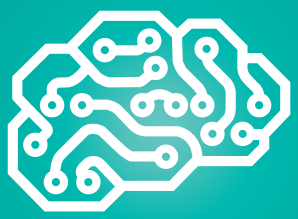
THE MICROSOFT JOURNAL FOR DEVELOPERS

JANUARY 2019 VOL 34 NO 1

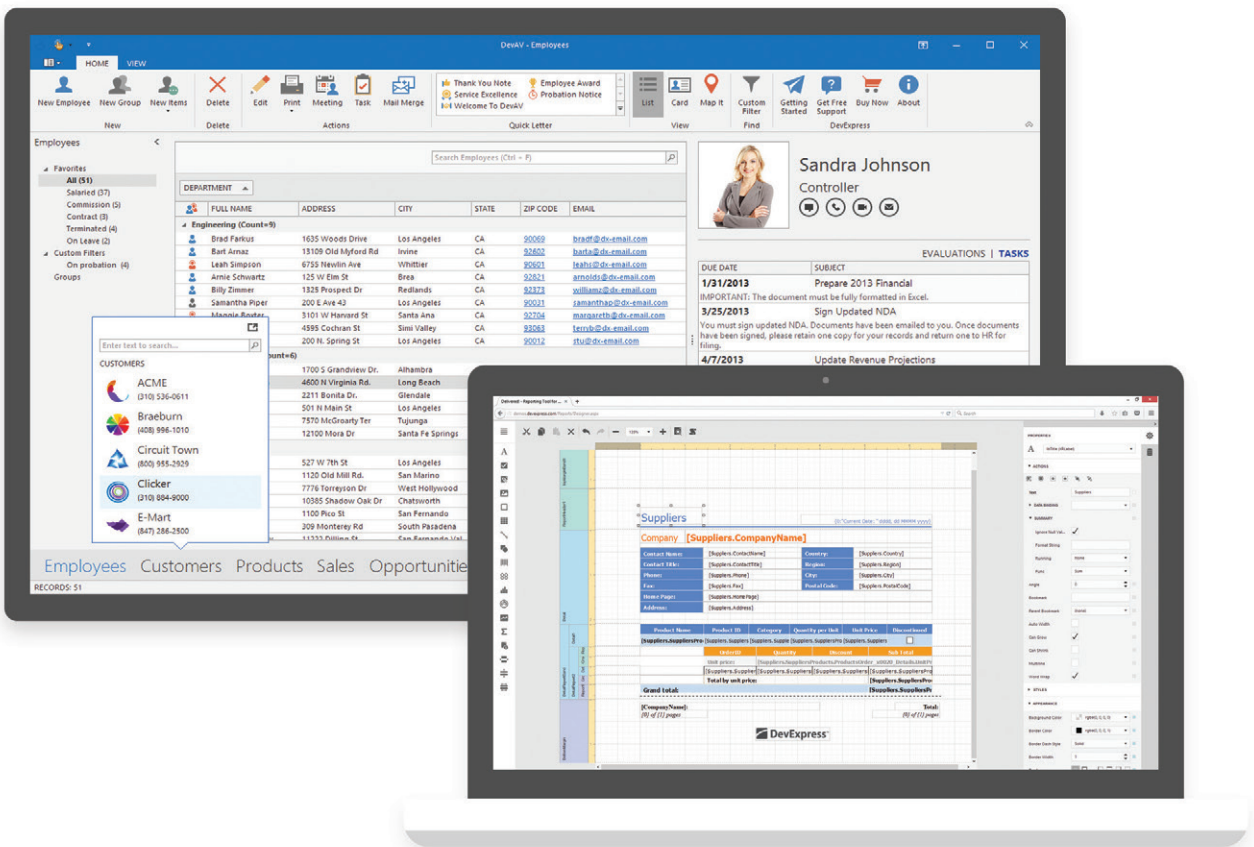
msdn magazine



DevExpress®

Your Next Great App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress UI components for desktops, the web and mobile world will help you build your best, without limits or compromise.



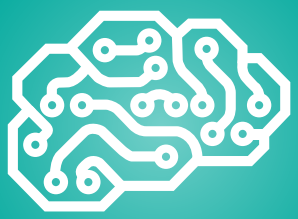
Experience the DevExpress Difference
Download Your Free 30-Day Trial Today

devexpress.com/trial

THE MICROSOFT JOURNAL FOR DEVELOPERS

JANUARY 2019 VOL 34 NO 1

msdn magazine



Machine Learning...16, 24, 32

COLUMNS

Microsoft

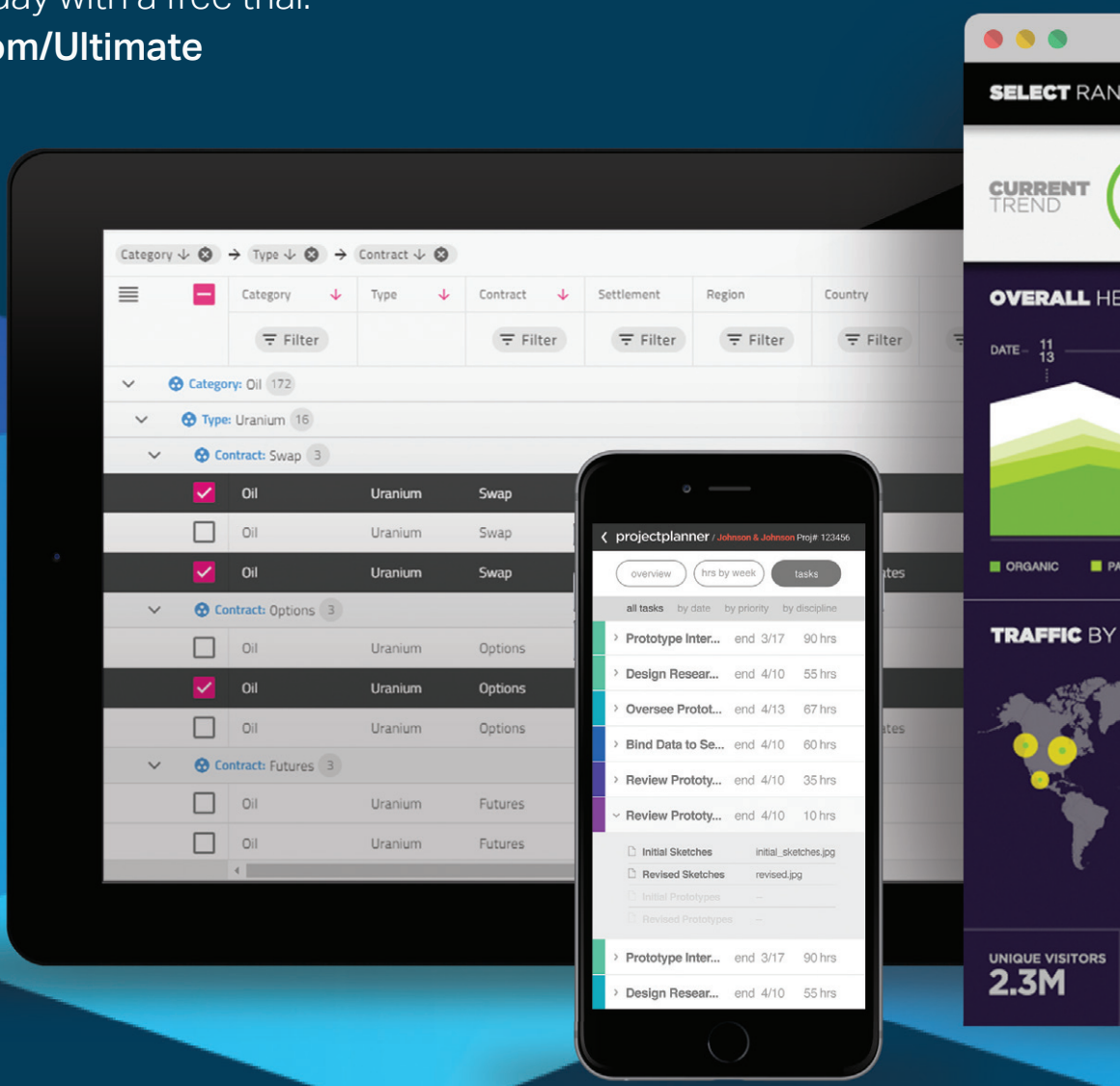
INFRAGISTICS

Faster Paths to Amazing Experiences

Infragistics Ultimate includes 100+ beautifully styled, high performance grids, charts, & other UI controls, plus visual configuration tooling, rapid prototyping, and usability testing.

Angular | JavaScript / HTML5 | ASP.NET | Windows Forms | WPF | Xamarin

Get started today with a free trial:
Infragistics.com/Ultimate



To speak with sales or request a product demo with a solutions consultant call 1.800.231.8588

New Release

Infragistics Ultimate 18.2

- Fastest **grids** & **charts** on the market for the Angular developer
- The most complete Microsoft Excel and Spreadsheet solution for creating dashboards and reports without ever installing Excel
- An end-to-end design-to-code platform with Indigo.Design
- Best-of-breed charts for financial services

msdn magazine

JANUARY 2019 VOLUME 34 NUMBER 1

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 6300 Canoga Avenue, Suite 1150, Woodland Hills, CA 91367. Periodicals postage paid at Woodland Hills, CA 91367 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, File 2272, 1801 W.Olympic Blvd., Pasadena, CA 91199-2272, email MSDNmag@1105service.com or call 866-293-3194 or 847-513-6011 for U.S. & Canada; 00-1-847-513-6011 for International, fax 847-763-9564. 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.
6300 Canoga Avenue, Suite 1150, Woodland Hills 91367
1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@Converge360.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
Web: 1105Reprints.com

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
Email: jlong@meritdirect.com
Web: meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail.
E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com
Irvine Office (weekdays, 9:00 a.m.-5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
2121 Alton Pkwy., Suite 240, Irvine, CA 92606
Corporate Office (weekdays, 8:30 a.m.-5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
6300 Canoga Ave., Suite 1150, Woodland Hills, CA 91367
The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

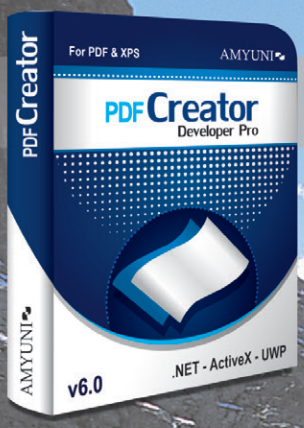
BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

Switch to Amyuni® PDF

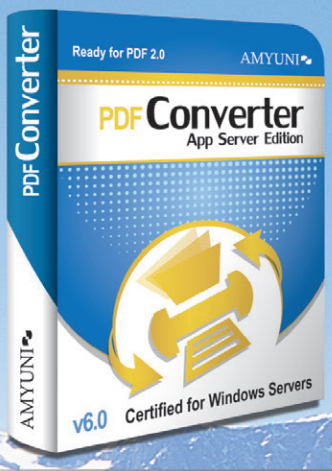
AMYUNI



Create and Process PDFs in .NET, COM/ActiveX and UWP

NEW v6.0

- Edit, process and print PDF 2.0 documents
- Create, fill-out and annotate PDF forms with Javascript support
- Fast and lightweight 64-bit components
- Universal Apps DLLs enable publishing C#, C++, CX or JS apps to Windows Store
- Plus a number of exciting features in v6.0



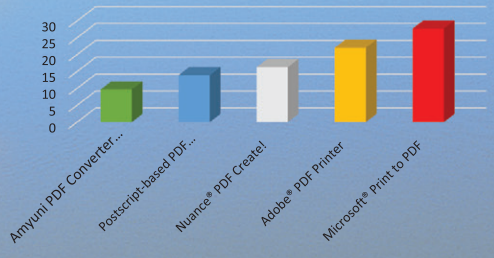
High Performance PDF Printer for Desktops and Servers

Print to PDF in a fraction of the time needed with other tools. WHQL certified for all Windows platforms. Version 6.0 updated for Server 2016.

Complete Suite of PDF, XPS and DOCX 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 and DOCX printer driver
- Export PDF documents into other formats such as images, HTML5 or editable DOCX
- Import/Export XPS files using our native libraries

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



Other Developer Components from Amyuni®

- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract 3 engine for accurate character recognition
- Javascript Engine: Integrate a full JS interpreter into your PDF processing applications
- USB Mobile Monitor: Mirror the display of your Windows or Linux system onto your Android device

Windows Server 2016 Certified

Windows Server 2012 Certified

Windows

All development tools available at
www.amyuni.com

AMYUNI Technologies

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

Editor's Note

MICHAEL DESMOND

Advancing AI

If you've been reading *MSDN Magazine* the past couple years, you know we've aggressively pursued artificial intelligence (AI) and machine learning (ML) as topics of inquiry. From early introductions to Cognitive Services in 2016, to in-depth explorations of the ML.NET framework and Automated Machine Learning in the Connect(); special issue of *MSDN Magazine* last month, our coverage mirrors the urgency Microsoft has shown in advancing the state of the art in AI and ML.

In this month's issue, we present a package of three feature articles focused on AI/ML. James McCaffrey's "Introduction to PyTorch on Windows" offers a glimpse at a promising neural network library that operates at a lower level than others like Microsoft Cognitive Services Toolkit, Google TensorFlow and scikit-learn. Yordan Zaykov, meanwhile, writes "Machine Learning Through Probabilistic Programming," which shows how the Microsoft Infer.NET framework provides algorithms to make probabilistic inferences from data—useful for building statistical models that solve ML problems. Finally, Arnaldo Peréz Castaño authors "Leveraging the Beliefs-Desires-Intentions Agent Architecture," where he walks through applying the ML architecture to a real-life scenario—specifically, implementing a travel assistant agent in C#.

The sustained focus on AI and ML reflects both the profound impact the technology is having on software development, and the speed with which it's evolving. McCaffrey is a senior contributing editor at *MSDN Magazine*, and a senior developer at Microsoft Research where he's deeply engaged in ML development. He says there are currently *dozens* of significant ML projects under development at Microsoft or strongly supported by Microsoft as open source. McCaffrey rattles off a list of projects he's engaged with, including ML.NET, Infer.NET, PyTorch, Project Brainwave and Azure Batch AI and Azure Automated ML.

How can developers stay ahead of this broad waterfront? It's an open question, says McCaffrey. "Developers have a limited amount of time that can be used for forward-looking activities. And because the field of AI/ML is so big and is increasing so rapidly, there's no clear learning roadmap, and everyone, including me, has to roll the dice a bit when making choices about where to focus."

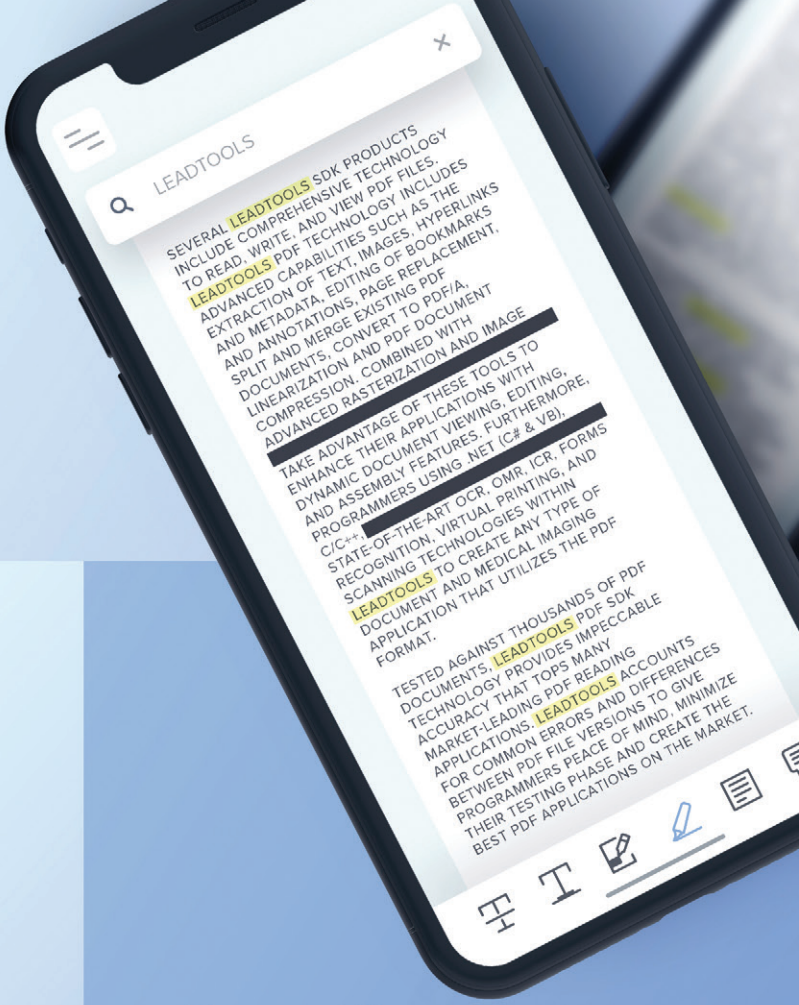
So how do you tilt the odds in your favor? McCaffrey offers a few tips:

- **Choose a Library:** The major deep neural libraries are so different that it's not feasible to learn them all. He advises developers to pick one or two at most, for instance Keras, PyTorch or scikit-learn.
- **Pick up Python:** Don't burn time learning Python, instead pick it up as you go. McCaffrey says most of his developer colleagues—like *MSDN Magazine* subscribers—are primarily C# programmers, and they pick up Python quickly on the fly.
- **Focus on Fundamentals:** Start your ML education by learning four fundamental techniques: binary classification using logistic regression, multiclass classification using a single hidden layer neural network (preferably on the Iris dataset), regression using a single hidden layer neural network (preferably on the Boston Housing dataset) and k-means data clustering.
- **Commit to Learning:** Mastering ML is not a sequential process. McCaffrey describes it as "a huge graph" that requires you to repeatedly examine topics, learning a bit more each time and understanding how topics are interrelated. For instance, most first neural network examples use uniform random initialization. There are many alternatives, but McCaffrey says you should avoid going deep on such topics immediately.
- **Be Wary of Workshops:** Many ML training programs are very expensive and can suffer from low technical quality. Make sure the instructor has a solid background and reputation before committing.

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

LEADTOOLS®
THE WORLD LEADER IN IMAGING SDKs

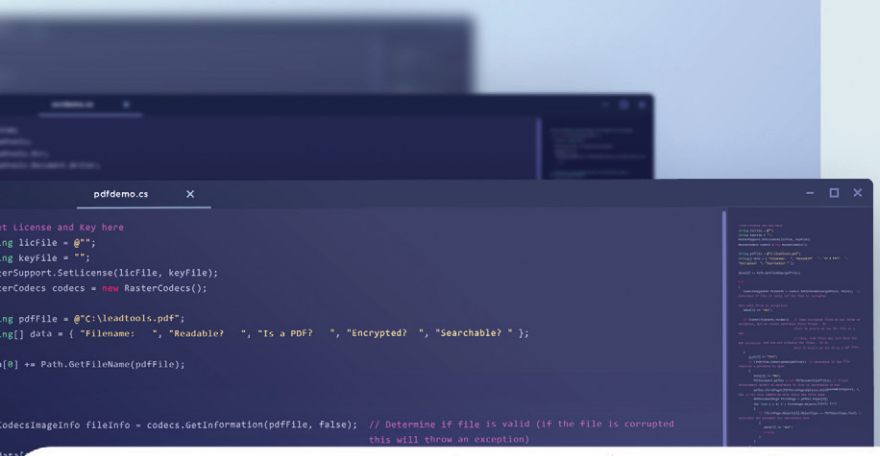
Multi-platform PDF SDK



Comprehensive PDF SDK technology for today's digital world. Develop multi-platform applications with support for extracting, editing, and writing text, hyperlinks, bookmarks, digital signatures, PDF forms, annotations, metadata, and pages from PDF files on any desktop, server, and mobile device.

Specific Features:

- View & Save on Desktop, Web & Mobile
- Extract, Delete, Insert & Replace Pages
- Convert Images to Searchable PDF
- Convert to PDF/A
- Load, Save & Convert Annotations
- PDF Forms



Microsoft .NET · macOS · iOS

Get Started Today
DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

JULIE LERMAN

A Peek at the EF Core Cosmos DB Provider Preview

Regular readers of this column will know that I do a lot of work with Entity Framework (EF) and the newer EF Core, and that also I'm a big fan of Azure Cosmos DB. At first glance, you might think that an Object Relational Mapper (ORM) and a document database such as Cosmos DB would have nothing to do with one another. ORMs are designed to solve the problem of mapping objects to relational databases. With a document database or other type of NoSQL databases, you can just store objects and graphs of objects as JSON and not have to worry about the constraints of a relational database. So why would I, or the EF team or other developers be putting the two together?

Why Use an ORM with a NoSQL Database?

When the earliest betas of EF Core arrived in the form of EF7, they included a proof-of-concept provider to interact with Azure Table Storage, the only NoSQL database on Azure at the time. After trying that out, I realized an important advantage that EF7 provided, and that same advantage exists now with EF Core and Azure Cosmos DB. The clients for interacting with the database, such as the .NET client or the Node.js SDK, require a whole lot of setup code to identify the database, the containers and the query objects. (Note that upcoming versions of the clients will reduce this complexity.) But with EF Core, the first thing I noticed was that I didn't have to write all of that extra code. I just provided a connection string and was good to go. In fact, while acknowledging that this data store isn't relational, I don't have to focus on that while doing perfunctory tasks like querying or saving data. I get to take all of my experience of interacting with data stores using EF Core and just point them to a different data store. The provider takes care of the interpretation of queries and the creation and reading of the JSON documents that get stored into the database.

While the provider can also create databases and containers using conventions, it's still up to you to create the Cosmos DB accounts for the databases and determine their configuration, as well as tune each database to improve performance and reduce costs. These are tasks you perform with any relational database, so it's no different when pointing EF Core to an Azure Cosmos DB database.

The new Cosmos DB provider for EF Core comes as a preview with EF Core 2.2. It's expected to be fully released with EF Core 3.0. But I've been so curious about using it ever since I tested out the proof of concept for Azure Table Storage more than two years ago, so I decided not to wait until it's fully released to check it out. And I'm sure many of you have been curious about it, as well.

The EF Core Cosmos DB provider is in preview. All information is subject to change.

Cosmos DB Client Tools

The Azure Portal has a great Data Explorer for viewing your Cosmos DB databases, containers and documents, but sometimes you don't want to go back and forth from your IDE to a Web site. Both Visual Studio Code and Visual Studio 2017 have great extensions for viewing and editing documents in your Cosmos DB databases. VS Code has the Azure Cosmos DB extension (bit.ly/2SuTXmS) and Visual Studio has the Cloud Explorer for Visual Studio 2017 extension (bit.ly/2G3SNxj). The Azure Cosmos DB extension also lets you work with pre-existing Cosmos DB accounts to create and remove databases and containers on the fly.

Getting the Provider into Your Solution

The provider works like any other EF Core provider. You reference its package in your project and then specify it in OnConfiguring or, if you're using ASP.NET Core, when defining the DbContext in Startup.

The provider is named Microsoft.EntityFrameworkCore.Cosmos. (The name has been shortened from earlier versions, if you were scratching your head over that.) And as with any package, there are a number of ways to add it to your project. In Visual Studio 2017, you can use the Package Manager. At the command line, you can add it with:

```
dotnet add package Microsoft.EntityFrameworkCore.Cosmos
```

Or if you want to simply open up the project file, you can add it in to an ItemGroup section with the following PackageReference:

```
<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore.Cosmos"/>
</ItemGroup>
```

Once you've added the package, you need to tell your DbContext to use the provider. I have a DbContext named ExpanseContext that maps my simplistic model of a TV show/book series I'm fond of: "The Expanse," to my data store—an Azure Cosmos DB database.

As with any of the other providers, this package will give you the UseCosmos extension method on the DbContextOptionsBuilder. From there you need to provide the three key elements of an Azure Cosmos DB connection string: the AccountEndpoint value, the Key value and the name of the database.

You can copy the connection string from within the Azure portal, the Azure CosmosDB extension for VS Code or the Cloud Explorer for Visual Studio 2017 extension. The format required by UseCosmos isn't the same format as the connection string, but the

connection string does give you a head start. You need to supply the three values as three comma-separated parameters. **Figure 1** shows an example where I'm configuring the connection directly in the OnConfiguring method of the ExpanseContext class.

The connection parameters are:

- My account endpoint
- A substitute for my key value and
- The name of the database, ExpanseCosmosDemo

If you're building an ASP.NET Core app, you can configure the context in the ConfigureServices method of the Startup class as follows:

```
services.AddDbContext<ExpanseContext>(options=>
  options.UseCosmos(
    "https://lermandatapoints.documents.azure.com:443",
    "theverylongaccesskeygoeshere",
    "ExpanseCosmosDemo"
);
```

There are some other things you can configure with respect to the Cosmos DB database, but I'd like to show you some of the basic, default behavior first. Part two of this article will demonstrate additional configurations.

