Enter the era of LINQ debugging with the OzCode Early Access Program

Enter the era of LINQ debugging with the OzCode Early Access Program

Jun 2 2016 12:11 AM

LINQ is awesome. It allows us to write powerful, succinct, and expressive code that's a pleasure to read and maintain. When LINQ was first introduced, way back when in the days of yore (circa 2007), it ushered in a new day in C# development and opened up the new and exciting world of functional constructs to the unsuspecting masses of imperative C# programmers. Nine-ish years later, and you'll be hard pressed to find a modern C# code-base that doesn't use LINQ all over the place, and with good reason: LINQ code tells you the story of *what we're doing*, instead of telling you the nuts and bolts of *how we're doing it*. It's the essence of *clean code*.

Debugging LINQ queries, on the other hand, is a completely different story. When you're using LINQ, you're essentially sacrificing the debuggability of your code for the readability of your code. If you F10 over a line of code that's using LINQ - you've just performed a whole bunch of complicated logic, but the debugger won't be helping you figure out what actually happened.

Over the years, developers have come up with a myriad of tricks to make debugging LINQ queries workable. Sometimes they put a breakpoint inside of a lambda expression and hit F5 repeatedly until the breakpoint hit is on the right item (or until their finger gets sore, whichever comes first). Sometimes they split a big query into several smaller queries to make things more workable. In some cases, they end up completely re-writing their LINQ query as imperative code (a bunch of foreach loops) to be able to figure out where things went wrong.

For the past several months, the OzCode team has been fully dedicated to changing the current state of LINQ debugging: our vision is that the more you use LINQ and the functional and beautiful declarative style of programming it enables, your code will actually become **easier** to debug and maintain.

Today, we're excited to announce a new way to debug LINQ statements and we'd like to offer you a chance to try it out, by entering the OzCode Early Access Program.

Getting started

► Click here to sign up for the Early Access Program and grab the latest build

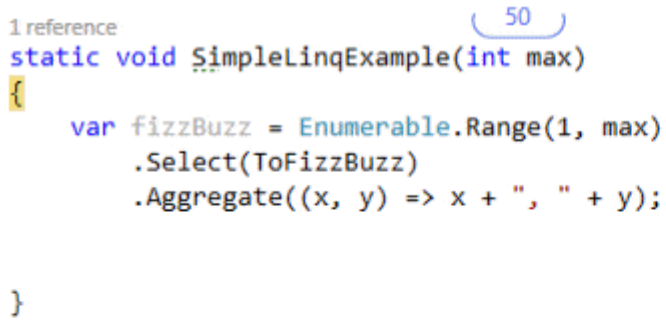
As a quick example, we've used a LINQ query to generate a FizzBuzz sequence.

Once you reach a line of code that contains a LINQ query, you'll notice the new numeric indicators:

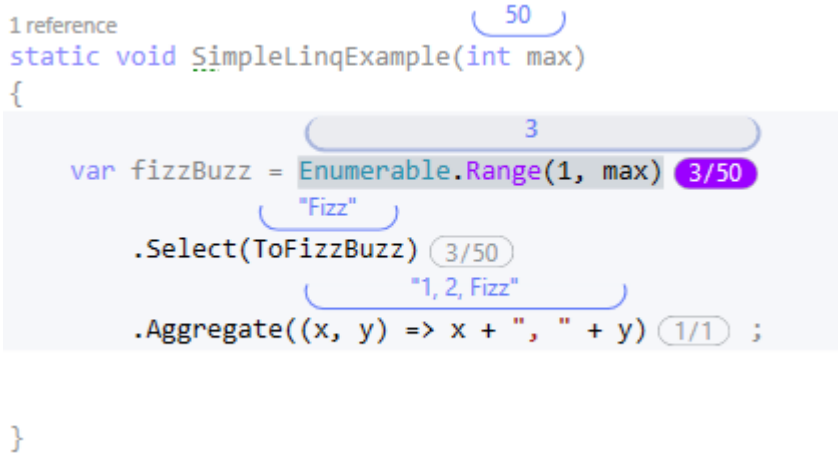
```
                                        50
static void SimpleLinqExample(int max)
{
                        "1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz...
    var fizzBuzz = Enumerable.Range(1, max) 50
        .Select(ToFizzBuzz) 50
        .Aggregate((x, y) => x + ", " + y) 1 ;

    Console.WriteLine(fizzBuzz);
}
```

