instruction and image. Pin also has one more not-so-obvious level—trace granularity. A trace is a straight-line instruction sequence with exactly one entry. It usually ends with an unconditional branch. A trace may include multiple exit points as long as they're conditional. Examples of unconditional branches include calls, returns and unconditional jumps. Note that a trace has exactly one entry point. If Pin detected a branch to a location within a trace, it will end that trace at that location and start a new trace.

Pin offers these instrumentation granularities to help you choose the appropriate trade-off between performance and level of detail. Instrumenting at the instruction level might result in severe performance degradation, because there could be billions of instructions. On the other hand, instrumenting at the function level might be too general and, therefore, it might increase the complexity of the analysis code. Traces help you instrument without compromising performance or detail.

## Write a Pintool

Now it's time to write a useful Pintool. The purpose of this Pintool example is to detect memory deallocation problems common to C/C++ programs. The simple Pintool I'm going to write can diagnose an existing program without having to modify the source code or recompile it, because Pin performs its work at run time. Here are the problems the Pintool will detect:

- Memory leaks: Memory allocated, but not freed.
- Double freeing: Memory deallocated more than once.
- Freeing unallocated memory: Deallocating memory that hasn't been allocated (such as calling free and passing NULL to it).

To simplify the code, I'll assume the following:

- The main function of the program is called main. I won't consider other variants.

Figure 1 **The RecordMalloc Analysis Routine Called Every Time Malloc Returns**

```
VOID RecordMalloc(ADDRINT addr) {
  if (!Record) return;
  if (addr == NULL) {
    cerr << "Heap full!";
    return;
  }

  map<ADDRINT, bool>::iterator it = MallocMap.find(addr);
  if (it != MallocMap.end()) {
    if (it->second) {
      // Allocating a previously allocated and freed memory.
      it->second = false;
    }
    else {
      // Malloc should not allocate memory that has
      // already been allocated but not freed.
      cerr << "Imposible!" << endl;
    }
  }
  else {
    // First time allocating at this address.
    MallocMap.insert(pair<ADDRINT, bool>(addr, false));
  }
}
```

- The only functions that allocate and free memory are new/malloc and delete/free, respectively. I won't consider calloc and realloc, for example.
- The program consists of one executable file.

Once you understand the code, you can modify it and make the tool much more practical.

## Define the Solution

To detect those memory problems, the Pintool must monitor calls to the allocation and deallocation functions. Because the new operator calls malloc internally, and the delete operator calls free internally, I can just monitor the calls to malloc and free.

Whenever the program calls malloc, I'll record the returned address (either NULL or the address of the allocated memory region). Whenever it calls free, I'll match the address of the memory being freed with my records. If it has been allocated but not freed, I'll mark it as freed. However, if it has been allocated and freed, that would be an attempt to free it again, which indicates a problem. Finally, if there's no record the memory being freed has been allocated, that would be an attempt to free unallocated memory. When the program terminates, I'll again check records for those memory regions that have been allocated but not freed to detect memory leaks.

## Choose a Granularity

Pin can instrument a program at four granularities: image, routine, trace and instruction. Which is best for this Pintool? While any of the granularities will do the job, I need to choose the one that incurs the least performance overhead. In this case, the image granularity would be the best. Once the image of the program is loaded, the Pintool can locate the malloc and free code within the image and insert the analysis code. This way, instrumentation overhead will be per-image instead of, for exmple, per-instruction.

To use the Pin API, I must include the pin.H header file in the code. The Pintool will be writing the results to a file, so I also have to include the fstream header file. I'll use the map STL type to keep track of the memory being allocated and deallocated. This type is defined in the map header file. I'll also use the cerr stream to show informative messages:

```
#include "pin.H"
#include <iostream>
#include <fstream>
#include <map>
```

I will define three symbols to hold the names of the functions malloc, free and main:

```
#define MALLOC "malloc"
#define FREE "free"
#define MAIN "main"
```

These are the required global variables:

```
bool Record = false;
map<ADDRINT, bool> MallocMap;
ofstream OutFile;
string ProgramImage;
KNOB<string> OutFileName(KNOB_MODE_WRITEONCE, "Pintool", "o", "memtrace.txt",
  "Memory trace file name");
```

The Record variable indicates whether I'm inside the main function. The MallocMap variable holds the state of each allocated memory region. The ADDRINT type is defined by pin.H and represents a memory address. If the value associated with a memory address it TRUE, it has been deallocated.

The ProgramImage variable holds the name of the program image. The last variable is a KNOB. This represents a command-line switch to the Pintool. Pin makes it easy to define switches for a Pintool. For each switch, define a KNOB variable. The template type parameter string represents the type of the values that the switch will take. Here, the KNOB lets you specify the name of the output file of the Pintool through the "o" switch. The default value is memtrace.txt.

> Pin lets you insert code into specific places in the program you're instrumenting—typically just before or after executing a particular instruction or function.

Next, I have to define the analysis routines executed at specific points in the code sequence. I need an analysis function, as defined in **Figure 1**, called just after malloc returns to record the address of the allocated memory. This function takes the address returned by malloc and returns nothing.

This function will be called every time malloc is called. However, I'm only interested in the memory if it's part of the instrumented program. So I'll record the address only when Record is TRUE. If the address is NULL, I'll just ignore it.

Then the function determines whether the address is already in MallocMap. If it is, then it must have been previously allocated and deallocated and, therefore, it's now being reused. If the address isn't in MallocMap, I'll insert it with FALSE as the value indicating it hasn't been freed.

Figure 2 **The RecordFree Analysis Routine**

```
VOID RecordFree(ADDRINT addr) {
  if (!Record) return;

  map<ADDRINT, bool>::iterator it = MallocMap.find(addr);
  if (it != MallocMap.end()) {
    if (it->second) {
      // Double freeing.
      OutFile << "Object at address " << hex << addr << "
        has been freed more than once."  << endl;
    }
    else {
      it->second = true; // Mark as freed.
    }
  }
  else {
    // Freeing unallocated memory.
    OutFile << "Freeing unallocated memory at " << hex << addr << "." << endl;
  }
}
```

I'll define another analysis routine, shown in **Figure 2**, that I'll have called just before free is called to record the address of the memory region being freed. Using MallocMap, I can easily detect if the memory being freed has already been freed or it hasn't been allocated.

Next, I'll need two more analysis routines to mark the execution and return of the main function:

```
VOID RecordMainBegin() {
  Record = true;
}

VOID RecordMainEnd() {
  Record = false;
}
```

Analysis routines determine the code to instrument the program. I also have to tell Pin when to execute these routines. That's the purpose of instrumentation routines. I defined an instrumentation routine as shown in **Figure 3**. This routine is called every time an image is loaded in the running process. When the program image is loaded, I'll tell Pin to insert the analysis routines at the appropriate points.

Figure 3 **The Image Instrumentation Routine**

```
VOID Image(IMG img, VOID *v) {
  if (IMG_Name(img) == ProgramImage) {
    RTN mallocRtn = RTN_FindByName(img, MALLOC);
    if (mallocRtn.is_valid()) {
      RTN_Open(mallocRtn);
      RTN_InsertCall(mallocRtn, IPOINT_AFTER, (AFUNPTR)RecordMalloc,
        IARG_FUNCRET_EXITPOINT_VALUE,
        IARG_END);
      RTN_Close(mallocRtn);
    }

    RTN freeRtn = RTN_FindByName(img, FREE);
    if (freeRtn.is_valid()) {
      RTN_Open(freeRtn);
      RTN_InsertCall(freeRtn, IPOINT_BEFORE, (AFUNPTR)RecordFree,
        IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
        IARG_END);
      RTN_Close(freeRtn);
    }

    RTN mainRtn = RTN_FindByName(img, MAIN);
    if (mainRtn.is_valid()) {
      RTN_Open(mainRtn);
      RTN_InsertCall(mainRtn, IPOINT_BEFORE, (AFUNPTR)RecordMainBegin,
        IARG_END);
      RTN_InsertCall(mainRtn, IPOINT_AFTER, (AFUNPTR)RecordMainEnd,
        IARG_END);
      RTN_Close(mainRtn);
    }
  }
}
```

The IMG object represents the executable image. All Pin functions that operate at the image level start with IMG_*. For example, IMG_Name returns the name of the specified image. Similarly, all Pin functions that operate at the routine level start with RTN_*. For example, RTN_FindByName accepts an image and a C-style string and returns an RTN object representing the routine for which I'm looking. If the requested routine is defined in the image, the returned RTN object would be valid. Once I find the malloc, free and main routines, I can insert analysis routines at the appropriate points using the RTN_InsertCall function.

This function accepts three mandatory arguments followed by a variable number of arguments:

- The first is the routine I want to instrument.
- The second is an enumeration of type IPOINT that specifies where to insert the analysis routine.
- The third is the analysis routine to be inserted.

Then I can specify a list of arguments to be passed to the analysis routine. This list must be terminated by IARG_END. To pass the return value of the malloc function to the analysis routine, I'll specify IARG_FUNCRET_EXITPOINT_VALUE. To pass the argument of the free function to the analysis routine, I'll specify IARG_FUNCARG_ENTRYPOINT_VALUE followed by the index of the argument of the free function. All these values starting with IARG_* are defined by the IARG_TYPE enumeration. The call to RTN_InsertCall has to be wrapped by calls to RTN_Open and RTN_Close so the Pintool can insert the analysis routines.

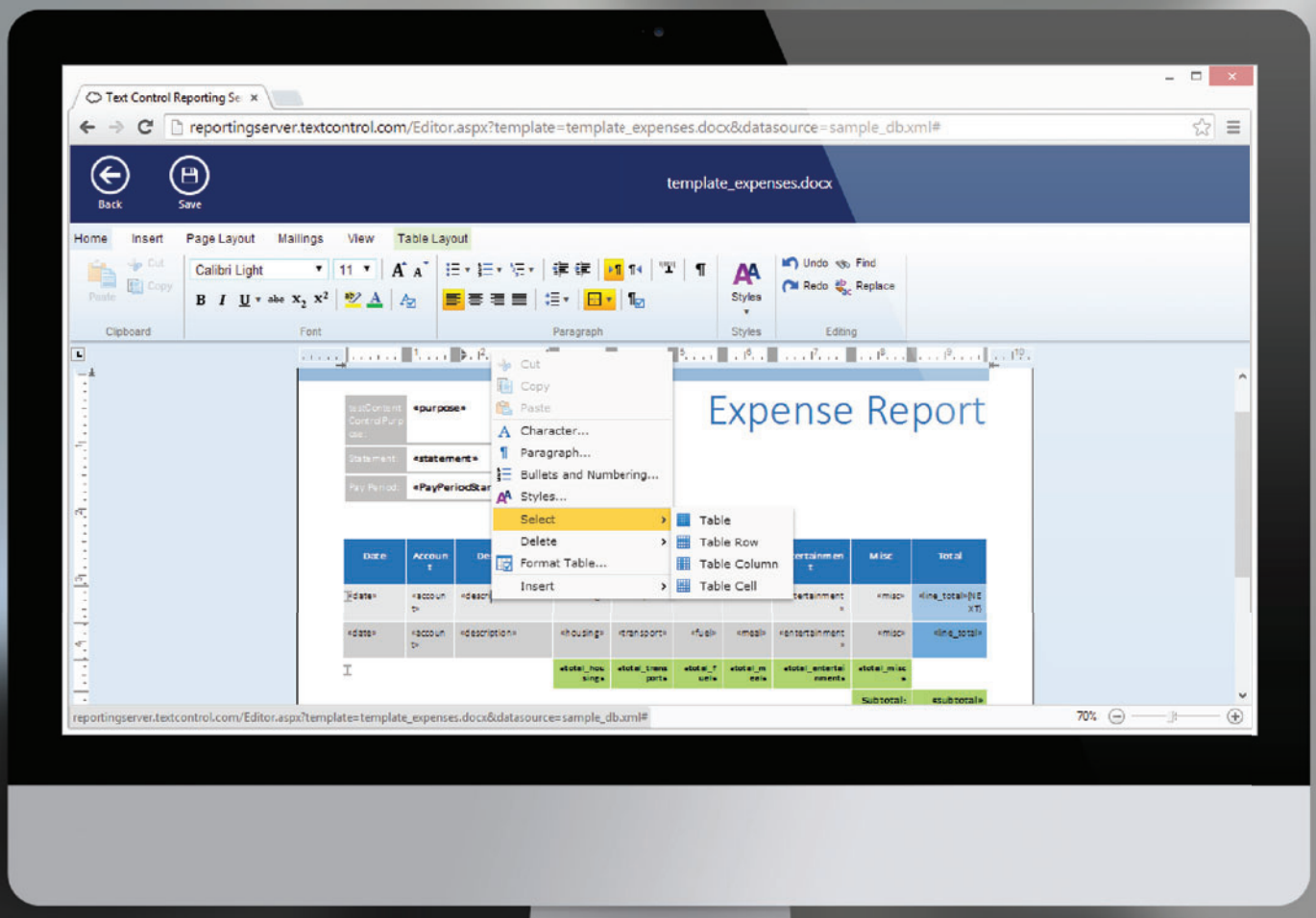
Now that I've defined my analysis and instrumentation routines, I'll have to define a finalization routine. This will be called upon termination of the instrumented program. It accepts two arguments, one being the code argument that holds the value returned from the main function of the program. The other will be discussed later. I've used a range-based for loop to make the code more readable:

```
VOID Fini(INT32 code, VOID *v) {
  for (pair<ADDRINT, bool> p : MallocMap) {
    if (!p.second) {
      // Unfreed memory.
      OutFile << "Memory at " << hex << p.first << "
        allocated but not freed." << endl;
    }
  }
  OutFile.close();
}
```

All I have to do in the finalization routine is to iterate over MallocMap and detect those allocations that haven't been freed. The return from Fini marks the end of the instrumentation process.

The last part of the code is the main function of the Pintool. In the main function, PIN_Init is called to have Pin parse the command line to initialize the Knobs. Because I'm searching for functions using their names, PIN has to load the symbol table of the program image. I can do this by calling PIN_InitSymbols. The function IMG_AddInstrumentFunction registers the instrumentation function Image to be called every time an image is loaded.

Also, the finalization function is registered using PIN_AddFiniFunction. Note that the second argument to these functions is passed to the v parameter. I can use this parameter to pass any additional information to instrumentation functions. Finally,




Cross-browser, cross-platform document and template editing

# HTML5-BASED TX TEXT CONTROL

The first true WYSIWYG, HTML5-based Web editor and reporting template designer for ASP.NET.

Give your users an MS Word compatible editor to create powerful reporting templates anywhere - in any browser on any device.

Download your 30-day trial version today:
www.textcontrol.com/html5

Visual Studio Partner

the new TEXT CONTROL

PIN_StartProgram is called to start the program I'm analyzing. This function actually never returns to the main function. Once it's called, Pin takes over everything:

```
int main(int argc, char *argv[]) {
  PIN_Init(argc, argv);
  ProgramImage = argv[6]; // Assume that the image name is always at index 6.
  PIN_InitSymbols();
  OutFile.open(OutFileName.Value().c_str());
  IMG_AddInstrumentFunction(Image, NULL);
  PIN_AddFiniFunction(Fini, NULL);
  PIN_StartProgram();
  return 0;
}
```

Assembling all these pieces of code constitutes a fully functional Pintool.

## Run the Pintool

You should be able to build this project without any errors. You'll also need a program to test the Pintool. You can use the following test program:

```
#include <new>
void foo(char* y) {
  int *x = (int*)malloc(4);
}

int main(int argc, char* argv[]) {
  free(NULL);
  foo(new char[10]);
  return 0;
}
```

Clearly, this program is suffering from two memory leaks and one unnecessary call to free, indicating a problem with the program logic. Create another project that includes the test program. Build the project to produce an EXE file.

The final step to run the Pintool is to add Pin as an external tool to Visual Studio. From the Tools menu, select External tools. A dialog box will open as shown in **Figure 4**. Click the Add button to add a new external tool. The Title should be Pin and the Command should be the directory of the pin.exe file. The Arguments include the arguments to be passed to pin.exe. The -t switch specifies the Pintool directory. Specify the program to be instrumented after the two hyphens. Click OK and you should be able to run Pin from the Tools menu.

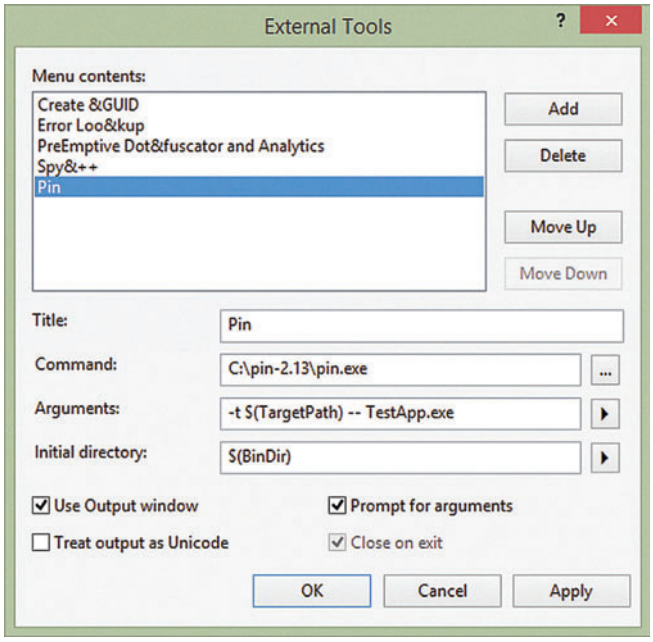



Figure 4 **Add Pin to Visual Studio Using the External Tools Dialog Box**

While running the program, the Output window will print anything you throw in the cerr and cout streams. The cerr stream usually prints informative messages from Pintool during execution. Once Pin terminates, you can view the results by opening the file the Pintool has created. By default, this is called memtrace.txt. When you open the file, you should see something like this:

```
Freeing unallocated memory at 0.
Memory at 9e5108 allocated but not freed.
Memory at 9e5120 allocated but not freed.
```

If you have more complex programs that adhere to the Pintool assumptions, you should instrument them using the Pintool, as you might find other memory issues of which you were unaware.

> You should be able to build this project without any errors.

## Debug the Pintool

When developing a Pintool, you'll stumble through a number of bugs. You can seamlessly debug it with the Visual Studio Debugger by adding the -pause_tool switch. The value of this switch specifies the number of seconds Pin will wait before it actually runs the Pintool. This lets you attach the Visual Studio Debugger to the process running the Pintool (which is the same as the process running the instrumented program). Then you can debug your Pintool normally.

The Pintool I've developed here assumes the name of the image is at index 6 of the argv array. So if you want to add the pause-tool switch, the image name will be at index 8. You can automate this by writing a bit more code.