The Model Classes

You'll need to be familiar with my model, shown in **Figure 2**. The Expanse tale is about two competing Consortiums, the United Nations and the Martian Congressional Republic. I've dumbed it down to reduce the complexity of changing allegiances or of characters who just go totally rogue and shun both consortiums. Respecting that in the model would be a Domain-Driven Design (DDD) lesson.

This leaves me with a handful of simplistic classes and, again, for the sake of focusing on the behavior of the provider, I've left these as CRUD classes with no real business logic. I'll skip the more interesting Space Station and character classes as I won't be working with them in the demo.

In addition to configuring the provider, the context class specifies DbSets for both the Consortium and Ship entities. Here's the full listing of my context class, ExpanseContext:

```
public class ExpanseContext : DbContext
{
  public DbSet<Consortium> Consortia{get;set;}
  public DbSet<Ship> Ships { get; set; }
  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
  {
    optionsBuilder.UseCosmos(
      "https://lermandatapoints.documents.azure.com:443",
      "theverylongaccesskeygoeshere",
      "ExpanseCosmosDemo");
  }
}
```

Letting EF Core Create the Cosmos DB Database

An Azure Cosmos DB database is tied to a Cosmos DB account and each database stores its documents in various subsets called containers. You may know these as collections in Cosmos DB, but the terminology has recently changed. The documents are now referred to as items. Don't think of containers as relational database tables, though. They are very different. The documentation refers to them as "the unit of scalability for both provisioned throughput and storage of items." Document databases don't predefine data structures, so it's possible to store differently structured documents in a single container. If you're new to document databases, check out my earlier article, "What the Heck Are Document Databases?" at msdn.com/magazine/hh547103. Because Cosmos DB is designed for storing large amounts of data, determining the balance between different containers and partitions is an important process. You should have some familiarity with it before embarking in order to manage performance and costs. You can, of course, fine-tune these aspects as you learn more about how your data is used. I recommend watching the video, "Modeling Data and Best Practices for the Azure Cosmos DB SQL API," at bit.ly/2FZtIDs.

The EF Core Database.EnsureCreated method can create an Azure Cosmos DB database, as well as any needed containers In Azure. If you're using the Windows-based Cosmos DB emulator (bit.ly/2sHNsAn), you can target a local version of the database during development. You'll need to have an existing Cosmos DB account in advance. Most of the important decisions you need to make that impact performance and costs are made when creating an Azure Cosmos DB Account and provisioning the throughput on containers. The fact that EF Core might be creating new databases or containers doesn't force you to accept some unknown defaults,

Figure 1 **Specifying the Cosmos DB Connection in the DbContext Class**

```
using Expanse.Classes;
using Microsoft.EntityFrameworkCore;
public class ExpanseContext : DbContext
{
  public DbSet<Consortium> Consortia{get;set;}
  public DbSet<Planet> Planets { get; set; }
  public DbSet<Character> Characters { get; set; }

  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
  {
    optionsBuilder.UseCosmos(
      "https://lermandatapoints.documents.azure.com:443",
      "theverylongaccesskeygoeshere",
      "ExpanseCosmosDemo"
  );
}
```

Figure 2 **The Expanse Object Model for My Demo**

```
public class Consortium
  {
    public Consortium()
    {
      Ships=new List<Ship>();
      Stations=new List<Station>();
    }
    public Guid ConsortiumId { get; set; }
    public string Name { get; set; }
    public List<Ship> Ships{get;set;}
    public List<Station> Stations{get;set;}
    public Origin Origin{get;set;
  }
  }
  public class Planet
  {
    public Guid PlanetId { get; set; }
    public string PlanetName { get; set; }
  }
  public class Ship
  {
    public Guid ShipId {get;set;}
    public string ShipName {get;set;}
    public Guid PlanetId {get;set;}
    public Origin Origin{get;set;}
  }
  public class Origin
  {
    public DateTime Date{get;set;}
    public String Location{get;set;}
  }
```

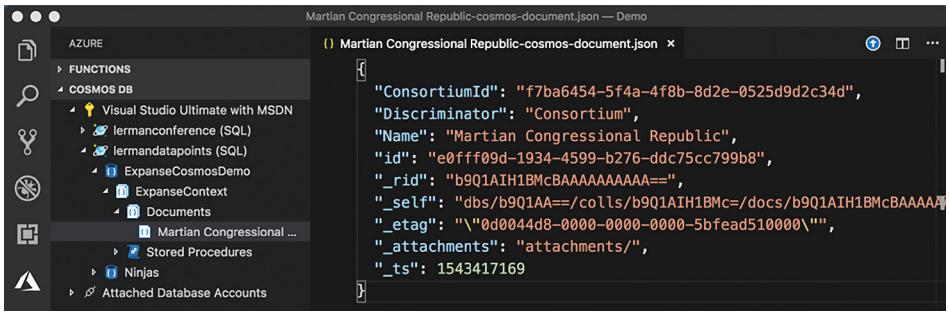


Figure 3 **The Item Created for the New Consortium Displayed by the Cosmos DB Extension**

but there are two defaults to be aware of: First, that EF Core will store all of the documents in a single partition and, second, that it will create a container using the name of your DbContext, storing all of the documents into that container. The reason for this is that the number of containers you have in your database impacts the cost. Therefore, if you don't already have a plan for distributing your items, you can start with everything in one container and one partition, then configure additional partitioned containers as needed to improve performance—again, striving for that balance between performance and cost.

My demo application is just a console app. Its main method triggers a call to my CreateDB method, which calls EF Core's EnsureCreated:

```
private static void CreateDB()
  {
    using(var context=new ExpanseContext())
    {
      context.Database.EnsureCreated();
    }
  }
```

As I'm using the defaults, the first time it's run EnsureCreated will create the new database and a single container, named ExpanseContext. If you explicitly add other containers down the road, EnsureCreated will recognize that the database and initial container already exist and just create the new container for you. I'll look more closely at this in part two of the article.

Storing Data into a Container

Your own code's interaction with EF Core is the same as with any other database. What's interesting, however, is how EF Core interacts with Cosmos DB under the covers.

For example, here's a method to create a single Consortium object, attach it to the context and call SaveChanges:

```
private static void AddObject () {
  var consortium = new Consortium { ConsortiumId = Guid.NewGuid (),
    Name = "Martian Congressional Republic" };
  using (var context = new ExpanseContext ()) {
    context.Consortia.Add (consortium);
    context.SaveChanges ();
  }
}
```

The code is the same as you'd write for a relational database provider. But in the underlying SaveChanges method, EF Core will transform that object to a JSON object and it's the JSON data that gets stored into the database.

But before sending it to Cosmos DB, EF Core adds two special properties to the JSON object. One is a property named id whose value is a GUID. The id will ensure each item in the container has a truly unique id even if they represent different entities. The other added property is named Discriminator and contains the name of the entity type this data represents. The discriminator enables EF Core to distinguish among the entity types the items represent. EF Core is able to create these extra properties, known only to the DbContext, thanks to its shadow properties feature (bit.ly/2PjUq9k). This also means that when you query data, the shadow properties will be returned to the DbContext but ignored when materializing the entity.

The id shadow property doesn't relieve the EF Core requirement that every entity needs a key property. The conventions are the same as always. EF Core still looks for a key property named Id or [entity]Id and, in my case, that's the ConsortiumId property. And you can always override that convention with mappings.

If you work with relational databases, you may be used to relying on the fact that they can generate the key values for you. Cosmos DB is not a relational database and won't populate ConsortiumId or the other keys. However, EF Core will supply values for any missing keys that are GUIDs. It doesn't have a key generator for integers, though, and keeping track of incrementing integers is no fun. So whether you plan to supply the keys or let EF Core do it, I'd highly recommend using GUIDs. My AddObject method creating the ConsortiumId value does this, even though I don't have a true use case for doing so since I'm not using the value elsewhere. It's just for demonstration. If I didn't supply that value when instantiating Consortium, EF Core would've done it for me. Keep in mind that if you're using the HasData method for seeding data, just as with the other providers, you're required to specify the values of primary and foreign key properties.

Figure 3 shows the item that was stored into the container as a result of the AddObject method. The first four properties came from EF Core. But what about the others? Cosmos DB always adds a number of metadata properties that it uses under the covers. However, when you query that data with EF Core, none of those metadata properties are returned to EF Core's DbContext.

What About Graphs of Data?

A consortium can have one or more ships. If I create a new consortium with a ship, my code might look like this;

```
var consortium=new Consortium{ConsortiumId= Guid.NewGuid(),
  ConsortiumName="United Nations"};
consortium.Ships.Add(
  new Ship{ShipId=Guid.NewGuid(),ShipName="Canterbury"});
```

After adding the consortium graph to the context and calling SaveChanges, two new items will get added to the container, as shown in **Figure 4**.

Notice that even though I didn't define a ConsortiumId foreign key property in the Ship class, EF Core knows it will be needed to keep track of the relationship, so it did what it's always done by applying the foreign key value, using its knowledge that the ship object belongs to the consortium object. Depending on my business logic, I often use and control the foreign key properties in my related types. But in case you don't, it's nice to see that the provider will take care of that for you.

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2019

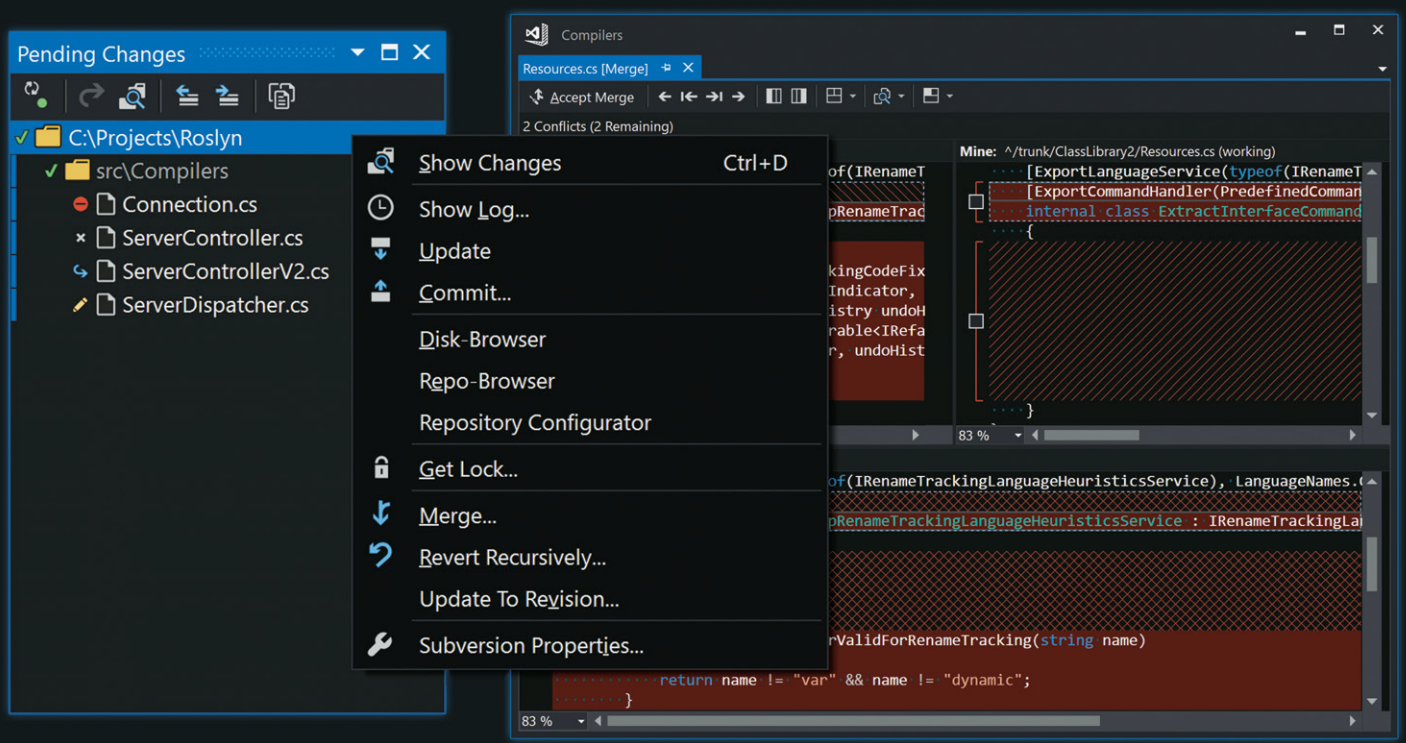
VisualSVN: Painless Version Control for Visual Studio

Six-time Award Winner
by Visual Studio Magazine
Reader's Choice Award

There is something in VisualSVN that makes professional developers love it.

We believe it is a fine-grained mix of a perfect fit with the Apache Subversion® workflow, high performance to support large projects, strong reliability to never crash or hang the IDE, beautiful look and effortless user experience. A number of unique usability features and excellent technical support make VisualSVN the favorite tool for thousands of software developers.

You might be surprised and we know why. If you're still there, give VisualSVN a try.



To learn more, please visit our website → www.visualsvn.com

But what EF Core doesn't do is create a single JSON document with a ship as a subdocument of the consortium. That's because both types have key properties and are true entities in my entity data model as defined by the ExpanseContext, so they'll always be represented as individual documents.

EF Core does, however, understand the concept of hierarchical JSON objects and you can see this in action when you use Owned Entities in your model. I'll look more closely at owned entities with this provider in the second article.

Retrieving Data from Cosmos DB

I'll cover one more topic in this column, with a quick look at data retrieval.

To retrieve data, you can just write LINQ queries as with any other provider. EF Core will use the Cosmos DB SQL API to transform the LINQ queries for getting the items.

And as EF Core does with other providers, any shadow properties defined explicitly by you or implicitly by the provider (such as the Discriminator and id properties), will always be returned to the context as part of the entry. That allows EF Core to materialize the correct object types and maintain the individual identities of each object.

You can see this by querying the ChangeTracker Entries after querying data from the database as I've done here:

```
private static void GetSomeDataBack () {
  using (var context = new ExpanseContext ()) {
    var consortia = context.Consortia.ToList ();
    var entries = context.ChangeTracker.Entries ().ToList ();
  }
}
```

By drilling into the properties of one of the entries, you can see there are five properties, even though Consortium only has two, ConsortiumId and Name. Notice there's even metadata explaining that they're shadow properties. One final shadow property is __jObject, which contains the string representation of the full JSON object of the entry.

Figure 4 **The Items Created Based on a New Consortium Graph Containing a Ship**

```
{
  "ConsortiumId": "09bf2c04-e951-41d7-b890-ea5bc27b5766",
  "ConsortiumName": "United Nations Thursay",
  "Discriminator": "Consortium",
  "id": "fa479b49-144f-47ee-9761-e4f6dfe94cb2",
  "_rid": "QOwDAKsiftgBAAAAAAAAAA==",
  "_self": "dbs/QOwDAA==/colls/QOwDAKsiftg=/docs/QOwDAKsiftgBAAAAAAAAAA==/",
  "_etag": "\"000058c7-0000-0000-0000-5bf80dcf0000\"",
  "_attachments": "attachments/",
  "_ts": 1542983119
}

{
  "ShipId": "581a5c65-8df7-4479-8626-9d8fd2b1c4c7",
  "ConsortiumId": "09bf2c04-e951-41d7-b890-ea5bc27b5766",
  "Discriminator": "Ship",
  "PlanetId": 0,
  "ShipName": "Canterbury 3rd",
  "id": "ebc2dcda-efb5-451b-a65d-f6fa0bb011a4",
  "Origin": null,
  "_rid": "QOwDAKsiftgCAAAAAAAAAA==",
  "_self": "dbs/QOwDAA==/colls/QOwDAKsiftg=/docs/QOwDAKsiftgCAAAAAAAAAA==/",
  "_etag": "\"000059c7-0000-0000-0000-5bf80dd00000\"",
  "_attachments": "attachments/",
  "_ts": 1542983120
}
```

Figure 5 **Loading Related Data Works Just the Same as with RDBMS Providers**

```
private static void EagerLoadInclude () {
  using (var context = new ExpanseContext ()) {
    var consortia = context.Consortia.Include (c => c.Ships).ToList ();
  }
}
private static void EagerLoadProjection () {
  using (var context = new ExpanseContext ()) {
    var consortia =
      context.Consortia.Select (c => new { c, c.Ships }).ToList ();
  }
}
private static void ExplicitLoad () {
  using (var context = new ExpanseContext ()) {
    var consortium =
      context.Consortia.FirstOrDefault (c => c.Name.Contains ("United"));
    context.Entry (consortium).Container (c => c.Ships).Load ();
  }
}
```

Querying Related Data

You can load related data using eager loading with Include or projections, as well as explicit loading. I tested out the proxy-based lazy loading, which didn't return the related data, and have been told that the dependency injection-based lazy loading isn't functional at this time, either.

The three methods in **Figure 5** show working examples of loading related ships using Include, a projection and explicit loading, along with a little bit of filtering, as well. The code is no different than if you were querying against a relational database provider.

Work with Cosmos DB Using Your Existing EF Core Knowledge

Even though Cosmos DB is a completely different type of data store—a document database that stores JSON documents—and is not at all like a relational database, you can leverage your existing knowledge of EF Core to work with it to store and retrieve data. But like any database—relational or not—you'll still need to work outside of EF Core to make sure that you're using the database efficiently and cost-effectively.

In part two of this article, I'll look at some more advanced features, like configuring containers and partitions, integrating owned entities into the mix and using logging to check out some of the SQL generated from the API. And perhaps by then a new version of the preview will be available to engage with additional functionality. If you want to keep an eye on the progress of the provider, there's a great "hit list" of features to work on and consider in the EF Core GitHub repository at bit.ly/2rmUpYN. ■

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

Thanks to the following Microsoft technical experts for reviewing this article: *Andriy Svyryd, Diego Vega*

DevExpress®

From Desktops to Web and Mobile

Your Next Great App Starts Here

Experience the DevExpress difference and see why your peers consistently vote our products #1. With our Universal Subscription, you will build your best, see complex software with greater clarity, increase your productivity and create stunning applications for Windows, Web and your Mobile world.

DevExpress Universal ships with 500+ UI controls.
It also includes our royalty-free reporting and dashboard platform.

WIN ASP MVC WPF UWP JS

Download your free 30-day trial today.

devexpress.com/try

TED NEWARD

Coding Naked

Readers who missed the last column may be surprised that I'm not still talking about the MEAN (Mongo, Express, Angular, Node) stack; that's because I wrapped up the series last time, and now it's time to turn the attention toward something a little different.

In the MEAN series I systematically deconstructed the entire stack and examined each of its constituent parts in detail. Now I'm going to take a look at a stack (well, technology, singular, really) that's designed to blend all of its parts into a single, seamless whole, one that's intended to be used to hide much of the low-level detail work required. In other words, sometimes developers "just want to work with objects," without having to worry about building out a UI, database or intermediate middleware. In many ways, this is the ultimate expression of what Alan Kay had in mind when he invented objects, back in the days of Smalltalk (which was Kay's original object-oriented language), and it's a natural, logical extension of the Domain-Driven Design (DDD) concept (or, perhaps more accurately, the other way around).

Smalltalk, when run, always executes inside a larger environment called a browser. In essence, the browser is a runtime execution environment, IDE and UI host all wrapped into one. (By comparison, an HTML browser is usually just a UI host, although several vendors, including Microsoft, are trying like mad to make the HTML browser into both an IDE and execution environment.) When a developer defines a new class in Smalltalk, the browser knows how to build a generic UI around a given object, and (depending on which vendor's Smalltalk) knows how to persist that to a relational store or to its own object database. The developer doesn't need to define "views" or "controllers," and "models" aren't anemic data-only classes that essentially just define a database schema. In many respects, this is object-oriented the way it was meant to be: Developers interact with customers, find the objects, define properties on those objects (and the relationships between them), define behaviors on those objects and ship the project. It was for this reason that Kay once said, "I invented the term object-oriented, and I can tell you that C++ was not what I had in mind." All jokes at the C++ language's expense aside, his disapproval was with all the thousands of objects that stood between the user and the actual domain object being manipulated. (Well, that and the whole directly manipulating-memory-through-pointers thing, but let's stay focused here.) Which means I can presume he'd be equally disappointed in C# and Java. (And Node.js, to boot.)

Naturally, developers have sought to recreate this Holy Grail of object programming, and I'm going to examine one of these attempts, which goes by the name of Naked Objects. (No, I'm serious, that's its name, which derives from the idea that developers should be focusing solely on the business domain and users should be able to work with the objects directly "without additional decoration," if you will. Or, put another way, users should work with objects in their natural, "naked" state.)

In essence, what's happening in this approach is equally intriguing and intimidating: Based on metadata gathered from an object at run time (via Reflection calls, usually), you generate a UI that knows how to display and edit the properties on the object, validate the edits based on additional metadata specified on the object (usually via custom attributes) and, if necessary, reject those changes that don't meet validation. From there, you query for and store the object to the database (typically through an object/relational mapping layer), along with any additional objects that might require updates, such as objects linked by ownership or some other relation.

Of course, part of the attraction to using the Naked Objects Framework is all the puns to be made. Ready to shuck it all and dive in?

Getting Naked

Fire up a browser, point it at nakedobjects.org and notice that the Web site is a simple redirect site that lets you choose where to go, depending on which platform holds your interest: There's a .NET flavor (which will be my focus) and a Java flavor, which also goes by the name Apache Isis. (Again, not kidding—the folks in charge of Isis are thinking about changing the name, but to be fair, they chose the name long before the folks in the Middle East did.)

When redirected to the .NET flavor, you end up on the GitHub project page for the NakedObjects Framework project, which as of this writing is at version 9. The project page has two notable links on the README homepage: One is to the Developer's Guide, which is a must-have when working with the framework (and a great example of documentation done well), and the other is a .ZIP file containing a template solution to use as a starting point. While the NakedObjects Framework (abbreviated NOF) assemblies are available through NuGet, it's generally easier to use the .ZIP template to begin your new NOF project, for reasons that will become more apparent later. For now, grab the .ZIP template, explode it into a subdirectory for code, and open up the Solution file in your favorite instance of Visual Studio.

When Visual Studio finishes loading, you'll notice that the solution is made up of several different projects, five to be exact. For the most part, their names are self-explanatory: Template.Client will be a Web client, Template.Server is the Web server, and

Template.DataBase and Template.SeedData represent the layers for talking to the database. (In essence, the last two are pretty straightforward Entity Framework projects, so if you already know EF, you've got the persistence part of NOF down, as well.)

The last project in the solution, Template.Model, is where most, if not all, of the developer work will take place. This is the collection of classes that represent the domain model of the project and, therefore, the bulk of the work that a developer will need to do should—and usually will—be done here. Fortunately, the NOF template already has a bit of sample code in it—a Student type, representing one of those creatures who loves to study—so let's just fire it up and watch it go. Make sure that Template.Server is set as the startup project (which it should be already), punch F5 and relax.

Running Naked

First, Visual Studio will fire up the server component and, owing to the EF default configuration, will take a few moments to build the database out of nothing to start. A few seconds after start, some JSON will appear in the browser window—this is because the Template.Server is actually a RESTful server, which means not only does it operate over HTTP, it serves back JSON that describes the entire collection of options that a user can take advantage of if they want. Notice the JSON consists of what basically look like hyperlinks: "rel" for "relation," "href" for the URL to use, "type" to describe what's expected and so on. This is so that developers who don't want to use the generic UI (which I'll examine next) can create their own UI that knows how to work from the JSON being handed back.

Let's look at the UI that NOF builds for you. In a new browser tab, navigate to http://localhost:5001. The result that comes back is … stark. It's clearly not built to be pretty, and to the unfamiliar eye, it looks like there's no real starting point. However, remember that REST (as Fielding originally intended it) and Smalltalk had similar aims: a universal UI, so that regardless of the domain, a user would know how to operate it. NOF is essentially building the UI collectively off of a number of static methods of classes, and these will be displayed in the top-level Menu from that homepage. Clicking it reveals three options: "All Students," "Create New Student" and "Find Student by Name." Pretty clearly, these are simple CRUD methods (well, C and R, anyway), so let's see who's already in the system. Click All Students, and three students (that are fed to EF on startup from the Template.SeedData project) will appear, along with a curious icon in the upper-right of the list returned. Clicking this icon will display the results as a table instead of just a list, but because Students only have a FullName property and nothing else, this won't seem all that interesting yet.

Selecting a student from that list will occupy the full page. Let's assume "James Java" has had a change of heart and wants a legal name change to go with it, so select him from the list, and when he comes up, select Edit. Notice the UI will change to make his name editable, so let's change it to "James Clr." Click Save, and you're back to a read-only UI. Let's go back and see the list of students again, so select the Home icon (the house in the lower-left corner), and you're back to that starting menu again. You could go looking for James by selecting Menu | All Students again, but the clipboard icon shows you a list of all the objects you've used recently. Select it and, sure enough, "James Clr" is there. (The "Student" there is the type of object James happens to be. That's not important in a demo that only has one kind of object, but as the system grows, so will the chances that objects named James might be both a Student and a RegistrationHistory, for example.)