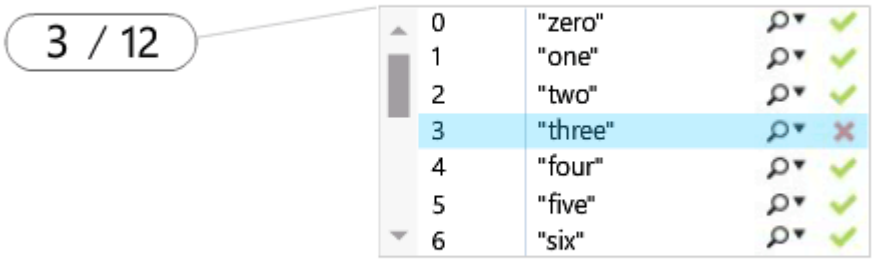
These indicators show you the number of items returned by each operator. Clicking on one of the numbers will open the LINQ DataTip, which will show you all the items that the LINQ operator produced, or all the items the operator consumed:



When we select an item from the list, the annotations on top of code will update to show us how that specific item “travelled” through the LINQ pipeline, showing us exactly what each lambda expression returned. This powerful visualization will help you instantly understand the **flow** of your LINQ query. For example, if we select the 3rd item, we can see how it *fizzed*:



You can quickly run through the different items by using your mouse wheel while hovering over the numeric indicator, like so:



You can also use the OzCode Search functionality to find a specific item in the list.

Get Ready to Fizz and Buzz

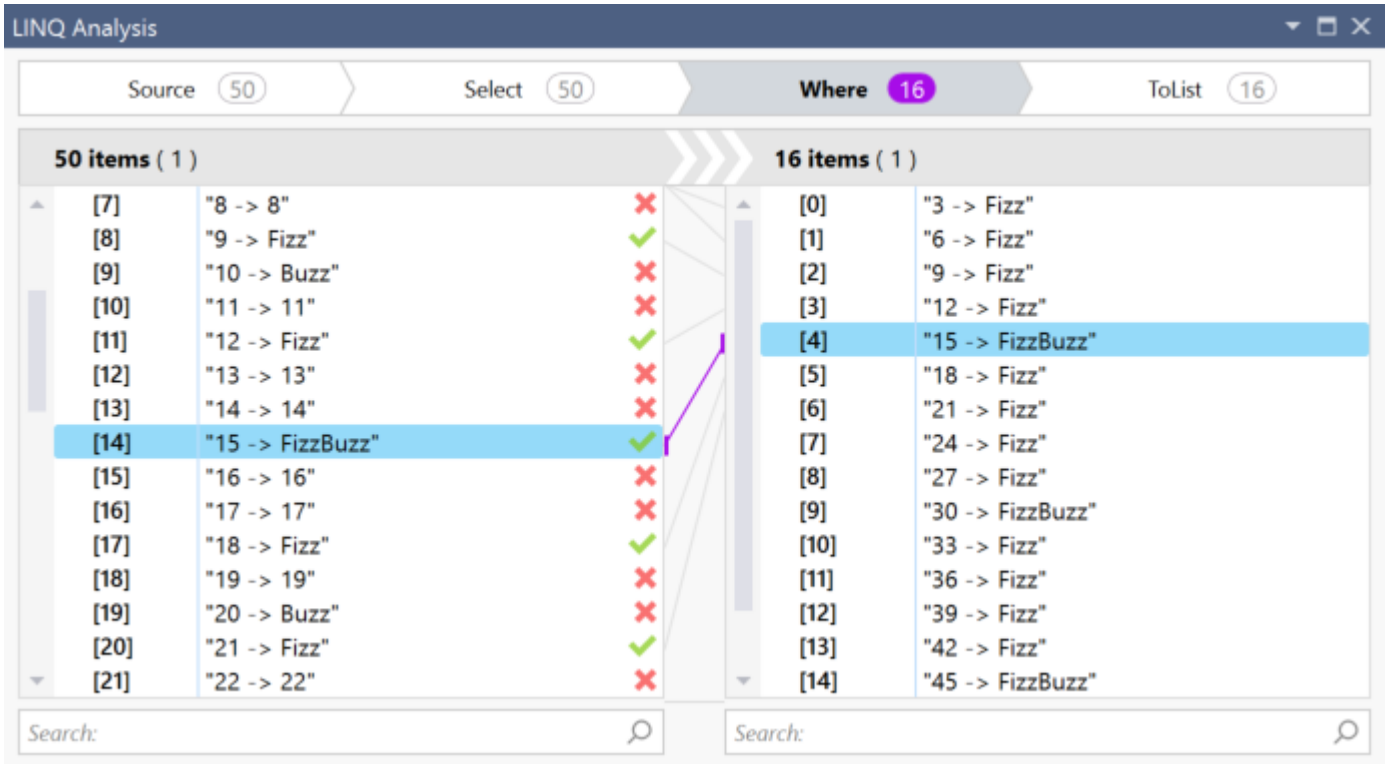
But that's not all!