## Wrapping Up

To further develop your skills, you can enhance the Pintool so it can detect other kinds of memory problems such as dangling pointers and wild pointers. Also, the Pintool output isn't very useful because it doesn't point out which part of the code is causing the problem. It would be nice to print the name of the variable causing the problem and the name of the function in which the variable is declared. This would help you easily locate and fix the bug in the source code. While printing function names is easy, printing variable names is more challenging because of the lack of support from Pin.

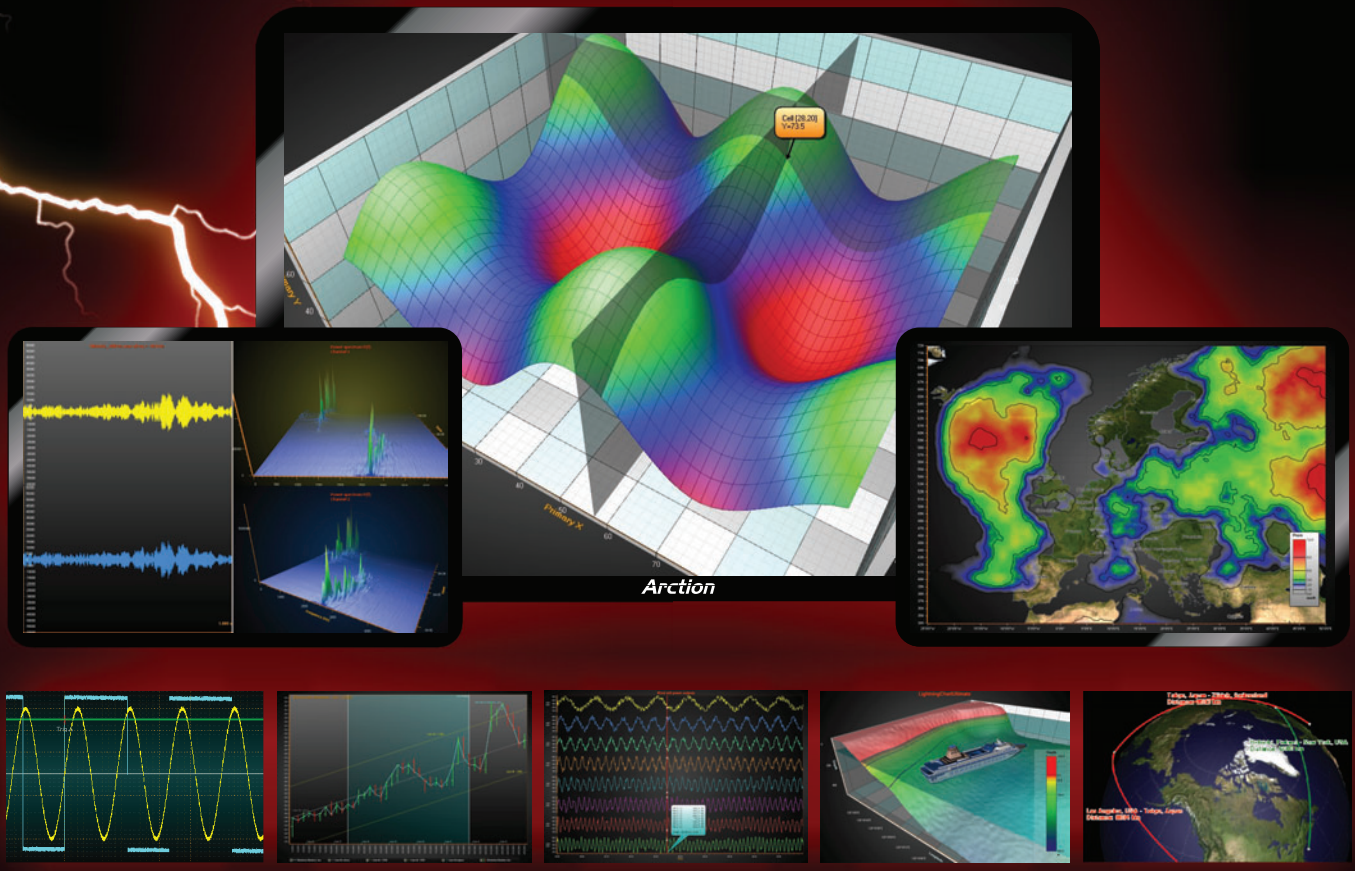
There are a lot of interactions happening between Pin, the Pintool and the instrumented program. It's important to understand these interactions when developing advanced Pintools. For now, you should work through the examples provided with Pin to gain a better understanding of its power. ■

**Hadi Brais** *is a Ph.D. scholar at the Indian Institute of Technology Delhi (IITD), researching optimizing compiler design for the next-generation memory technology. He spends most of his time writing code in C/C++/C# and digging deep into the CLR and CRT. He blogs at hadibrais.wordpress.com. Reach him at hadi.b@live.com.*

**Thanks** to the following technical expert for reviewing this article: *Preeti Ranjan Panda*

The fastest rendering data visualization components
for WPF and WinForms...

# LightningChart






***HEAVY-DUTY DATA VISUALIZATION TOOLS FOR SCIENCE, ENGINEERING AND TRADING***

**WPF charts performance comparison**

| | |
|---|---|
| Opening large dataset | LightningChart is up to 977,000 % faster |
| Real-time monitoring | LightningChart is up to 2,700,000 % faster |

**Winforms charts performance comparison**

| | |
|---|---|
| Opening large dataset | LightningChart is up to 37,000 % faster |
| Real-time monitoring | LightningChart is up to 2,300,000 % faster |

*Results compared to average of other chart controls. See details at www.LightningChart.com/benchmark. LightningChart results apply for Ultimate edition.*

- Entirely DirectX GPU accelerated
- Superior 2D and 3D rendering performance
- Optimized for real-time data monitoring
- Touch-enabled operations
- Supports gigantic data sets
- On-line and off-line maps
- Great customer support
- Compatible with Visual Studio 2005...2013

Download a free 30-day evaluation from

**www.LightningChart.com**

Arction

Pioneers of high-performance data visualization

# Load Testing Made Easy with Microsoft Azure and Visual Studio Online

Charles Sterling

The Internet is the great equalizer in the business world. It helps the smallest companies compete with the largest corporations in the world. There is as much opportunity as there is risk. Applications must perform on time all the time. Downtime equates directly to lost revenue. To help development teams understand the performance characteristics of their apps under duress and heavy usage, Microsoft released load testing tools with Visual Studio Team System in 2005.

Even before that release, Visual Studio load testing was one of the most widely internally used test tools. Which begs the question, regardless of the tool used, why didn't load testing gain the same acceptance by the world at large? Turns out load testing isn't done by most development teams due to the effort of setting up and managing the infrastructure required. That's an unnecessary risk.

With Visual Studio 2013, Microsoft released cloud-based load testing as part of Visual Studio Online. This solves the challenge of using Microsoft Azure to automatically provision and host a load testing infrastructure. With Visual Studio 2013 Update 3, cloud load testing in Visual Studio Online was further enhanced with Application Insights integration. This helps demonstrate internal diagnostics such as performance counters, exceptions and objects allocated during the run.

This article discusses:

- Effective load testing techniques
- Performing cloud-based load testing
- Integrating results with Application Insights

Technologies discussed:

Microsoft Azure, Application Insights, Visual Studio Online

## Create a Load Test

If you aren't familiar with the load testing process in Visual Studio, you should start with a Web Performance and Load Test Project. Do this from the File | New Project menu. Select the Web Performance Test and Load Test to create one or more Web Tests.

## Data Binding Web Tests for Cloud-Based Load Testing

While a static Web test can give you detailed performance information about your application, you'll be able to derive more insight from a dynamic Web test. These dynamic load tests typically supply different values for each run. For example, instead of sending the search term "Frog" thousands of times, it would make more sense to use other likely search terms and names that test boundary conditions (such as "Toad," "Frog" and so on). You can accomplish this with an advanced technique called data binding.

To data bind a Web test, you'll need to add a data source to your Web test from the Web Test toolbar. This type of dynamic test will support XML, CSV and database as data sources. Keep in mind, for XML and CSV data sources the entire file is copied to the agent virtual machine. Also, for cloud-based load tests, the test agent needs to be able to connect to the data source. Therefore, the database data source needs to be running on Microsoft Azure SQL Database.

The properties of the data source you're using for the Web test determine how the test will traverse the data source: Do Not Move, Random, Sequential or Unique. Note that the Unique property doesn't yet work with cloud-based load tests. After you've added a data source, the value property of your query string parameter (previously the static string "Frog") will now be a dropdown menu item. This will help you to bind the test to a particular column of your data source.

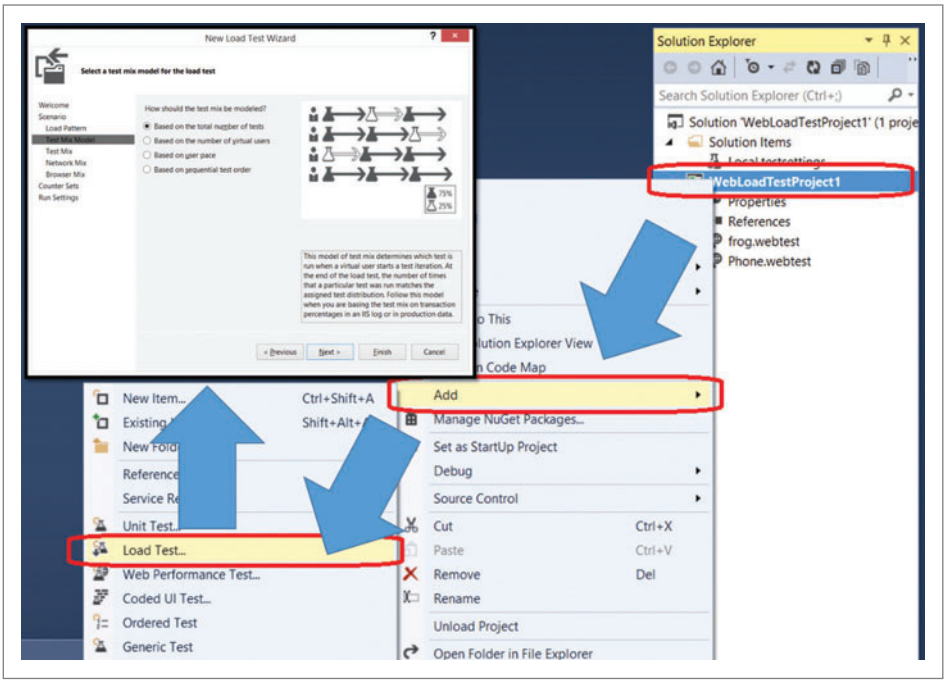



Figure 1 **Add Load Test to a Project**

To drive these Web Performance Tests, add a load test to the project (see **Figure 1**). Open Solution Finder, right-click on your project (mine is named WebLoadTestProject1), select Add, then Load Test and follow the directions in the New Load Test Wizard. During the load test, the system collects information such as how many users, duration, the type of network warm-up time, test distribution and so on. Another option is to create a dynamic load test using data binding (see "Data Binding Web Tests for Cloud-Based Load Testing").

## Put Azure to Work

Typically, the real work begins after you create the load test. At this point, the test team needs computers to host the Test Agents that actually run the load test. For example, if you want to put a 5,000-user load on your Web site and you can support 500 users per computer running the required tests, then your test team will need to acquire, provision and manage an additional 10 computers.

However, using the option in the Local.TestingSettings file, Visual Studio Online cloud load test can take care of all this for you. By default, cloud load test will automatically start an Azure-hosted agent for every 500 users. Each agent has two cores.

You can override this behavior through a new Run Settings property called Agent Count (Total Cores). The current implementation of cloud load test limits you to a maximum of 10 Agents or 20 cores. Cloud load test also requires at least 25 users per core. So while you can set the Agent Count (Total Cores) property to 10 for a load with only 10 users, cloud load test would only allocate one Agent and inform you of this with the following message:

> "This load test will run using one agent cores. Agent Core Count value has been modified from '10' to 'one' as a load test optimally needs '25' virtual users per core. Learn more about it here at go.microsoft.com/fwlink/?LinkID=310093."

For tests that incorporate more than 5,000 users, the users are distributed across the 10 agents and 20 cores.

## Understand Your Load Test

You'll typically start your load testing efforts by asking yourself questions the load test should answer like:

- Is my application as fast as I expect?
- Is my performance getting better or worse?
- Is my code generating errors under load?

To answer these questions, Visual Studio load test provides several different reports. It will then automatically store them on the run data for both you and your team in Visual Studio Online.

For example, using the Throughput report and having that displayed during the load test execution, you'll be able to tell at a glance if your application is running as expected (see **Figure 2**). Unexpectedly fast response times coupled with an unexpected number of failures would suggest a systemic failure (like a service being down or an incorrect server name). Then you can abort your load test.

Another challenging aspect of traditional load testing that Visual Studio load test solves is collaboration in a team environment, such

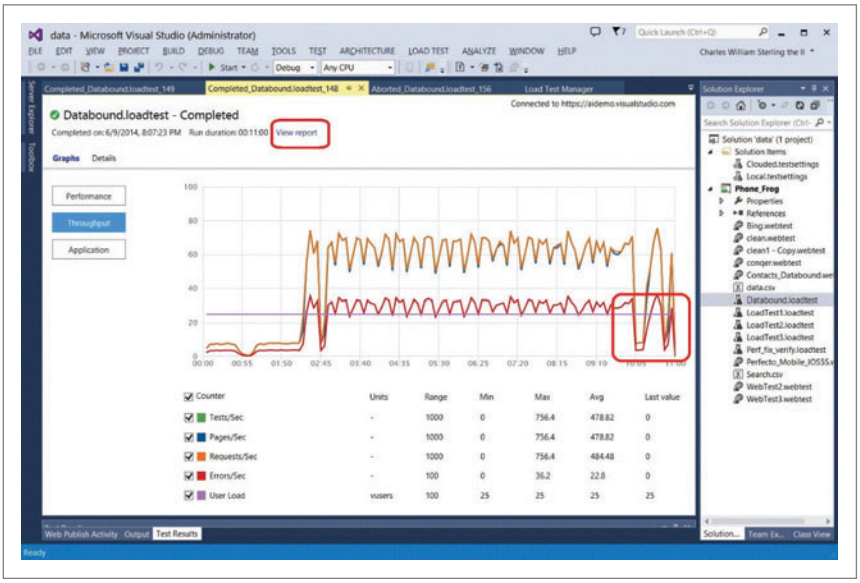



Figure 2 **The Visual Studio Online Load Test Report Page**

as sharing and storing load test results. In the Load Test Manager in **Figure 3**, you can only see two team members are doing performance testing. The last four runs were aborted and they've just queued another run. Something isn't going well there.

Agile teams are typically interested in knowing the impact of their current sprint efforts on performance. To answer this, Visual Studio provides two different Excel-based reports that help the team compare runs and examine trends. To view this in-depth information, download the run data from Visual Studio Online. Click on the view report link. This opens a report view of your data—including the option to open and compare multiple sets of run data in Excel.

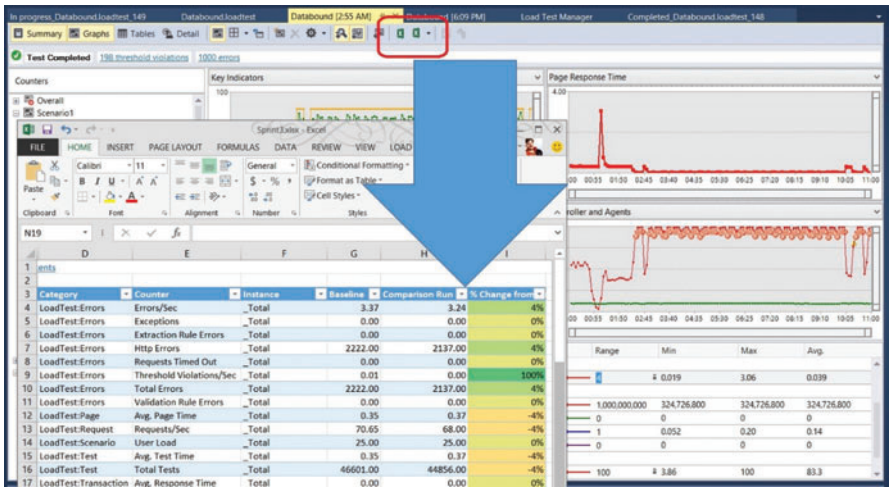


Figure 3 **The Load Test Manager**

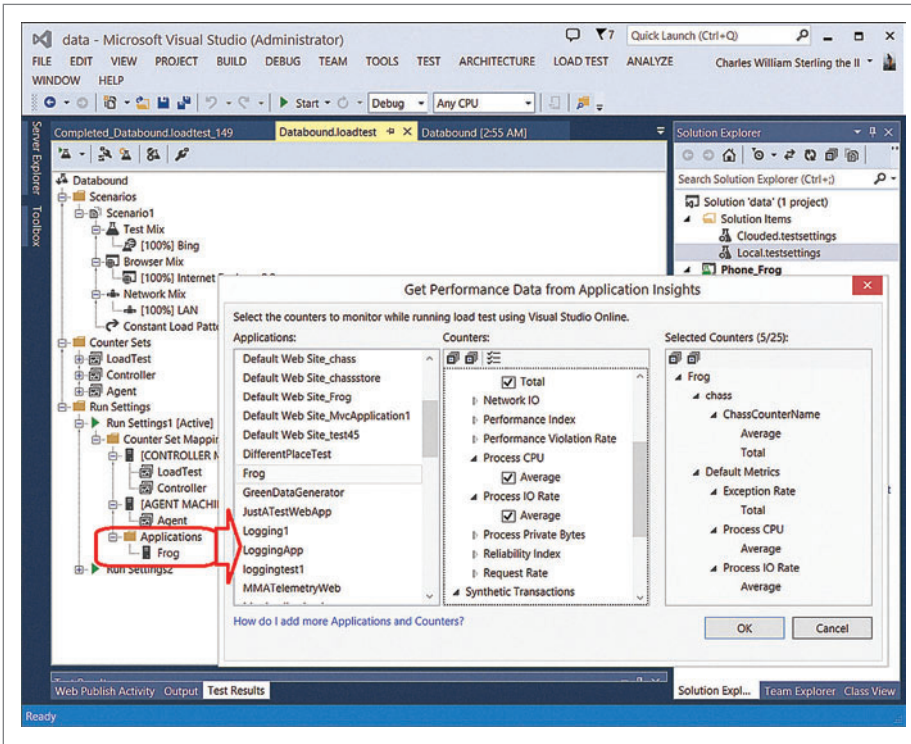



Figure 4 **An Example of a Completed Load Testing Report**

The report in **Figure 4** compares Sprint2 versus Sprint3. You can see that while the performance is 4 percent slower, there are far fewer errors. That's a manageable trade-off to the team.