Figure 1 **The Basic Student Type**

```
using NakedObjects;

namespace Template.Model
{
  public class Student
  {
    // All persisted properties on a domain object must be 'virtual'

    NakedObjectsIgnore] // Indicates that this property
                        // will never be seen in the UI
    public virtual int Id { get; set; }

    [Title]// This property will be used for the object's title at
           // the top of the view and in a link
    public virtual string FullName { get; set; }
  }
}
```

Students and Studying

The Student model seems pretty anemic—students are more than just a name! They're also a subject, so let's add that to the model. Stop Visual Studio, and open up the Student.cs file in the Template.Model project. Student currently looks like the code shown in **Figure 1**.

Add a new property to Student, say "Subject," also a string, and also public and virtual, and then punch F5 again. Thanks to the development-default settings in EF, James Java will lose his name change, but notice what you've gained: Throughout the UI and the database, you now have students who have subjects. All of them are currently empty, mind you, but then again, lots of college students have no idea what they're studying, either. The larger point is that with that single property, you've gained complete UI support, as well as database support, without having to write any code to create the UI, validate the input, or persist the data.

Wrapping Up

There's obviously a lot here that I'm just skipping over, and it would be teasing to not explore it further, so I'll spend a few articles examining NOF and determining what its limitations are. The important point is that particularly for applications that aren't consumer-facing, not everything has to be a hand-crafted artisanal UI and database; sometimes the speed with which an application can be generated is far more important than how pretty it looks. NOF and other DDD-inspired kinds of tools can be exactly what the doctor ordered in those kinds of situations, and you even have some room to make things "prettier" by building a more customized Angular (or other SPA framework) front end. Next time I'll explore some of the options and capabilities of domain classes in NOF, and we'll see just how well NOF can handle common UI concerns. In the meantime … Happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor. He has written a ton of articles, authored and co-authored a dozen books, and speaks all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

NEW IN 2019!

On-Demand Session Recordings for One Year!

Get access to all sessions (not including Hands-On Labs or Workshops) at each show for a year. Learn more at vslive.com.

Developer Training Conferences and Events

Choose VSLive! For:

- ✔ In-depth developer training
- ✔ Unparalleled networking
- ✔ World-class speakers
- ✔ Exciting city adventures

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 An 1105 MEDIA company

JOIN US IN 2019!

February 6-7, 2019

Microtek Training Center Dallas

Want to learn more on ASP.NET Core in the Microsoft Cloud? Check out Visual Studio Live!'s upcoming Training Seminar to expand your knowledge and accelerate your career.

REGISTER NOW!
vslive.com/dallas

March 3-8, 2019

Bally's Hotel & Casino

Visual Studio Live! kicks off 2019 in the heart of Las Vegas with 6 days of hard-hitting Hands-On Labs, workshops, 60+ sessions, expert speakers and several networking opportunities included! Register to join us today!

REGISTER NOW!
vslive.com/lasvegas

April 22-26, 2019

Hyatt Regency

For the first time in our 20-year history, Visual Studio Live! is heading down south to New Orleans for intense developer training, bringing our hard-hitting sessions, well-known coding experts and unparalleled networking to the Big Easy!

REGISTER NOW!
vslive.com/neworleans

June 9-13, 2019

Hyatt Regency Cambridge

Join Visual Studio Live! for an amazing view of Beantown, bringin g our infamous speakers for intense developer training, Hands-On Labs, workshops, sessions and networking adventures to the Northeast.

REGISTER NOW!
vslive.com/boston

August 12-16, 2019

Microsoft HQ

Join our Visual Studio Live! experts at the Mothership for 5 days of developer training and special Microsoft perks unique to our other show locations. Plus, we are adding the ever-so popular full-day Hands-On Labs to the agenda in Redmond for the first time this year!

REGISTER NOW!
vslive.com/microsofthq

September 29, 2019

Westin Gas Lamp

Head to the heart of the San Diego Gaslamp District with Visual Studio Live! this Fall as we immerse ourselves with all things for developers, including several workshops, sessions and networking opportunities to choose from.

DETAILS COMING SOON!

October 6-10, 2019

Swissotel

Head to the Windy City and join Visual Studio Live! this October for 5 days of unbiased, developer training and bringing our well-known Hands-On Labs to the city for the first time.

DETAILS COMING SOON!

November 17-22, 2019

Royal Pacific Resort at Universal

Visual Studio Live! Orlando is a part of Live! 360, uniquely offering you 6 co-located conferences for one great price! Stay ahead of the current trends and advance your career – join us for our last conference of the year!

DETAILS COMING SOON!

Introduction to PyTorch on Windows

James McCaffrey

It's possible, though quite difficult, to create neural networks from raw code. Luckily, there are many open source code libraries you can use to speed up the process. These libraries include CNTK (Microsoft), TensorFlow (Google) and scikit-learn. Most neural network libraries are written in C++ for performance but have a Python API for convenience.

In this article I'll show you how to get started with the popular PyTorch library. Compared to many other neural network libraries, PyTorch operates at a lower level of abstraction. This gives you more control over your code and allows you to customize more easily, at the expense of having to write additional code.

The best way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo program reads the well-known Iris dataset into memory. The goal is to predict the species of an Iris flower (setosa, versicolor or virginica) from four predictor values: sepal length, sepal width, petal length and petal width. A sepal is a leaf-like structure.

The complete Iris dataset has 150 items. The demo program uses 120 items for training and 30 items for testing. The demo first creates a neural network using PyTorch, then trains the network using 600 iterations. After training, the model is evaluated using the test data. The trained model has an accuracy of 90.00 percent, which means the model correctly predicts the species of 27 of the 30 test items.

The demo concludes by predicting the species for a new, previously unseen Iris flower that has sepal and petal values (6.1, 3.1, 5.1, 1.1). The prediction probabilities are (0.0454, 0.6798, 0.2748), which maps to a prediction of versicolor.

This article assumes you have intermediate or better programming skills with a C-family language, but doesn't assume you know anything about PyTorch. The complete demo code is presented in this article. The source code and the two data files used by the demo are also available in the download that accompanies this article. All normal error checking has been removed to keep the main ideas as clear as possible.

This article discusses:

- Installing PyTorch
- Preparing the Iris dataset
- Defining the neural network
- Loading the data into memory
- Training the neural network
- Evaluating and using the model

Technologies discussed:

PyTorch, Python, Ananconda3 5.2.0

Code download available at:

msdn.com/magazine/0119magcode

Installing PyTorch

Installing PyTorch involves two steps. First you install Python and several required auxiliary packages such as NumPy and SciPy, then you install PyTorch as an add-on package. Although it's possible to install Python and the packages required to run PyTorch separately, it's much better to install a Python distribution. I strongly

recommend using the Anaconda distribution of Python, which has all the packages you need to run PyTorch, plus many other useful packages. In this article, I address installation on a Windows 10 machine. Installation on macOS and Linux systems is similar.

Coordinating compatible versions of Python, required auxiliary packages and PyTorch is a non-trivial challenge. Almost all the installation failures I've seen have been due to version incompatibilities. At the time I'm writing this article, I'm using Ananconda3 5.2.0 (which contains Python 3.6.5 and NumPy 1.14.3 and SciPy 1.1.0) and PyTorch 0.4.1. These are all quite stable, but because PyTorch is relatively new and under continuous development, by the time you read this article there could be a newer version available.

Before starting, I recommend you uninstall any existing Python systems you have on your machine, using the Windows Control Panel | Programs and Features. I also suggest creating a C:\PyTorch directory to hold installation files and project (code and data) files.

To install the Anaconda distribution, go to repo.continuum.io/archive and look for file Anaconda3-5.2.0-Windows-x86_64.exe, which is a self-extracting executable. If you click on the link, you'll get a dialog with buttons to Run or Save. You can click on the Run button.

The Anaconda installer is very user-friendly. You'll be presented with a set of eight installation wizard screens. You can accept all defaults and just click the Next button on each screen, with one exception. When you reach the screen that asks you if you want to add Python to your system PATH environment variable, the default is unchecked (no). I recommend checking that option so you don't have to manually edit your system PATH. The default settings will place the Python interpreter and 500+ compatible packages in the C:\Users\<user>\AppData\Local\ Continuum\ Anaconda3 directory.

To install the PyTorch library, go to pytorch.org and find the "Previous versions of PyTorch" link and click on it. Look for a file named torch-0.4.1-cp36-cp36m-win_amd64.whl. This is a Python "wheel" file. You can think of a .whl file as somewhat similar to a Windows .msi file. If you click on the link, you'll get an option to Open or Save. Do a Save As and place the .whl file in your C:\PyTorch directory. If you can't locate the PyTorch .whl file, try bit.ly/2SUiAuj, which is where the file was when I wrote this article.

You can install PyTorch using the Python pip utility, which you get with the Anaconda distribution. Open a Windows command shell and navigate to the directory where you saved the PyTorch .whl file. Then enter the following command:

```
C:\PyTorch> pip install torch-0.4.1-cp36-cp36m-win_amd64.whl
```

Installation is quick, but there's a lot that can go wrong. If installation fails, read the error messages in the shell carefully. The problem will almost certainly be a version compatibility issue.

To verify that Python and PyTorch have been successfully installed, open a command shell and enter "python" to launch the Python interpreter. You'll see the ">>>" Python prompt. Then enter the following commands (note there are two consecutive underscore characters in the version command):

```
C:\>python
>>> import torch as T
>>> T.__version__
'0.4.1'
>>> exit()
C:\>
```

If you see the responses shown here, congratulations, you're ready to start writing neural network machine learning code using PyTorch.

Preparing the Iris Dataset

The raw Iris dataset can be found at bit.ly/1N5br3h. The data looks like this:

```
5.1, 3.5, 1.4, 0.2, Iris-setosa
4.9, 3.0, 1.4, 0.2, Iris-setosa
...
7.0, 3.2, 4.7, 1.4, Iris-versicolor
6.4, 3.2, 4.5, 1.5, Iris-versicolor
...
6.2, 3.4, 5.4, 2.3, Iris-virginica
5.9, 3.0, 5.1, 1.8, Iris-virginica
```

The first four values on each line are a flower's sepal length, sepal width, petal length and petal width. The fifth item is the species to predict. The raw data has 50 setosa, followed by 50 versicolor, followed by 50 virginica. The training file is the first 40 of each species (120 items), and the test file is the last 10 of each species (30 items). Because there are four predictor variables, it's not feasible to graph the dataset. But you can get a rough idea of the structure of the data by examining the graph in **Figure 2**.

Neural networks only understand numbers so the species must be encoded. With most neural network libraries, you'd replace setosa with (1, 0, 0), versicolor with (0, 1, 0) and virginica with (0, 0, 1). This is called 1-of-N encoding or one-hot encoding. However, PyTorch performs one-hot encoding behind the scenes and expects 0, 1 or 2 for the three classes. Therefore, the encoded data for PyTorch looks like:

```
5.1, 3.5, 1.4, 0.2, 0
4.9, 3.0, 1.4, 0.2, 0
...
7.0, 3.2, 4.7, 1.4, 1
6.4, 3.2, 4.5, 1.5, 1
...
6.2, 3.4, 5.4, 2.3, 2
5.9, 3.0, 5.1, 1.8, 2
```

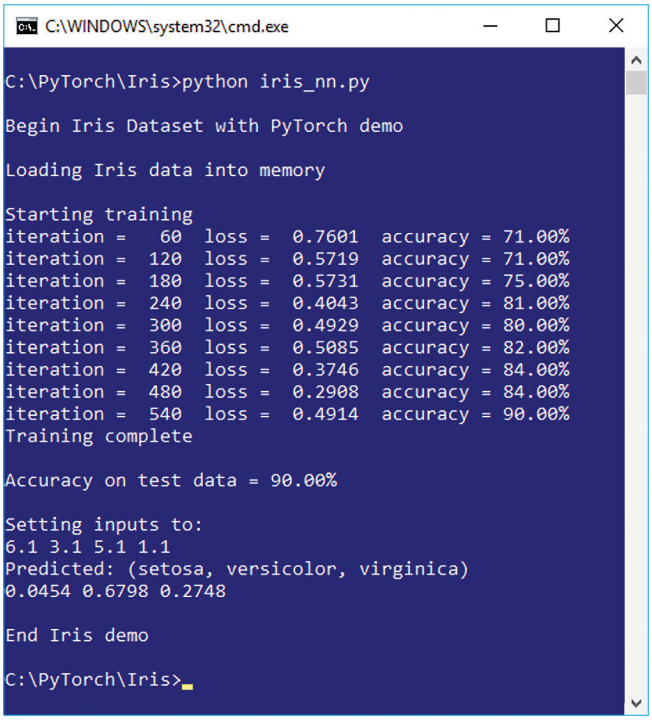


Figure 1 **The Iris Dataset Example Using PyTorch**

In most situations you should normalize the predictor variables, typically by scaling so that all values are between 0.0 and 1.0, using what's called min-max normalization. I didn't normalize the Iris data in order to keep the demo a bit simpler. When working with neural networks, I usually create a root folder for the problem, such as C:\PyTorch\Iris, and then a subdirectory named Data to hold the data files.

The Demo Program

The complete demo program, with a few minor edits to save space, is presented in **Figure 3**. I indent two spaces rather than the usual four spaces to save space. I used Notepad to edit the demo program, but there are dozens of Python editors that have advanced features. Note that Python uses the '\' character for line continuation.

The structure of a PyTorch program differs somewhat from that of other libraries. In the demo, program-defined class Batch serves up a specified number of training items for training. Class Net defines a 4-7-3 neural network. Function accuracy computes the classification accuracy (percentage of correct predictions) of data using a specified model/network. All of the control logic is contained in a main function.

Because PyTorch and Python are being developed so quickly, you should include a comment that indicates what versions are being used. Many programmers who are new to Python are surprised to learn that base Python doesn't support arrays. NumPy arrays are used by PyTorch so you'll almost always import the NumPy package.

Defining the Neural Network

The definition of the neural network begins with:

```
class Net(T.nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.fc1 = T.nn.Linear(4, 7)
    T.nn.init.xavier_uniform_(self.fc1.weight)
    T.nn.init.zeros_(self.fc1.bias)
...
```

The first line of code indicates that the class inherits from a T.nn.Module class, which contains functions for creating a neural network. You can think of the __init__ function as a class constructor. Object fc1 ("fully connected layer 1") is the network hidden layer, which expects four input values (the predictor values) and has seven processing nodes. The number of hidden nodes is a hyperparameter and must be determined by trial and error. The hidden layer weights are initialized using the Xavier uniform algorithm, which is called Glorot uniform in most other libraries. The hidden layer biases are all initialized to zero.

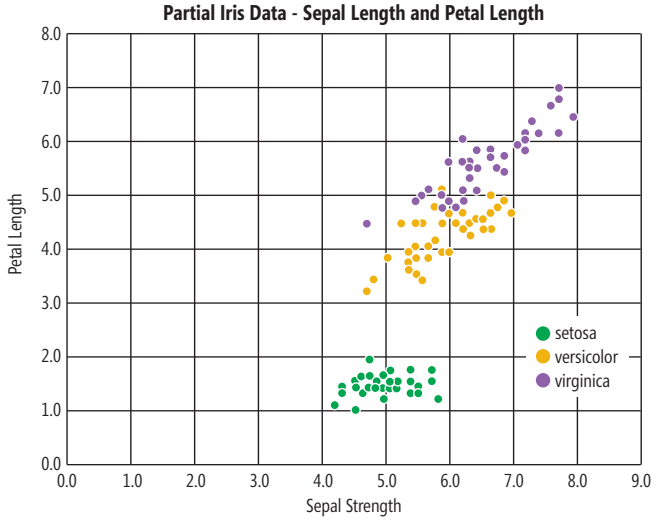


Figure 2 **Partial Iris Data**

> Because PyTorch and Python are being developed so quickly, you should include a comment that indicates what versions are being used.

The network output layer is defined by:

```
self.fc2 = T.nn.Linear(7, 3)
T.nn.init.xavier_uniform_(self.fc2.weight)
T.nn.init.zeros_(self.fc2.bias)
```

The output layer expects seven inputs (from the hidden layer) and produces three output values, one for each possible species. Notice that the hidden and output layer aren't logically connected at this point. The connection is established by the required forward function:

```
def forward(self, x):
  z = T.tanh(self.fc1(x))
  z = self.fc2(z)  # no softmax!
  return z
```

The function accepts x, which is the input predictor values. These values are passed to the hidden layer and the results are then passed to the tanh activation function. That result is passed to the output layer, and the final results are returned. Unlike many neural network libraries, with PyTorch you don't apply softmax activation to the output layer because softmax will be automatically applied by the training loss function. If you did apply softmax to the output layer, your network would still work, but training would be slower because you'd be applying softmax twice.

Loading the Data into Memory

When using PyTorch, you load data into memory in NumPy arrays and then convert the arrays to PyTorch Tensor objects. You can loosely think of a Tensor as a sophisticated array that can be handled by a GPU processor.

There are several ways to load data into a NumPy array. Among my colleagues, the most common technique is to use the Python Pandas (originally "panel data," now "Python data analysis") package. However, Pandas has a bit of a learning curve, so for simplicity the demo program uses the NumPy loadtxt function. The training data is loaded like so:

```
train_file = ".\\Data\\iris_train.txt"
train_x = np.loadtxt(train_file, usecols=range(0,4),
  delimiter=",",  skiprows=0, dtype=np.float32)
train_y = np.loadtxt(train_file, usecols=[4],
  delimiter=",", skiprows=0, dtype=np.float32)
```

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2019

ActivePDF®

14-year Award Winner by Visual Studio Magazine Reader's Choice Award

ActivePDF would like to take this opportunity to thank all of our customers worldwide and VSM subscribers for your continued support and confidence in our entire PDF family of products. For 14 years, **DocGenius™ Toolkit** has been recognized for its powerful .NET PDF API capabilities, enabling developers to create, modify, view, extract, and automate data to and from PDF files within software applications.

Explore the ActivePDF Family of Products

DocGenius™
Rock solid, scalable PDF tools for Windows developers.

- **Toolkit** — Full PDF Manipulation [.NET/COM API]
- **Rasterizer** — PDF to IMG [.NET/COM API]
- **Redactor** — PDF Redaction [.NET API]
- **Server** — In-App PDF Generation [.NET/COM API]
- **Spooler** — Print PDF to Paper [.NET/COM API]
- **Compressor** — PDF Compression [.NET/COM API]
- **WebGrabber** — HTML to PDF [.NET/COM API]
- **Xtractor** — PDF Data Extraction [.NET/COM API]

DocSight™
Server-based PDF content curation automation for digital business-critical applications.

- **DocConverter** — PDF Conversion [Solution w/ API]
- **Meridian** — Network PDF printer [Solution w/ API]
- **OCR** — Full-Text & Zonal OCR [Solution w/ API]

DocSpace™
A family of server applications with user interfaces for viewing, creating, and interacting with PDF.

- **ReaderPlus** — PDF Viewer/Editor [Solution w/ API]

Figure 3 **The Iris Dataset Demo Program**

```
# iris_nn.py
# PyTorch 0.4.1 Anaconda3 5.2.0 (Python 3.6.5)

import numpy as np
import torch as T

# ----------------------------------------------------------

class Batch:
  def __init__(self, num_items, bat_size, seed=0):
    self.num_items = num_items; self.bat_size = bat_size
    self.rnd = np.random.RandomState(seed)

  def next_batch(self):
    return self.rnd.choice(self.num_items, self.bat_size,
      replace=False)

# ----------------------------------------------------------

class Net(T.nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.fc1 = T.nn.Linear(4, 7)
    T.nn.init.xavier_uniform_(self.fc1.weight)  # glorot
    T.nn.init.zeros_(self.fc1.bias)

    self.fc2 = T.nn.Linear(7, 3)
    T.nn.init.xavier_uniform_(self.fc2.weight)
    T.nn.init.zeros_(self.fc2.bias)

  def forward(self, x):
    z = T.tanh(self.fc1(x))
    z = self.fc2(z)  # see CrossEntropyLoss() below
    return z

# ----------------------------------------------------------

def accuracy(model, data_x, data_y):
  X = T.Tensor(data_x)
  Y = T.LongTensor(data_y)
  oupt = model(X)
  (_, arg_maxs) = T.max(oupt.data, dim=1)
  num_correct = T.sum(Y==arg_maxs)
  acc = (num_correct * 100.0 / len(data_y))
  return acc.item()

# ----------------------------------------------------------

def main():
  # 0. get started
  print("\nBegin Iris Dataset with PyTorch demo \n")
  T.manual_seed(1);  np.random.seed(1)

  # 1. load data
  print("Loading Iris data into memory \n")
  train_file = ".\\Data\\iris_train.txt"
  test_file = ".\\Data\\iris_test.txt"

  train_x = np.loadtxt(train_file, usecols=range(0,4),
    delimiter=",",  skiprows=0, dtype=np.float32)
  train_y = np.loadtxt(train_file, usecols=[4],
    delimiter=",", skiprows=0, dtype=np.float32)
  test_x = np.loadtxt(test_file, usecols=range(0,4),
    delimiter=",",  skiprows=0, dtype=np.float32)
  test_y = np.loadtxt(test_file, usecols=[4],
    delimiter=",", skiprows=0, dtype=np.float32)

  # 2. define model
  net = Net()

# ----------------------------------------------------------

  # 3. train model
  net = net.train()  # set training mode
  lrn_rate = 0.01; b_size = 12
  max_i = 600; n_items = len(train_x)

  loss_func = T.nn.CrossEntropyLoss()  # applies softmax()
  optimizer = T.optim.SGD(net.parameters(), lr=lrn_rate)
  batcher = Batch(num_items=n_items, bat_size=b_size)

  print("Starting training")
  for i in range(0, max_i):
    if i > 0 and i % (max_i/10) == 0:
      print("iteration = %4d" % i, end="")
      print("  loss = %7.4f" % loss_obj.item(), end="")
      acc = accuracy(net, train_x, train_y)
      print("  accuracy = %0.2f%%" % acc)

    curr_bat = batcher.next_batch()
    X = T.Tensor(train_x[curr_bat])
    Y = T.LongTensor(train_y[curr_bat])
    optimizer.zero_grad()
    oupt = net(X)
    loss_obj = loss_func(oupt, Y)
    loss_obj.backward()
    optimizer.step()
  print("Training complete \n")

  # 4. evaluate model
  net = net.eval()  # set eval mode
  acc = accuracy(net, test_x, test_y)
  print("Accuracy on test data = %0.2f%%" % acc)

  # 5. save model
  # TODO

# ----------------------------------------------------------

  # 6. make a prediction
  unk = np.array([[6.1, 3.1, 5.1, 1.1]], dtype=np.float32)
  unk = T.tensor(unk)  # to Tensor
  logits = net(unk)  # values do not sum to 1.0
  probs_t = T.softmax(logits, dim=1)  # as Tensor
  probs = probs_t.detach().numpy()    # to numpy array

  print("\nSetting inputs to:")
  for x in unk[0]: print("%0.1f " % x, end="")
  print("\nPredicted: (setosa, versicolor, virginica)")
  for p in probs[0]: print("%0.4f " % p, end="")

  print("\n\nEnd Iris demo")

if __name__ == "__main__":
  main()
```

PyTorch expects the predictor values to be in an array-of-arrays-style matrix and the class values to predict to be in an array. After these statements are executed, matrix train_x will have 120 rows and four columns, and train_y will be an array with 120 values. Most neural network libraries, including PyTorch, use float32 data as the default data type because the precision gained by using 64-bit variables isn't worth the performance penalty incurred.

Training the Neural Network

The demo creates the neural network and then prepares training with these statements:

```
net = Net()
net = net.train()  # set training mode
lrn_rate = 0.01; b_size = 12
max_i = 600; n_items = len(train_x)
loss_func = T.nn.CrossEntropyLoss()  # applies softmax()
optimizer = T.optim.SGD(net.parameters(), lr=lrn_rate)
batcher = Batch(num_items=n_items, bat_size=b_size)
```

Setting the network into training mode isn't required for the demo because training doesn't use dropout or batch normalization, which have different execution flows for training and evaluation. The learning rate (0.01), batch size (12) and maximum training iterations (600) are hyperparameters. The demo uses iterations rather than epochs because one epoch usually refers to processing

File Format APIs

Open, Create, Convert, Print and Save files from your applications!

Try risk free - 30 day trial

ASPOSE

Download a Free Trial at

https://downloads.aspose.com

Aspose.Words

Create, edit, convert or print Word documents (DOC, DOCX, RTF etc.) in your .NET, Java and Android applications.

Aspose.Cells

Develop high performance .NET, Java and Android applications to Create, Edit or Convert Excel worksheets (XLS, XLSX, ODS etc).

Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS etc.) using our native APIs for .NET, Java and Android platforms.

Aspose.Slides

Create, edit or convert PowerPoint presentations (PPT, PPTX, ODP etc.) in your .NET, Java and Android applications.

Aspose.Email

Create, Edit or Convert Outlook Email file formats (MSG, PST, EML etc.) and popular network protocols.

Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet etc.) using our native APIs for .NET and Java.

Aspose.Imaging

Deliver efficient applications to Create, Draw, Manipulate or Convert image file formats.

Aspose.Tasks

Develop high performance apps to Create, Edit or Convert Microsoft Project® document formats.

- Aspose.Diagram
- Aspose.Note
- Aspose.3D
- Aspose.CAD
- Aspose.HTML
- Aspose.GIS

Americas: +1 903 306 1676 EMEA: +44 141 628 8900 Oceania: +61 2 8006 6987
sales@asposeptyltd.com

all training items one time each. Here, one iteration means processing only 12 of the training items.

The CrossEntropyLoss function is used to measure error for multiclass classification problems where there are three or more classes to predict. A common error is to try and use it for binary classification. The demo uses stochastic gradient descent, which is the most rudimentary form of training optimization. For realistic problems, PyTorch supports sophisticated algorithms including adaptive moment estimation (Adam), adaptive gradient (Adagrad) and resilient mean squared propagation (RMSprop).

The program-defined Batch class implements the simplest possible batching mechanism. On each call to its next_batch function, 12 randomly selected indices from the 120 possible training data indices are returned. This approach doesn't guarantee that all training items will be used the same number of times. In a non-demo scenario, you'd likely want to implement a more sophisticated batcher that randomly selects different indices until all have been selected once, and then resets itself.

Training is performed exactly 600 times. Every 600 / 10 = 60 iterations, the demo displays progress information:

```
for i in range(0, max_i):
  if i > 0 and i % (max_i/10) == 0:
    print("iteration = %4d" % i, end="")
    print("  loss = %7.4f" % loss_obj.item(), end="")
    acc = accuracy(net, train_x, train_y)
    print("  accuracy = %0.2f%%" % acc)
```

The average cross entropy loss/error value for the current batch of 12 training items can be accessed through the object's item function. In general, cross entropy loss is difficult to interpret during training, but you should monitor it to make sure that it's gradually decreasing, which indicates training is working.

Somewhat unusually, at the time I'm writing this article, PyTorch doesn't have a built-in function to give you classification accuracy. The program-defined accuracy function computes the classification accuracy of the model using the current weights and biases values. Accuracy is much easier to interpret than loss or error, but it's a cruder metric.

Inside the training loop, a batch of items is selected from the 120-item dataset and converted into Tensor objects:

```
curr_bat = batcher.next_batch()
X = T.Tensor(train_x[curr_bat])
Y = T.LongTensor(train_y[curr_bat])
```

Recall that curr_bat is an array of 12 indices into the training data so train_x[curr_bat] has 12 rows and four columns. This matrix is converted into PyTorch Tensor objects by passing the matrix to the Tensor function. For a classification problem, you must convert the encoded class label values into LongTensor objects rather than Tensor objects.

The actual training is performed by these five statements:

```
optimizer.zero_grad()
oupt = net(X)
loss_obj = loss_func(oupt, Y)
loss_obj.backward()
optimizer.step()
```

You can essentially consider these statements as magic PyTorch incantations that perform training using back-propagation. You must first zero-out the weight and bias gradient values from the previous iteration. The call to the net function passes the current batch of 12 Tensor objects to the network and computes the 12 output values using the forward function. The calls to backward and step compute the gradient values and use them to update weights and biases.

Evaluating and Using the Model

After training completes, the demo computes model accuracy on the test data:

```
net = net.eval()  # set eval mode
acc = accuracy(net, test_x, test_y)
print("Accuracy on test data = %0.2f%%" % acc)
```

As before, setting the model to evaluation mode isn't necessary in this example, but it doesn't hurt to be explicit. The demo program doesn't save the trained model, but in a non-demo scenario you might want to do so. PyTorch, along with most other neural network libraries (with the notable exception of TensorFlow) supports the Open Neural Network Exchange (ONNX) format.

Accuracy is much easier to interpret than loss or error, but it's a cruder metric.

The demo uses the trained model to predict the species of a new, previously unseen Iris flower:

```
unk = np.array([[6.1, 3.1, 5.1, 1.1]], dtype=np.float32)
unk = T.tensor(unk)  # to Tensor
logits = net(unk)  # values do not sum to 1.0
probs_t = T.softmax(logits, dim=1)  # as Tensor
probs = probs_t.detach().numpy()    # to numpy array
```

The call to the net function returns three values that don't necessarily sum to 1.0, for example (3,2, 4.5, 0.3), so the demo applies softmax to coerce the output values so that they do sum to 1.0 and can be loosely interpreted as probabilities. The values are Tensor objects so they're converted into a NumPy array so they can be displayed more easily.

Wrapping Up

This article has just barely scratched the surface of the PyTorch library, but should give you all the information you need to start experimenting. As this article demonstrates, PyTorch is quite different from and operates at a lower level than CNTK, TensorFlow and scikit-learn. A common question is, "Which neural network library is best?" In a perfect world you could dedicate time and learn all the major libraries. But because these libraries are quite complicated, realistically most of my colleagues have one primary library. In my opinion, from a technical point of view, the three best libraries are CNTK, Keras/TensorFlow and PyTorch. But they're all excellent and picking one library over another really depends mostly on your programming style and which one is most used by your colleagues or company. ■

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

Thanks to the following Microsoft technical experts who reviewed this article: *Brian Broll, Yihe Dong, Chris Lee*

Machine Learning Through Probabilistic Programming

Yordan Zaykov

Programming that's probabilistic? Really? That doesn't make much sense ... Or that's what I thought when I started to work in this domain. The researchers I was listening to didn't have the traditional view of placing machine learning (ML) problems into categories. Instead, they just expressed the real-world processes that lead to the generation of their data in a computer-readable form. That's what they call a model and its representation—a probabilistic program.

The paradigm is actually quite appealing. First, you don't need to learn the hundreds of ML algorithms available out there. You just have to learn how to express your problems in a probabilistic program. This involves some knowledge of statistics, because you're modeling the uncertainty of the real world. Let's say, for example, you want to predict the price of a house, which you decide is a linear combination of some features (location, size and so forth). Your model is that the price is the sum of the products of each feature and some weight, with noise added. This so happens to be called a linear regressor. In abstract terms:

```
For each house:
  score = Weights * house.Features;
  house.Price = score + Noise;
```

You can learn the weights during training and then use them directly in prediction. If you make a tiny change in your model by having the noisy score at the end thresholded against zero, the new label is suddenly one of two classes. You've just modeled a binary linear classifier, maybe without even knowing what it's called.

Second, you don't need to try and adapt your problem and data to one of the existing ML algorithms. This should be obvious—you designed the model for your problem, so it fits your data. Modern probabilistic programming tools can automatically generate an ML algorithm from the model you specified, using a general-purpose inference method. You don't even need to know much about it, because it's already implemented for you. So, an application-specific model combined with a general-purpose inference method gives an application-specific ML algorithm.

Finally, the approach seems to withstand the test of time. Most successful ML algorithms fit within this paradigm—a model, representing a set of assumptions, plus an inference method, which does the computations. The methodology has evolved over time. These days, deep neural networks are popular; the model is a composition of thresholded linear functions, and the inference method is called stochastic gradient descent (SGD). Modifying the structure of the

This article discusses:

- An introduction to probabilistic programming
- Creating a sample probabilistic model to represent a skill-rating system
- Using Infer.NET to implement a skill-rating system
- Benefits of probabilistic programming

Technologies discussed:

Infer.NET, C#

network to be recurrent or convolutional; that is, changing the model allows targeting different applications. But the inference method can remain the same—SGD, as shown in **Figure 1**. So, though the choice of model has evolved over the years, the general approach of designing a model and applying an inference method has remained constant.

So, the task for the developer is to come up with a model for his application. Let's learn how to do this.

Hello, Uncertain World!

The first program everyone writes in a new language is "Hello world." The equivalent in a probabilistic setup is, of course, "Hello uncertain world." You can think of the probabilistic program as a simulation or a data sampler. I'll start with some parameters and use them to generate data. For instance, let's have two strings that are completely unknown. That is, they can likely be any string, or in more statistical terms, they're string random variables drawn from a uniform distribution. I'll concatenate them with a space in the middle and constrain the result to be the string, "Hello, uncertain world":

```
str1 = String.Uniform()
str2 = String.Uniform()
Constrain(str1 + " " + str2 == "Hello, uncertain world")
Print(str1)
Print(str2)
```

This is a probabilistic model—I laid out my assumptions of how the data got generated. Now I can run some inference method that will do the necessary computations. The space between the two strings can be either of the two spaces—the one between "Hello" and "uncertain," or the one between "uncertain" and "world." So I can't be sure about the value of either of the two variables. Therefore, the result is a distribution that captures the uncertainty about the possible values. Str1 is equally likely to be "Hello" or "Hello uncertain," and str2 is 50 percent "uncertain world" and 50% percent "world."

A Real Example

Let's move on to a more realistic example now. Probabilistic programming can be used to solve an enormous range of ML problems. For instance, my team developed a recommender system some time ago and shipped it in Azure Machine Learning. Before that, we productized an e-mail classifier in Exchange. Now, we're working on improving the player matchmaking in Xbox by upgrading the skill-rating system. We're also in the process of developing a system for Bing that automatically extracts knowledge from the Internet by modeling unstructured text. All of this is achieved using the same paradigm—define the model as a set of assumptions represented in a probabilistic program, and then have a general-purpose inference method do the necessary computations.

The skill-rating system—called TrueSkill—demonstrates many of the advantages of probabilistic programming, including the ability to interpret the behavior of the system, to incorporate domain knowledge in the model, and to learn as new data arrives. So, let's implement a simplified version of the model that's running in production in blockbuster titles like "Halo" and "Gears of War."

Problem and Data The problem to solve is rating players in games. This has many uses, such as player matchmaking (where fair and enjoyable games are achieved by matching players or teams with similar skills). For simplicity, let's assume there are only two participants in each match and that the outcome is a win or a loss with no draws. So, the data for each match will be the unique identifiers of the two players and an indicator of who wins. I'll use a small handmade dataset in this article, but the approach scales to hundreds of millions of matches in Xbox.

What I'd like to do is learn the skills of all players and be able to predict the winner of future matchups. A naïve approach would be to simply count the number of wins and losses of each player, but this doesn't consider the strength of the opponents in these matches. There's a much better way.

> Probabilistic programming can be used to solve an enormous range of machine learning problems.

Model Design Let's start by making assumptions about how the data came into existence. First, each player has some hidden (or latent) skill that's never observed directly—you only see the effects of their skill. I'll assume this is a real number, but I also need to specify how it was generated. A reasonable choice is that it was generated from a normal (or Gaussian) distribution. In a more complex example, the parameters of this Gaussian would be unknown and learnable. For simplicity I'll set them directly.

The model of the skill can be represented graphically, as shown in the leftmost sketch in **Figure 2**, which simply says that the random variable skill is drawn from a normal distribution.

Another assumption I can make is that in each match players have a performance number. This number is close to their latent skill, but can vary above or below it, depending on whether the player plays better or worse than their typical level. In other words, the performance is a noisy version of the skill. And noise is typically also modeled to be Gaussian, as depicted in the center diagram in **Figure 2**. Here, the performance is a random variable drawn from a Gaussian whose mean is the player's skill and whose variance is fixed, indicated by the hatched noise variable. In a more complicated model I'd try to learn the noise variance from the data, but for simplicity here I'll fix it at 1.

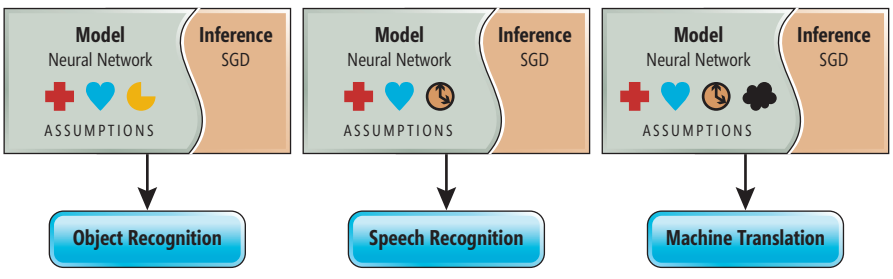


Figure 1 **Different Models for Different Applications, All Using the Same Inference Method**

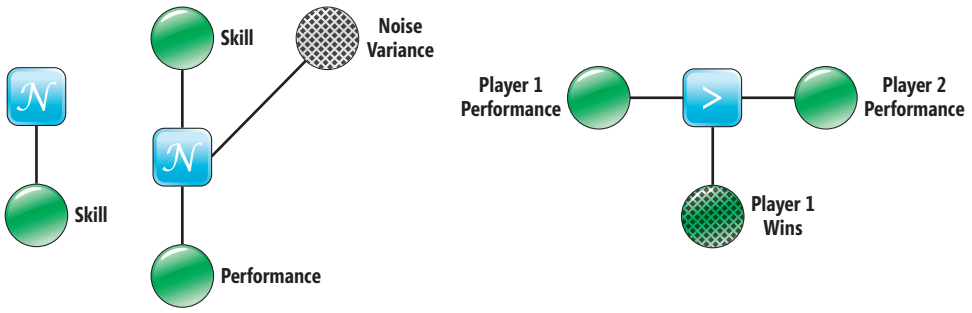


Figure 2 **The Composition of a Two-Player Game**

The player who performs better wins. In a two-player match I can represent this graphically as shown in the rightmost diagram in **Figure 2**. I cheated a bit in this notation by having the Boolean Player 1 Wins variable "somewhat" hatched. That's because its value is observed during training, where the outcomes of the matches are given, but it's not observed during prediction.

Before putting this all together, I need to introduce a couple of new notations. The first is called a *plate* and represents a foreach loop. It's a rectangle that captures parts of the model that need to be repeated over a given range (for example, over players or over games). Second, I'll use dashed lines to indicate selection, as when I select the skills of the two players in each game. The simplified TrueSkill model is shown in **Figure 3**.

Here, I use a plate over the range of players. Then, for each game, I select the latent skills of the two players in the game, add some noise and compare the performances. The noise variable isn't inside any plates because its value is assumed not to change per player or game.

This visualization of the model, also called a factor graph, is a convenient representation in this simple case. But when models grow larger, the diagram becomes confusing and difficult to maintain. That's why developers prefer to express it in code—as a probabilistic program.

Infer.NET

The probabilistic programming framework for .NET is called Infer.NET. Developed by Microsoft Research, it went open source a couple of months ago. Infer.NET provides a modeling API to specify the statistical model, a compiler to generate the ML algorithm from the user-defined model, and a runtime on which the algorithm executes.

Infer.NET is becoming more tightly integrated with ML.NET and is now under the Microsoft.ML.Probabilistic namespace. You can install Infer.NET by running:

```
dotnet add package Microsoft.ML.Probabilistic.Compiler
```

The compiler will automatically pull down the runtime package. Note that Infer.NET runs on .NET Standard and, therefore, on Windows, Linux and macOS.

Let's use C#, and start by including the Infer.NET namespaces:

```
using Microsoft.ML.Probabilistic.Models;
using Microsoft.ML.Probabilistic.Utilities;
using Microsoft.ML.Probabilistic.Distributions;
```

Then, I'll implement the model I defined earlier and see how to train it and how to make predictions.

Model Implementation The model is written like a program that simulates a game, in terms of player skill, performance and outcome, but this program will not actually be run. Behind the scenes, it builds up a data structure that represents the model. When this data structure is given to an inference engine, it's compiled into ML code, which is then executed to perform the actual computation. Let's implement the model in three steps: define the skeleton of the plates over players and games, define the content of the player plate, and define the content of the game plate.

Plates are implemented in Infer.NET using the Range class. Its instances are basically control variables in probabilistic foreach loops. I need to define the size of these plates, which is the number of players and games, respectively. Because I don't know these upfront, they'll be variables used as placeholders for these values. Conveniently, Infer.NET provides the Variable class for just this purpose:

```
var playerCount = Variable.New<int>();
var player = new Range(playerCount);
var gameCount = Variable.New<int>();
var game = new Range(gameCount);
```

With the plates defined, let's focus on their contents. For the player plate, I'll need an array of double random variables for the skills of each player. In Infer.NET this is done using Variable.Array<T>. I'll also need an array of Gaussian random variables for the prior

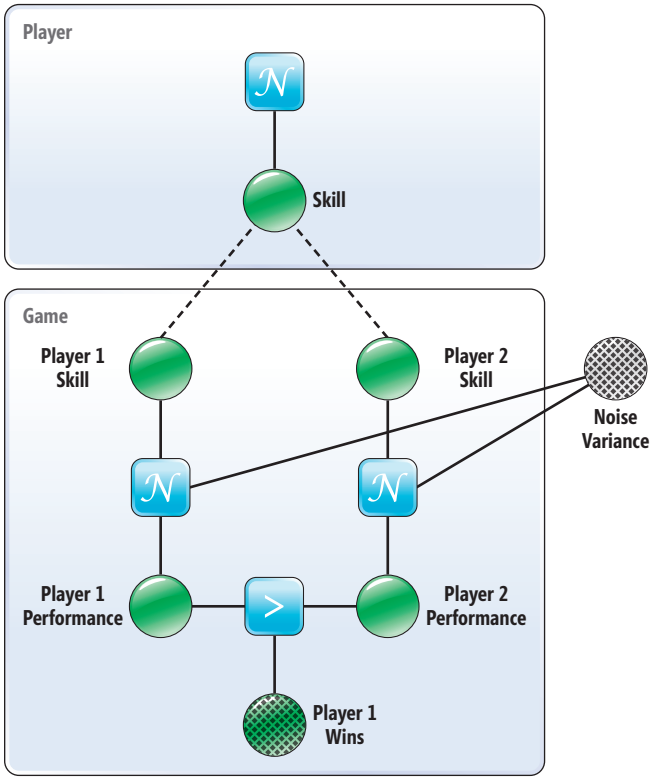


Figure 3 **The Simplified TrueSkill Model**

distribution over the skills of the players. Then, I'll go over the players and connect their skills to the priors. This is achieved using the Variable<T>.Random method. Note how Variable.ForEach(Range) provides the means to implement the internals of the plate:

```
var playerSkills = Variable.Array<double>(player);
var playerSkillsPrior = Variable.Array<Gaussian>(player);
using (Variable.ForEach(player))
{
  playerSkills[player] = Variable<double>.Random(playerSkillsPrior[player]);
}
```

> Notice how I'm creating a model specifically for my data, as opposed to trying to rejig the data to fit some algorithm.

The last piece of the model is the game plate. I'll start by defining the arrays that will contain the training data—the first and second player in each game and the outcomes of the games. Notice how I'm creating a model specifically for my data, as opposed to trying to rejig the data to fit some algorithm. With the data containers ready, I need to go over the games, select the skills of the players in each game, add some noise to them, and compare the performances to generate the game outcome:

```
var players1 = Variable.Array<int>(game);
var players2 = Variable.Array<int>(game);
var player1Wins = Variable.Array<bool>(game);
  const double noise = 1.0;
    using (Variable.ForEach(game))
{
  var player1Skill = playerSkills[players1[game]];
  var player1Performance =
    Variable.GaussianFromMeanAndVariance(player1Skill, noise);
  var player2Skill = playerSkills[players2[game]];
  var player2Performance =
    Variable.GaussianFromMeanAndVariance(player2Skill, noise);
      player1Wins[game] = player1Performance > player2Performance;
}
```

Interestingly, the same model is used for both training and prediction. The difference is that the observed data will be different. During training, you know the match outcomes, while in prediction you don't. So, while the model remains the same, the generated algorithms will differ. Luckily, the Infer.NET compiler takes care of all that.

Training All queries to the model (training, prediction and so forth) go through three steps: set priors, observe data, run inference. Both training and prediction are called "inference" because fundamentally they do the same thing: They use observed data to move from a prior distribution to a posterior distribution. In training, you start from a broad prior distribution over skills, indicating that uncertainty over the skills is high. I'll use a Gaussian distribution for the prior. After observing data, I obtain a narrower Gaussian posterior distribution over the skills.

For the prior on skills I'll just borrow parameters learned from "Halo 5"—a good choice for mean and variance are 6.0 and 9.0, respectively. I set these values by assigning them to the ObservedValue property of the variable holding the prior. For four players, the code would look like this:

```
const int PlayerCount = 4;
playerSkillsPrior.ObservedValue =
  Enumerable.Repeat(Gaussian.FromMeanAndVariance(6, 9),
  PlayerCount).ToArray();
```

Next comes the data. For each game, I have the two players and the outcome. Let's work with a fixed example. **Figure 4** shows three matches played by four players, with arrows indicating a match that was played, each pointing to the loser of the match.

To simplify the code, I'll assume each player has been assigned a unique id. In this example, the first game is between Alice and Bob, and the arrow indicates that Alice wins. In the second game Bob beats Charlie, and finally Donna beats Charlie. Here it's expressed in code, again using ObservedValue:

```
playerCount.ObservedValue = PlayerCount;
gameCount.ObservedValue = 3;
players1.ObservedValue = new[] { 0, 1, 2 };
players2.ObservedValue = new[] { 1, 2, 3 };
player1Wins.ObservedValue = new[] { true, true, false };
```

Finally, I run inference by instantiating an inference engine and calling Infer on the variables I want to obtain. In this case, I'm interested only in the posterior over the player skills:

```
var inferenceEngine = new InferenceEngine();
var inferredSkills = inferenceEngine.Infer<Gaussian[]>(playerSkills);
```

The Infer statement here actually does a lot of work because it performs two compilations (Infer.NET and C#) and an execution. What happens is that the Infer.NET compiler traces the passed playerSkills variable to the probabilistic model, builds an abstract syntax tree from the model code and generates an inference algorithm in C#. Then the C# compiler gets invoked on the fly and the algorithm is executed against the observed data. Because of all that, you might find the calls to Infer somewhat slow, even though you're operating on very little data here. The Infer.NET manual explains how to speed this process up, by working with precompiled algorithms. That is, you perform the compilation steps upfront, so that only the computation part is executed in production.

For this example, the inferred skills are:

```
Alice: Gaussian(8.147, 5.685)
Bob: Gaussian(5.722, 4.482)
Charlie: Gaussian(3.067, 4.814)
Donna: Gaussian(7.065, 6.588)
```

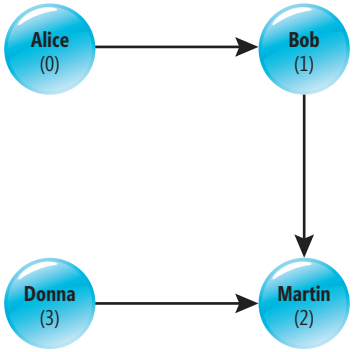


Figure 4 **The Results of Three Matches**

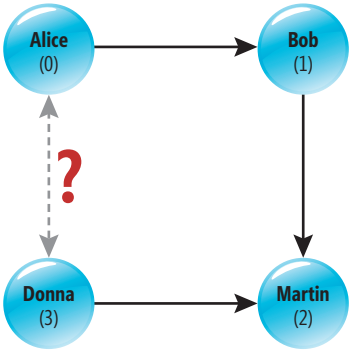


Figure 5 **Predicting the Outcome of a Match**

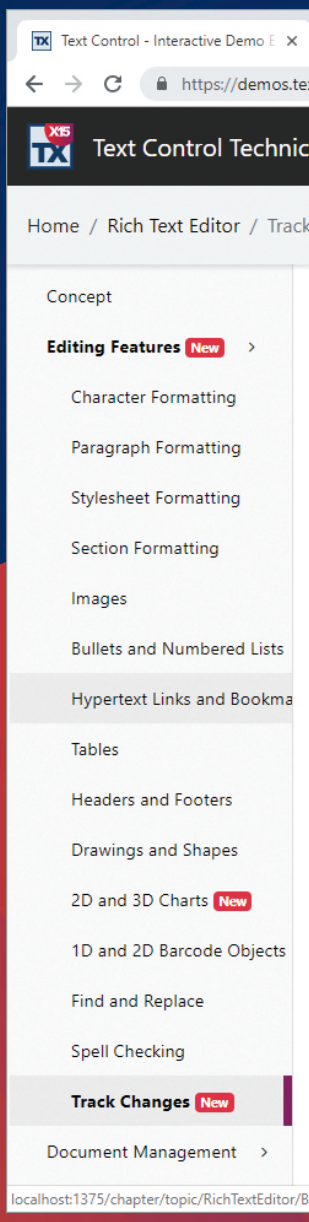
TEXT CONTROL

INTEGRATE DOCUMENT COLLABORATION

Integrate MS Word compatible track changes into cross-platform web applications. Share and review documents with a true WYSIWYG document editor.