We've also added a new Tool Window that allows you to move through the query pipeline and see exactly what happened in each stage. On the left hand side, you'll see all the items that **came into** the operator, and on the right hand side, you'll see the items that **came out** of it.

Consider the following query:

```
var fizzy = Enumerable.Range(1, max)
   .Select(i => $"{i} -> {ToFizzBuzz(i)}")
   .Where(s => s.Contains("Fizz"))
   .ToList();
```

Opening the new tool window will show us exactly what items passed through the `Where` predicate and which items were rejected:

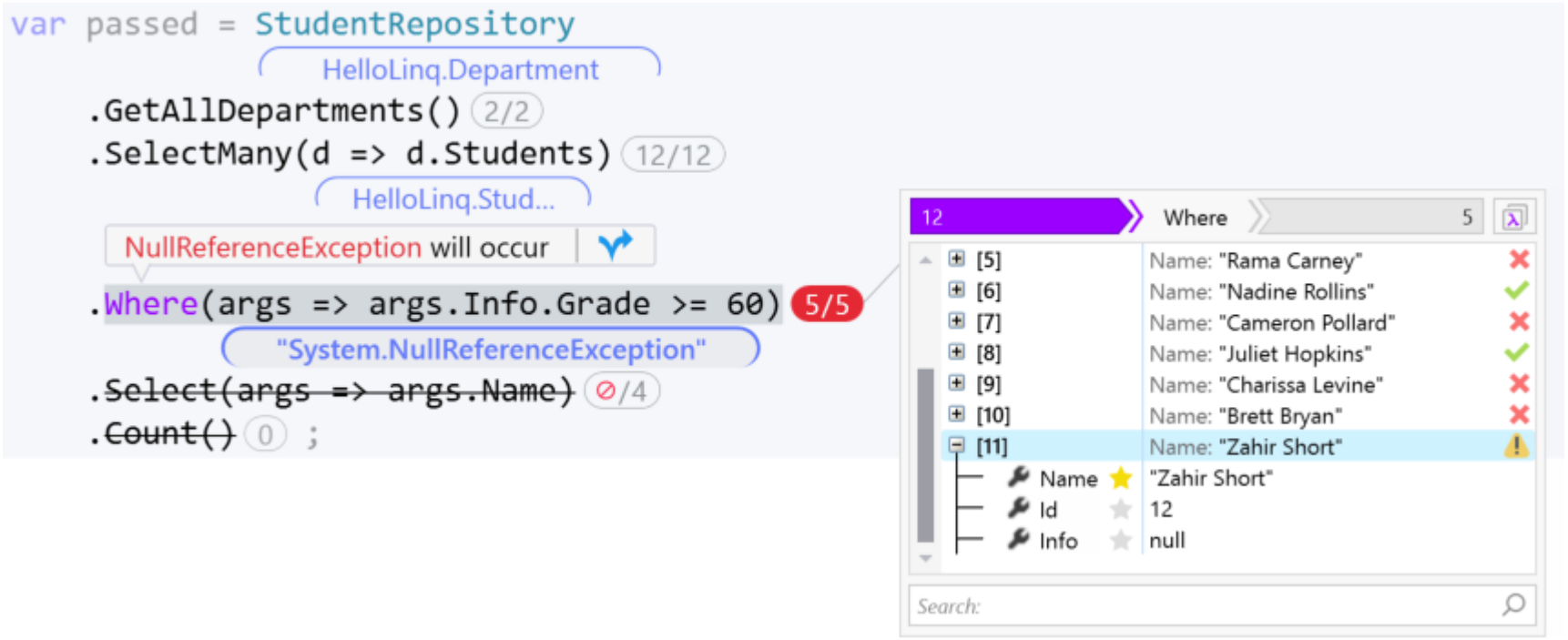


Debugging Exceptions in LINQ Queries

Some of the hardest bugs to tackle are when one of the items inside of a LINQ query causes an exception to be thrown...

Not anymore! OzCode will warn about the exception **before** it's thrown, and show you what item caused the exception, enabling you to drill down and find all the information you need to make sure the bug won't happen again.

In this example, OzCode is showing us that a **NullReferenceException** is about to occur inside the **Where** predicate, and we can quickly see that what's causing the issue here is that the `Info` Property of one specific Student object is `null`:



Wait, there's more

As part of the Early Access Preview you'll also get to see some of the new features that we're going to introduce in OzCode 2.1:

For example, the new & improved **Compare**, which shows the differences between two (or three, or four) instances, or the same instance over time:

```
var phoneBook = LoadPhoneBook();

using (var memoryStream = new MemoryStream())
{
    phoneBook.Save(memoryStream);
    memoryStream.Seek(0, SeekOrigin.Begin);

    var loadedPhoneBook = PhoneBook.CreateFromStream(memoryStream);

                  OzCo...      OzCodeDemo....      "Something went wrong!!!"
    Debug.Assert(phoneBook == loadedPhoneBook, "Something went wrong!!!");
}
```

and **Export**, which enables you to create a textual representation of an object as C# code, Json or XML - this is super-useful when you integrate this new tooling into your Unit Testing workflow.