## Integrate Application Insights

Errors discovered during load testing are often questioned due to questionable diagnostics collection techniques. Worse yet are those errors that are difficult or impossible to diagnose due to insufficient diagnostics collection information.

With Visual Studio 2013 Update 2, cloud load testing was enhanced with the integration of Application Insights. See the May 2014 article, "Next-Generation Development with Application Insights" (msdn.microsoft.com/magazine/dn683794.aspx). This integration with production monitoring tools enables testing applications in development or production. It also helps development teams find and begin to address any errors they discover, regardless of when they were found—whether an hour, day or week earlier.

To include integrated performance and diagnostics information into your cloud load test using Application Insights, right-click on the Applications node of the load test definition. Then select the Application Insights Component you'd like to include in your load test reporting.

## Wrapping Up

Cloud load testing in Visual Studio Online removes much of the work of setting up and load testing your application. While Azure makes it easy to scale applications and potentially "buy" your way out of poorly performing code, without load testing, it's often difficult or impossible to know how much this will cost. It's also difficult to tell if you'll need to do this before there's a negative impact on your customers. Full cloud load testing can help you identify live site bugs that can have an impact on your business. ■

**Charles Sterling** *has been at Microsoft for 20 years. Before Microsoft he was a marine biologist working for the United States National Marine Fisheries doing marine mammal research on the Bering Sea. After a six-year tour in Australia as a Microsoft product manager, Sterling is currently back in Redmond as a senior program manager on the Visual Studio development team. Reach him at chass@microsoft.com.*

**Thanks** to the following Microsoft technical expert for reviewing this article:
*Manas Maheshwari*

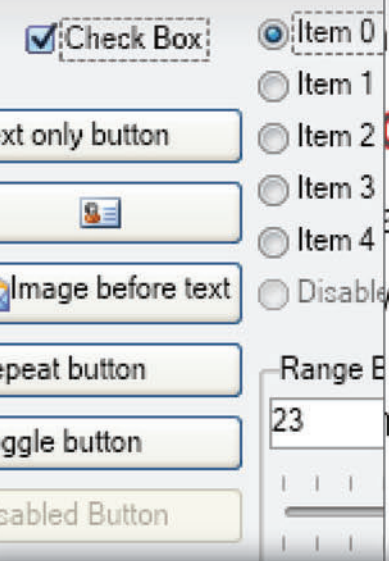
**the myth:** “To create a .NET application that runs on WinForms, WPF, Silverlight, MonoMac, Xamarin.Mac you have to write it 5 times (once for each platform)”

**the truth:** Don’t trust old-school devs. With Nevron Open Vision you only write once!

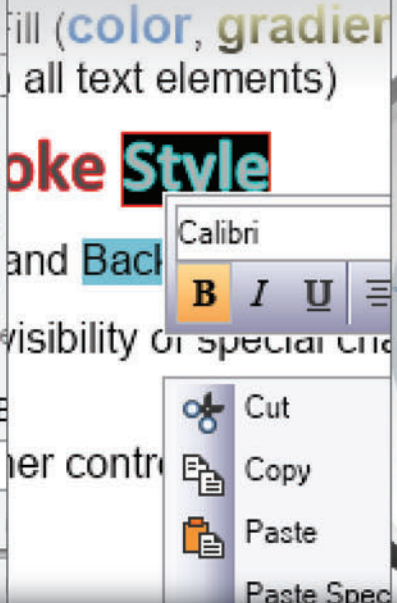
# NEVRON OPEN VISION for .NET

comes with:

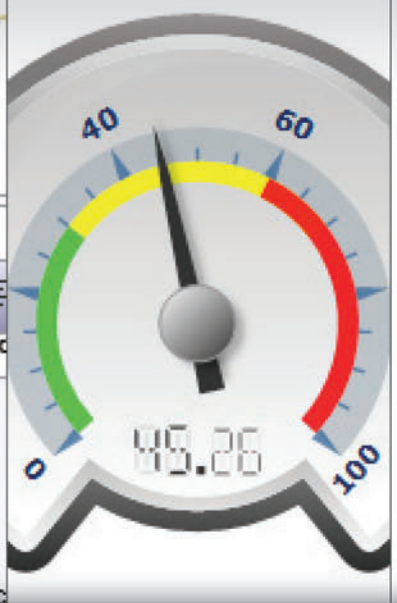
## UI SUITE




## TEXT EDITOR




## GAUGE




## BARCODE


COMING SOON: NOV CHART for .NET  NOV DIAGRAM for .NET

## Learn more at **www.nevron.com** today

**www.nevron.com** | email@nevron.com | +1 888-201-6088 (Toll free, USA and Canada)

# WPF Commanding with the State Machine Pattern

Tarquin Vaughan-Scott

Windows Presentation Foundation (WPF) has a powerful commanding framework that lets you separate the UI and the command logic. When you use the Model-View-ViewModel (MVVM) design pattern, a command is exposed on the ViewModel as a property that implements the ICommand interface. Controls on the View bind to these properties. When a user interacts with that control, it executes the command.

As always, the devil is in the details. The true challenge isn't in executing the command, but ensuring it executes when the ViewModel is in a valid state for that action. Typically, the "can execute" validation is implemented with a conditional expression using local variables such as IsLoading, CanDoXYZ, SomeObject != null and so on. Each new state requires additional conditions that need evaluation, so this strategy can quickly become overly complex.

This article discusses:

- How to use state machines to control app actions
- Binding commands to state machines
- App design for efficient command structure

Technologies discussed:

Windows Presentation Foundation, Model-View-ViewModel, Microsoft .NET Framework

Code download available at:

msdn.microsoft.com/magazine/msdnmag1114

A state machine solves the "can execute" problem, as it restricts the actions allowed for a specific state. Binding commands directly to the state machine provides a good solution to this tricky problem.

This article will demonstrate how to design and implement a state machine by analyzing your application. It will also show you how to bind commands to the state machine. There are also some additional benefits such as preventing reentrancy of asynchronous methods and showing and hiding controls for specific states.

## State Machines

State machines come in different flavors, but they're essentially a design pattern that represents a process that moves from one state to another. User actions (also called triggers) cause the state machine to transition between states. Rules restrict which actions are allowed for each state.

Finite state machines only allow one state at a time. Hierarchical state machines allow states and sub-states. Hierarchical state machines are often more useful because the sub-states inherit all the attributes of their super-state. This can reduce the amount of configuration required.

For example, consider a user clicking a Search button: The application transitions to a Searching state. While in this state, the user should be prevented from performing other actions (with the possible exception of a Cancel action). The Searching state would therefore have a rule that prevents or ignores all actions except for the Cancel action. Some state machine implementations also

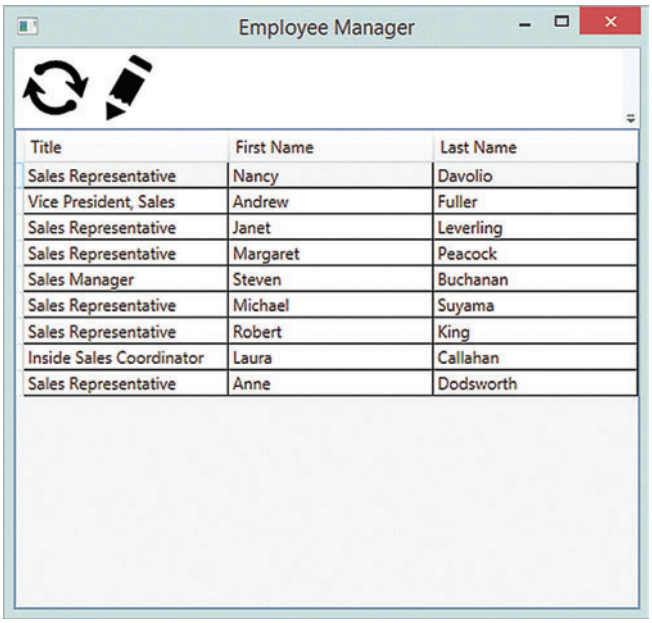



Figure 1 **The List of Employees Loaded and Ready for Editing**

allow entry and exit actions. You can have this logic executed when a state is entered and exited.

## Command Structure

Like a genie in a bottle, commands are there to do their master's bidding, but what is a command? Commands have their origin in command-line interfaces, in which the user is required to type in a directive that is then interpreted by the application.

This concept has evolved in modern interfaces to abstract a user's intended action. For example, if a user wants to copy some text, the command is Copy Text. The user can achieve this by clicking a menu item, right-clicking the mouse button or even using keyboard controls (CTRL+C). How the command is implemented depends on the underlying application and the framework upon which it's built.

WPF implements the command concept via the ICommand interface. This is part of the Microsoft .NET Framework. This interface has two methods and an event:

- void Execute(object parameter)—This executes code when the command is invoked.
- bool CanExecute(object parameter)—This determines if the command can be invoked.
- event EventHandler CanExecuteChanged—This informs the framework that conditions affecting the CanExecute method have changed. This is generally handled by the WPF framework.

Basically, a command source such as a button or menu item implements the ICommandSource interface. This interface has a property named Command of type ICommand. By binding this property to an ICommand implementation on the ViewModel, the control will invoke the Execute method. You can also have it enabled and disabled based on the CanExecute method result. If a control acting as a command source doesn't have an ICommand property (which, unfortunately, is fairly common), then you can use an event to command technique.

The implementations of ICommand built in to the .NET Framework are the RoutedCommand and RoutedUICommand. These route events up and down the visual tree and aren't suitable for use with the MVVM pattern. The RelayCommand by Josh Smith and the DelegateCommand that's part of the Prism framework are commonly used implementations for the MVVM pattern.

> State machines come in different flavors, but they're essentially a design pattern that represents a process that moves from one state to another.

The source code accompanying this article contains a Visual Studio 2013 solution that demonstrates the principles of the state machine and how to use it to manage commands. The sample application lets a user search through a list of employees, select an employee and then display a dialog for editing employee details (see **Figure 1**).

## Design the State Machine

The first step in the design process is to define the section of the application using a flow chart or state diagram. **Figure 2** shows a flow chart diagram for the employee manager screen. It also includes blocks that indicate a potentially long-running process such as Searching.

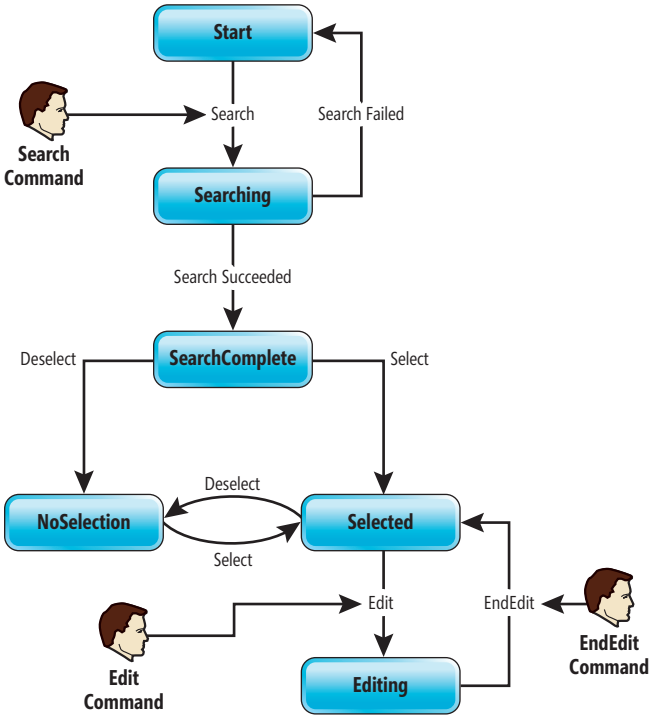



Figure 2 **Flow Chart Showing Process of Employee Manager Screen**

You need to handle these in a special way and prevent the user from re-running the operation while it's busy. Creating these intermediary "busy" blocks is especially important with the new async functionality in the .NET Framework, as control returns to the user and he could potentially try and run the same action again resulting in reentry.

The first step in the design process is to define the section of the application using a flow chart or state diagram.

The second step in the design process is to define any commands that let the user interact with the application. From the employee manager screen, the user can Search, Edit an Employee and then End Editing. The user also drives the process of Selecting and Deselecting an employee, however, these aren't managed as commands. They're handled as data binding on a DataGrid control. The commands are now mapped onto the workflow arrows (or flow of control) at the relevant points, as shown in **Figure 2**.

## Implement the State Machine

You don't have to create a state machine framework from scratch, as there are many freely available libraries. The only option within the .NET Framework is the Workflow Foundation (WF) State Machine Activity. This is too complex for the problem I'm trying to solve here, but it's great for long-running persistent workflows.

## Additional References

**State Machine:** It's surprisingly difficult to find a well-written introduction to state machines, but games developer Bob Nystrom has written a pretty good one (bit.ly/1uGxVv6).

**WPF MVVM:** The "father" of WPF-MVVM Josh Smith provides an implementation of the ICommand interface, the RelayCommand, in a February 2009 *MSDN Magazine* article (msdn.microsoft.com/magazine/dd419663).

**Commands:** Official MSDN documentation on WPF commanding discusses routed commands and events that are part of the .NET Framework (bit.ly/1mRCOTv).

**Routed Commands:** Brian Noyes covers routed commands in a September 2008 *MSDN Magazine* article. It's a good starting point to understand the difference between routed commands and the MVVM command pattern (bit.ly/1CihBVZ).

**Commands, RelayCommands and EventToCommand:** Laurent Bugnion's May 2013 article shows how to convert events to commands, which is useful for controls that don't implement the ICommandSource interface (msdn.microsoft.com/magazine/dn237302).

**Prism Framework:** The official documentation for the Prism framework discusses the MVVM pattern and the DelegateCommand (bit.ly/1k2Q6sY).

**NuGet Stateless Package:** The Stateless package on the NuGet Web site comes with download instructions (bit.ly/1sXBQI2).

After some preliminary research, I settled on the Stateless state machine library, which is available as a NuGet package at bit.ly/ZL58MG.

I can now use the flow chart I created in the design phase to create a list of states (each block in the flow chart) and the triggers (each arrow in the flow chart). Stateless uses generic types for states and triggers, so I'm going to go ahead and use an enumeration for both:

```
public enum States
{
  Start, Searching, SearchComplete, Selected, NoSelection, Editing
}
public enum Triggers
{
  Search, SearchFailed, SearchSucceeded, Select, DeSelect, Edit, EndEdit
}
```

Once the enumerations are defined, I need to configure each state using the fluent interface. The important options are:

- SubstateOf(TState state)—indicates a state has a super-state and will inherit all its configurations.
- Permit(TTrigger trigger, TState targetState)—allows a state to transition to the target state via the trigger.
- Ignore(TTrigger trigger)—causes the state to ignore the trigger if its fired.
- OnEntry(Action entryAction)—causes an action to execute when the state is entered.
- OnExit(Action exitAction)—causes an action to execute when the state is exited.

There will be an exception raised by the state machine if a trigger is fired in a state without a valid configuration. This is the configuration for the Searching state:

```
Configure(States.Searching)
  .OnEntry(searchAction)
  .Permit(Triggers.SearchSucceeded, States.SearchComplete)
  .Permit(Triggers.SearchFailed, States.Start)
  .Ignore(Triggers.Select)
  .Ignore(Triggers.DeSelect);
```

The OnEntry action executes the search process and the Permits allow the relevant triggers to fire. The Ignores prevent the View from firing the Select and DeSelect triggers when the DataGrid control

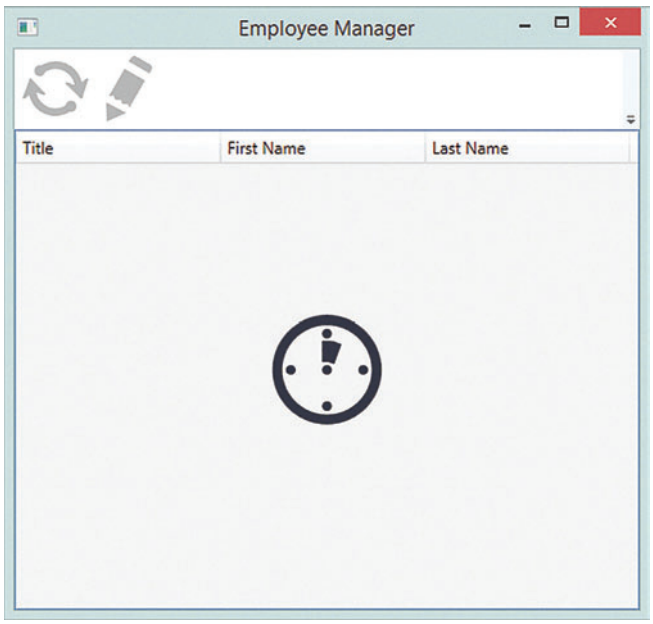



Figure 3 **Busy Animation in the Search Dialog**

binds to the underlying data source (an annoyance that happens with most list controls in WPF).

The state machine also exposes two important methods:

- void Fire(TTrigger)—This transitions the state machine using the previous configurations.
- bool CanFire(Trigger trigger)—This returns true if the current state allows the trigger to be fired.

These are the key methods required for creating commands. They perform the execute and can execute logic.

## Bind Commands to the State Machine

The MVVM pattern exposes a property implementing the ICommand interface on the ViewModel. Creating this command property is now a simple matter of binding its Execute and CanExecute methods to the state machine's Fire and CanFire methods, respectively. Create an extension method to centralize this logic:

```
public static ICommand CreateCommand<TState, TTrigger>(
  this StateMachine<TState, TTrigger> stateMachine, TTrigger trigger)
    {
      return new RelayCommand
        (
          () => stateMachine.Fire(trigger),
          () => stateMachine.CanFire(trigger)
        );
    }
```

Once this extension method is in place, create ICommand properties on the ViewModel (refer back to the flow diagram in **Figure 2** for the commands identified during the analysis phase):

```
SearchCommand = StateMachine.CreateCommand(Triggers.Search);
EditCommand = StateMachine.CreateCommand(Triggers.Edit);
EndEditCommand = StateMachine.CreateCommand(Triggers.EndEdit);
```

## Bind the View to the ViewModel Command

For a control that acts as a command source, you can have its Command property bound to the ViewModel's ICommand property. The sample application has two buttons and menu items bound to the command properties on the ViewModel:

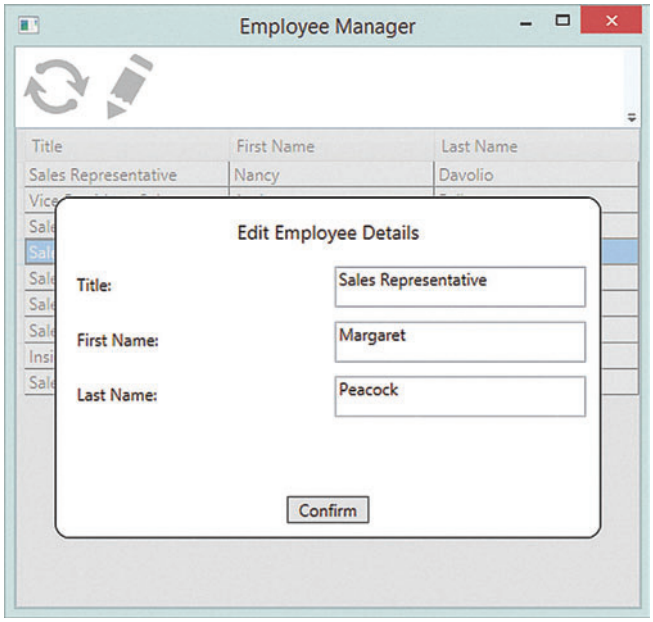



Figure 4 **The Editing Dialog State**

```
<Button ToolTip="Search" VerticalAlignment="Center"
Style="{StaticResource ButtonStyle}"
  Command="{Binding SearchCommand}">
  <Image Source="Images\Search.png"></Image>
</Button>
```

Now the command is bound directly to the state machine. It will fire the configured trigger when executed, but more important, it will be disabled if that trigger isn't allowed for the current state. This takes care of the tricky can execute logic, and it's all neatly configured within the state machine. **Figure 3** shows the employee manager screen while it's busy searching. Note the Search command is disabled as the Search trigger isn't allowed for the Searching state.