See our technology live:

demos.textcontrol.com

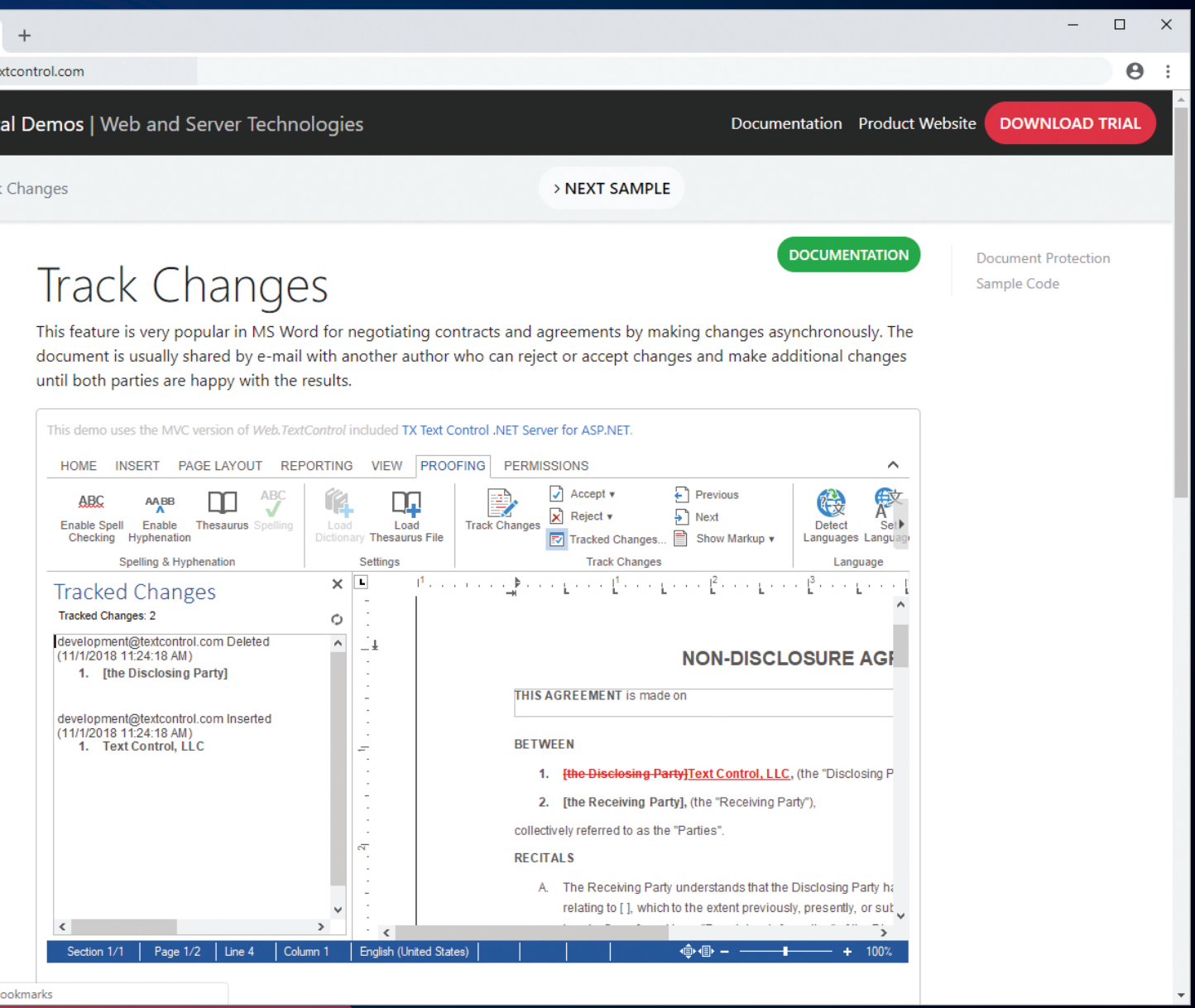


WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio
Partner

TX Text Control X16 Released

Evaluate our technology and test the most sophisticated cross-browser, and true WYSIWYG, rich text editor. Merge MS Word compatible templates with JSON data and create pixel-perfect Adobe PDF documents on-the-fly. See what's possible today!



Prediction Having inferred the player skills, I can now make predictions for future matches. I'll follow the same three steps as in training, but this time I'll be interested in inferring the posterior over the player1Wins variable. The way I like to think about this is that in training the information flows upward in the factor graph—from the data at the bottom to the model parameters at the top. By contrast, in prediction you already have the model parameters (learned in training) and information flows downward.

In my dataset, both Alice and Donna have one win and no losses. However, intuitively it feels like in a match between the two, Alice has higher chances of winning because her win is of more significance—it's against Bob, who is more likely a stronger player than Charlie. Let's try to predict the outcome of the match between Alice and Donna (see **Figure 5**).

In this case, the prior distribution over skills is the posterior distribution inferred in training. The observed data is one game between player 0 (Alice) and player 3 (Donna), with an unknown outcome. To make the outcome unknown, I need to clear the previously observed value of player1Wins, because this is what I want a posterior over:

```
playerSkillsPrior.ObservedValue = inferredSkills;

gameCount.ObservedValue = 1;
players1.ObservedValue = new[] { 0 };
players2.ObservedValue = new[] { 3 };
player1Wins.ClearObservedValue();

var player0Vs3 = inferenceEngine.Infer<Bernoulli[]>(player1Wins).First();
```

It's worth mentioning that uncertainty is propagated through this model all the way to the outcome of the match. This means that the obtained posterior over the predicted result is not simply a Boolean variable, but a value indicating the probability of the first player to win the match. Such a distribution is called a Bernoulli.

The value of the inferred variable is Bernoulli(0.6127). This means that Alice has a greater than 60 percent chance of winning the match against Donna, which aligns with my intuition.

Evaluation This example demonstrates how to build a well-known probabilistic model—TrueSkill. In practice, coming up with the right model requires multiple iterations over its design. Different models are compared by carefully selecting a set of metrics that indicate the model performance on the given data.

Evaluation is a central topic in ML, far too broad to cover here. It's also not something specific to probabilistic programming. It's worth mentioning, though, that having a model allows you to compute a unique "metric"—the *model evidence*. This is the probability that the training data was produced by this particular model. It's great for comparing different models—no need for a test set!

Benefits of Probabilistic Programming

The approach shown in this article probably looks more difficult than what you've seen before. For instance, the Connect(); special issue of the magazine (msdn.com/magazine/mt848634) introduced you to ML.NET, which focuses less on model design and more on data transformation. And in many cases, that's the right path to follow—if the data seems to match an existing algorithm and you're comfortable with treating the model as a black box, start with something off the shelf. However, if you need a bespoke model, probabilistic programming may be the right choice for you.

There are other cases when you might want to use probabilistic programming. One of the main advantages of having a model you understand is your improved ability to explain the behavior of the system. When the model isn't a black box, you can examine its internals and look at the learned parameters. These would make sense to you because you designed the model. For instance, in the previous example you can look at the learned skills of the players. In a more advanced version of TrueSkill, called TrueSkill 2, many more aspects of the game are modeled, including how performance in one game mode is connected to that in another. Understanding this connection helps game designers realize how similar the different game modes are. Interpretability of an ML system is also crucial for debugging. When a black-box model doesn't produce the desired results, you may not even know where to start looking for the problem.

Another advantage of probabilistic programming is the capability to incorporate domain knowledge in the model. This is captured both in the structure of the model and the ability to set priors. By contrast, traditional approaches usually only look at data, without allowing the domain expert to inform the behavior of the system. This capability is required in certain domains, such as health care, where there's strong domain knowledge and data can be scarce.

An advantage of the Bayesian approach, and one that is very well supported in Infer.NET, is the ability to learn as new data arrives. This is called online inference, and it's particularly useful in systems that interact with user data. For example, TrueSkill needs to update the skills of the players after each match, and the knowledge extraction system needs to continuously learn from the Internet as it grows. This is all so easy, though—you just plug in the inferred posteriors as the new priors, and the system is ready to learn from new data!

Probabilistic programming also naturally applies to problems with certain data traits, such as heterogeneous data, scarce data, unlabeled data, data with missing parts and data collected with known biases.

What Next?

There are two ingredients to successfully building a probabilistic model. The first one, obviously, is to learn how to model. In this article I introduced the main concepts and techniques of the methodology, but if you want to learn more, I recommend a freely available online book, "Model-Based Machine Learning" (mbmlbook.com). It gives a gentle introduction to model-based ML in production and is targeted at developers (as opposed to scientists).

Once you know how to design models, you have to learn how to express them in code. The Infer.NET user guide is an excellent resource for this. There are also tutorials and examples that cover many scenarios, which can all be reached from our repository at github.com/dotnet/infer, where we also warmly welcome you to join and contribute. ■

Yordan Zaykov *is the principal research software engineering lead of the Probabilistic Inference Development team at Microsoft Research Cambridge. He focuses on applications based on the Infer.NET ML framework, including work on e-mail classification, recommendations, player ranking and matchmaking, health care, and knowledge mining.*

Thanks to the following Microsoft technical experts who reviewed this article: *John Guiver, James McCaffrey, Tom Minka, John Winn*

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2019

FLEXERA

InstallShield

Installation Made Easy, MSIX Ready

InstallShield delivers the ultimate installation experience for your customers. And it makes life easier for you.

Your choice of installers and packages. Simplify complex installers and bundle multiple installers into one using Suite projects.

Then there's the big bonus: it's all set for building MSIX packages.

No coding for standard projects. And smooth deployment to the Windows Store.

It's fast. It's easy. Tightly integrated with Microsoft Visual Studio.

To learn more, try an evaluation today.
https://info.flexerasoftware.com/IS-EVAL-InstallShield-Premier

To learn more, please visit our website → **www.flexera.com**

Leveraging the Beliefs-Desires-Intentions Agent Architecture

Arnaldo Pérez Castaño

In this article, I describe a Travel Assistant Agent (TAA) that uses a Beliefs-Desires-Intentions (BDI) agent architecture for its decision-making process.

The BDI agent architecture is based on Michael Bratman's philosophical theory (Bratman 1987) that explains reasoning through the following attitudes: beliefs, desires and intentions. Consequently, a BDI agent is an autonomous agent that makes use of these three concepts for its functioning. Let's examine these concepts one by one:

- **Beliefs** are the agent's model of the environment, basically what it believes to be true. It's not knowledge as some of its beliefs might be false. This component of the BDI architecture is usually represented as a dataset of facts like breeze(1, 2), danger(2, 3), safe(0, 0), safe(0, 1) and so on.
- **Desires** represent the ideal state of the environment for the agent. Like in the human mind, these represent things we would like to see accomplished in the future. A desire might be realistic or not, as it occurs with human thinking, and may or may not be achievable. Desires can be mutually inclusive or exclusive.
- **Intentions** represent a subset of desires that the agent has taken as goals to be accomplished soon. These intentions cannot go against its beliefs—for instance, the agent cannot have an intention that makes it go through a dead zone or otherwise prevents it from reaching the goal that the intention represents. Such an intention would be discarded.

This article discusses:

- Understanding the Beliefs-Desires-Intentions agaent architecture
- How the Beliefs-Desires-Intentions agent architecture can be used to automate decision making
- Building a Travel Assistant Agent in C#

Technologies discussed:

C#, BDI Architecture, Artificial Intelligence, Robot Process Automation

The BDI agent architecture is based on Michael Bratman's philosophical theory (Bratman 1987) that explains reasoning through the following attitudes: beliefs, desires and intentions.

Beliefs consist of a fact dataset that's updated during the agent's lifetime. Regarding desires and intentions, a pair of questions arises: How do you select desires to become intentions, and how do you select later intentions to become agent actions? To answer this, I must present the two components of practical reasoning in the BDI model of the agent:

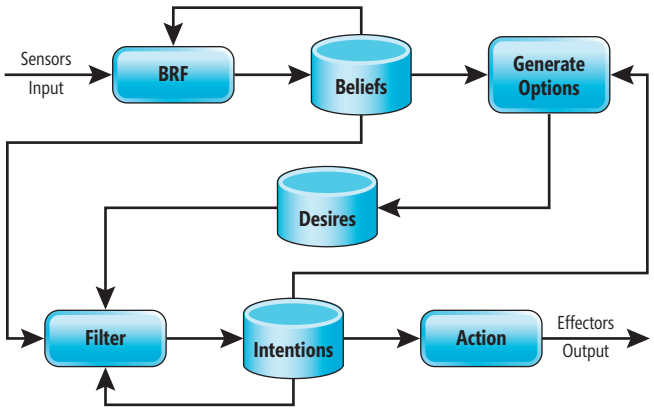


Figure 1 **The BDI Agent Architecture**

- **Deliberation:** This component deals with strategic thinking and decides what desires are to be accomplished now. The result is a set of intentions.
- **Means-Ends Reasoning:** This component deals with tactical planning and decides what actions should be performed to fulfill the set of committed intentions. The result is a set of plans and actions.

The diagram in **Figure 1** illustrates the workings of a BDI architecture. Inputs or percepts received through sensors from the environment are submitted to the Belief-Revision-Function (BRF). This function supervises updates to the current beliefs database, making sure that the recently received set of percepts doesn't contradict any beliefs in the belief dataset. The function accepts as arguments the set of inputs and the set of beliefs: Brf(inputs, beliefs).

> Beliefs consist of a fact dataset that's updated during the agent's lifetime.

Afterward, a set of options is generated that eventually becomes desires and enters the desire dataset. In order to generate these options, the function looks at the belief dataset and the intention dataset. As a result, its takes as parameters the belief dataset, the desire dataset and the intention dataset: generateOptions (beliefs, desires, intentions). Why do you need intentions to generate options? Because you don't want to generate options that contradict or go against the current set of intentions.

Figure 2 **Enum Types Used as T Parameter in BDI Class**

```
public enum DesireType
  {
    Visit, Budget, Date
  }

  public enum BeliefType
  {
    TourPackages
  }

  public enum IntentionType
  {
    BookTourPackage
  }
```

Using a filter function, the previously obtained desires are filtered and become intentions. To filter you usually exclude desires that aren't realistic or are very hard to fulfill at the moment. The filter function would have the signature: filter (beliefs, desires, intentions). Finally, from the set of intentions and a means-ends reasoning approach an action is taken using the agent's effectors (such as mechanical arms and screens).

To summarize, BDI is a paradigm and set of general principles, which can be used to design the architecture of a software system for decision making. You may be familiar with other design paradigms, such as three-tier architecture or Model-View-Controller for Web systems. Like these other paradigms, BDI is a general design aide rather than a well-defined prescriptive blueprint.

It's time to move beyond the theory of the BDI model and look at how to apply it to a Robotic Process Automation (RPA) application—specifically, a Travel Assistant Agent.

Travel Assistant Agent (TAA)

The goal is to create a TAA where tourists can input their holiday desires and obtain a holiday travel plan. For this example, the desires will be:

- Travel to Cuba
- Specifically, to the cities of Havana and Varadero
- On a maximum budget of $1,200 USD
- For a week
- Depart in December 2018

Using its beliefs database, the TAA transforms these desires into intentions and presents a realistic holiday plan, which might look something like this: Book the Cuba Beaches Holiday tour package with cubamaniatour.com on Dec. 15 for 8 days at a price of $800.

This plan can be obtained using a belief database that resembles the following:

- Tour Package, Essential Cuba Tour, 7 days, Available always, $800.
- Tour Package, Cuba Beaches Holidays Tour, 8 days, Available always, $800.
- Tour Package, Havana Explorer Cuba Tour, 3 days, Available 12-10-2018, $500.
- Tour Package, Havana Varadero Tour, 10 days, Available 17-10-2018, $800.

Note that any plan provided by TAA must take into consideration all the desires submitted by the customer, including the maximum budget desire ($1,200) that must be considered by the agent when providing the plan. This specific desire might not be realistic, preventing the agent using its BDI model to turn the desire into an intention. No plan that doesn't fit within the parameters of the price and other desires can be accomplished.

TAA Implementation in C#

Now that you have an idea of how the TAA works and how it will interact with tourists, let's explore how to code such an agent in C#. I'll start by creating a generic Attitude<TA, T> class that, being as general as possible, will comprehend all characteristics of

any attitude (belief, desire, intention) in a BDI agent architecture. This class is presented here:

```
public class Attitude<TA, T>
  {
    public T AttitudeRepresentation;
    public TA Label;

    public Attitude(TA label, T attitudeRepr)
    {
      Label = label;
      AttitudeRepresentation = attitudeRepr;
    }
  }
```

In this class I have two generic fields: TA that's used to identify the type of attitude in the BDI architecture and T that represents the data associated with it. The T generic parameter is substituted by one of the enum types listed in **Figure 2**.

These enums can be seen as the set of desires, beliefs and intentions of the agent at any given moment. Finally, the abstract Bdi<T> class is shown in **Figure 3**.

The Bdi<T> class relates to the BDI architecture and its philosophy but has no direct relation with TAA. Note that in the Bdi<T> class, I simply provide a map (abstract versions of Deliberate and MeansEndsReasoning methods) for creating BDI models.

Figure 3 **Bdi<T> Class**

```
public abstract class Bdi<T>
  {
    public IEnumerable<Attitude<BeliefType, T>> Beliefs { get; set; }
    public IEnumerable<Attitude<DesireType, T>> Desires { get; set; }
    public IEnumerable<Attitude<DesireType, T>> Intentions { get; set; }

    protected Bdi(IEnumerable<Attitude<BeliefType, T>> beliefs)
    {
      Beliefs = new List<Attitude<BeliefType, T>>(beliefs);
      Desires = new List<Attitude<DesireType, T>>();
      Intentions = new List<Attitude<DesireType, T>>();
    }

    protected abstract IEnumerable<Attitude<IntentionType, T>> Deliberate(
      IEnumerable<Attitude<DesireType, T>> desires);

    protected abstract T MeansEndsReasoning(IEnumerable<Attitude<IntentionType,
      T>> intentions);
}
```

Figure 4 **Taa<T> Class Constructor and Inherited Methods**

```
public class Taa<T> : Bdi<T> where T: Dictionary<string, string>
  {

    public Taa(IEnumerable<Attitude<BeliefType, T>> beliefs) : base(beliefs)
    {
    }

    public T GetPlan(IEnumerable<Attitude<DesireType, T>> desires)
    {
      return MeansEndsReasoning(Deliberate(desires));
    }

    protected override IEnumerable<Attitude<IntentionType, T>>
      Deliberate(IEnumerable<Attitude<DesireType, T>> desires)
    {
      return LookForTours(desires.ToList());
    }

    protected override T MeansEndsReasoning(IEnumerable<Attitude<IntentionType,
      T>> intentions)
    {
      return intentions.FirstOrDefault() == null ?  null :
        intentions.First().AttitudeRepresentation;
    }
  }
```

Therefore, the Bdi<T> class is used as a blueprint to implement the BDI agent architecture on any device, program and the like. Note that one method that was omitted in this class is UpdateBeliefs. This procedure would update the Beliefs dataset after a certain condition is met (such as deliberation or mean-ends reasoning) and matches the Brf function detailed earlier. It has been left to the reader and its specific conditions to create such an update method.

> Using different strategies, you could return the tour package or intention with the lowest price or longest duration.

The Taa<T> class that represents the Travel Assistant Agent inherits from Bdi<T> and its constructor, and the inherited methods implementation is illustrated in **Figure 4**.

In the Taa class, deliberation occurs in the Deliberate method, which ultimately calls the LookForTours method, as shown in **Figure 5**. It's in this method that desires inputted by tourists seek possible satisfaction by being put against beliefs in the Beliefs dataset. Recall that in the sample model, each belief represents a tour package. If such a package is found, then it's added to a result list of intentions, all of which consist of booking a certain tour package. Then, in the

Figure 5 **LookForTours Method**

```
private IEnumerable<Attitude<IntentionType, T>>
  LookForTours<T>(List<Attitude<DesireType, T>> desires) where T :
Dictionary<string, string>
  {
    var visitDesire = desires.First(d => d.Label == DesireType.Visit);
    var dateDesire = desires.First(d => d.Label == DesireType.Date);
    var maxBudgetDesire = desires.First(d => d.Label == DesireType.Budget);

    var citiesToVisit = visitDesire.AttitudeRepresentation["visiting"]
      .Split(',');
    var dateFrom = dateDesire.AttitudeRepresentation["from"];
    var days = int.Parse(dateDesire.AttitudeRepresentation["days"]);
    var maxBudget =
      double.Parse(maxBudgetDesire.AttitudeRepresentation["max"]);

    var tourPackages = Beliefs.Where(b => b.Label == BeliefType.TourPackages);
    var result = new List<Attitude<IntentionType, T>>();

    foreach (var tourPackage in tourPackages)
    {
      var data = tourPackage.AttitudeRepresentation as Dictionary<string, string>;
      var starts = data["starts"];
      var daysTour = int.Parse(data["days"]);
      var cities = data["cities"].Split(',');
      var price = double.Parse(data["price"]);

      if (daysTour <= days &&
        cities.Intersect(citiesToVisit).Count() == cities.Length &&
        starts == dateFrom &&
        price < maxBudget)
      {
        result.Add(new Attitude<IntentionType, T>(IntentionType.BookTourPackage,
          tourPackage.AttitudeRepresentation as T));
      }
    }

    return result;
  }
```

Figure 6 **Program Class**

```
class Program
  {
    static void Main(string[] args)
    {
      var beliefs = new List<Attitude<BeliefType, Dictionary<string, string>>>
      {
        new Attitude<BeliefType, Dictionary<string, string>>(
          BeliefType.TourPackages,
          new Dictionary<string, string> { { "tour", "Essential-Cuba" },
          { "starts" , "Anytime" }, { "days" , "7"}, { "cities", "HAV, VAR, TRI" },
          { "price" , "800"}, { "operator" , "www.cubamaniatour.com" }}),
        new Attitude<BeliefType, Dictionary<string, string>>(
          BeliefType.TourPackages,
          new Dictionary<string, string> { { "tour", "Cuba Beaches Holiday" },
          { "starts", "10-12-2018" }, { "days" , "8"}, { "cities", "HAV, VAR" },
          { "price" , "800"}, { "operator" , "www.cubamaniatour.com" }}),
        new Attitude<BeliefType, Dictionary<string, string>>(
          BeliefType.TourPackages,
          new Dictionary<string, string> { { "tour", "Havana & Varadero Tour" },
          { "starts", "12-15-2018" }, { "days" , "15"}, { "cities", "HAV,
            VAR, TRI" },
          { "price" , "800"}, { "operator" , "www.cubamaniatour.com" }}),
        new Attitude<BeliefType, Dictionary<string, string>>(
          BeliefType.TourPackages,
          new Dictionary<string, string> { { "tour", "Discover Cuba" },
          { "starts", "12-15-2018" }, { "days", "15"}, { "cities", "HAV,
            VAR, TRI" },
          { "price" , "800"}, { "operator" , "www.cubamaniatour.com" }}),
        new Attitude<BeliefType, Dictionary<string, string>>(
          BeliefType.TourPackages,
          new Dictionary<string, string> { { "tour", "Classic Car Tour" },
          { "starts", "12-15-2018" }, { "days", "10"}, { "cities", "HAV,
            VAR, TRI" },
          { "price" , "800"}, { "operator" , "www.cubamaniatour.com" }}),
        new Attitude<BeliefType, Dictionary<string, string>>(
          BeliefType.TourPackages,
          new Dictionary<string, string> { { "tour", "Havana Explorer" },
          { "starts", "12-15-2018" }, { "days", "3"}, { "cities", "HAV,
            VAR, TRI" },
          { "price" , "800"}, { "operator" , "www.cubamaniatour.com" }}),
        new Attitude<BeliefType, Dictionary<string, string>>(
          BeliefType.TourPackages,
          new Dictionary<string, string> { { "tour", "Trinidad Time" },
          { "starts", "12-15-2018" }, { "days", "7"}, { "cities", "HAV,
            VAR, TRI" },
          { "price" , "800"}, { "operator" , "www.cubamaniatour.com" }}),
      };

      var taa = new Taa<Dictionary<string, string>>(beliefs);

      var desires = new List<Attitude<DesireType, Dictionary<string, string>>>
      {
        new Attitude<DesireType, Dictionary<string, string>>(DesireType.Visit,
          new Dictionary<string, string> { {"visiting", "HAV, VAR" } }),
        new Attitude<DesireType, Dictionary<string, string>>(DesireType.Budget,
          new Dictionary<string, string> { {"max", "1000" } }),
        new Attitude<DesireType, Dictionary<string, string>>(DesireType.Date,
          new Dictionary<string, string> { {"from", "10-12-2018" },
            {"days" , "9"  }})
      };

      var tourPackage = taa.GetPlan(desires);

      Console.WriteLine(tourPackage == null ?
        "Sorry, no plan goes according to your details" : PrintPlan(tourPackage));
      Console.ReadLine();
    }
```

MeansEndsReasoning method, to have a simple logic I returned the first feasible tour package as the holiday intention for the tourist.

Using different strategies, you could return the tour package or intention with the lowest price or longest duration. In this case, to keep it simple, I outputted the first intention. The GetPlan method is supposed to return the action or set of actions to carry out tourist desires. In this case, the action (booking a tour package) coincides with the set of intentions and the result of the MeansEndsReasoning method.

To conclude, I tested my TAA in a Console Application as illustrated in **Figure 6**. There I created some dummy data as the set of beliefs, created a list of tourist desires, and then got a plan for these desires.

The PrintPlan method is included at the bottom of the Program. It prints the dictionary containing data belonging to the tour operator whose tour package has been selected by TAA as a booking option. The PrintPlan method looks like this:

```
private static string
PrintPlan(Dictionary<string, string>
toPrint)
  {
    var result = "";

    foreach (var keyValue in toPrint)
    {
      result += keyValue.Key + ", " +
keyValue.Value + '\n';
    }

    return result;
  }
}
```

The result obtained after executing the previous experiment code is shown in **Figure 7**.

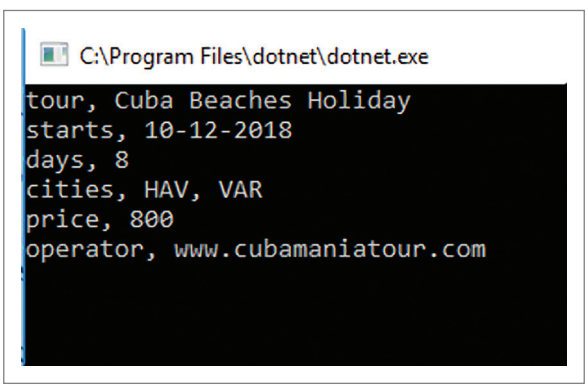


Figure 7 **Output of TAA for Tourist Desires**

In this article I introduced the Beliefs-Desires-Intentions (BDI) architecture. I described this agent architecture from a theoretical perspective, explained the idea behind the TAA and the way it would work, and presented an implementation in C# where the TAA was implemented using the BDI architecture. Finally, the TAA code was tested and I showed, through a simple Console Application, how it allows tourists to obtain options to book tour packages as holiday plans that matched their inputted desires (max budget, cities to visit, date, time and more). It's now up to you to extend the simple BDI model introduced here and adapt it to your own needs. ■

Arnaldo Pérez Castaño *is a computer scientist based in Belgrade, Serbia, where he works for P3 Digital Services, a subsidiary of P3 Group, a German multinational company with headquarters in Aachen. He's the author of "Practical Artificial Intelligence-Machine Learning, Bots and Agent Solutions Using C#" (Apress, 2018), "PrestaShop Recipes" (Apress 2017) and also a series of programming books—"JavaScript Fácil", "HTML y CSS Fácil" and "Python Fácil" (Marcombo S.A.)—and he writes for VisualStudioMagazine.com and Smashing Magazine. He's one of the co-founders of Cuba Mania Tour (cubamaniatour.com) and his expertise includes Visual Basic, C#, .NET Framework and Artificial Intelligence. Cinema and music are some of his passions. Contact him at arnaldo.skywalker@gmail.com.*

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

DevExpress DXperience 18.2 | from $1,439.99

DevExpress

A comprehensive suite of .NET controls and UI libraries for all major Microsoft dev platforms.

- WinForms – New Sunburst Chart, Office Navigation UX, SVG Office 2019 skins
- WPF – New Gantt control, improved Data Filtering UX and App Theme Designer
- ASP.NET & MVC – New Adaptive Layouts, improved Rich Text Editor and Spreadsheet
- Reporting – Vertical Band support, Free-hand drawing and improved Report wizards
- JavaScript – New HTML/Markdown WYSIWYG editor, Improved Grid and TreeList performance

LEADTOOLS Document Imaging SDKs V20 | from $2,995.00 SRP

LEADTOOLS
THE WORLD LEADER IN IMAGING SDKs

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

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 39, Code 128, QR, Data Matrix, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

PBRS (Power BI Reports Scheduler) | from $9,811.51

Christiansteven

Date & time Scheduling for Power BI reports with one Power BI License.

- Exports reports to PDF, Excel, Excel Data, Word, PowerPoint, CSV, JPG, HTML, PNG and ePub
- Send reports to email, printer, Slack, Google Sheets, folder, FTP, DropBox & SharePoint
- Uses database queries to automatically populate report filters, email addresses & body text
- Adds flexibility with custom calendars e.g. 4-4-5, holidays, "nth" day of the month, etc.
- Responds instantly by firing off reports when an event occurs e.g. database record is updated

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

- 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 responsive HTML, CHM, PDF, MS Word, ePUB, Kindle or print
- Styles and Templates give you full design control

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

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

Europe (UK)
ComponentSource
The White Building
33 King's Road
Reading, Berkshire
RG1 3AR
United Kingdom

Europe (Ireland)
ComponentSource
Unit 1E & 1F
Core B, Block 71
The Plaza, Park West
Dublin 12
Ireland

Asia Pacific
ComponentSource
7F Kojimachi Ichihara Bldg
1-1-8 Hirakawa-cho
Chiyoda-ku
Tokyo, 102-0093
Japan

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

GSA Schedule Contract GS-35F-0188R

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

MARCH 3-8, 2019 ➤ BALLY'S HOTEL AND CASINO

AN OASIS OF EDUCATION DAZZLING IN THE DESERT

Las Vegas

INTENSE DEVELOPER TRAINING CONFERENCE

In-depth Technical Content On:

- AI, Data and Machine Learning
- Cloud, Containers and Microservices
- Delivery and Deployment
- Developing New Experiences
- DevOps in the Spotlight
- Full Stack Web Development
- .NET Core and More

Register by January 11 & Save Up To $400!

Use Promo Code MSDN

Your Adventure STARTS HERE!

vslive.com/lasvegas

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360

AGENDA AT-A-GLANCE

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS — LAS VEGAS

DevOps in the Spotlight	Cloud, Containers and Microservices	AI, Data and Machine Learning	Developing New Experiences	Delivery and Deployment	.NET Core and More	Full Stack Web Development

START TIME	END TIME				
		Full Day Hands-On Labs: Sunday, March 3, 2019 *(Separate entry fee required)*			
8:00 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**HOL01** Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 1) - *Brock Allen*	**HOL02** Full Day Hands-On Lab: Xamarin and Azure: Build the Mobile Apps of Tomorrow - *Laurent Bugnion & Brandon Minnick*	**HOL03** Full Day Hands-On Lab: Develop an ASP.NET Core 2 and EF Core 2 App in a Day - *Philip Japikse*	
4:00 PM	6:00 PM	Conference Registration Open			
START TIME	**END TIME**	**Pre-Conference Workshops: Monday, March 4, 2019** *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	06:00 PM	**HOL01** Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 2) - *Brock Allen*	**M02** Workshop: ASP.NET Core with Azure DevOps - *Brian Randell*	**M03** Workshop: Cross-Platform C# Using .NET Core and WebAssembly - *Rockford Lhotka & Jason Bock*	
7:00 PM	09:00 PM	Dine-A-Round - Carlos & Charlies in the Flamingo, meet at conference registration desk at 6:45pm to walk over with the group.			
START TIME	**END TIME**	**Day 1: Tuesday, March 5, 2019**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**T01** Take Advantage of HTML5 Features - *Paul Sheriff*	**T02** Azure 101 - *Laurent Bugnion*	**T03** Visual Studio Productivity Tips and Tricks - *Allison Buchholtz-Au*	**T04** How Microsoft Does DevOps - *Tiago Pascoal*
9:30 AM	10:45 AM	**T05** Cool Features in CSS 3 - *Paul Sheriff*	**T06** Microservices with ACS (Managed Kubernetes) - *Vishwas Lele*	**T07** C# 7.x Like a Boss! - *Adam Tuliper*	**T08** To Be Announced
11:00 AM	12:00 PM	Keynote: To Be Announced			
12:00 PM	1:00 PM	Lunch - *Platinum*			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - *Grand Salon*			
1:30 PM	2:45 PM	**T09** Xamarin.Forms Takes You Places! - *Sam Basu*	**T10** Busy Developer's Guide to NoSQL - *Ted Neward*	**T11** Improving Performance in .NET Applications - *Jason Bock*	**T12** To Be Announced
3:00 PM	4:15 PM	**T13** Bringing History to Life with the HoloLens - *Adam Tuliper*	**T14** Busy .NET Developer's Guide to Python - *Ted Neward*	**T15** Concurrent Programming in .NET - *Jason Bock*	**T16** SQL Server Database DevOps - *Brian Randell*
4:15 PM	5:30 PM	Welcome Reception - Grand Salon			
START TIME	**END TIME**	**Day 2: Wednesday, March 6, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** Choosing a Front-End JavaScript Framework - *Ben Hoelting*	**W02** The Next Frontier - Conversational Bots - *Sam Basu*	**W03** Introduction to Windows Containers and Docker - *Marcel de Vries*	**W04** To Be Announced
9:30 AM	10:45 AM	**W05** JavaScript Patterns for the C# Developer - *Ben Hoelting*	**W06** Getting Started with Artificial Intelligence - *Jen Stirrup*	**W07** Agile Failures: Stories From The Trenches - *Philip Japikse*	**W08** DevOps on Azure with Containers, K8s, and Azure DevOps - *Brian Randell*
11:00 AM	12:00 PM	General Session: .NET Today and Tomorrow - *Jon Galloway, Senior Program Manager & Beth Massi, Product Marketing Manager, .NET Team, Microsoft*			
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - *Platinum*			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:10pm (Must be present to win) - *Grand Salon*			
1:30 PM	2:45 PM	**W09** Securing Web APIs from JavaScript/SPA Applications - *Brock Allen*	**W10** Taking Advantage of AI easily with Azure Cognitive Services - *Laurent Bugnion*	**W11** Go Serverless with Azure Functions - *Eric D. Boyd*	**W12** .NET Development Using Azure Dev Spaces - *Lisa Guthrie*
3:00 PM	4:15 PM	**W13** What's New in ASP.NET Core 2.2 to Secure Web Applications and APIs - *Brock Allen*	**W14** Applying ML to Software Development - *Vishwas Lele*	**W15** Secure Your App with Azure AD B2C - *Oren Novotny*	**W16** Bolting Security Into Your Development Process - *Tiago Pascoal*
4:30 PM	5:45 PM	**W17** Cross-Platform C# Using .NET Core and WebAssembly - *Rockford Lhotka*	**W18** Solving the 80% Problem. Data Preparation for Microsoft AI - *Euan Garden*	**W19** Monitor Your Applications and Infrastructure - *Eric D. Boyd*	**W20** Signing Your Code The Easy Way - *Oren Novotny*
7:00 PM	8:30 PM	Evening Out TBD			
START TIME	**END TIME**	**Day 3: Thursday, March 7, 2019**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** Science of Great UI - Part 1 (Efficiency in Thought & Motion) - *Mark Miller*	**TH02** Data Visualization Principles for Artificial Intelligence in Business - *Jen Stirrup*	**TH03** Building Resilient Microservices with .NET on Azure Service Fabric - *Mark Fussell*	**TH04** C# 7, Roslyn and You - *Jim Wooley*
9:30 AM	10:45 AM	**TH05** Science of Great UI - Part 2 (Design Like a Pro) - *Mark Miller*	**TH06** Understanding, Selecting, Visualizing and Finalizing Data Science Models for Data Professionals - *Jen Stirrup*	**TH07** Microservice Architecture in .NET - *Rockford Lhotka*	**TH08** (WPF + WinForms) * .NET Core = Modern Desktop - *Oren Novotny*
11:00 AM	12:15 PM	**TH09** Building Cross Device Experiences with Project Rome - *Tony Champion*	**TH10** Introduction to the CNTK Neural Network Library - *James McCaffrey*	**TH11** How to Observe and Talk to Users - *Billy Hollis*	**TH12** Entity Framework Core Performance Monitoring and Tuning - *Jim Wooley*
12:15 PM	1:15 PM	Lunch - Platinium			
1:15 PM	2:30 PM	**TH13** Building UWP Apps for Multiple Devices - *Tony Champion*	**TH14** Modern SQL Server Security Features for Developers - *Leonard Lobel*	**TH15** The Most Important Lessons I've Learned in Forty Years of Developing Software - *Billy Hollis*	**TH16** Architecting Systems for DevOps and Continuous Delivery - *Marcel de Vries*
2:45 PM	4:00 PM	**TH17** To Be Announced	**TH18** Introduction to Cosmos DB - *Leonard Lobel*	**TH19** To Be Announced	**TH20** Security in Your Pipelines. The shift to Rugged DevOps - *René van Osnabrugge*
START TIME	**END TIME**	**Post-Conference Workshops: Friday, March 8, 2019** *(Separate entry fee required)*			
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries			
8:00 AM	5:00 PM	**F01** Workshop: Building, Running, & Continuously Deploying Microservices with Docker Containers on Azure - *Marcel de Vries and René van Osnabrugge*		**F02** Workshop: Developer Dive into SQL Server - *Leonard Lobel*	

Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

Introducing Azure SQL Database Hyperscale

Kevin Farlee

At the Ignite conference, we announced the public preview of Azure SQL Database Hyperscale, a new storage architecture providing a SQL-based and highly scalable service tier for databases that adapts on-demand to your workload's needs. With Azure SQL Database Hyperscale, databases can quickly auto-scale up to 100TB, eliminating the need to pre-provision storage resources, and significantly expand the potential for app growth without being limited by storage size.

Compared to current Azure SQL Database service tiers, Azure SQL Database Hyperscale provides the following additional capabilities:

- Support for 100TB+ database size
- Rapid scale up/down and point-in-time restore, regardless of the database size
- Fewer size-of-data operations
- Higher log throughput than current service tiers
- Scale-out read-only workload with read-scale replicas without data copy

Azure SQL Database Hyperscale is currently in preview. All information is subject to change.

This article discusses:

- Overview of the Azure SQL Hyperscale technology
- The primary components of the Azure SQL Database Hyperscale model
- High availability and disaster recovery features
- Creating a Hyperscale database

Technologies discussed:

Azure SQL Database Hyperscale, SQL Server, PowerShell

New Architecture

Azure SQL Database Hyperscale is a fundamental rearchitecting of the storage engine within the SQL database that was accomplished without making changes in the query engine that processes queries and determines behaviors. So, we've added significant new capabilities without introducing any compatibility challenges.

> The compute nodes look like a traditional SQL Server, but without local data files or log files.

The Azure SQL Database Hyperscale architecture breaks the monolithic SQL engine into several microservices designed for the cloud environment; these services decouple compute, log and storage.

As shown in **Figure 1**, the Azure SQL Database Hyperscale model comprises three primary components:

- **Compute** is the query engine from SQL Server, the portion of the database engine that executes the query logic, and

determines compatibility with other SQL implementations, as well as behaviors when the query is evaluated.

- **Page Servers** is a new scale-out architecture for storing the data that takes the traditional Storage Engine component and splits it into a scale-out set of services, each one managing a defined set of data pages (nominally totaling 1TB of data pages).
- **Log Service** is a new logging architecture that manages the data flow for log data, ensuring that logged updates to the data get propagated to all of the replicas of a changed page, and persisted durably for future needs.

Compute Nodes

The compute nodes look like a traditional SQL Server, but without local data files or log files. Compute nodes are the "server" that applications and users interact with, whether updating the data (via the Primary compute node) or doing strictly read-only transactions via one of the readable secondary compute nodes.

The primary compute node writes transaction log records to the Landing Zone of the log service, and fetches data pages from page servers if they're not found in the local data cache or Resilient Buffer Pool Extension (RBPEX). This is where all of the query execution logic resides, and that logic is unchanged from other SQL deployment modes. By preserving this top layer of the SQL Engine largely unchanged, you can maintain complete compatibility with other deployment modes of SQL while delivering the revolutionary new capabilities the separate storage components offer. Because we've separated the compute nodes containing the query engine from the storage, they're effectively stateless, meaning you can lose a compute node without endangering any data at all. It also means that you can scale the compute resources up or down without moving any data, which provides unparalleled agility for scaling resources up or down at will.

I'll discuss more high availability (HA) features later.

Log Service

The log service externalizes the transactional log of a Hyperscale database. The primary compute instance writes the log records directly to Azure Premium Storage managed by the log service (The "Landing Zone" in **Figure 1**). The log service retrieves log records from the landing zone and makes it available to page servers and secondary compute. The log service also offloads log records to cheaper long-term storage to support point-in-time restore.

Because the landing zone is Azure Premium Storage, which has resiliency and HA built into it, log records are persisted and safe as soon as they're written to the landing zone. With the comparable AlwaysOn Availability Groups HA architecture for on-premises SQL Server, a transaction commit needs to be sent to a quorum of secondary replicas via network protocol, and an acknowledgement received back from that quorum of replicas before the transaction can be deemed to be complete, and the success returned to the calling application. This can take a significantly longer period of time than with the Hyperscale architecture. You can have full HA without waiting for secondary nodes to acknowledge the receipt of the log records. This means that even with full HA, you have end-to-end log latencies in the 2ms range. When the Azure Ultra SSD storage technology becomes available, this latency will shrink from 2.5ms to around 0.4ms for remote, fully durable and resilient data storage.

Finally, log records are persisted in Azure Standard storage for long-term storage, and the space in the landing zone is reclaimed. The log records in the Azure storage are kept as long as the backup retention period configured for the database, so there's no need to do transaction log backups. You just keep the log data online, which gives you in effect an infinite transaction log (within the bounds of the user-configured backup retention period).

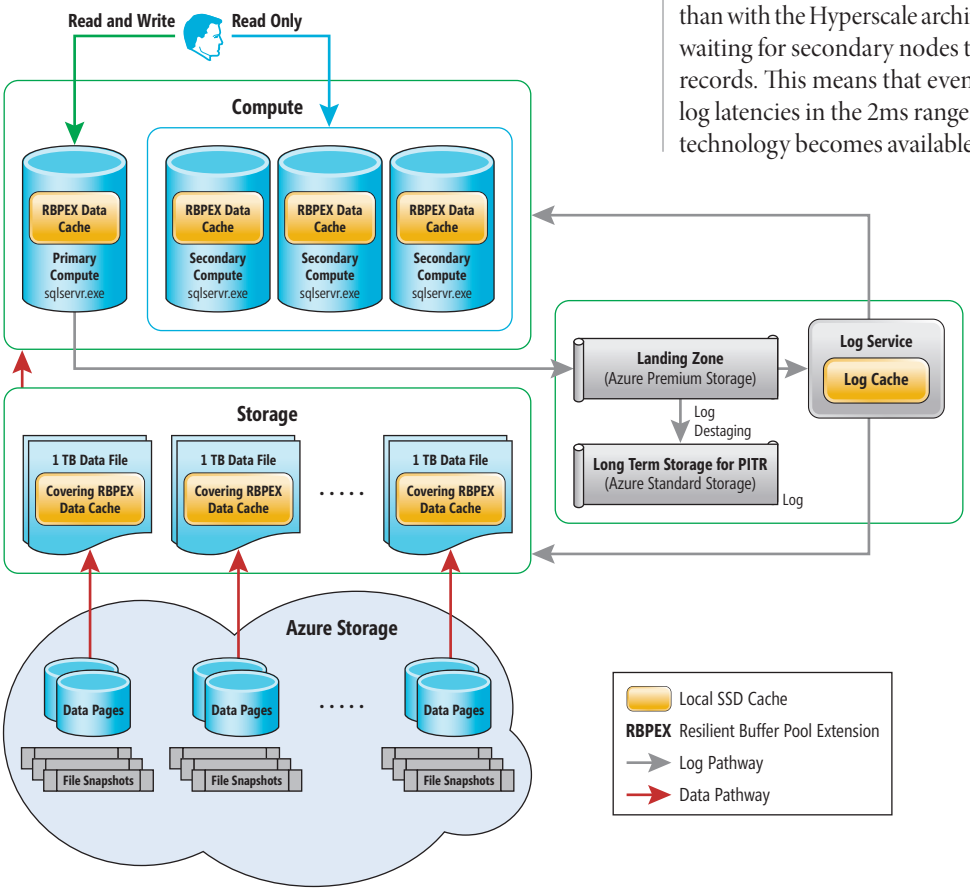


Figure 1 **The New Azure SQL Database Hyperscale Architecture**

Page Servers

The page servers host and maintain the data files. They consume the log stream from the log service and apply the data modifications described in the log stream to data files. Read requests of data pages

that aren't found in the compute's local data cache or RBPEX are forwarded to the page servers that own the relevant pages. In page servers, the data files are persisted in Azure Storage and are heavily cached through their own RBPEX SSD-based caches.

Each page server manages pages representing approximately 1TB of data, so the page servers are scaled horizontally in units of approximately 1TB. The Compute Nodes/query engine models each page server as a 1TB data file, so from the page ID (File ID: Page within file), you know exactly which page server hosts the page being requested.

> Each page server manages pages representing approximately 1TB of data, so the page servers are scaled horizontally in units of approximately 1TB.

The data for each page server is ultimately persisted in Azure Standard storage, which provides full resilience and availability. Thus, you can survive the loss of an entire page server without risking any data loss, as the data is fully available from the Azure storage, and you can completely rebuild the page server from that data if needed.

The data for each page server is fully cached in its own local SSD RBPEX cache, so that in operation, no data request is ever forwarded to the Azure storage backing the page server, because it can always be satisfied out of the local RBPEX cache.

Multiple page servers will be created for a large database. When the database is growing and available space in existing page servers is lower than a threshold, a new page server is automatically added to the database. Because page servers are working independently, you can grow the database with no local resource constraints.

High Availability and Disaster Recovery (DR)

Automated Backup and Point-in-Time Restore In a Hyperscale database, snapshots of the data files are taken from the page servers, leveraging Azure blob storage capabilities periodically to replace the traditional streaming backup. This allows you to back up a very large database in just a few seconds with no impact to the running workload. Together with the log records stored in the log service, you can restore the database to any point in time during retention—which is seven days in public preview and will be configurable up to 35 days at general availability (GA)—in a very short time, regardless of database size. Each page server's data is snapshotted independently, with no requirement to synchronize the snapshots, so that potential source of delay is eliminated, as well.

Highly Available Components Each of the components in the Azure SQL Database Hyperscale architecture is designed to be highly available so that there will be no interruptions in database access:

- Compute nodes each have at least one replica running and hot at all times. In the event of a compute node failure or failover, the replica would immediately take over the Primary role, keeping the database online and available. A replacement replica can be started up very quickly, and can warm up its caches as a background task without impacting production performance. Other replicas may be configured for read scale-out purposes. With this architecture, the compute nodes are effectively stateless.
- Page servers each have a standby replica online with their RBPEX cache fully populated, so they're available to take over for the active page server in the event of failure. Again, because they're stateless outside of cached data, a replacement can be online very quickly, without any risk of data loss.
- The log service doesn't have a hot standby, but this is unnecessary as the log service has no cached data and can be replaced as quickly as you could failover to a standby replica. The data that's managed by the log service resides first in the landing zone, which is resilient Azure Premium Storage (soon Ultra SSD storage), and is ultimately persisted in Azure Standard storage for long-term retention.

So, as you can see, there's no component that represents a single point of failure. All Hyperscale

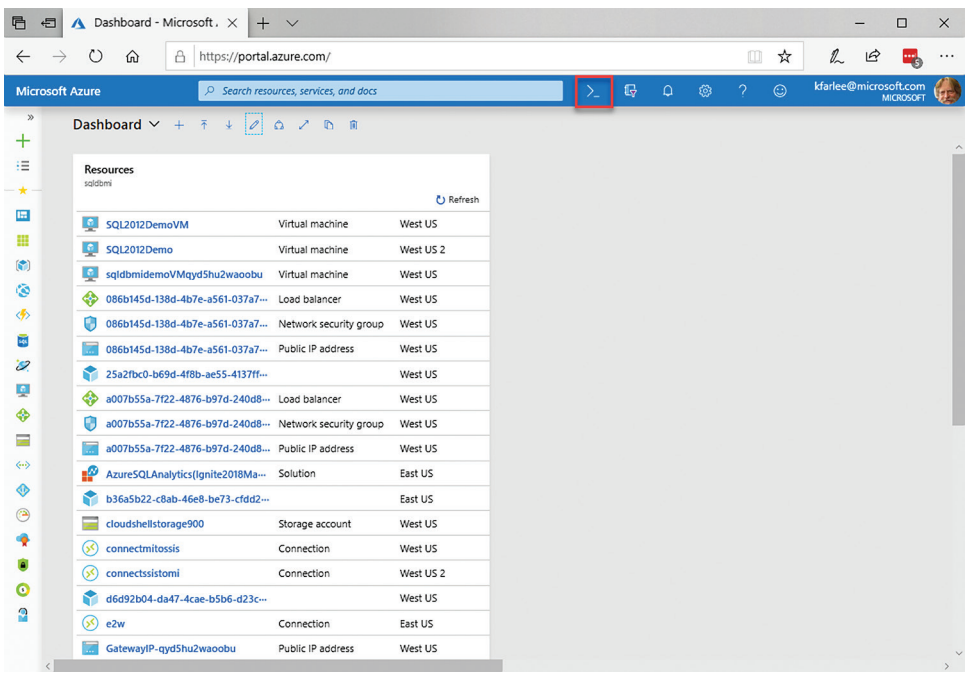


Figure 2 **Launching Cloud Shell from the Azure Portal**

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2019

VisualSVN Server:

Universal versioned storage

for CODE

and EVERYTHING ELSE

- High-performance terabyte scale replication for distributed teams
- Secure Single Sign-On with Active Directory
- Beautiful web interface with preview for PDF and Word* files
- Incredibly easy to install, configure and maintain
- Completely automatable with PowerShell
- Fast and reliable. Certified for Windows Server. More than **3,000,000 downloads**

To learn more, please visit our website → **www.visualsvn.com**

* Coming soon

components have active standbys, with the exception of the log service, and all data is backed by fully resilient Azure storage.