## Additional Benefits

Once you've implemented the state machine, you can also bind other visual elements to its state. When the search function is busy executing (and keep in mind this could be a long-running operation within an asynchronous method), it's desirable to show the user a busy indicator. **Figure 3** shows an animated image displayed only when the state machine is in the Searching state. Do this by creating a custom converter that converts the state machine state into a Visibility value:

```
public class StateMachineVisibilityConverter : IValueConverter
  {
    public object Convert(object value, Type targetType,
      object parameter, CultureInfo culture)
    {
      string state = value != null ? value.ToString() : String.Empty;
      string targetState = parameter.ToString();

      return state == targetState ? Visibility.Visible : Visibility.Collapsed;
    }
  }
```

The animated image is then bound to the state machine state using the custom converter:

```
<local:AnimatedGIFControl Visibility="{Binding StateMachine.State,
                        Converter={StaticResource StateMachineConverter},
                        ConverterParameter=Searching}"/>
```

You can apply the same principle to the editing dialog shown over the employee manager screen in **Figure 4**. When the state machine is in the Editing state, the dialog becomes visible and the user can edit the selected employee's details.

## State of Mind

The state machine pattern not only solves the problems associated with command logic, but also creates a state of mind that allows for better application analysis. It's easy to think of commands as controls that execute some sort of logic. What's often neglected is determining when to allow the command to execute. The command-state machine pattern addresses these issues early on in the design process. They're naturally part of the state machine configuration. States machines have simplified my ViewModel code and can do the same for your applications. ■

**Tarquin Vaughan-Scott** *is the lead developer at InfoVest (infovest.co.za), based in Cape Town, South Africa, which creates data management solutions for the financial industry. His main interests are database design and architectures—especially the differences between transactional and data warehouse systems. Reach him at tarquin@infovest.co.za.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Nicholas Blumhardt*

# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com

## 2015 Dates Announced


**Visual Studio Live!** is hittin' the open road on the ultimate code trip! For 22 years, we have helped tens of thousands of developers navigate the .NET Highway, featuring code-filled days, networking nights and the best in independent training:

- Multi-track Events
- Focused, Cutting-edge .NET Education
- Covering the Hottest Topics
- Relevant and Useable Content
- Expert Speakers, Including Many Microsoft Instructors
- Interactive Sessions
- Discuss Your Challenges
- Find Actionable Solutions

**MAKE PLANS TO JOIN US IN 2015!**


SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

# NAVIGATE THE .NET HIGHWAY

## 5 Locations to Choose From

### New York

**The Code That Never Sleeps**
**September 28 – October 1**
NY Marriott at Brooklyn
Bridge Brooklyn, NY

NEW YORK Code Trip

### Orlando

***Code in the Sun***

**PART OF LIVE! 360**
*Tech Events with Perspective*

**SharePoint Live!**
**SQL Server Live!**
**Modern Apps Live!**
**TechMentor**

**November 16 – 20**
Loews Royal Pacific Resort
Orlando, FL

ORLANDO Code Trip



JAMES MCCAFFREY

# Consensus Classification Using C#

In machine learning (ML), classification is the process of creating a model (typically a math equation of some sort, or a set of rules) that predicts a value that can take on discrete, non-numeric values. For example, you might want to predict the political party (Democrat or Republican) of a member of Congress based on his or her voting record. Training the model is the process of finding the set of constants (for a math equation model) or set of rules (for a rule-based model) so that when presented with training data with known output-dependent variable values, the computed outputs closely match the known outputs. Then the model can be used to make predictions for new data with unknown outputs.

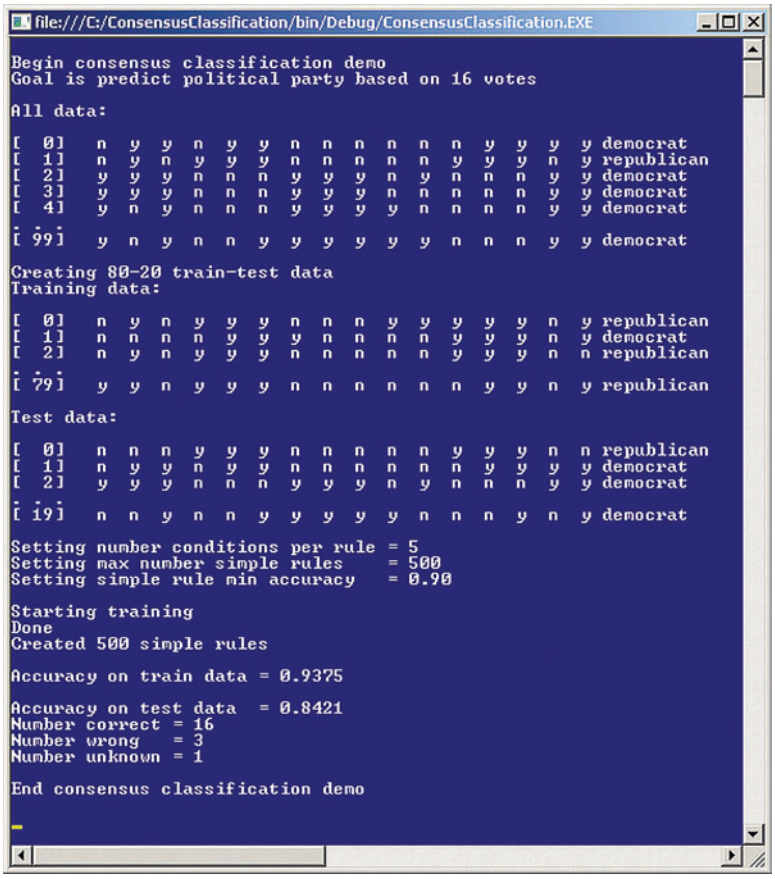



Figure 1 **Consensus Classification in Action**

> This article presents a custom technique that, for lack of a better name, I call consensus classification.

Although there are many standard classification algorithms and techniques, including Naive Bayes classification, logistic regression classification and neural network classification, for some problems a custom classification algorithm is useful. This article presents a custom technique that, for lack of a better name, I call consensus classification. The best way to get a feel for what consensus classification is and to see where this article is headed is to take a look at the demo program in **Figure 1**.

The goal of the demo program is to create a model that predicts the political party, Democrat or Republican, of a member of the U.S. House of Representatives based on the representative's voting record on 16 legislative bills. The raw data set consists of 100 items, each of which corresponds to a Representative and has 17 fields. The first 16 fields in each item are votes where character "y" is a yes vote and character "n" is a no vote. The last field in each item is the Representative's actual political party. For example, the first two data items are:

```
n, y, y, n, y, y, n, n, n, n, n, n, y, y, y, y, democrat
n, y, n, y, y, y, n, n, n, n, n, y, y, y, n, y, republican
```

The demo data is a subset of a well-known benchmark set called the Congressional Voting Records Data Set. The complete benchmark data set has 435 items, some of which have a vote value of "?," indicating an unknown vote, or abstention. The demo subset consists of the first 100 items, from the full set, that have no "?" votes. Additionally, the source data set has political party in the first column; I moved party to the last column, which is much more convenient when programming.

Although it's not necessary to know what legislative bill each of the 16 votes correspond to, the topic of each bill is suggested by the 16 following short terms: handicapped-infants, water-project,

Code download available at msdn.microsoft.com/magazine/msdnmag1114.

# We know how application development can be.

## Let DotImage take some of the bite out of your challenges.

Atalasoft DotImage

Connecting the dots is a no-brainer. DotImage image-enables your .NET-based web application faster, more cost effectively, and less painfully than if done on your own. This proven SDK is versatile, with options including OCR capabilities, WingScan compatibility, and support for a range of formats. Coupled with dedicated assistance from our highly knowledgeable and skilled engineers, DotImage helps your business connect with powerful information hidden inside your documents, making the big picture much easier to see.

Image-enabling experts & bacon connoisseurs. Visit us online to see our full line of SDK products for .NET and Java

Atalasoft
A Kofax Company

www.atalasoft.com

adopt-budget, physician-fees, el-salvador, school-religion, anti-satellite, nicaraguan-contras, mx-missile, immigration-bill, synfuels-cutback, education-spending, superfund-sue, crime-bill, duty-free, south-africa.

The demo program splits the 100-item data set into an 80-item set used to train the model, and a 20-item data set used to estimate the accuracy of the resulting model. A consensus classification model consists of a set of simple rules such as, "If the Representative voted 'y' on bill 0 and 'n' on bill 3 and 'y' on bill 15, then the Representative is a Republican." In the demo, the number of Boolean conditions in each simple rule is set to 5, and the total number of rules is set to 500. Furthermore, each simple rule is required to be at least 90 percent accurate on the training data items for which the rule is applicable.

Behind the scenes, the training process generated the 500 simple rules. After the model was created, the demo program applied the model rules to the training data and got a 93.75 percent accuracy. Then the model was applied to the 20-item test set, resulting in 84.21 percent accuracy—16 correct predictions, three incorrect predictions, and one data item where none of the 500 rules in the model was applicable.

This article assumes you have at least intermediate programming skills and a basic knowledge of ML classification. The demo is coded using C#, but you shouldn't have much trouble refactoring the code to another language, such as Visual Basic .NET or Python. The demo code is a bit too long to present in its entirety, but the entire source code is available in the download that accompanies this article at msdn.microsoft.com/magazine/msdnmag1114.

## Overall Program Structure

The overall structure of the demo program, with some WriteLine statements removed and minor edits to save space, is presented in **Figure 2**. To create the demo, I launched Visual Studio and created a new C# console application and named it ConsensusClassification. The demo has no significant Microsoft .NET Framework dependencies, so any relatively recent version of Visual Studio will work. After the template-generated code loaded, in the Solution Explorer window I renamed file Program.cs to the more descriptive ConsensusProgram.cs and Visual Studio automatically renamed class Program for me.

> Each simple rule is required to be at least 90 percent accurate on the training data items for which the rule is applicable.

All the program logic is contained in the Main method. The demo has two static helper methods, MakeTrainTest and ShowData. The classification logic is contained in a single program-defined class named ConsensusClassifier. The classifier exposes six public methods: a single constructor, RuleCount (the actual number of simple rules in the model), ComputeOutput

Figure 2 **Overall Program Structure**

```
using System;
using System.Collections.Generic;
namespace ConsensusClassification
{
  class ConsensusProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin consensus classification demo");
      Console.WriteLine("Goal is predict political party");
      string[][] allData = new string[100][];
      allData[0] = new string[] { "n", "y", "y", "n", "y", "y", "n", "n",
        "n", "n", "n", "n", "y", "y", "y", "y", "democrat" };
      allData[1] = new string[] { "n", "y", "n", "y", "y", "y", "n", "n",
        "n", "n", "n", "y", "y", "y", "n", "y", "republican" };
      // Etc.
      allData[99] = new string[] { "y", "n", "y", "n", "n", "y", "y", "y",
        "y", "y", "y", "n", "n", "n", "y", "y", "democrat" };

      Console.WriteLine("All data: ");
      ShowData(allData, 5, true);

      Console.WriteLine("Creating 80-20 train-test data");
      string[][] trainData;
      string[][] testData;
      MakeTrainTest(allData, 0, out trainData, out testData); // 0 = seed

      Console.WriteLine("Training data: \n");
      ShowData(trainData, 3, true);
      Console.WriteLine("Test data: \n");
      ShowData(testData, 3, true);

      int numConditions = 5; // Conditions per rule
      int maxNumRules = 500;
      double minAccuracy = 0.90; // Min % rule accuracy

      Console.WriteLine("Setting number conditions per rule = " + numConditions);
      Console.WriteLine("Setting max number simple rules    = " + maxNumRules);
      Console.WriteLine("Setting simple rule min accuracy   = " +
        minAccuracy.ToString("F2"));
      ConsensusClassifier cc =
        new ConsensusClassifier(numConditions, maxNumRules);

      Console.WriteLine("Starting training");
      cc.Train(trainData, minAccuracy);
      Console.WriteLine("Done");
      Console.WriteLine("Created " + cc.RuleCount() + " simple rules");

      double trainAcc = cc.Accuracy(trainData);
      Console.WriteLine("Accuracy on train data = " + trainAcc.ToString("F4"));

      int numCorrect, numWrong, numUnknown;
      double testAcc = cc.Accuracy(testData, out numCorrect,
        out numWrong, out numUnknown);
      Console.WriteLine("Accuracy on test data  = " + testAcc.ToString("F4"));
      Console.WriteLine("Number correct = " + numCorrect);
      Console.WriteLine("Number wrong   = " + numWrong);
      Console.WriteLine("Number unknown = " + numUnknown);

      Console.WriteLine("End consensus classification demo\n");
      Console.ReadLine();
    }

    static void MakeTrainTest(string[][] allData, int seed,
      out string[][] trainData, out string[][] testData) { . . }
    static void ShowData(string[][] rawData, int numRows,
      bool indices) { . . }
  } // Program

  public class ConsensusClassifier { . . }

} // ns
```

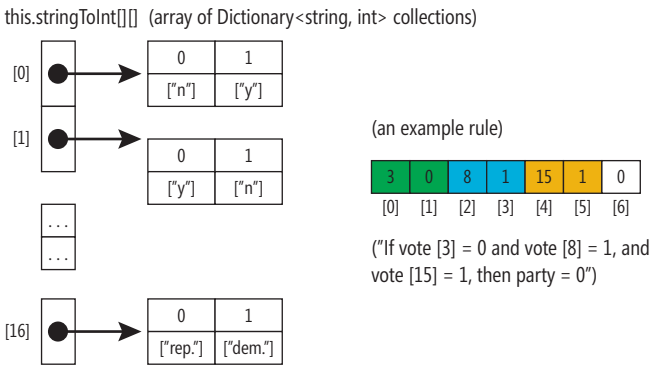



Figure 3 **Consensus Classification Data Structures**

(to make predictions after the model has been created), Train (to create the model) and an overloaded Accuracy (to compute the percentage of correct predictions).

In the Main method, the demo program hardcodes the 100-item source data into an array-of-arrays-style string matrix:

```
string[][] allData = new string[100][];
allData[0] = new string[] { "n", "y", "y", "n", "y", "y", "n", "n",
  "n", "n", "n", "n", "y", "y", "y", "y", "democrat" };
...
```

In a non-demo scenario, your data would likely be in a text file and you'd load the data into memory using a helper method. The source data is split into a training set and a test set, like so:

```
string[][] trainData;
string[][] testData;
MakeTrainTest(allData, 0, out trainData, out testData);
```

The 80/20 split percentages are hardcoded. The 0-value argument passed to method MakeTrainTest is a seed value for a Math.Random object so that the train-test split can assign data items randomly. An alternative is to use a stratified approach so that the proportions of Democrat and Republican data items in the training and test matrices are roughly the same as the percentages in the overall source data.

The demo creates the model with this code:

```
int numConditions = 5;
int maxNumRules = 500;
double minAccuracy = 0.90;
ConsensusClassifier cc = new ConsensusClassifier(numConditions, maxNumRules);
cc.Train(trainData, minAccuracy);
```

Here, I decided to pass values for the number of Boolean conditions in each rule and the maximum number of rules to create to the constructor, and pass a reference to the training data and the minimum accuracy each rule must meet to method Train. Many developers prefer to pass all relevant parameters to the constructor (at the expense of a constructor with a large number of parameters). Here, I passed values most directly related to each method (at the expense of a rather arbitrary-looking calling interface).

The demo program concludes by computing and displaying the accuracy of the model:

```
double trainAcc = cc.Accuracy(trainData);
Console.WriteLine("Accuracy on train data = " + trainAcc.ToString("F4"));

int numCorrect, numWrong, numUnknown;
double testAcc = cc.Accuracy(testData, out numCorrect, out numWrong,
  out numUnknown);
Console.WriteLine("Accuracy on test data  = " + testAcc.ToString("F4"));
Console.WriteLine("Number correct = " + numCorrect);
Console.WriteLine("Number wrong   = " + numWrong);
Console.WriteLine("Number unknown = " + numUnknown);
```

The first overloaded Accuracy returns just the percentage of correct classifications. The second overloaded Accuracy returns, in addition, the number of correct, incorrect and non-applicable data items, where non-applicable means that none of the model's rules apply to a particular data item. For example, suppose the number-of-conditions parameter was set to 3 and one of the data items has vote0 = 'y', vote1 = 'y' and vote2 = 'n'. Even with 500 rules, it's possible for none of the rules to take this combination of votes into account. An alternative is to design method Accuracy so that non-applicable data items aren't possible. One common way to do this is to add a final rule that resembles, "...else Representative is a Democrat," where Democrat is a default value, which is typically the most common dependent-variable value.

The demo program doesn't use the model to predict the political party of a Representative. Predicting the political party of a Representative who voted "yes" on bills [0] through [7], and "no" on bills [8] through [15] could resemble this:

```
string[] newData = new string[] { "y", "y", "y", "y", "y", "y", "y", "y",
  "n", "n", "n", "n", "n", "n", "n", "n" };
int party = cc.ComputeOutput(newData);
Console.WriteLine("Predicted party = " + party);
```

Method ComputeOutput returns an integer value, 0 or 1, where 0 indicates Republican and 1 indicates Democrat.