Creating Your Own Hyperscale Database

You can create a new Hyperscale Database in the same ways you create any other Azure SQL Database, using the Microsoft Azure Portal, PowerShell or Azure Resource Manager (ARM).

Here, I'll show the method to create a Hyperscale Database using PowerShell. Note that this is no different from the standard script for creating any Azure SQL Database, with the exception of the *-RequestedServiceObjectiveName* parameter. In the example that follows, I'll use the HS_Gen5_8, which is a Hyperscale SQL Database, on Generation 5 hardware, with eight cores. Of course, if you already have a resource group and logical server, you can skip those portions of the script.

The first thing you'll need to do is to launch Azure Cloud Shell. Cloud Shell is a free, interactive shell that you can use to run the steps mentioned in this article. It has common Azure tools pre-installed and configured to use with your account. You can just click the Copy button to copy the code, then paste the code into the Cloud Shell and press enter to run it.

There are a number of ways to launch Cloud Shell. Among the easiest are to either open Cloud Shell in your browser by going to shell.azure.com/powershell and clicking the Launch Cloud Shell button, or by clicking the Cloud Shell button on the menu in the upper right of the Azure Portal. I recommend you use the second method. Log in to the Azure Portal at portal.azure.com using the account that has access to your Azure subscription. Then click on the link as shown in **Figure 2**.

You can then paste the script shown in **Figure 3** into the Cloud Shell window and execute it to create a resource group, a SQL Logical Server and a database. Note that the resource group name is set to "myResourceGroup-" plus a random number, for example "MyResourceGroup-2088327474." Other objects are named similarly. However, you can customize these names if it's more convenient.

In the script, the server is created in the WestUS2 region. At this time, Hyperscale is not available in all Azure regions, so you can either keep it in WestUS2, which is active, or try another region. If you get an error that the "Specified edition is not available. Please choose a different edition," this indicates that the region you've chosen has not yet enabled Hyperscale. You can either try another region, or use WestUS2.

Another object that's configured for you is the firewall rule that grants access to your server. The example script sets both startip and endip to 0.0.0.0, which doesn't allow any access. You can add the address for your specific computer by finding the new server in the Portal, clicking on "show firewall settings" and then clicking on "Add client IP," which will add a firewall rule allowing your current machine to connect to the database.

Once your database and server are created, and the firewall rules are adjusted, you can connect to your database using SQL Server Management Studio (SSMS). The script prints out the name it assigned to the SQL Server. Simply append ".database.windows.net" to the server name when you connect. The user name and password are contained in this script (of course, you can and should change them to something only you know). At that point, you're connected to your new Hyperscale database.

Figure 3 **Creating a Resource Group, SQL Logical Server and Database**

```
# Set the resource group name and location for your server
$resourcegroupname = "myResourceGroup-$(Get-Random)"
$location = "westus2"
# Set an admin login and password for your server
$adminlogin = "ServerAdmin"
$password = "ChangeYourAdminPassword1"
# Set server name - the logical server name has to be unique in the system
$servername = "server-$(Get-Random)"
# The sample database name
$databasename = "mySampleDatabase"
# The IP address range that you want to allow to access your server
$startip = "0.0.0.0"
$endip = "0.0.0.0"

# Create a resource group
$resourcegroup = New-AzureRmResourceGroup -Name $resourcegroupname
-Location $location

# Create a server with a system-wide unique server name
$server = New-AzureRmSqlServer -ResourceGroupName $resourcegroupname `
  -ServerName $servername `
  -Location $location `
  -SqlAdministratorCredentials $(New-Object -TypeName
    System.Management.Automation.PSCredential -ArgumentList $adminlogin,
    $(ConvertTo-SecureString -String $password -AsPlainText -Force))

# Create a server firewall rule that allows access from the specified IP range
$serverfirewallrule = New-AzureRmSqlServerFirewallRule
  -ResourceGroupName $resourcegroupname `
    -ServerName $servername `
    -FirewallRuleName "AllowedIPs" -StartIpAddress $startip -EndIpAddress $endip

# Create a blank database with the Hyperscale, Gen5, two-core performance level
$database = New-AzureRmSqlDatabase  -ResourceGroupName $resourcegroupname `
  -ServerName $servername `
  -DatabaseName $databasename `
  -RequestedServiceObjectiveName "HS_Gen5_2" `
  -SampleName "AdventureWorksLT"

Echo $servername

# Clean up deployment
# Remove-AzureRmResourceGroup -ResourceGroupName $resourcegroupname
```

Other methods for creating Azure Hyperscale SQL Databases can be found at bit.ly/2QoTLbj.

The Next-Generation Database Architecture

With this brief tour, you've seen that Azure SQL Database Hyperscale is a revolutionary new architecture that has the unique benefit of providing full compatibility with previous generations of SQL engines. Azure SQL Database Hyperscale is truly cloud-born, and has significant advantages in scale, performance and manageability. By eliminating common size-of-data operations, it's able to deliver Very Large Database (VLDB) capabilities without the typical VLDB challenges. For more details, please refer to the "Hyperscale Service Tier (Preview) for up to 100TB" page at bit.ly/2TTJkeK. ■

Kevin Farlee *has more than 30 years in the industry, in both database and storage management software. In his current role as a princial program manager on the Microsoft SQL Database team, he's engaged in developing the Hyperscale VLDB features of Azure SQL DB.*

Thanks to the following Microsoft technical experts for reviewing this article: *Alain Dormehl, Xiaochen Wu*

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2019

DevExpress®

Visual Studio MAGAZINE 1 READER'S 2019 CHOICE

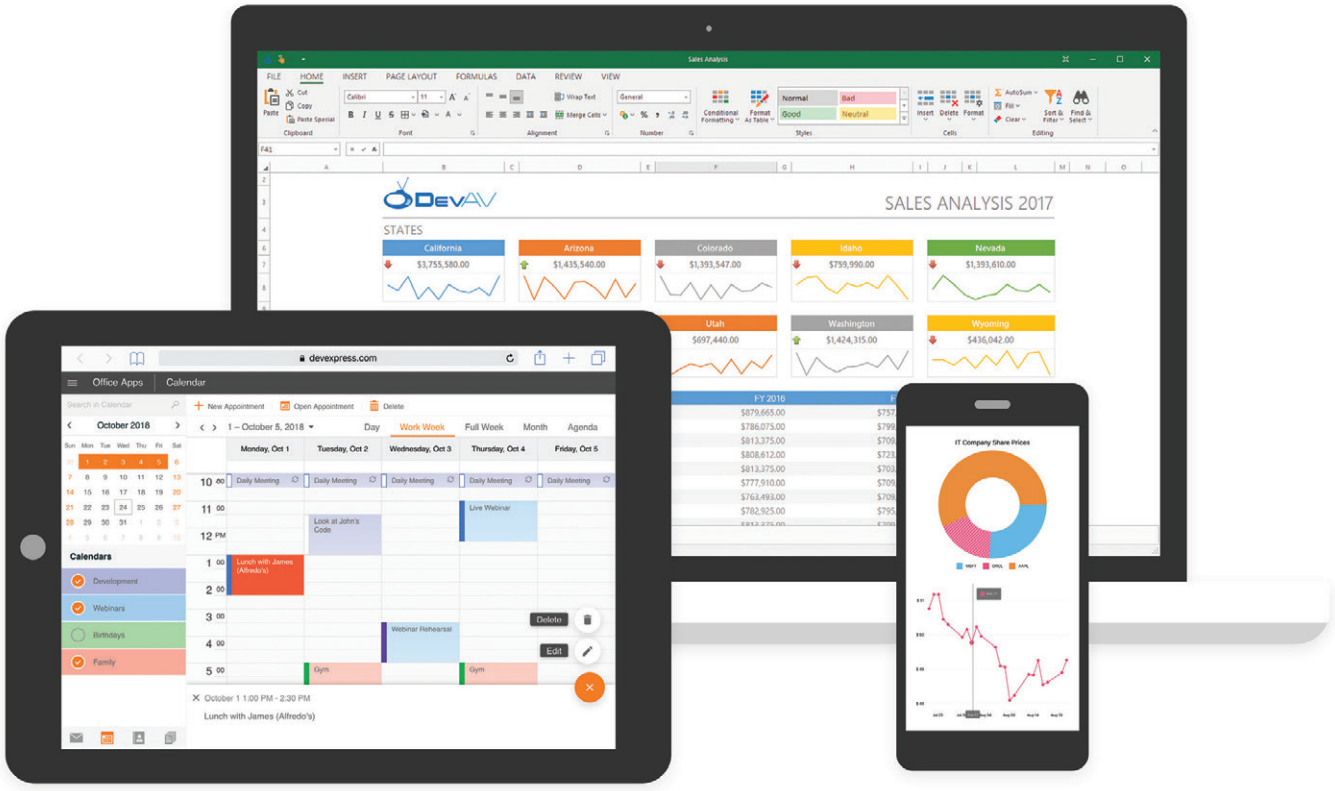
Thank You x 20

We are grateful to everyone who voted for our products in this year's VSM Readers' Choice Awards. Thank you for your continued support and for the confidence you've placed in DevExpress over the last 20 years.

Since 1998, our goal has been steadfast and unshakeable: Create best-of-breed software development tools so you can deliver high-impact business solutions that amaze. From the desktop and web to your mobile world, we remain fully committed to your needs and will do everything possible to earn your trust in the years to come.

Thank you once again from all of us at DevExpress.
management@devexpress.com

WIN ASP MVC WPF UWP JS



To learn more, please visit our website → www.devexpress.com

DINO ESPOSITO

Template-Based Components in Blazor

It's been nearly a year since the first public build of Blazor was released in early 2018. Designed to be a client-side Web framework capable of running C# and .NET code from within the host browser, the platform has evolved in a number of directions.

Blazor remains a plain client-side framework processing data downloaded from any reachable back-end services, but it also had its underpinnings modified to run entirely from the server through a SignalR connection. The recent introduction of the SignalR Azure service just confirms, in my opinion, Microsoft's intention of pushing Blazor more and more as a modern development platform. A scalable cloud service between the server application and myriad clients guarantees that .NET Core code can be effectively hosted on the server and interactively run on the client through the intermediation of C# instead of JavaScript.

As a client-side Web framework, Blazor (part of which will ship with the upcoming ASP.NET Core 3.0 release) can't do its job well without components. In version 0.6.0 of the framework, the Blazor team introduced a particular flavor of components: template-based. In this article, I'll explore how they work by updating the type-ahead example featured in my past two columns to be a template-based component.

Adding Templates to Blazor

Simple components can be configured through properties, but realistic components often need more rendering flexibility, and templates are a canonical way to achieve that. For example, think of a data grid component. With Razor, or any other data-binding infrastructure, you can easily build a data table linked to a known data source. Here's a quick example of how to build an HTML table out of a collection of data items:

```
<table>
@foreach(var item in Items)
{
  <tr>
    <td>@item.FirstName</td>
    <td>@item.LastName</td>
  </tr>
}
</table>
```

It's quick and easy, but there's not much there that can be reused. Now imagine you make the grid structure richer by adding a header, a footer and perhaps a search bar on top and a pager bar at the bottom. Graphical layout and code behind both the search bar and the pager bar won't change with the actual data being searched and paged. Yet you're going to rewrite search and pager bar each and every time you use a data grid to display a different type of data.

With this minimal level of abstraction, a grid of countries and a grid of customers are completely different entities, although the core code behind the various internal parts is nearly the same. Template-based components address this specific scenario and show a way to have a single DataGrid component capable of presenting, searching, and paging both countries and customers with a single codebase.

Figure 1 **The DataSource Template Component**

```
@typeparam TItem
@inject HttpClient HttpExecutor

<div style="border: solid 4px #111;">
  <div class="table-responsive">
    <table class="table table-hover">
      <thead>
        <tr>
          @if (HeaderTemplate != null)
          {
            @HeaderTemplate
          }
        </tr>
      </thead>

      <tbody>
        @foreach (var item in Items)
        {
          <tr>
           @RowTemplate(item)
          </tr>
        }
      </tbody>

      <tfoot>
        <tr>
          @if (FooterTemplate != null)
          {
            @FooterTemplate(Items)
          }
        </tr>
      </tfoot>
    </table>
  </div>
</div>


@functions {

  [Parameter]
  RenderFragment HeaderTemplate { get; set; }

  [Parameter]
  RenderFragment<TItem> RowTemplate { get; set; }

  [Parameter]
  RenderFragment<IList<TItem>> FooterTemplate { get; set; }

  [Parameter]
  IList<TItem> Items { get; set; }

}
```

Figure 2 **Declaring a DataSource Template Component**

```
<DataSource Items="@Countries" TItem="Country">
  <HeaderTemplate>
    <th>Name</th>
    <th>Capital</th>
  </HeaderTemplate>
  <RowTemplate>
    <td>@context.CountryName</td>
    <td>@context.Capital</td>
  </RowTemplate>
  <FooterTemplate>
    <td colspan="2">
      @context.Count countries found.
    </td>
  </FooterTemplate>
</DataSource>
```

Let's say you want to have a rich grid component in your Blazor front-end views. **Figure 1** shows the source code of a brand new template-based DataSource Blazor component.

As you can see, the DataSource component is built around the skeleton of an HTML table in which the body is constructed by iterating table rows on top of data records in the bound collection. The header and footer are defined as table rows, with the details of the actual content left to client pages. Client pages can interact and further customize the component through the public interface defined via the @functions section. There you've defined three templates—HeaderTemplate, FooterTemplate and RowTemplate—and a property, called Items, that act as the actual data source and data provider for the component.

You can have template-based components bound to a fixed data type, or you can have components bound to a generic data type that's specified declaratively. The same grid component can realistically be used to present, search and page different collections of data. To endow the component with this capability, in Blazor you use the @typeparam directive, like so:

```
@typeparam TItem
```

Any reference to the TItem moniker found in the Razor source code is treated as a reference to the dynamically determined type of a C# generic class. You specify the actual type being used within the component through the TItem property. **Figure 2** shows how a client page would declare a DataSource template component.

The name of the generic type property—TItem in the previous code snippet—will match the name of the type parameter as declared through the @typeparam directive within the source code of the component. Note that frequently the generic type parameter is just inferred by the framework and may not be specified.

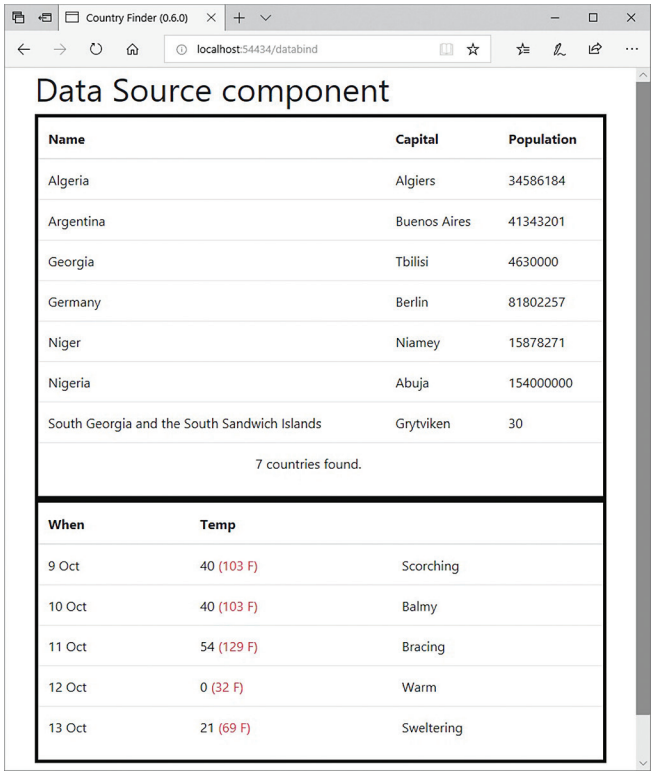


Figure 3 **DataSource Generic Component**

Programming Aspects of Templates

A Blazor template is an instance of the RenderFragment type. Put another way, it's a chunk of markup being rendered by the Razor view engine that can be treated like a plain instance of a .NET type. Most templates are parameter-less, but you can make them generic, too. A generic template will receive an instance of the specified type as an argument and can use that content to render its output. In the sample in **Figure 2**, the header template is parameter-less, but the row and the footer templates are generic.

> Simple components can be configured through properties, but realistic components often need more rendering flexibility, and templates are a canonical way to achieve that.

In particular, the RowTemplate property takes an instance of TItem whereas the FooterTemplate property receives a collection of TItem instances. If needed, you can also define a template to receive an instance of a fixed type. For example, the FooterTemplate could instead be passed only an integer denoting the number of items being rendered in the page. This is shown in the code here:

```
RenderFragment<TItem> RowTemplate { get; set; }
RenderFragment<IList<TItem>> FooterTemplate { get; set; }
```

A template can be made optional to implement in a client page, by wrapping its call with a plain check for existence before use. Here's how a Blazor component can make one of its template properties optional:

```
@if (FooterTemplate != null)
{
  @FooterTemplate(Items)
}
```

When rendering a parametric template, you use the "context" implicit name to reference the argument of the template. For example, when rendering a table row through the RowTemplate property, you use the context parameter to refer to the item being rendered, like so:

```
<RowTemplate>
  <td>@context.CountryName</td>
  <td>@context.Capital</td>
</RowTemplate>
```

Note that the name of the context argument can be declaratively changed through the Context property of the template, as shown here:

```
<RowTemplate Context="dataItem">
  <td>@dataItem.CountryName</td>
  <td>@dataItem.Capital</td>
</RowTemplate>
```

As a result, the DataSource generic component can be used in the same Blazor view to populate data grids of different data types, as depicted in **Figure 3**. The structure of the code you write is summarized like this:

```
<DataSource Items="@Countries" TItem="Country">
  ...
</DataSource>
<DataSource Items="@Forecasts" TItem="WeatherForecast">
  ...
</DataSource>
```

Any surrounding markup and code you may have around the displayed grid (for example, pager bar, search bar, sorting buttons) are fully reused. As a personal note, this deeper level of markup customization reminds me of the old days of ASP.NET Web Forms, where custom server controls, via templates and custom properties, defined their own domain-specific language. This made the process of outlining the desired UI quite smooth and fluent. The advent of MVC and the subsequent shift to plain, client-side Web development brought us closer to the HTML machinery and away from abstraction. Framework components are just trying to recover that level of expressivity.

Rewriting the TypeAhead Blazor Component

Last month (msdn.com/magazine/mt830376) I presented a type-ahead component entirely written in Blazor providing the same functionality as the popular—and JavaScript-based—Twitter TypeAhead plug-in. In that implementation, the server endpoint in charge of returning hints was actually forced to return a compact and general-purpose data transfer object with three properties: the unique ID of the object identified in the query, a display text, and a third property rendering of the markup to show in the dropdown box for each hint.

In last month's demo, I used the type-ahead component to find a country name. The server endpoint returned an object made of the ISO country code, the country name, and an HTML snippet with country name, capital, and continent. The HTML snippet, however, was under the full control of the server implementation and there was no chance for the author of the client page using the type-ahead component to control the layout of the HTML snippet. That's just the perfect scenario to see template-based components in action outside the realm of a basic demo as I've done so far.

Figure 4 presents the HTML layout of the new type-ahead component. It's the same code from last month, with a notable exception: the format of the data returned by the hint provider. In the original implementation, the hint provider (a sample controller) returned a data transfer object and therefore was the single point of control of the actual data being selected by the user. Having the hint provider return a list of countries instead of a list of tailor-made type-ahead items enables the Blazor component to expose an ItemTemplate property for callers to decide the layout of each dropdown menu item. Receiving a list of countries enables the component to raise its OnSelection event, passing on directly the instance of the selected data item to any interested listeners.

Figure 4 **A Template-Based TypeAhead Component**

```
<div>
  <div class="input-group">
    <input type="text" class="@Class"
           oninput="this.blur(); this.focus();"
           bind="@SelectedText"
           onblur="@(ev => TryAutoComplete(ev))" />

    <div class="input-group-append">
      <button class="btn dropdown-toggle"
              type="button"
              data-toggle="dropdown"
              style="display: none;">
        <i class="fa fa-chevron-down"></i>
      </button>
      <div class="dropdown-menu @(_isOpen ? "show" : "")"
           style="width: 100%;">
        <h6 class="dropdown-header">
          @Items.Count item(s)
        </h6>
        @foreach (var item in Items)
        {
          <a class="dropdown-item"
           onclick="@(() => TrySelect(item))">
            @ItemTemplate(item)
          </a>
        }
      </div>
    </div>
  </div>
</div>
```

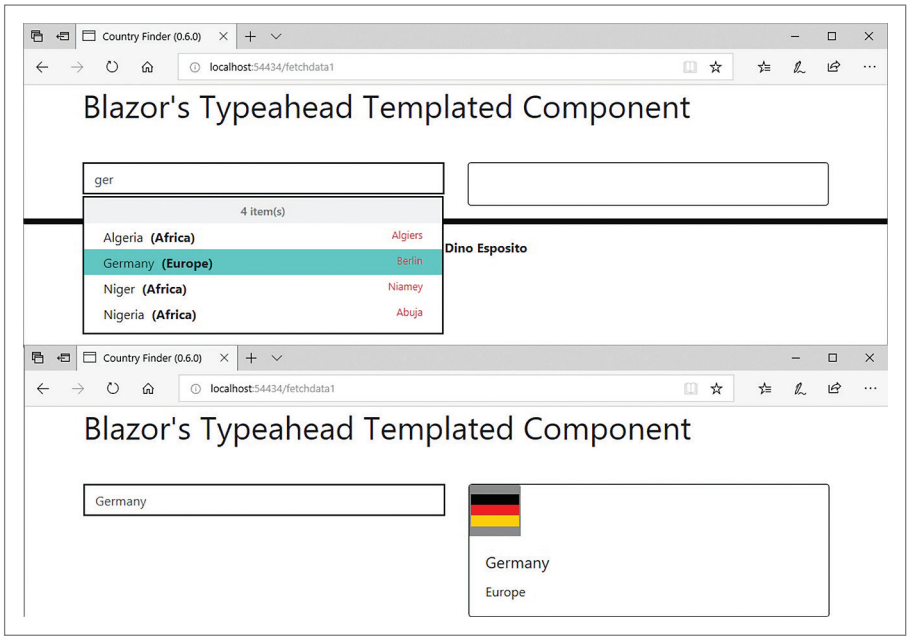


Figure 5 **The Country Dropdown List**

To be precise, a hidden field to collect via code the unique ID of the selected data item may still be necessary to ensure that once used within an HTML form the type-ahead component can successfully post its content through the normal channel of the browser. However, now the hidden field can be put more naturally outside the boundaries of the type-ahead component, and be under the full control of the client developer.

The type-ahead component defines a template property named ItemTemplate, which is defined as follows:

```
[Parameter]
RenderFragment<TItem> ItemTemplate { get; set; }
```

The TItem parameter is defined by the caller. In summary, here's the markup that sets up a type-ahead component with item templates:

```
<Typeahead TItem="Country"
           url="/hint/countries"
           name="country"
           onSelectionMade="@ShowSelection">
  <ItemTemplate>
    <span>@context.CountryName</span> 
    <b>(@context.ContinentName)</b>
    <small class="pull-right">@context.Capital</small>
  </ItemTemplate>
</Typeahead>
```

The TItem parameter tells the component it will be handling objects of type Country and receiving hints from the specified URL. Whenever hints based on the entered text are received, they're rendered in a dynamic dropdown list using the markup in the ItemTemplate section. By design, the item template receives an instance of the current data item and builds up an HTML row. Needless to say, the shape of the dropdown list is now entirely under the control of the page author. And this is a huge step forward (see **Figure 5**).

Wrapping Up

The TypeAhead component is an interesting example of component programming in Blazor. It incorporates the logic to retrieve data over the HTTP protocol, as well as templates and some internal interaction between the elements (the input field and the dropdown list). It then communicates with the outside world via events.

Born as a catchy experiment, Blazor is growing significantly, though the way ahead isn't completely clear and may well change in the coming months. At this time, the primary goal of the team is to ship support for running Blazor client-side in the browser over WebAssembly.

At the same time, embedding Blazor in ASP.NET Core has many benefits, not the least of which is a much faster application load time. The Blazor components that will be integrated in ASP.NET Core 3.0 will be renamed to Razor Components. It's just a different name chosen to keep things clear and possibly avoid confusion between what runs in the browser and what runs on the server producing output for the client. At any rate, the component model is expected to stay the same regardless of whether you're running on the server or the client. ■

Dino Esposito *has authored more than 20 books and 1,000-plus articles in his 25-year career. Author of "The Sabbatical Break," a theatrical-style show, Esposito is busy writing software for a greener world as the digital strategist at BaxEnergy. Follow him on Twitter: @despos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Daniel Roth*

JAMES MCCAFFREY

Self-Organizing Maps Using C#

A self-organizing map (SOM) is a relatively simple machine learning (ML) technique/object. However, SOMs are a bit difficult to describe because there are so many variations, and also because SOMs have characteristics that resemble several other ML techniques, including unsupervised clustering and supervised classification. Briefly, I usually think of a SOM as a kind of exploratory clustering analysis where the goal is to assign data items that are similar to each other to a map node.

The best way to get an idea of what a SOM is and to see where this article is headed is to take a look at the demo program in **Figure 1** and the diagram in **Figure 2**. The demo creates a 5x5 SOM for the UCI digits dataset. The dataset has 1,797 items and the SOM has 25 nodes. The SOM is created so that each data item is associated with exactly one map node.

```
C:\SOmap\bin\Debug\SOmap.exe

Begin self-organizing map demo

Reading UCI digits data into memory
Constructing 5x5 SO Map
step = 20000
step = 40000
step = 60000
step = 80000
Map construction complete

Value of map[1][1] vector is:
0.0000  0.0237 . . 0.0086
 [0]     [1]   . .  [63]

Assigning data indices to map

Data indices assigned to map[3][3]:
  265    275    325    329    348    361    384
  393    403    413    429    430    432    438
  472    480    494    498    504   1523   1529
 1542   1580   1633   1665

Most common labels for each map node:
3  2  0  0  6
3  9  9  1  4
3  8  5  9  4
2  2  5  7  4
1  1  8  7  7

End self-organizing map demo
```

Figure 1 **Self-Organizing Map (SOM) Demo**

Code download available at msdn.com/magazine/0119magcode.

After constructing the SOM, the demo computes the most common digit associated with each map node and that information is displayed in **Figure 2**. The result of a SOM analysis is usually visual and must be analyzed somewhat subjectively. For example, you can see most data items that represent a particular digit are assigned to SOM nodes that are close to each other. And, the data items that represent the digit 0 and the digit 6 are close to each other in the SOM, which suggests they're more similar to each other in some way than the digit 4 and the digit 8 are.

This article assumes you have intermediate or better programming skills, but doesn't assume you know anything about SOMs. The demo program is coded using C#, but you shouldn't have too much trouble refactoring the code to another language, such as Visual Basic or Python. The complete code for the demo program, with some minor edits to save space, is presented in **Figure 3**. The code and the data file are also available in the download that accompanies this article.

Understanding SOMs

The first step when creating a SOM is to make sure you understand the target data. The UCI Digits Dataset has 1,797 items. The data looks like:

```
0, 12, 0, 8, . . 16, 7
0, 9, 11, 5, . . 13, 4
...
```

Each item has 65 values. The first 64 values are grayscale pixel values, between 0 and 16, that represent a crude 8x8 handwritten digit. The last value is the digit label, 0 to 9, which corresponds to the pixel value. So, the two items in the previous dataset represent a 7 and a 4. You can create a SOM for data that has labels, or data without labels, but for basic SOMs the non-label data must be strictly numeric.

The demo program creates a 5x5 SOM. The dimensions of a SOM are arbitrary to a large extent and are mostly a matter of trial and error. Each cell of a SOM is called a node. Each node holds a vector value, all with the same

3	2	0	0	6
3	9	9	1	4
3	8	5	9	4
2	2	5	7	4
1	1	8	7	7

Figure 2 **One Possible Visualization of the SOM**

Manipulating Documents?

APIs to view, convert, annotate, compare, sign, assemble and search documents in your applications.

Try GroupDocs APIs for FREE

Download a Free Trial at

https://downloads.groupdocs.com

GROUPDOCS

GroupDocs.Viewer

View over 50 documents and image formats in any application using document viewer APIs.

GroupDocs.Annotation

Add annotations to specific words, phrases and any region of the document.

GroupDocs.Conversion

Fast batch document conversion APIs for any .NET, Java or Cloud app.

GroupDocs.Comparison

Compare two documents and get a difference summary report.

GroupDocs.Signature

Digitally sign Microsoft Word, Excel, PowerPoint and PDF documents.

GroupDocs.Assembly

Document automation APIs to create reports from templates and various data sources.

GroupDocs.Metadata

Organize documents with metadata within any cross platform application.

GroupDocs.Search

Transform your document search process for advance full text search capability.

- GroupDocs.Text
- GroupDocs.Editor
- GroupDocs.Parser
- GroupDocs.Watermark

Americas: +1 903 306 1676 EMEA: +44 141 628 8900 Oceania: +61 2 8006 6987
sales@asposeptyltd.com

Figure 3 **Self-Organizing Map Demo Program**

```
using System;
using System.Collections.Generic;
using System.IO;
namespace SOmap
{
  class SOmapProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("\nBegin self-organizing map demo \n");
      Random rnd = new Random(0);
      int Rows = 5, Cols = 5;
      int RangeMax = Rows + Cols;
      double LearnRateMax = 0.5;
      int StepsMax = 100000;

      // Initialize SOM nodes to random values
      double[][][] map = new double[Rows][][];  // [r][c][vec]
      for (int i = 0; i < Rows; ++i) {
        map[i] = new double[Cols][];
        for (int j = 0; j < Cols; ++j) {
          map[i][j] = new double[64];
          for (int k = 0; k < 64; ++k)
            map[i][j][k] = rnd.NextDouble();
        }
      }

      // Read data and labels into memory
      Console.WriteLine("Reading UCI digits data into memory");
      double[][] data = new double[1797][];
      for (int i = 0; i < 1797; ++i)
        data[i] = new double[64];

      int[] labels = new int[1797];

      FileStream ifs = new FileStream("..\\..\\digits_uci.txt",
        FileMode.Open);
      StreamReader sr = new StreamReader(ifs);
      string line = ""; string[] tokens = null;
      int row = 0;
      while ((line = sr.ReadLine()) != null)  {
        tokens = line.Split(',');
        for (int j = 0; j < 64; ++j)
          data[row][j] = double.Parse(tokens[j]) / 16.0;
        labels[row] = int.Parse(tokens[64]);
        ++row;
      }
      sr.Close(); ifs.Close();

      // Construct the SOM
      Console.WriteLine("Constructing 5x5 SO Map");
      for (int s = 0; s < StepsMax; ++s)  // main loop
      {
        if (s % (int)(StepsMax/5) == 0 && s > 0)
          Console.WriteLine("step = " + s);

        double pctLeft = 1.0 - ((s * 1.0) / StepsMax);
        int currRange = (int)(pctLeft * RangeMax);
        double currLearnRate = pctLeft * LearnRateMax;

        // Pick random data index
        int t = rnd.Next(0, 1797);
        // Get (row,col) of closest map node -- 'bmu'
        int[] bmuRC = ClosestNode(data, t, map);
        // Move each map mode closer to the bmu
        for (int i = 0; i < Rows; ++i) {
          for (int j = 0; j < Cols; ++j) {
            if (ManDist(bmuRC[0], bmuRC[1], i, j) <= currRange)
              for (int k = 0; k < 64; ++k)
                map[i][j][k] = map[i][j][k] +
                  currLearnRate * (data[t][k] - map[i][j][k]);
          } // j
        } // i
      } // s(tep)
      Console.WriteLine("Map construction complete \n");

      // Show one map node
      Console.WriteLine("Value of map[1][1] vector is: ");
      Console.WriteLine(map[1][1][0].ToString("F4") + "  " +
        map[1][1][1].ToString("F4") + " . . " +
        map[1][1][63].ToString("F4"));
      Console.WriteLine(" [0]     [1]   . .  [63] \n");

      // Map has been created. assign data items to map
      Console.WriteLine("Assigning data indices to map \n");
      List<int>[][] mapping = new List<int>[Rows][];
      for (int i = 0; i < Rows; ++i)
        mapping[i] = new List<int>[Cols];
      for (int i = 0; i < Rows; ++i)
        for (int j = 0; j < Cols; ++j)
          mapping[i][j] = new List<int>();

      for (int t = 0; t < 1797; ++t)  // each data item
      {
        // Find node map coords where node is closest to D(t)
        int[] rc = ClosestNode(data, t, map);
        int r = rc[0]; int c = rc[1];
        mapping[r][c].Add(t);
      }

      Console.WriteLine("Data indices assigned to map[3][3]: ");
      foreach (int idx in mapping[3][3])
        Console.Write(idx.ToString().PadLeft(5) + "  ");
      Console.WriteLine("\n");

      // Show one possible visualization
      Console.WriteLine("Most common labels for each map node: ");
      for (int i = 0; i < Rows; ++i) {
        for (int j = 0; j < Cols; ++j) {
          List<int> members = new List<int>();  // '0'- '9'
          foreach (int idx in mapping[i][j])
            members.Add(labels[idx]);
          int mcv = MostCommonVal(members);
          Console.Write(mcv + "  ");
        }
        Console.WriteLine("");
      }

      Console.WriteLine("\nEnd self-organizing map demo");
      Console.ReadLine();
    } // Main

    static int ManDist(int x1, int y1, int x2, int y2)
    {
      return Math.Abs(x1 - x2) + Math.Abs(y1 - y2);
    }

    static double EucDist(double[] v1, double[] v2)
    {
      double sum = 0;
      for (int i = 0; i < v1.Length; ++i)
        sum += (v1[i] - v2[i]) * (v1[i] - v2[i]);
      return Math.Sqrt(sum);
    }

    static int[] ClosestNode(double[][] data, int t,
      double[][][] map)
    {
      // Coords in map of node closest to data[t]
      double smallDist = double.MaxValue;
      int[] result = new int[] { 0, 0 };  // (row, col)
      for (int i = 0; i < map.Length; ++i) {
        for (int j = 0; j < map[0].Length; ++j)  {
          double dist = EucDist(data[t], map[i][j]);
          if (dist < smallDist) {
            smallDist = dist; result[0] = i; result[1] = j;
          }
        }
      }
      return result;
    }

    static int MostCommonVal(List<int> list)
    {
      if (list.Count == 0) return -1;
      int largestCount = 0; int mostCommon = 0;
      int[] counts = new int[10];
      foreach (int val in list) {
        ++counts[val];
        if (counts[val] > largestCount) {
          largestCount = counts[val]; mostCommon = val;
        }
      }
      return mostCommon;
    }

  } // Program
} // ns
```

Visual Studio LIVE! TRAINING SEMINAR

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

ASP.NET Core in the Microsoft Cloud

Dallas, Texas

February 6 - 7, 2018

Two Hot Topics in Two Days

> February 6: ASP.NET Core with Azure DevOps

> February 7: ASP.NET Core Microservices with Docker, K8s, and AKS

TOP REASONS TO JOIN US IN DALLAS

- In-Depth Training
- Two Hot Topics in Two Days
- Networking
- Flexibility

REGISTER TODAY!

Use Promo Code MSDN

VSLIVE.COM/DALLAS

length, and the data items, excluding the label. So, each node in the demo SOM has a vector with 64 values. For example, If you examine **Figure 1**, you'll see that after the SOM was created, the map node at location [1][1] is (0.000, 0.0237, . . 0.0086).

A SOM is created so that each node vector is representative of some of the data items but also so that map nodes that are close to each other geometrically represent data items that are similar. The algorithm used to create the demo SOM, in very high-level pseudo-code, is:

```
create map with random node vectors
loop while s < StepsMax times
  compute what a "close" node is, based on s
  compute a learn rate, based on s
  pick a random data item
  determine map node closest to data item ("BMU")
  for-each node close to the BMU
    adjust node towards data item
end-loop
```

Creating a SOM is really more of a meta-heuristic than a rigidly prescribed algorithm. Each of the statements in the pseudo-code can be implemented in several ways.

The Demo Program

To create the demo program, I launched Visual Studio and created a new C# console application project named SOmap. I used Visual Studio 2017 but the demo has no significant .NET dependencies so any version of Visual Studio will work fine. After the template code loaded, I renamed file Program.cs to SOmapProgram.cs and allowed Visual Studio to automatically rename class Program for me. In the Editor window, I removed all unneeded using statements, leaving just the reference to the System namespace. Then I added references to the Collections.Generic and IO namespaces because the demo uses a List<int> collection and reads data from a text file.

The demo program Main method begins with:

```
Random rnd = new Random(0);
int Rows = 5, Cols = 5;
int RangeMax = Rows + Cols;
double LearnRateMax = 0.5;
int StepsMax = 100000;
```

A key part of the SOM algorithm is determining what it means for two map nodes to be close together. The demo uses Manhattan distance; for example, map nodes at [1][1] and [3][4] have a Manhattan distance of 2 + 3 = 5. For a 5x5 map, the farthest apart any two nodes can be is 5 + 5 = 10.

Inside the main processing loop, the current maximum distance that defines "close" nodes is computed as:

```
double pctLeft = 1.0 - ((s * 1.0) / StepsMax);
int currRange = (int)(pctLeft * RangeMax);
double currLearnRate = pctLeft * LearnRateMax;
```

The pctLeft variable computes the percentage of steps left. For example, if StepsMax was set to 100 and the current loop counter variable is s = 33, then the percentage of steps remaining is 0.67. The maximum neighbor distance for step s is computed using the percentage. Similarly, when neighbor nodes are updated, the percentage of steps remaining is used to gradually decrease the learning rate.

In each training iteration step, the one node in the map that's closest to the randomly selected data item, in terms of Euclidean distance, is called the best matching unit (BMU). The code that updates each map node that's close to the current BMU is:

```
for (int i = 0; i < Rows; ++i) {
  for (int j = 0; j < Cols; ++j) {
    if (ManDist(bmuRC[0], bmuRC[1], i, j) <= currRange)
      for (int k = 0; k < 64; ++k)
        map[i][j][k] = map[i][j][k] +
          currLearnRate * (data[t][k] - map[i][j][k]);
```

In words, for each map node, if the Manhattan distance is less than the current maximum range that defines closeness, then the map node vector is adjusted a small fraction of the difference between the vector's current value and the current data item.

Using a SOM

Suppose you've created a SOM—an n-by-m grid of nodes where each node is a numeric vector that has the same length as the data being analyzed (excluding any labels). Now what? Using a SOM is problem-dependent. If your data doesn't have labels, one common technique is to create a U-Matrix, which is an n-by-m grid where the value of each node is the average distance between the corresponding SOM node vector and neighbor node vectors. A U-Matrix is usually displayed so that each value is interpreted as a grayscale pixel value.

Another possibility for displaying a SOM created from data without labels is to map each node vector to a color. For example, if a SOM node vector has size six, you color each SOM cell using the RGB color model by assigning the average of the first two vector values to R, the second two values to G, and the last two values to B. The demo SOM node vectors have size 64, so there's no obvious way to associate each vector to a color.

If your data has labels, you can map each SOM node to a label. The demo program determines all labels from the data under analysis associated with each SOM node, and then computes the most frequent label for each node, and then colors nodes according to an arbitrary color assignment.

Wrapping Up

There are dozens of variations of the basic self-organizing map structure used in the demo program. Instead of using an n-by-m rectangular grid, you can use a layout where each cell in the SOM is a hexagon. You can use a toroidal geometry where edges of the SOM connect. You can use three dimensions instead of two. And there are many ways to define a close neighborhood for nodes. And so on.

Although SOMs are widely known in the field of ML, they aren't used very often, at least among my colleagues. I suspect there are several reasons for this, but perhaps the main cause is that SOMs have so many variations it's rather confusing to select one particular design. Additionally, based on my experience, in order for a SOM to give useful insights into a dataset, a custom SOM is needed (as opposed to using an off-the-shelf SOM from an ML library). All that said, however, for certain problems scenarios, SOMs can provide useful insights. ■

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

Thanks to the following Microsoft technical experts who reviewed this article: *Chris Lee, Ricky Loynd, Ken Tran*

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

Training Conference for IT Pros at Microsoft HQ!

The FUTURE of TECH is HERE

MICROSOFT HEADQUARTERS
REDMOND, WA
AUGUST 5-9, 2019

In-depth Technical Tracks on:

- Client and EndPoint Management
- Cloud
- Infrastructure
- PowerShell and DevOps
- Security
- Soft Skills for IT Pros

SAVE $400
WHEN YOU REGISTER BY JUNE 14
Use Promo Code MSDN

EVENT PARTNER
Microsoft

SUPPORTED BY
Redmond
Redmond Channel Partner
Virtualization & Cloud Review

PRODUCED BY
Converge 360
An 1105 Media company

TechMentorEvents.com/MicrosoftHQ

DAVID S. PLATT

Mentoring Again

Here I go again. It was two years ago that I wrote about mentoring the software group of my daughter's For Inspiration and Recognition of Science and Technology (FIRST) Robotics team (see msdn.com/magazine/mt790211). Now I'm gearing up for my third run at it. We'll be deep in the fray by the time you read this. When am I ever going to learn?

That first year the team did OK. We improved on our previous year's performance, advancing into the elimination rounds for the first time. We came within three points of reaching the final round in one tournament. But last year we backslid, never reaching the elimination rounds, never even cracking the top half in any tournament.

I tried to spin it as a learning experience. "Look," I told the kids. "Engineering projects sometimes fail. If you want to be an engineer, you have to learn to live with that. It's like being a surgeon: No matter how good you are, no matter how hard you work, some of your patients are going to die. You need to brutally self-analyze, learn what there is to learn from your experience, and move on with your head held high. If you can't do that, stay home and be an accountant." I'm wracking my brain as to how we can reverse our fortunes.

I have four students in my software group, one each from senior to freshman. They all have some notion of basic coding. But it's a huge leap from introductory coding samples to building an embedded real-time program running on rickety hardware, against a hard deadline. I often point to my rewrite of Ecclesiastes in my January 2011 column, saying: There's a time for bulletproof code, and a time for desperate hacks (msdn.com/magazine/gg535678). There's a time for solving a general problem, such as rotating the robot to any arbitrary angle, and a time to hand-optimize the 15 seconds of autonomous operation we need to advance in the standings.

A mentor's role is complex. It contains aspects of teaching, although it's not exactly a teacher. It has aspects of coaching as well, although it's not exactly a coach. I have to keep the students' spark alive, while also introducing them to reality. I try not to write code for them. Instead, I ask questions that point them in the right direction, and then let them run: "Our inertial guidance code isn't integrating distance correctly? [No.] How often does it run? [I don't know.] How could you measure it, quickly and easily? [A simple static counter, written to console at the end.] Fine, do that. [20 times per second.] Jeez, is that all? How can we get more? [Threads?] Maybe. Try looking them up. Is there an example? Careful, those little bastards will turn on you and bite without warning." And so on.

One of the best ways to improve your own skills is to help someone else improve theirs. FIRST Robotics calls that "gracious professionalism." Figuring we'd do better if we put more heads together, I started a software consortium with three other teams in nearby towns. I call it NSFSA (North Shore FIRST Software Alliance, see nsfsa.org). We started with a Java language bootcamp last summer for complete newbies. We then met on alternate Saturdays throughout the fall, discussing topics of interest—sometimes basic program structure, sometimes movement, sometimes vision. Several of the more advanced students helped present the topics. It benefitted them as much as it did the students that they helped, if not more.

I often point to my rewrite of Ecclesiastes in my January 2011 column, saying: There's a time for bulletproof code, and a time for desperate hacks.

Believe it or not, our biggest hurdle is getting every student's PC running with the tools and configurations that they need. The undocumented settings, the development environments and libraries that don't like each other, the firewalls put up by schools with the best of intentions—they all get in the way. I'd love to see the whole shebang in one virtual machine that everyone could just download and run. Or better yet, hosted in the cloud—a huge virtual machine, that we'd only pay to run on the days we needed it. Any sponsors out there?

This is Annabelle's senior year. I want to send her out on a high note. Here we go. ■

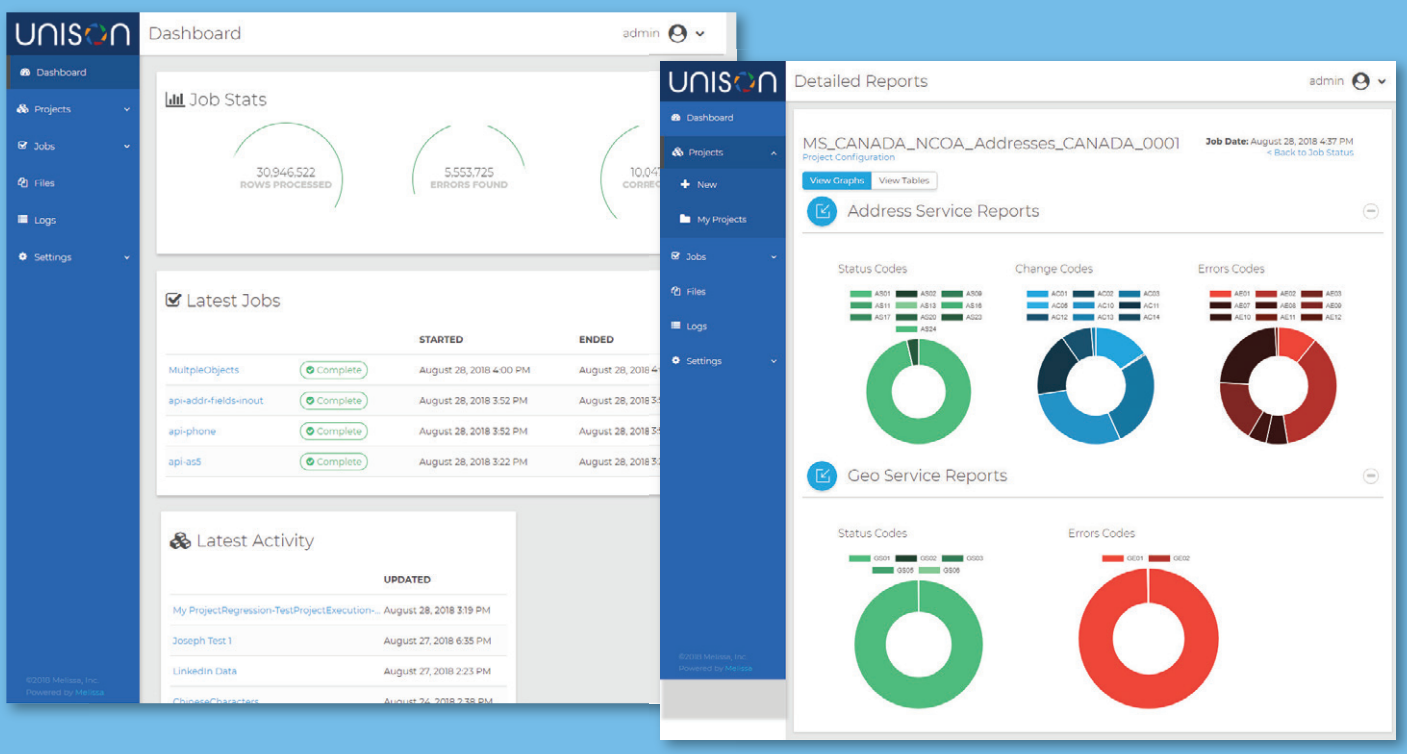
DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should have taped down two of his daughter's fingers so she would learn how to count in octal. You can contact him at rollthunder.com.*

Multiplatform Data Quality Management

Unison – Speedy, Secure, Scalable

Melissa's Unison is a data steward's best friend. It's the ideal multiplatform solution to establish and maintain contact data quality at higher speeds – processing 30+ million addresses per hour – while meeting the most stringent security requirements. With Unison, you can design, administer and automate data quality routines that cleanse, validate and enrich even your most sensitive customer information, as data never leaves your organization. Streamline data prep workflows, reduce analytics busy work, gain more insights and increase efficiency!

- Verify and standardize names, addresses, emails and phone numbers, plus append lat/long coordinates and Census data.
- On-Premise platform requiring no coding or programming with automatic updates from Melissa.
- Works offline, scalable across multiple servers and allows users to script batch jobs with various levels of data access.
- Access, manage and visually analyze the quality of your data over time, enjoy project collaboration and more!



Request a Demo

Melissa.com/msdn-unison

1-800-MELISSA

melissa®