## The Consensus Algorithm

The heart of the consensus classifier is a rule set. The two primary issues to deal with are deciding how to represent a rule, and generating rules. The demo program represents a rule as an integer array. The scheme is best explained using a concrete example, as shown in **Figure 3**. Suppose the number of Boolean conditions in each rule is set to three. A single rule would be represented as an array with seven cells. If the array had values { 3, 0, 8, 1, 15, 1, 0 }, then the rule corresponds to "If vote [3] is 0 and vote [8] is 1 and vote [15] is 1, then party is 0." The rule set is a generic List collection of rules.

> An alternative is to design method Accuracy so that non-applicable data items aren't possible.

The meaning of 0 and 1 can vary for each column/vote. The demo program constructs an array of Dictionary collections, one per column/vote. Zero-based integer IDs are assigned to each string value as they're encountered in each column in the training data. For example, in column [0], for vote [0], in the training data, "n" is encountered first and assigned to 0 and "y" is encountered next and assigned to 1. But in column [1], "y" is encountered first and assigned to 0 and "n" is encountered second and assigned to 1.

Similarly, in column [16] of the training data, the last column, which holds dependent variable values "democrat" and "republican," "republican" is encountered first and so is 0, and "democrat" is 1. The string-to-integer mapping information is stored in a class

member array named stringToInt, as shown in **Figure 3**. So the expression stringToInt[1]["y"] returns the zero-based index value for a yes vote for vote [1], which, for the demo training data, is 0.

Here's the high-level pseudo-code for creating the rules that define the classification model:

```
loop while numRules < maxRules && trial < maxTrials
  ++trial
  select a random row of training data
  select numConditions random columns of selected row
  create candidate rule from selected columns of selected row
  if candidate rule already in rule list, continue
  if candidate rule does not meet minimum accuracy, continue
  candidate rule is good so add rule to rule list
end loop
```

A concrete example may help clarify. In the main loop, suppose row [0] of the training data is randomly selected. Next, suppose numConditions has been set to 3 and the three randomly selected columns are [1], [2] and [15]. In the training data, these columns have values "y," "n," and "y" and the dependent variable has value "republican." The column values are looked up in the array-of-Dictionary collections and are 0, 0 and 0. Therefore, the candidate rule is an integer array with values { 1, 0, 2, 0, 15, 0, 0 }, which can be interpreted as "If column 1 is 0 and column 2 is 0 and column 15 is 0, then party is 0."

Next, the 80 items in the training data are scanned to see if the candidate rule meets the minimum accuracy criterion. Note that the candidate rule will be correct for at least one data item—the item that was used to create the rule. However, the candidate rule will not necessarily be applicable to all training items. For example, the candidate rule { 1, 0, 2, 0, 15, 0, 0 } that was generated from data item [0] isn't applicable to data item [1] because column [2] has value 1 rather than the required 0.

> Combining several rules or models to generate a classification result is often called ensemble learning in ML vocabulary.

Once the rule set has been created, the output for a given set of input data is determined by what might be described as counting votes. For each data item, all of the rules in the rule set are analyzed. Suppose, as in the demo, the rule set has 500 rules. And suppose that for one data item, 220 of the rules predict a political party of Republican, 250 of the rules predict Democrat, and 30 of the rules aren't applicable. The classifier would predict the person associated with the data is a Democrat. In other words, the output is a consensus of the predictions from the rule set.

## A Few Comments

The motivation for the consensus classification technique presented in this article came from a research project I was working on some time ago. In particular, I was trying to predict the sex—male or female—of an instant message user based on variables such as the region of the user, the age category of the user and so on. I tried every kind of standard classification approach I could think of, using existing ML tools, but none of my approaches were able to generate an effective prediction model, even though my gut feeling was that the data contained enough information to yield a signal.

> For each data item, all of the rules in the rule set are analyzed.

I noticed that using Naive Bayes classification seemed to be the best of my attempts. Naive Bayes assumes each predictor variable is mathematically independent. Even though this assumption is often not true, Naive Bayes sometimes works quite well. For my problem, the predictor variables were almost certainly related in some way, but I wasn't sure which variables were related. So I decided to use parts of several ML techniques to create the approach described in this article. Ultimately, I was able to create a prediction model that was significantly more accurate than any standard-technique model.

The demo problem is a binary classification problem because the dependent variable can be either Democrat or Republican. Furthermore, all of the predictors are binary categorical variables because they can be either yes or no. Consensus classification can deal with multinomial classification problems (for example, Democrat, Republican, Independent), and can handle categorical predictors that have more than two values (for example, yes, no, absent, abstained). In my experiments, it was unclear how effective consensus classification is when presented with problems where the predictor variables are numeric, as with age, and have been binned into categories (for example, young, medium and old).

Combining several rules or models to generate a classification result is often called ensemble learning in ML vocabulary. The idea of generating many simple rules is used in ML techniques, and these are sometimes referred to as boosting techniques. Let me note that in my Test Run column I almost always present ML techniques that have a solid research basis. This article is an exception. The technique presented here hasn't been studied formally or subjected to serious research. Several of my academic colleagues have taken me to task for having the colossal gall to use custom, non-traditional ML techniques, but my usual somewhat snarky response is that I'm typically more concerned with getting results than with establishing an elegant math theorem. In my opinion, using standard ML techniques is the best way to initially approach an ML problem, but when traditional techniques fail, creating a custom solution can sometimes be surprisingly effective. ■

**Dr. James McCaffrey** *works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. He can be reached at jammc@microsoft.com.*

**Thanks** to the following technical experts at Microsoft Research for reviewing this article: *Marciano Moreno Diaz Covarrubias, Delbert Murphy, David Raskino and Alisson Sol*



TED NEWARD AND
JOE HUMMEL

# Rise of Roslyn

For a few years now, various computer professionals, thought leaders and pundits have advocated the idea of domain-specific languages (DSLs) as a way of approaching solutions to software problems. This seems particularly appropriate if the DSL syntax is something "casual users" can use to adapt and modify the business rules in a system. This is the Holy Grail of software to many developers—building a system people can maintain on their own when the business needs change.

One of the principal criticisms of DSL, however, is the fact that writing a compiler is a "lost art." It's not uncommon for programmers from all walks of life to look upon creating a compiler or interpreter as some kind of Dark Art.

At the Build 2014 conference this year, Microsoft formally announced one of the worst-kept secrets in the Microsoft .NET Framework development ecosystem—the open sourcing of Roslyn. This is the revamped/rebuilt compiler system that underlies the C# and Visual Basic languages. For some, this is a chance for Microsoft to put its languages into the open source community and reap the benefits—bug fixes, enhancements, public review of new language features and so on. For developers, it's an opportunity to take a deeper look at how compilers (and interpreters—although Roslyn is focused on compilation given the languages in question) work under the hood.

For more background (and installation tips) check out the Roslyn CodePlex page at roslyn.codeplex.com. As always with not-yet-released bits, it's highly recommended you do this on a virtual machine or a machine you don't care too much about.

## Roslyn Fundamentals

At a high level, the goal of a compiler is to translate programmer input (source code) into executable output, such as a .NET assembly or native .exe file. While the exact names for modules within a compiler vary, we typically think of a compiler as broken into two fundamental parts: a front end and back end (see **Figure 1**).

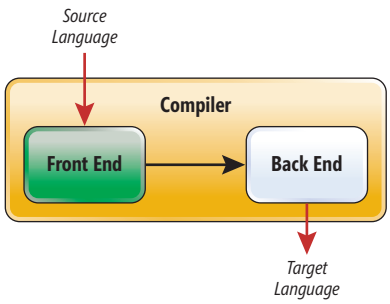



Figure 1 **High-Level Compiler Design**

One of the key responsibilities of the front end is to verify the formatting accuracy of the inbound source code. As with all programming languages, there's a specific format programmers must follow to keep things clear and unambiguous to the machine. For example, consider the following C# statement:

```
if x < 0                    <-- syntax error!
  x = 0;
```

This is syntactically incorrect, because if conditions must be surrounded by ( ), like so:

```
if (x < 0)
  x = 0;
```

Once the code is parsed, the back end is responsible for deeper validation of the source, such as type-safety violations:

```
string x = "123";

if (x < 0)                  <-- semantic error!
  x = 0;                    <-- semantic error!
```

Incidentally, these examples are deliberate design decisions by the language implementer. They're subject to long debates as to which are "better" than others. To hear more, visit any online programming forum and type in, "D00d ur language sux." You'll soon find yourself embroiled in an "educational" session that will certainly be one to remember.

> Microsoft formally announced one of the worst-kept secrets in the .NET development ecosystem—the open-sourcing of Roslyn.

Assuming there are no syntactic or semantic errors, compilation continues and the back end translates the input into an equivalent program in the desired target language.

## Deeper into the Depths

Although you might take the two-part approach with the simplest languages, most often a language compiler/interpreter is broken down into much more than that. At the next level of complexity, most compilers arrange themselves to act in six major phases, two in the front end and four in the back end (see **Figure 2**).

The front end performs the first two phases: lexical analysis and parsing. The goal of lexical analysis is to read the input program and output the tokens—the keywords, punctuation, identifiers and so on. The location of each token is also maintained, so the format

of the program isn't lost. Suppose the following program fragment starts at the beginning of the source file:

```
// Comment
if (score>100)
  grade = "A++";
```

The output from lexical analysis would be this sequence of tokens:

```
IfKeyword               @ Span=[12..14)
OpenParenToken          @ Span=[15..16)
IdentifierToken         @ Span=[16..21), Value=score
GreaterThanToken        @ Span=[21..22)
NumericLiteralToken     @ Span=[22..25), Value=100
CloseParenToken         @ Span=[25..26)
IdentifierToken         @ Span=[30..35), Value=grade
EqualsToken             @ Span=[36..37)
StringLiteralToken      @ Span=[38..43), Value=A++
SemicolonToken          @ Span=[43..44)
```

Each token carries additional information, such as start and end position (Span) as measured from the start of the source file. Notice IfKeyword starts at position 12. This is due to the comment that spans [0..10) and the end-of-line characters that span [10..12). While technically not tokens, output from the lexical analyzer typically includes information about whitespace, including comments. In the .NET compiler, whitespace is conveyed as syntax trivia.

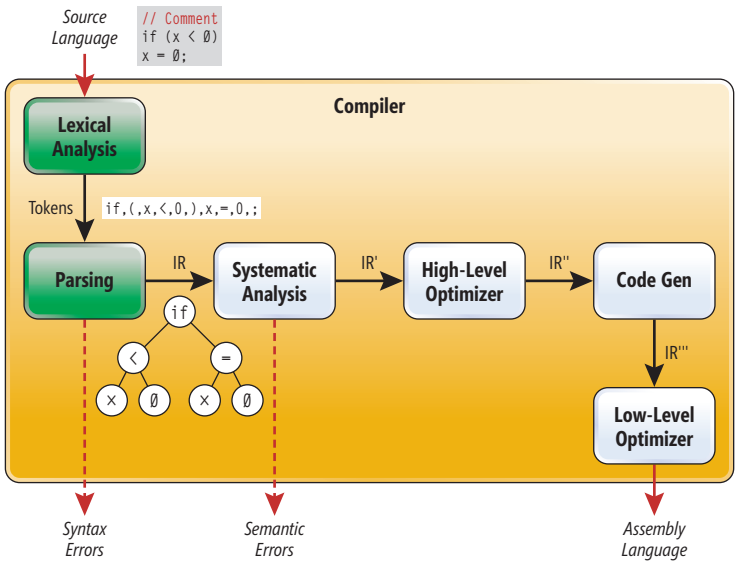



Figure 2 **The Main Phases of a Compiler**

The second phase of the compiler is parsing. The parser works hand-in-hand with the lexical analyzer to perform syntax analysis. The parser does the vast majority of the work, requesting tokens from the lexical analyzer as it checks the input program against the various grammar rules of the source language. For example, C# programmers all know the syntax of an if statement:

```
if  (  condition  )  then-part  [ else-part ]
```

The [ … ] symbolizes the else-part is optional. The parser enforces this rule by matching tokens, and applying additional rules for the more complex syntactic elements such as condition and then-part:

```
void if( )
{
  match(IfKeyword);
  match(OpenParenToken);
  condition();
  match(CloseParenToken);
  then_part();
  if (lookahead(ElseKeyword))
  else_part();
}
```

The function match(T) calls the lexical analyzer to get the next token, and checks to see if this token matches T. Compilation continues normally if it matches. Otherwise, it reports a syntax error. The simplest of parsers use a match function to throw an exception upon a syntax error. This effectively halts compilation. Here's one such implementation:

```
    void match(SyntaxToken T)
    {
var next = lexer.NextToken();

if (next == T)
  ;  // Keep going, all is well:
else
  throw new SyntaxError(...);
    }
```

Lucky for us, the .NET compiler contains a much more sophisticated parser. It's capable of continuing in the face of gross syntax errors.

Assuming there were no syntax errors, the front end is essentially done. It has but one task remaining—to convey its efforts to the back end. The form in which this is stored internally is known as its intermediate representation—or IR. (Despite the similarity in terminology, an IR has nothing to do with the .NET Common Intermediate Language.) The parser in the .NET compiler builds an Abstract Syntax Tree (AST) as the IR, and passes this tree to the back end.

Trees are a natural IR, given the hierarchical nature of C# and Visual Basic programs. A program will contain one or more classes. A class contains properties and methods, properties and methods contain statements, statements often contain blocks, and blocks contain additional statements. The goal of an AST is to represent the program based on its syntactic structure. The "abstract" in AST denotes the absence of syntactic sugar such as ; and ( ). For example, consider the following sequence of C# statements (assume these compile without error):

```
sum = 0;
foreach (var x in A)   // A is an array:
  sum += x;

avg = sum / A.Length;
```

At a high level, the AST for this code fragment would look like **Figure 3**.

The AST captures the necessary information about the program: the statements, the order of the statements, the pieces of each statement and so on. The unnecessary syntax is discarded, such as all semicolons. The key feature to understand about the AST in **Figure 3** is that it captures the syntactic structure of the program.

In other words, it's how the program is written, not how it executes. Consider the foreach statement, which loops zero or more

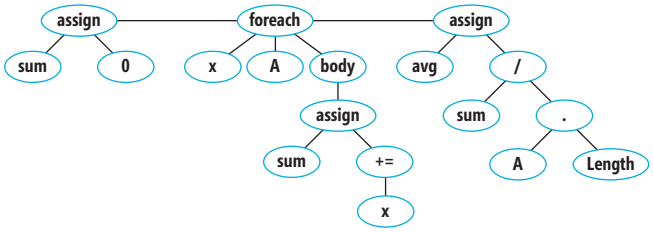



Figure 3 **High-Level Abstract Syntax Tree for C# Code Fragment (Details Missing for Simplicity)**

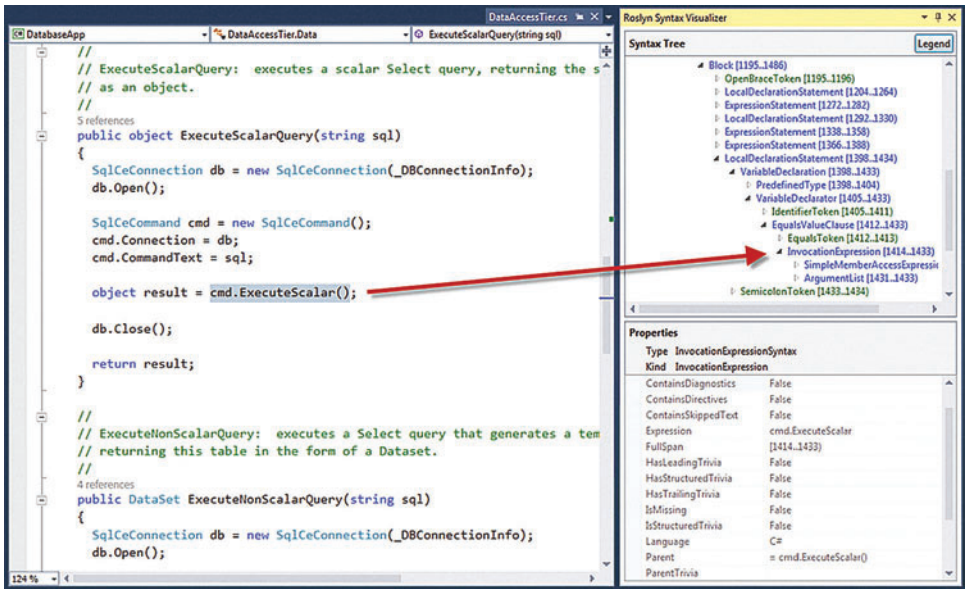



Figure 4 **The Roslyn Syntax Visualizer in Visual Studio 2013**

As a concrete example, consider the if statement we parsed earlier:

```
// Comment
if (score>100)
  grade = "A++";
```

**Figure 5** shows the corresponding AST fragment built by the .NET compiler.

As with a lot of things, the tree seems overwhelming at first. Remember two things, though. One, the tree is simply an expansion of the earlier source statements, so it's actually quite easy to walk through the tree and see how it maps back to the original source. And two, the AST is intended for machine consumption, not for humans. Generally the only time a human looks at the AST is to debug a parser. Bear in mind, as well, that a more complete course on lexing and parsing is well beyond the scope of the space we have here. There are lots of resources available to those who want to dive deeper into this exercise. The goal here is a gentle introduction, not a deep dive.

times as it iterates through a collection. The AST captures the components of the foreach statement—the loop variable, the collection and the body. What the AST doesn't convey is that foreach may repeat over and over again. In fact, if you look at the tree, there's no arrow in the tree to signify how foreach executes. The only way to know is by knowing the foreach keyword == loop.

ASTs are a perfectly good IR, with one main advantage: They're easy to build and understand. The disadvantage is that more sophisticated analyses, such as those used in the back end of the compiler, are more difficult to perform on an AST. For this reason, compilers often maintain multiple IRs, including a common alternative to the AST. This alternative is the control flow graph (CFG), which represents a program based on its flow of control: loops, if-then-else statements, exceptions and so on. (We'll cover this more in the next column.)

The best way to learn how the AST is used in the .NET compiler is through the Roslyn Syntax Visualizer. This is installed as part of the Roslyn SDK. Once installed, open any C# or Visual Basic program in Visual Studio 2013, position your cursor at the source line of interest and open the visualizer. You'll see the View menu, Other Windows and Roslyn Syntax Visualizer (see **Figure 4**).

## Wrapping Up

We're not done with Roslyn by any stretch of the imagination, so stay tuned. It you're interested in diving deeper into Roslyn, may we suggest installing Roslyn. Then have a look at some of the documentation, starting with the Roslyn CodePlex page.

If you want to dive more deeply into parsing and lexing, there are numerous books available. There's the venerable "Dragon Book," also known as "Compilers: Principles, Techniques & Tools" (Addison Wesley, 2006). If you're interested in a more .NET-centric approach, consider "Compiling for the .NET Common Language Runtime (CLR)" by John Gough (Prentice Hall, 2001), or Ronald Mak's "Writing Compilers and Interpeters: A Software Engineering Approach" (Wiley, 2009). Happy coding! ■

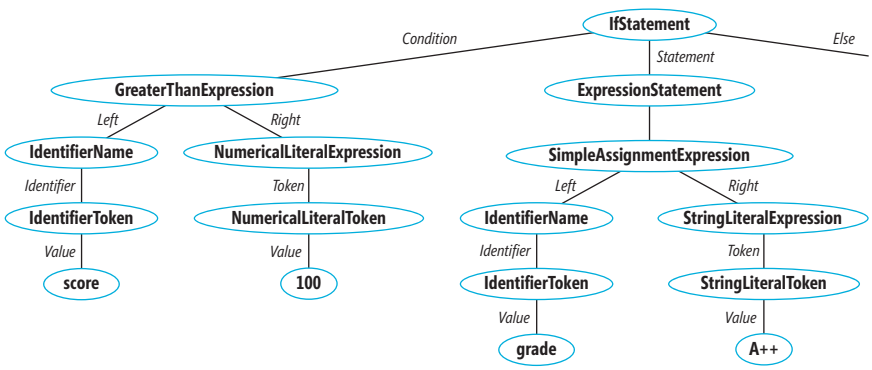



Figure 5 **Abstract Syntax Tree Built by .NET Compiler for IfStatement**

**Joe Hummel** *is a research associate professor at the University of Illinois, Chicago, a content creator for Pluralsight.com, a Visual C++ MVP, and a private consultant. He earned a Ph.D. at UC Irvine in the field of high-performance computing and is interested in all things parallel. He resides in the Chicago area, and when he isn't sailing can be reached at joe@joehummel.net.*

**Ted Neward** *is the CTO at iTrellis, a consulting services company. He has written more than 100 articles and authored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He's an F# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or ted@itrellis.com if you're interested.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Dustin Campbell*

# Spreadsheets Made Easy.

## Fastest Calculations

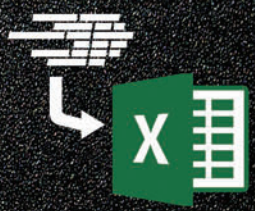
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.


## Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.

## Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.




## Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/lasvegas

## Las Vegas MARCH 16 - 20

BALLY'S HOTEL & CASINO LAS VEGAS, NV


# Code on the Strip

**Visual Studio Live!'s** first stop on its 2015 Code Trip is Las Vegas, situated fittingly along Historic Route 66. Developers, software architects, engineers, and designers will cruise onto the Strip for five days of unbiased and cutting-edge education on the Microsoft Platform. Navigate the .NET Highway with industry experts and Microsoft insiders in 60+ sessions and fun networking events – all designed to make you better at your job.

SUPPORTED BY

Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

# NAVIGATE THE .NET HIGHWAY

## TRACKS INCLUDE:

- Visual Studio / .NET
- JavaScript/HTML5
- ASP.NET
- Cross-Platform Mobile Development
- Windows 8.1/WinRT
- Data Management
- Cloud Computing
- Windows Phone

## *Register NOW and Save $500!*

Use promo code VSLNOV2

CONNECT WITH VISUAL STUDIO LIVE!

**twitter.com/vslive – @VSLive**

**facebook.com – Search "VSLive"**

**linkedin.com – Join the "Visual Studio Live" group!**

Scan the QR code to register or for more event details.



RACHEL APPEL

# Mobile Web Sites vs. Native Apps vs. Hybrid Apps

One question routinely surfaces in today's modern development landscape—whether to build a mobile Web site versus a native app versus a hybrid app. As a developer, you need to take the time to think through a few considerations before running off to develop software. One such consideration is determining your target audience. To a large degree, this will determine your target platforms.

Your users will use many different devices to access your software. Some will access apps through a corporate network only, while other apps are consumer-focused. Once you've determined the audience and platforms, you must figure out what kind of software will serve the needs of those audiences, potentially with platform-specific features.

There are three main types of modern apps: mobile Web apps, native apps and hybrid apps. I'll review each type, their pros and cons, and how to get started developing them. For the purposes of this article, I'm not considering traditional desktop (native) apps created with Windows Forms or Windows Presentation Foundation (WPF). They're not considered modern, as they only run on large desktop screens and not across a multitude of devices.

## Mobile Web Sites

Mobile Web sites have the broadest audience of the three primary types of applications. Any smartphone can at least display content and let the user interact with a mobile page, although some do so better than others. Along with reach, another benefit is easy deployment. Just update in one location and all users automatically have access to the latest version of the site.

If you already have a Web site and want a companion app or want to expand into the app market, you can start by making your Web site mobile-friendly. This means making a few modifications, but there's a big payoff for a small effort, especially when compared to building a complete native set of apps. Web sites that target desktop or large monitors are hard to use on small devices. Modifying them so they're easy to use on mobile devices will directly affect customer satisfaction.

Making mobility a first-class feature in your site also increases reach. It's easier to use mobile Web sites. There are fewer pop-ups and distractions. Also, mobile design generally leans toward large square or rectangle buttons that are easy to tap.

You can use all your current Web development skills to build a mobilized version of your Web site. That means using HTML, JavaScript, CSS and perhaps a few of your favorite frameworks. The knowledge required to mobilize apps isn't limited to a certain platform or vendor.

Two big things to note about going mobile are integrating a responsive design and restructuring the content so it works on small hardware. CSS media queries cover the responsive design. Media queries are a way to code CSS to define style rules that target specific device form factors. For example, your site should have media queries for several device form factors, including phones, tablets, phablets, laptops and large screens.

Fortunately, you can build media queries that work for several devices within a category. Restructuring the content entails changing the layout to something a tiny screen can display that's easy for users to view. This changes the data volume, as well. There are default media queries that come with Twitter Bootstrap, a popular library that contains responsively designed CSS and styles to get you started.

For example, **Figure 1** contains CSS that works on a large swath of devices. The code in **Figure 1** doesn't cover every scenario, but it covers most of them. You might have some modifications to make to the code to fit with your needs.

The CSS in **Figure 1** not only works in mobile Web apps, but native apps, as well. That means it applies to all three types of apps covered in this article. On the Windows platform, you can use it in Windows Library for JavaScript (WinJS) projects and hybrid apps in C#. For a more in-depth look at responsive app design, see my October 2013 column, "Build a Responsive and Modern UI with CSS for WinJS Apps" (msdn.microsoft.com/magazine/dn451447).

Mobile site UIs and UXes likely won't match that of the host OS, as the Web and native platforms tend to surface certain design patterns and techniques. Many folks try to cram a Web site that targets desktop monitors into the tiny screens of the smartphone or phablet. This rarely works well. Be sure to consider how users consume information on small devices.

One downside to mobile Web sites is that many features available to native apps simply aren't available to mobile Web sites. Even some of the native features hybrids enjoy are out of reach for mobile Web sites. This is primarily for security reasons.

Access to the file system and local resources isn't available in Web sites, whether or not they're mobile. This will change when browsers widely adopt the File API. For now, Mobile IE, Opera Mini and some iOS Safari versions don't support it. Code can't call on the webcam, sensors or other hardware components. At some point, browsers will expose more of the hardware features, but for now it's mostly off-limits.

Figure 1 **CSS Media Queries for Popular Form Factors**

```
Smartphones
Portrait and Landscape
@media only screen and (min-device-width : 320px) and
(max-device-width : 480px) { ... }

Landscape
@media only screen and (min-width : 321px) { ... }

Portrait
@media only screen and (max-width : 320px) { ... }

Tablets, Surfaces, iPads
Portrait and landscape
@media only screen and (min-device-width : 768px) and
(max-device-width : 1024px) { ... }

Landscape
@media only screen and (min-device-width : 768px) and
(max-device-width : 1024px) and (orientation : landscape) { ... }

Portrait
@media only screen and (min-device-width : 768px) and
(max-device-width : 1024px) and (orientation : portrait) { ... }}

Desktops, laptops, larger screens
@media only screen  and (min-width : 1224px) { ... }

Large screen
@media only screen  and (min-width : 1824px) {  ... }

Smartphones
@media only screen and (min-device-width : 320px) and (max-device-width : 480px)
and (orientation : landscape) and (-webkit-min-device-pixel-ratio : 2) { ... }

Portrait
@media only screen and (min-device-width : 320px) and (max-device-width : 480px)
and (orientation : portrait) and (-webkit-min-device-pixel-ratio : 2) { ... }
```

To enable offline capabilities, mobile Web sites have to use Web technologies such as Web Storage, IndexedDb and AppCache. Mobile sites can't take advantage of file system resources, but their sandbox model still allows for some client-based storage. Many existing Web sites don't support offline capabilities, rendering them useless when they're disconnected.

## Native Apps

For most platforms you're targeting, you should be able to retain your skills. If you're developing on Windows, you can power your app with C#, Visual Basic, or C++, alongside XAML for the UI. You could also write in JavaScript, alongside HTML, CSS and WinJS for the UI. On Android, you can write in Java and Objective-C for the iOS.

When going the native route, you can leverage the marketing power of the app store. It doesn't really matter which store. The fact is, they all try to help market your app with free exposure or promos you wouldn't get otherwise. Of course, the downside of an app store is a potential user has to find and install your app. Even with the boost in marketing the store gives you, there will be users who won't find your app.

Something that often blocks you from targeting multiple platforms is the need to recreate at least part of the UI for each targeted platform. That means you'll need a UI for Windows Store and Windows Phone, iOS and Android. It can be quite a challenge to build a flexible UI that works well on dozens of slightly different screens. However, the result is users get that rich, native



msdn magazine

**STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION**

1. Publication Title: MSDN Magazine
2. Publication Number: 1528-4859
3. Filing Date: 9/30/14
4. Frequency of Issue: Monthly
5. Number of Issues Published Annually: 12
6. Annual Subscription Price: US $35, International $60
7. Complete Mailing Address of Known Office of Publication: 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311
8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher: Same as above.
9. Full Name and Complete Mailing Address of Publisher, Editor, and Managing Editor:
   Henry Allain, President, 16261 Laguna Canyon Rd., Ste. 130, Irvine, CA 92618
   Michael Desmond, Editor-in-Chief, 600 Worcester Rd., Ste. 204, Framingham, MA 01702
   Wendy Hernandez, Group Managing Editor, 4 Venture, Ste. 150, Irvine, CA 92618
10. Owner(s): 1105 Media, Inc. dba: 101 Communications LLC, 9201 Oakdale Ave, Ste. 101, Chatsworth, CA 91311. Listing of shareholders in 1105 Media, Inc.
11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or more of the Total Amount of Bonds, Mortgages or Other Securities:
    Nautic Partners V, L.P., 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Kennedy Plaza Partners III, LLC, 50 Kennedy Plaza, 12th Flr., Providence, RI 02903
    Alta Communications 1X, L.P., 1X-B, L.P., Assoc., LLC, 28 State St., Ste. 1801, Boston, MA 02109
12. The tax status has not changed during the preceding 12 months.
13. Publication Title: MSDN Magazine
14. Issue date for Circulation Data Below: September 2014
15. Extent & Nature of Circulation:

| | Average No. Copies Each Month During Preceding 12 Months | No. Copies of Single Issue Published Nearest to Filing Date |
|---|---|---|
| a. Total Number of Copies (Net Press Run) | 83,312 | 81,764 |
| b. Legitimate Paid/and or Requested Distribution | | |
| 1. Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541 | 65,505 | 65,905 |
| 2. In-County Paid/Requested Mail Subscriptions Stated on PS Form 3541 | 0 | 0 |
| 3. Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS® | 4,035 | 3,853 |
| 4. Requested Copies Distributed by Other Mail Classes Through the USPS | 0 | 0 |
| c. Total Paid and/or Requested Circulation | 69,540 | 69,758 |
| d. Nonrequested Distribution | | |
| 1. Outside County Nonrequested Copies Stated on PS Form 3541 | 11,344 | 10,971 |
| 2. In-County Nonrequested Copies Distribution Stated on PS Form 3541 | 0 | 0 |
| 3. Nonrequested Copies Distribution Through the USPS by Other Classes of Mail | 0 | 0 |
| 4. Nonrequested Copies Distributed Outside the Mail | 2,271 | 881 |
| e. Total Nonrequested Distribution | 13,615 | 11,852 |
| f. Total Distribution | 83,155 | 81,610 |
| g. Copies not Distributed | 157 | 154 |
| h. Total | 83,312 | 81,764 |
| i. Percent paid and/or Requested Circulation | 83.63% | 85.48% |

16. Electronic Copy Circulation
    a. Requested and Paid Electronic Copies
    b. Total Requested and Paid Print Copies (Line 15c) + Requested/Paid Electronic Copies
    c. Total Requested Copy Distribution (Line15f) + Requested/Paid Electronic Copies (Line 16a)
    d. Percent Paid and/or Requested Circulation (Both print & Electronic Copies) (16b divided by 16c x 100)
    ☐ **I certify that 50% of all my distributed copies (electronic and paid print are legitimate request or paid copies.**
17. Publication of Statement of Ownership for a Requester Publication is required and will be printed in the November 2014 issue of this publication.
18. I certify that all information furnished on this form is true and complete:
    David Seymour, Director, Print and Online Production

experience they expect from quality apps. In the end, the ratings in the app store reflect apps that provide an excellent UX.

When going native, you'll have to map out a cross-platform strategy. Consider which of the platforms you'll target and the order in which you'll publish them. From the perspective of the mobile Web versus native versus hybrid apps question, the smoothest path is mobile Web to hybrid to native.

Even though you might publish native apps, you'll want to keep the mobile Web site well maintained, as mobile accounts for moat traffic. Then choose a platform, perhaps one that meets your developer background, and start developing. For more guidance on the various considerations when creating cross-platform apps, see my May 2014 column, "Design a Cross-Platform Modern App Architecture" (msdn.microsoft.com/magazine/dn683800).

Visual Studio contains many project templates for creating native apps for the Windows platform. Under C#, Visual Basic and C++, you'll find Windows Store and Windows Phone apps. Visual Studio also contains templates for JavaScript apps. You must first determine the language you'll use, as there are many considerations here, including whether the app will be cross-platform. My September 2013 column, "Understanding Your Language Choices for Developing Modern Apps" (msdn.microsoft.com/magazine/dn385713), can help you decide which language to use. As you might expect, there's a rich ecosystem of tools around native apps including APIs and controls targeting each platform of interest.

Most native apps within a particular platform have a similar navigation paradigm. For example, the Windows Store platform employs app bars and a strategically placed back button. Taking advantage of built-in navigation schemes lets your app give users a familiar feel and no learning curve. This results in better ratings and more downloads. My April 2014 column, "Navigation Essentials in Windows Store Apps" (msdn.microsoft.com/magazine/dn342878), contains all the facts about Windows Store app navigation.

## Hybrid Apps

In that place between mobile Web sites and native apps lie the hybrid apps. Hybrid apps are a way to expose content from existing Web sites in app format. They're a great way to take your Web content and package it up for publishing in an app store. You can publish hybrid apps in any of the major app stores: Microsoft Windows Store, Google Play, Apple App Store, Amazon Appstore and even BlackBerry World.

The great thing about a hybrid app is it can be a published app or a stopgap measure to fill the store while you're working on creating a set of native apps. It gives you great headway into the market to publish something and get the marketing process started while you work on completing a native app set, if that's your goal. If not, a hybrid app can serve as a way to have something formally listed in the app stores for the exposure.

Hybrid apps may enjoy a few more privileges with local resources than mobile Web sites, depending on the host OS rules. That means things such as webcam use or certain sensors might not work everywhere.

The good news if you're considering hybrid apps is you get to use familiar Web development skills. Hybrids are essentially Web site wrappers. Their foundation is the same old HTML, JavaScript and CSS you already know.

There's an entire third-party ecosystem around building hybrid apps for the various app stores. As you might expect, there are templates for creating hybrid apps in Visual Studio. Popular vendors such as Xamarin, Telerik, DevExpress and Infragistics all have tools and controls that speed up the hybrid app development process.

Using an iFrame in Visual Studio JavaScript apps, you can create a hybrid app completely from Web languages. You can also build a Hybrid app using the Windows Phone HTML5 project template with C# or Visual Basic .NET. Finally, take any XAML-based app and add a WebView control for the same effect. The WebView control behaves as if it were a browser. This means you control it by calling methods like Navigate, Refresh or Stop, often mapping to an equivalent user-driven action. Here's a sample of the WebView control and some basic code that navigates to a start page for the app:

```
In MainPage.Xaml
<WebView x:Name="webView"/>
In MainPage.Xaml.Cs
public MainPage()
{
  this.InitializeComponent();
  Uri targetUri = new Uri("http://rachelappel.com");
  webView.Navigate(targetUri);
}
```

You can tap into WebView events to perform navigation, page load or other tasks. For example, you can tap into navigation to log the popular links, as in this example:

```
private void webView_NavigationCompleted(Windows.UI.Xaml.Controls.WebView sender,
  WebViewNavigationCompletedEventArgs args)
{
  logNavigation(args.Uri.ToString());
}
```

This is exactly the kind of event you'd expect when controlling a Web browser. The WebView makes it much easier to combine existing content on the Web with native app capabilities.

## Wrapping Up

Each way of designing and building apps comes with its own set of benefits and drawbacks. The app store concept, for example, is both a pro and a con. The upside is targeted visibility. The downside is you have to develop multiple UIs, although back-end services code is often sharable.

Regardless of whether you're going to build a native or hybrid app, you should have a mobile version of your Web site. Mobile Web sites offer the largest immediate and instant reach of all the types of apps. You don't get to leverage the store's marketing efforts, which can boost sales. Hybrid apps help you enter a marketplace earlier while developing native apps. This is a great way to collect download and usage data, to determine if that's a viable market. Finally, responsive design and responsive CSS add richness to any of the apps discussed here that support Web technologies. ■

**Rachel Appel** *is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, MIX and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Frank La Vigne*



# Next Flight Out

Gisli Olafsson's picture (on.fb.me/111PoWd) doesn't much resemble an angel. But when the massive earthquake hit Haiti in 2010, and Gisli came out of the sky on an Icelandair cargo jet, leading a search and rescue team with tons of relief supplies, he sure looked like one to the victims trapped in the rubble.

Tiny Iceland, population 325,000, hammered by economic meltdown just 15 months previously, put the first rescuer boots on Haitian soil before the ground had even stopped shaking. The trained urban search and rescue team had practiced using listening tools to locate buried survivors. They'd learned how to break through concrete and extract victims without further injury, and without bringing the whole pile down on themselves—they hoped. They risked their lives under unimaginable conditions to rescue strangers.

Fighting disasters requires more than cots and food. Yes, you need those, and drinking water and bandages, too. But without IT, you won't know who brought in the supplies or where you've stored them.

Gisli was a Microsoft employee at the time. A month later, after the Icelandic team had rescued everyone they could, he was back in Reykjavik, setting up my Why Software Sucks talk for local developers. When I enthused, "I'm gonna bring down the house," he grimaced and said, "You might just want to change your phrasing a teeny bit."

I first met Gisli in 2005, at another Microsoft speaking engagement in Iceland. I had dinner with him and his wife a few years later, when he had become Microsoft's Technical Advisor for Disaster Management. In addition to helping with Microsoft's own disaster planning, Microsoft also sent him to help governments and other organizations, such as the United Nations. "Microsoft actually wanted to pay me a salary to combine these two passions" of disaster relief and technology, he said. (Just can the Windows Vista jokes, OK?)

He later moved on to his current position as Emergency Response Director at NetHope (nethope.org), a non-profit consortium of major humanitarian organizations looking to combine their IT forces to meet their world development and disaster relief goals. I doubt they'd decline a donation, if you're feeling generous today.

Fighting disasters requires more than cots and food. Yes, you need those, and drinking water and bandages, too. But without IT, you won't know who brought in the supplies or where you've stored them. You won't get them to the right people in the right places at the right times, and you'll have trouble convincing the outside world to send them. It takes IT, good IT under extremely tough conditions, to coordinate, to deliver, to advocate.

Gisli now lives with his family near Seattle, Wash., with his wife, Sonja Petursdottir. He became a grandfather for the first time this past summer.

He wrote a book on his experiences, "The Crisis Leader" (amzn.to/ZyBeL9), which I'm finding fascinating. He speaks of the shift in a person's life goals, from success to significance. That one is resonating for me.

Sometimes I think our work in the software industry is a waste of time. For example, I was just talking to a major corporation about helping them make their online advertising more effective. Sure, I know that pays for the Web content we all enjoy without coughing up money directly. And, sure, I'll cash their check; I may be crazy, but I'm not stupid. However, I have a hard time believing that's why God put me on earth. Or you, either. When I see a guy in our industry putting it on the line like this, I feel humbled. And if you've read these columns regularly, you'll know I don't humble easily.

I didn't interview Gisli specifically for this column. I tried to do that a year ago, but he was tied up in the Philippines in the aftermath of Typhoon Haiyan. He's too busy to talk again today, and I have to finish this column right now.

Because Gisli is heading out for Liberia as I write this. The fight against Ebola needs IT. He leaves tomorrow. Godspeed, friend. ■

**DAVID S. PLATT** *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*


# .NET Controls for the Professional Developer

ComponentOne delivers Data Visualization, UI, Spreadsheet and Reporting Controls for Microsoft Visual Studio Development

| Reporting | Data Visualization | Spreadsheet |
|---|---|---|

### ActiveReports

Easily create a variety of form-based, analytical and transaction reports. Visualize data with graphical components, design reports in the IDE & extend the reach of your reports with versatile viewers.

### Studio Enterprise

Hundreds of data and UI controls for all .NET platforms including grids, charts, reports and schedulers. Make building next-gen UIs in today's apps easier with an array of samples with source code.

### Spread Studio

A Microsoft Excel-like spreadsheet control with the functionally of an advanced data grid, for adding power to any .NET application. Includes a formula engine and flexible tabular layout.

Download your free trials at
**ComponentOne.com**

Visual Studio Partner

ComponentOne a division of GrapeCity

# SYNCFUSION
# +
# XAMARIN.FORMS


## SYNCFUSION ESSENTIAL STUDIO FOR XAMARIN

- Create cross-platform applications in 100% C#.
- Reuse your existing .NET code assets.
- Design an attractive, flexible native UI for any device.
- $99 for new customers, free for Essential Studio Enterprise Edition license holders.

To learn more about creating stunning cross-platform user interfaces with Xamarin.Forms, please visit

www.syncfusion.com/products/xamarin

Syncfusion®

# Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

**ASPOSE.CELLS IS A PROGRAMMING API** that allows developers to create, manipulate and convert Microsoft Excel spreadsheet files from within their own applications. Its powerful features make it easy to convert worksheets and charts to graphics or save reports to PDF.

A flexible API for simple and complex spreadsheet programming.

Aspose.Cells speeds up working with Microsoft Excel files. The API is a flexible tool for simple tasks such as file conversion, as well as complex tasks like building models. Developers control page layout, formatting, charts and formulas. They can read and write spreadsheet files and save out to a wide variety of image and text file formats.

Fast and reliable, Aspose.Cells saves time and effort compared to using Microsoft Office Automation.

G2 =LINEST(E2:E12,A2:D12,TRUE,TRUE)

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Floor Space (x1) | Offices (x2) | Entrances (x3) | Age (x4) | Assessed Value (y) | | |
| 2 | 2310 | 2 | 2 | 20 | 142000 | | -264.334 |
| 3 | 2333 | 2 | 2 | 12 | 144000 | | 13..26801 |
| 4 | 2366 | 3 | 1.5 | 33 | 151000 | | 0.996748 |
| 5 | 2379 | 3 | 2 | 43 | 150000 | | |
| 6 | 2402 | 2 | 3 | 53 | 139000 | | |
| 7 | 2425 | 4 | 2 | 23 | 169000 | | |

Aspose.Cells lets developers work with data sources, formatting, even formulas.

## Common Uses

- Building dynamic reports on the fly.
- Creating Excel dashboards with charts and pivot tables.
- Rendering and printing spreadsheets and graphics with high fidelity.
- Exporting data to, or importing from, Excel spreadsheets.
- Generating, manipulating and editing spreadsheets.
- Converting spreadsheets to images or other file formats.

## Key Features

- A complete spreadsheet manipulation solution.
- Flexible data visualization and reporting.
- Powerful formula engine.
- Complete formatting control.

## Supported File Formats

XLS, XLSX, XLSM, XMPS, XLTX, XLTM, ODS, SpreadsheetML, tab delim., CSV, TXT, PDF, HTML, and many image formats including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

## Platforms

| Pricing Info | | | | | |
|---|---|---|---|---|---|
| | Standard | Enhanced | | Standard | Enhanced |
| Developer Small Business | $999 | $1498 | Site Small Business | $4995 | $7490 |
| Developer OEM | $2997 | $4494 | Site OEM | $13986 | $20972 |

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

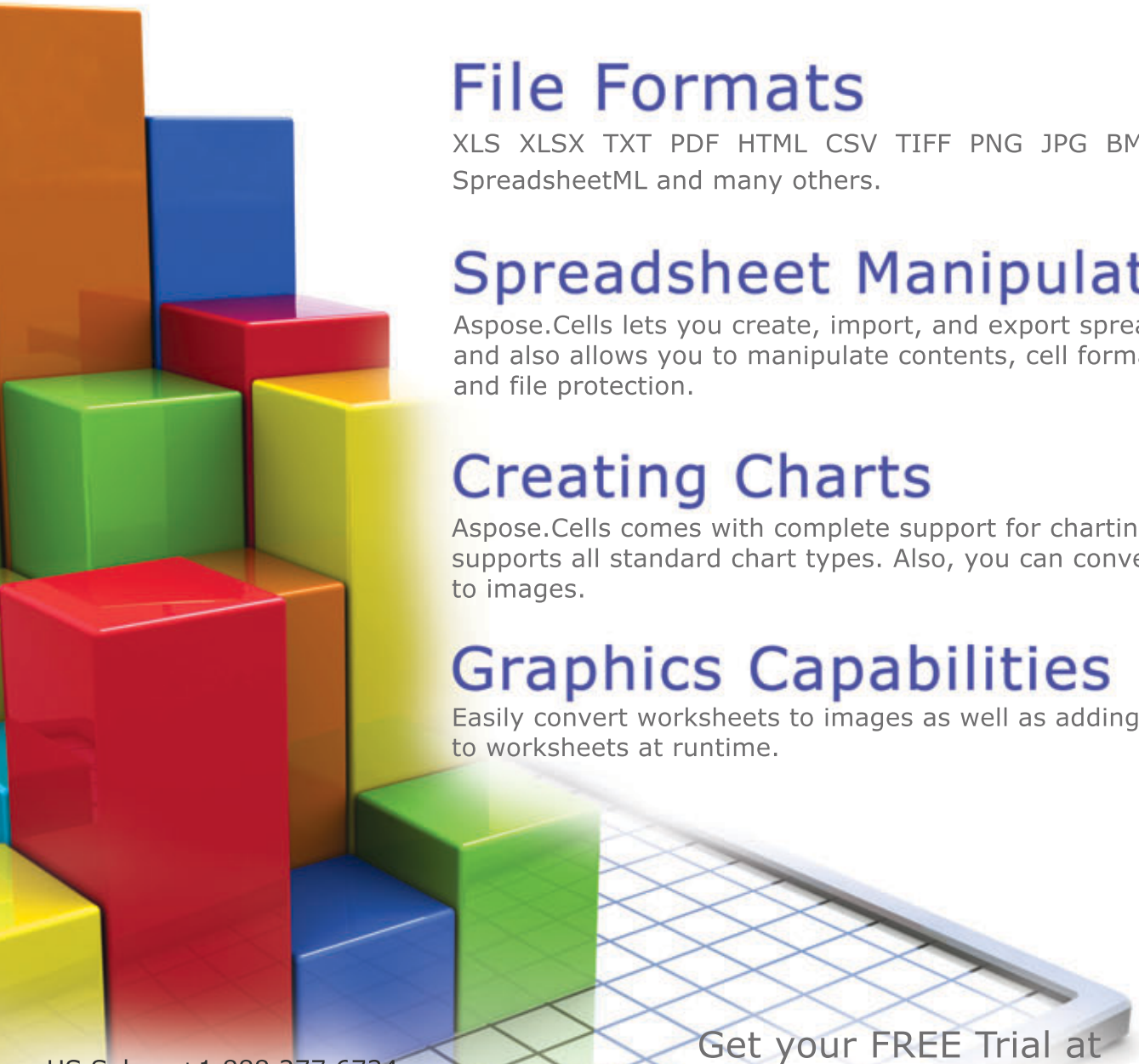
**US:** +1 888 277 6734
sales@aspose.com

**EU:** +44 141 416 1112
sales.europe@aspose.com

**Oceania:** +61 2 8003 5926
sales.asiapacific@aspose.com

# Aspose.Cells for

## .NET, Java, Cloud & Android



## File Formats

XLS XLSX TXT PDF HTML CSV TIFF PNG JPG BMP SpreadsheetML and many others.

## Spreadsheet Manipulation

Aspose.Cells lets you create, import, and export spreadsheets and also allows you to manipulate contents, cell formatting, and file protection.

## Creating Charts

Aspose.Cells comes with complete support for charting and supports all standard chart types. Also, you can convert charts to images.

## Graphics Capabilities

Easily convert worksheets to images as well as adding images to worksheets at runtime.

Get your FREE Trial at
http://www.aspose.com

US Sales: +1 888 277 6734
FAX: +1 866 810 9465
sales@aspose.com

EU Sales: +44 141 416 1112
sales.europe@aspose.com

ASPOSE
Your File Format APIs

### No Office Automation

Aspose.Cells does not require Microsoft Office to be installed on the machine in order to work.

# Aspose.Words

Program with word processing documents independently of Microsoft Word

- Solution for document creation, manipulation and conversion.
- Advanced mail merge functionality.

**ASPOSE.WORDS IS AN ADVANCED PROGRAMMING API** that lets developers perform a wide range of document processing tasks with their own applications. Aspose.Words makes it possible to generate, modify, convert, render and print documents without Microsoft Office Automation. It provides sophisticated and flexible access to, and control over, Microsoft Word files.

Aspose. Words is powerful, user-friendly and feature rich. It saves developers time and effort compared to using Microsoft Office Automation and makes gives them powerful document management tools.

Generate, modify, convert, render and print documents without Microsoft Office Automation.

Aspose.Words makes creating, changing and converting DOC and other word processing file formats fast and easy.

| | Table | | | |
|---|---|---|---|---|
| | **Column 1** | **Column 2** | **Column 3** | **Column 4** |
| **Row 1** | Cell 1 | Cell 2 | Cell 3 | Cell 4 |
| **Row 2** | Cell 1 | Cell 2 | Cell 3 | |
| **Row 3** | Cell 1 | Cell 2 | | |

Aspose.Words has sophisticated controls for formatting and managing tables and other content.

## Common Uses

- Generating reports with complex mail merging; mail merging images.
- Populating tables and documents with data from a database.
- Inserting formatted text, paragraphs, tables and images into Microsoft Word documents.
- Adding barcodes to documents.
- Inserting diagrams and watermarks into Word documents.
- Formatting date and numeric fields.

## Key Features

- A complete Microsoft Word document manipulation solution.
- Extensive mail merge features.
- Complete formatting control.
- High-fidelity conversion, rendering and printing.

## Supported File Formats

DOC, DOCX, ODT, OOXML, XML, HTML, XHTML, MHTML, EPUB, PDF, XPS, RTF, and a number of image formats, including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

## Platforms

| Pricing Info | | | | | |
|---|---|---|---|---|---|
| | Standard | Enhanced | | Standard | Enhanced |
| Developer Small Business | $999 | $1498 | Site Small Business | $4995 | $7490 |
| Developer OEM | $2997 | $4494 | Site OEM | $13986 | $20972 |

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

**US:** +1 888 277 6734
sales@aspose.com

**EU:** +44 141 416 1112
sales.europe@aspose.com

**Oceania:** +61 2 8003 5926
sales.asiapacific@aspose.com

# Case Study: Aspose.Words for .NET

## Lulu helps authors, publishers, businesses, and educators publish and sell print on demand books and ebooks. Why do they use Aspose?

**LULU IS A TECHNOLOGY COMPANY THAT PROVIDES AN OPEN PUBLISHING PLATFORM** where customers from all over the world can create, publish and sell print-on-demand books, ebooks, photobooks and calendars.

The basic function of Lulu's publishing platform is to receive manuscripts from customers and send them to printers for printing. The printers receive the manuscripts in PDF format but that is not always its original format.

> It took only 10 lines of code to integrate Aspose.Words for .NET into the new solution.

Customers can submit manuscripts in any number of formats: many use Microsoft Word. Lulu's publishing platform converts incoming manuscripts to PDFs that can then be sent to the printer. The conversion is automatic: the document comes in, is converted and goes off to print withouthuman intervention or review.

### Updating the Platform

Lulu has been running for several years. The original conversion platform depended on Microsoft Automation for converting DOC files to PDFs. As the business grew and had to accommodate a much higher number of manuscripts, some problems with the existing platform became apparent.

- It did not scale,
- it did not support the latest Microsoft Word document formats, and
- it was not robust.

### Looking for a Solution

The company decided to build a new platform using components that could support their continued growth.

Lulu's engineering team tested Aspose.Words for .NET alongside the existing Microsoft Automation system and other applications. Each solution had its strengths and weaknesses but in the end, Aspose.Words for .NET won because

- it took only 10 lines of code to integrate it into the new platform,
- it is robust and scalable,
- it supports all the file formats that Lulu needs, and
- the licensing structure is straight-forward and cheaper over time than other solutions.

### Outcome

The result was a product that can take any Microsoft Word document that a customer submits, regardless of how the customer may have embellish their manuscript, and convert it to a PDF file that can be printed anywhere.

This is an extract from a case study on our website. For the full version, go to: www.aspose.com/corporate/customers/case-studies.aspx

Customers create manuscripts that can be printed by any printer.

GroupDocs™
work faster. together..

http://groupdocs.com
sales@groupdocs.com

# POWERFUL DOCUMENT MANAGEMENT LIBRARIES AND CLOUD APIS FOR YOUR WEB AND MOBILE APPLICATIONS

.NET, JAVA, CLOUD APIS

## GroupDocs Viewer

True text, high-fidelity embedded document viewer with support for over 50 file formats.

## GroupDocs Signature

Electronic signature capture API that gives your apps legally binding e-signature capabilities.

## GroupDocs Annotation

A powerful API that lets you annotate Microsoft Office, PDF and other documents within your own applications.

## GroupDocs Assembly

Incorporates data entered by users in online forms into PDF and Microsoft Office documents using merge fields.

## GroupDocs Conversion

Universal document converter with an independent engine for fast conversion between more than 50 file formats.

## GroupDocs Comparison

A diff view API that allows end users to quickly find differences between two revisions of a document.

Sales: +1 (214) 329-9760

# Powerful Document Management Solutions for Your Web and Mobile Applications

**GroupDocs offers professional stand-alone .NET & Java libraries along with cloud APIs that allow end-users to view, annotate, convert, e-sign, assemble and compare documents and images of more than 35 file formats within your own web and mobile applications. Key benefits include:**

- All GroupDocs APIs are 100% independent, and don't require any 3rd party software installation.
- Being extremely lightweight, .NET & Java libraries can be integrated with just a single DLL.
- Cloud APIs are supported by SDKs to help developers on .NET, Java, JavaScript, PHP, Python and Ruby seamlessly integrate GroupDocs solutions into any web or mobile apps.
- No need for client-side installation. End-users can work with documents from any web-enabled device and modern web-browser.
- All GroupDocs' products come with a 30-day fully-functional trial and free support during the integration period.

## Stand-Alone .NET & Java Libraries Pricing

GroupDocs .NET & Java licenses are based on the number of developers and the number of locations where the components will be used:

| | Developer Small Business | Developer OEM | Site Small Business | Site OEM |
|---|---|---|---|---|
| License for one developer | ✓ | ✓ | | |
| Licenses for up to 10 developers | | | ✓ | ✓ |
| Use derived work at one location | ✓ | | | |
| Use derived work at up to 10 locations | | | ✓ | |
| Royalty free/deploy to unlimited locations | | ✓ | | ✓ |
| Discount applied to multiple purchases | ✓ | ✓ | ✓ | ✓ |
| Can be used to create unlimited applications | ✓ | ✓ | ✓ | ✓ |
| Updates and hotfixes for one year | ✓ | ✓ | ✓ | ✓ |
| Free technical support | ✓ | ✓ | ✓ | ✓ |
| **Price** | **$2,499** | **$7,497** | **$9,996** | **$29,988** |

## Cloud API Pricing

GroupDocs' cloud APIs use a different licensing model. Instead of licenses, they are charged by use: the number of calls made to the API. To find out more about our cloud API pricing, please visit our website: www.groupdocs.com

### GroupDocs Viewer

A powerful document viewer API that allows you to display over 35 document formats in your web or mobile application. The viewer can both rasterize documents and convert them to SVG+HTML+CSS, delivering true-text high-fidelity rendering.
Supported file formats include: Microsoft Office, Visio, Project and Outlook documents, PDFs, AutoCAD, image files (TIFF, JPG, BMP, GIF, TIFF, etc.) and more.

### GroupDocs Signature

GroupDocs Signature API is an easy way to give your apps legally binding e-signature capabilities. Your users are then able to get documents signed electronically using only a web-browser.
The API gives developers access to sophisticated online signature features, from e-signature capture control and different signing workflows, to reminder management, contact management and signer roles.

### GroupDocs Annotation

With support for over 35 file formats, this API allows your app users to annotate documents of all common business formats, including Microsoft Office and PDF. And thanks to the advanced document management options, users can store, share, print, download and export the annotated documents easily - all from within your own application.

### GroupDocs Assembly

GroupDocs Assembly automatically incorporates data submitted through online forms into existing document templates in PDF or Microsoft Word formats. For each completed form a new custom document is generated. The API lets you build document assembly solutions without getting into the details of working with templates, fields and merging data.

### GroupDocs Conversion

GroupDocs Conversion API allows end users to convert back and forth between over 25 document formats within your own application. It supports all Microsoft Office document formats as well as PDF, HTML and common image file formats (TIFF, JPEG, GIF, PNG, BMP). Your users can convert documents one by one on the fly, or add several documents at a time to a conversion queue.

### GroupDocs Comparison

A document comparison API that allows users to quickly and easily find differences between two revisions of a document right in your web or mobile app. It merges two uploaded documents into a single one and displays it, highlighting differences with the redline view approach - similar to the Microsoft Word change tracking feature, but online. Works with Microsoft Word, Excel, and PowerPoint documents, as well as Adobe Acrobat PDF files.

# Adding File Conversion and Manipulation to Business Systems

How often do people in your organization complain that they can't get information in the file format and layout they want? Converting documents from one format to another without losing layout and formatting should be simple, but it can be frustrating for both users and developers.

**EXTRACTING DATA FROM A DATABASE AND DELIVERING IT TO THE SALES TEAM AS A REPORT**, complete with charts and corporate branding, is fine. Until the sales team says that they want it as a Microsoft Excel file, and could you add a dashboard?

Using information from online forms in letters that can are printed and posted is easy. But what if you also want to add tracking barcodes and archive a digital copy as a PDF?

Aspose creates APIs that work independently of Microsoft Office Automation.

Ensuring that your business system supports all the different Microsoft Office file formats your users want can be difficult. Sometimes the native file format support of your system lets you down. When that is the case, use tools that extend that capability. A good tool can save you time and effort.

## Document Conversion Options

**Building your own solution**: Time-consuming and costly, this option is only sensible if the solution you develop is central to your business.

**Using Microsoft Office Automation**: Microsoft Office Automation lets you use Microsoft Office programs server-side. It is not how the Office products were designed to be used. It can work well but you might notice issues with the stability, security and speed of the system, as well as cost.

**Using an API**: The API market has lots of free and commercial solutions, some very focused, some feature-rich. An API integrates with your code and gives you access to a range of new features.

## Look to Aspose

Aspose are API experts. We create APIs, components and extensions that work independently of Microsoft Automation to extend a platform's native file format manipulation capabilities.

Aspose have developed APIs for .NET, Java, Cloud and Android that lets developers convert, create and manipulate Microsoft Office files – Microsoft Word, Excel, PowerPoint, Visio and Project – and other popular business formats, from PDFs and images to emails. We also have APIs for working with images, barcodes and OCR. The APIs are optimised for stability, speed and ease of use. Our APIs save users weeks, sometimes months, of effort.

### Finding the Right Tool

To find the product that's right for you, take a systematic approach:

- List must-have and nice-to-have features.
- Research the market.
- Ask for recommendations.
- Select a few candidates .
- Run trials.
- Evaluate
  - ease of use,
  - support and documentation,
  - performance, and
  - current and future needs.

# Aspose.BarCode

A complete toolkit for barcode generation and recognition

- Generate barcodes with customer defined size and color.
- Recognize a large number of barcode types from images.

**ASPOSE.BARCODE IS A ROBUST AND RELIABLE BARCODE GENERATION AND RECOGNITION API** that allows developers to add barcode generation and recognition functionality to their applications quickly and easily.

Aspose.BarCode supports most established barcode specifications. It can export generated barcodes to multiple image formats, including BMP, GIF, JPED, PNG and TIFF.

Aspose.BarCode gives you full control over every aspect of the barcode image, from background and bar color, through image quality, rotation angle, X-dimension, captions, and resolution.

**Robust and reliable barcode generation and recognition.**

Aspose.BarCode can read and recognize most common 1D and 2D barcodes from any image and at any angle. Filters help developers clean up difficult to read images to improve recognition.


Aspose.BarCode offers a large number of symbologies and formatting options.

## Common Uses

- Generating and recognizing barcode images.
- Printing barcode labels.
- Enhancing workflow by adding barcode functionality.
- Using recognition functions to drive real-life work processes.

## Key Features

- Barcode generation and recognition.
- Comprehensive support for 1D and 2D symbologies.
- Image processing for improved recognition.

## Supported File Formats

JPG, TIFF, PNG, BMP, GIF, EMF, WMF, EXIP and ICON.

Format support varies across platforms.

## Supported Barcodes

**Linear**: EAN13, EAN8, UPCA, UPCE, Interleaved2of5, Standard2of5, MSI, Code11, Codabar, EAN14(SCC14), SSCC18, ITF14, Matrix 2 of 5, PZN, Code128, Code39 Extended, Code39 Standard, OPC, Code93 Extended, Code93 Standard, IATA 2 of 5, GS1Code128, ISBN, ISMN, ISSN, ITF6, Pharmacode, DatabarOmniDirectional, VIN, DatabarTruncated, DatabarLimited, DatabarExpanded, PatchCode, Supplement **2D**: PDF417, MacroPDF417, DataMatrix, Aztec, QR, Italian Post 25, Code16K, GS1DataMatrix **Postal**: Postnet, Planet, USPS OneCode, Australia Post, Deutsche Post Identcode, AustralianPosteParcel, Deutsche Post Leticode, RM4SCC, SingaporePost, SwissPostParcel

## Platforms

| Pricing Info | | | | | |
|---|---|---|---|---|---|
| | Standard | Enhanced | | Standard | Enhanced |
| Developer Small Business | $599 | $1098 | Site Small Business | $2995 | $5490 |
| Developer OEM | $1797 | $3294 | Site OEM | $8386 | $15372 |

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.


www.aspose.com

**US**: +1 888 277 6734
sales@aspose.com

**EU**: +44 141 416 1112
sales.europe@aspose.com

**Oceania**: +61 2 8003 5926
sales.asiapacific@aspose.com

# Aspose.Email

Work with emails and calendars without Microsoft Outlook

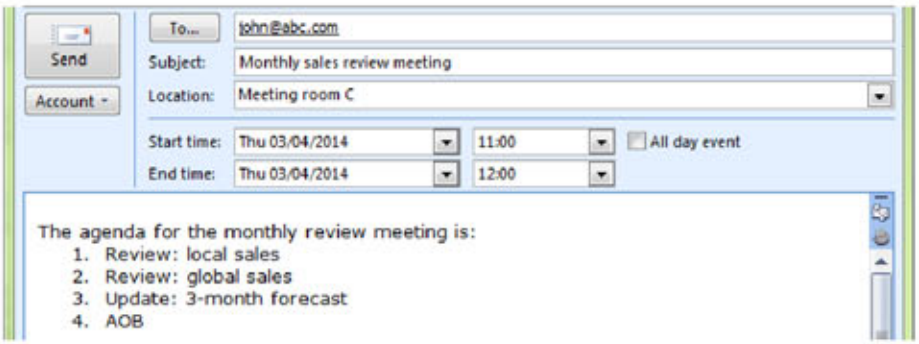
- Complete email processing solution.
- Message file format support.

**ASPOSE.EMAIL IS AN EMAIL PROGRAMMING API** that allows developers to access and work with PST, EML, MSG and MHT files. It also offers an advanced API for interacting with enterprise mail systems like Exchange and Gmail.

Aspose.Email can work with HTML and plain text emails, attachments and embedded OLE objects. It allows developers to work against SMTP, POP, FTP and Microsoft Exchange servers. It supports mail merge and iCalendar features, customized header and body, searching archives and has many other useful features.

Aspose.Email works with HTML and plain text emails, attachments and embedded OLE objects.

Aspose.Email allows developers to focus on managing email without getting into the core of email and network programming. It gives you the controls you need.




**Aspose.Email lets your applications work with emails, attachments, notes and calendars .**

## Common Uses

- Sending email with HTML formatting and attachments.
- Mail merging and sending mass mail.
- Connecting to POP3 and IMAP mail servers to list and download messages.
- Connecting to Microsoft Exchange Servers to list, download and send messages.
- Create and update tasks using iCalendar.
- Load from and save messages to file or stream (EML, MSG or MHT formats).

## Key Features

- A complete email processing solution.
- Support for MSG and PST formats.
- Microsoft Exchange Server support.
- Complete recurrence pattern solution.

## Supported File Formats

MSG, MHT, OST, PST, EMLX, TNEF, and EML.

Format support varies across platforms.

## Platforms

| Pricing Info | | | | | |
|---|---|---|---|---|---|
| | Standard | Enhanced | | Standard | Enhanced |
| Developer Small Business | $599 | $1059 | Site Small Business | $2995 | $5490 |
| Developer OEM | $1797 | $3294 | Site OEM | $8386 | $15372 |

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

# Aspose.Pdf

Create PDF documents without using Adobe Acrobat

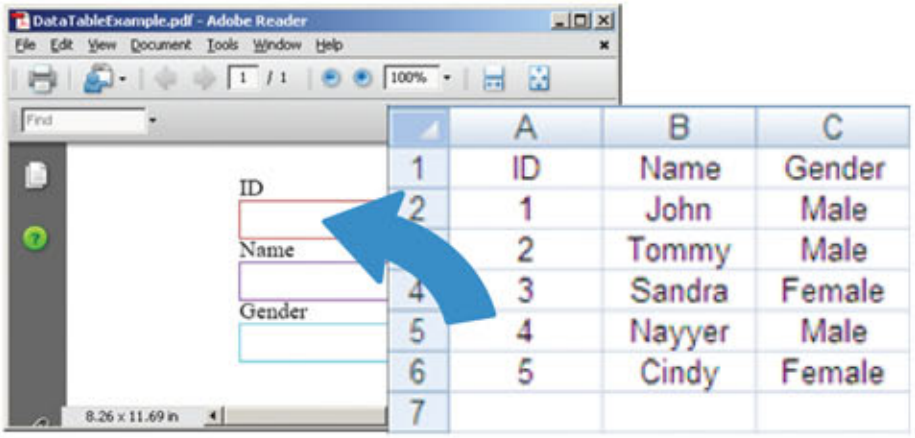
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

**ASPOSE.PDF IS A PDF DOCUMENT CREATION AND MANIPULATION API** that developers use to read, write and manipulate PDF documents without using Adobe Acrobat. Aspose.Pdf is a sophisticated product that integrates with your application to add PDF capabilities.

Aspose.Pdf offers a wealth of features that lets developers compress files, create tables, work with links, add and remove security, handle custom fonts, integrate with external data sources, manage bookmarks, create table of contents, create forms and manage form fields.

Read, write and manipulate PDF documents independently of Adobe Acrobat.

It helps developers add, work with attachments, annotations and PDF form data, add, replace or remove text and images, split, concatenate, extract or inset pages, and print PDF documents.




Aspose.Pdf can be used to automatically complete PDF forms with external data.

## Common Uses

- Creating and editing PDF files.
- Inserting, extracting, appending, concatenating and splitting PDFs.
- Working with text, images, tables, images, headers, and footers.
- Applying security, passwords and signatures.
- Working with forms and form fields.

## Key Features

- PDF creation from XML or XLS-FO documents.
- PDF form and field support.
- Advanced security and encryption.
- High-fidelity printing and conversion.
- Supported File Formats
- PDF, PDF/A, PDF/A_1b, PCL, XLS-FO, LaTeX, HTML, XPS, TXT and a range of image formats.

Format support varies across platforms.

## Platforms

| Pricing Info | | | | | |
|---|---|---|---|---|---|
| | Standard | Enhanced | | Standard | Enhanced |
| Developer Small Business | $799 | $1298 | Site Small Business | $3995 | $6490 |
| Developer OEM | $2397 | $3894 | Site OEM | $11186 | $18172 |

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.


www.aspose.com

**US:** +1 888 277 6734
sales@aspose.com

**EU:** +44 141 416 1112
sales.europe@aspose.com

**Oceania:** +61 2 8003 5926
sales.asiapacific@aspose.com

# Banckle
Customer Service 2.0

http://banckle.com/
sales@banckle.com


Browser Based

Powerful APIs

Device Compatibility

## Customer Service Tools that Run in a Web-Browser

Banckle's online customer service tools help companies large and small engage with customers. Just log in and go: no plugin or software installation necessary. Platform, OS and browser independent.

## Build powerful SaaS apps with Banckle's robust Cloud APIs.

Contact **sales@banckle.com** for API pricing information and an extended 30 day free trial. Alternatively, visit **banckle.com** and use promo code **MiniMag2014** at checkout for a 20% discount.

Banckle.com - Customer Service 2.0

# Customer Service Apps that Help Grow Your Business

Banckle's professional web apps help companies engage with customers.

## Banckle Chat

Chat live with your customers to give them the help they want, when they need it. Embeddable HTML makes it easy to integrate Banckle's live chat service into your website.

Prices start from $6.30/month.

## Banckle Meeting

Give your customers a great online meeting experience with an intuitive interface. Share screen, presentations, work with a whiteboard and use video conferencing from a web browser.

Prices start from $10.50/month.

## Banckle CRM

Keep the team up to date with projects and customer contacts. Keep an eye on the pipeline, manage tasks and log contacts in one central place to improve team work and customer focus.

Prices start from $4.20/month.

## Banckle Helpdesk

Stay on top of your customers' issues with an online service desk and ticketing system. Never lose a ticket but let the team collaborate to deliver the best possible customer support.

Prices start from $11.00/month.

## Banckle Campaign

Find out how effective your email campaigns are and keep in touch with your customers. Design, test and send campaigns, manage mailing lists and learn from campaign reports.

Prices start from $9.80/month.

## Banckle Email

Manage email from an affordable, secure and intuitive online platform. Use all the email features you are used to: attachments, folders and simple account administration.

Prices start from $13.30/month.

## Banckle Total

All Banckle's customer service tools in one package, Banckle Total helps you take customer support to the next level.

Subscriptions start at $31.00/month.

## Try Our APIs

Want to build your own solution?, Consider our RESTful API's

Contact **sales@banckle.com** for API pricing information and an extended 30 day free trial. Alternatively, visit **banckle.com** and use promo code **MiniMag2014** at checkout for a 20% discount.

# Aspose.Slides

Work with presentations without using Microsoft PowerPoint

- Complete solution for working with presentation files.
- Export presentations and slides to portable or image formats.

**ASPOSE.SLIDES IS A FLEXIBLE PRESENTATION MANAGEMENT API** that helps developers read, write and manipulate Microsoft PowerPoint documents. Slides and presentations can be saved to PDF, HTML and image file formats without Microsoft Office Automation.

Aspose.Slides offers a number of advanced features that make it easy to perform tasks such as rendering slides, exporting presentations, exporting slides to SVG and printing. Developers use Aspose.Slides to build customizable slide decks, add or remove standard graphics and automatically publish presentations to other formats.

Aspose.Slides gives you the tools you need to work with presentation files.

Aspose.Slides gives developers the tools they need to work with presentation files. It integrates quickly and saves time and money.


Aspose.Slides has advanced features for working with every aspect of a presentation.

## Common Uses

- Creating new slides and cloning existing slides from templates.
- Handling text and shape formatting.
- Applying and removing protection.
- Exporting presentations to images and PDF.
- Embedding Excel charts as OLE objects.
- Generate presentations from database.

## Key Features

- A complete presentation development solution.
- Control over text, formatting and slide elements.
- OLE integration for embedding external content.
- Wide support for input and output file formats.

## Supported File Formats

PPT, POT, PPS, PPTX, POTX, PPSX, ODP, PresentationML, XPS, PDF and image formats including TIFF and JPG.

Format support varies across platforms.

## Platforms

| Pricing Info | | | | | |
|---|---|---|---|---|---|
| | Standard | Enhanced | | Standard | Enhanced |
| Developer Small Business | $799 | $1298 | Site Small Business | $3995 | $6490 |
| Developer OEM | $2397 | $3894 | Site OEM | $11186 | $18172 |

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.


www.aspose.com

**US:** +1 888 277 6734
sales@aspose.com

**EU:** +44 141 416 1112
sales.europe@aspose.com

**Oceania:** +61 2 8003 5926
sales.asiapacific@aspose.com

# Support Services

Get the assistance you need, when you need it, from the people who know our products best.

- Use experienced Aspose developers for your projects
- Get the level of support that suits you and your team

**NO ONE KNOWS OUR PRODUCTS AS WELL AS WE DO.** We develop them, support them and use them. Our experience is available to you, whether you want us to develop a solution for you, or you just need a little help to solve a particular problem.

## Consulting

Aspose's developers are expert users of Aspose APIs. They understand how to use our products and have hands-on experience of using them for software development. Aspose's developers are skilled not just with Aspose tools but in a wide range of programming languages, tools and techniques.

Aspose's file format experts are here to help you with a project or your support questions

When you need help to get a project off the ground, Aspose's developers can help.

Work with the most experienced Aspose developers in the world.

## Consulting Benefits

- Use Aspose engineers to work on your products
- Get peace of mind from a fully managed development process
- Get a custom-built solution that meets your exact needs

## Support Options

### *Free*

Everyone who uses Aspose products have access to our free support. Our software developers are on stand-by to help you succeed with your project, from the evaluation to roll-out of your solution.

### *Priority*

If you want to know when you'll hear back from us on an issue, and know that your issue is prioritized, Priority Support is for you. It provides a more formal support structure and has its own forum that is monitored by our software engineers.

### *Enterprise*

Enterprise customers often have very specific needs. Our Enterprise Support option gives them access to the product development team and influence over the roadmap. Enterprise Support customers have their own, dedicated issue tracking system.


| Pricing Info |
| --- |
| Each consulting project is evaluated individually; no two projects have exactly the same requirements. |
| To see the Priority and Enterprise support rates, refer to the product price list, or contact our sales team. |

# We're Here to Help YOU

## Aspose has 4 Support Services to best suit your needs

| | |
|---|---|
| Free Support | Support Forums with no Charge |
| Priority Support | 24 hour response time in the week, issue escalation, dedicated forum |
| Enterprise Support | Communicate with product managers, influence the roadmap |
| Sponsored Support | Get the feature you need built now |

**Technical Support** is an issue that Aspose takes very seriously. Software must work quickly and dependably. When problems arise, developers need answers in a hurry. We ensure that our clients receive useful answers and solutions quickly.

Email • Live Chat • Forums


CONTACT US

US Sales: +1 888 277 6734
sales@aspose.com

EU Sales: +44 141 416 1112
sales.europe@aspose.com

AU Sales: +61 2 8003 5926
sales.asiapacific@aspose.com