Found a bug in your debugging session? Spent 10 long minutes just tracking it down, and don't feel like doing it all over again? Simply *Export* the faulty object which caused the error, copy-paste the exported output into a new Unit Test, and voilà! Instant regression test.

A few known limitations to LINQ Debugging

- The LINQ feature only works in VS2015.
- The feature works only with the Fluent Syntax of LINQ, and does not work with the Query Comprehension Syntax (The SQL-esque syntax).
- When using a user-defined LINQ operator (meaning one that isn't a part of Microsoft's System.Linq), the feature does not show the links between the before and after of that operator (the purple lines in the above figure).
- When the query needs to process a large set of data, the numeric indicators will show a question mark instead of the actual numbers, and only after clicking them to go into LINQ Analysis Mode the real numbers will be revealed.

Keep in mind that this is still an Early-Access-Preview release and as such may contain bugs. If you do find a bug, please let us know - we would like to fix it. Same goes with any thoughts you have on how we could improve the cool new LINQ visualization to better fit your needs!

That's it for now, so go ahead, [download the newly LINQified version of OzCode] and let us know what you think!

7 Comments **OzCode** **1 Login**

♡ **Recommend** 1 Tweet Share **Sort by Best**

Łukasz Kurzyniec • 3 years ago • edited

Awesome! I have been waiting for it

1 ^ | v • Share ›

Michael Taylor • 3 years ago

In order to show the count of items, doesn't this require that the LINQ query be executed? This fundamentally changes the behavior of your code under test since, in the debugger it would trigger execution but LINQ itself defers execution until the point that we trigger an enumeration. It is a common bug in code that contexts and other required objects are released before the query runs resulting in a runtime exception. How does this feature work around that or, can we enable the enumeration only if we do something explicit so that we do not modify the behavior under test?

^ | v • Share ›

Omer Raviv Mod → Michael Taylor • 3 years ago • edited

That's a great question! One of the coolest things about this feature is that it is able show you the numeric indicators without actually re-evaluating the query, using the same emulation technique we use in our Predict feature. We described this technique in more detail in our Back to the Future webinar (see below) around the 9 minute mark. This technique does have some limitations though, such as the fact it won't work if the LINQ query calls out to native code.

However, when you actually click the numeric annotation to enter LINQ analysis mode, that's when we re-evaluate the query, which means you should avoid using it if the query has side effect.

OzCode v2.0 Webinar: Back to the Future!

see more

Share ›

Brian Kraemer • 3 years ago

Will this be part of 2.x license?

Share ›

Omer Raviv Mod → Brian Kraemer • 3 years ago • edited

Hi Brian. No, the LINQ debugging experience specifically will be part of OzCode v3.0 and existing v2.x license holders will need to upgrade to keep using it once the v3.0 ships (and there is no release date for v3.0 yet). Having said that, existing customers are welcome to try out the LINQ debugging magic for free while it is still part of the Early Access Preview!

Share ›

Sven Heitmann • 3 years ago • edited

great feature. please make sure your controls can handle dark themes. export results in v2.0 beta is hard to read

Share ›

Omer Raviv Mod → Sven Heitmann • 3 years ago • edited

Hi Sven! The support for Dark Theme in Export has already been fixed in the EAP release, so please try it out! Also, we've worked very hard to make sure the LINQ debugging looks nicely in Dark Theme, so please let us know if you have any feedback about the design!

Share ›

ALSO ON **OZCODE**

First Impressions of Visual Studio 2017 RC

4 comments • 3 years ago

Ryan Edwards — Seems like they are struggling to find new stuff to add! Don't get me wrong, I'll still get it and use it, but two years in the making and these aren't really must-have features for me.

Honest Review of Visual Studio 2019 Preview 1

5 comments • 9 months ago

Omer Raviv — Hi Andre, apologies! We've now updated both places in that blog post to include link to just one installer that supports VS2015, VS2017, and the VS2019 Preview.

Analyzing GitHub LINQ usage - the results

New Subscription Model

Analyzing GitHub LINQ usage – the results

2 comments • 3 years ago

New Subscription Model

43 comments • 3 years ago

Refund policy: we offer refunds for the first payment of a subscription for up to 30 days after payment, subscription renewal payment ar non-refundable.

Privacy Policy Terms of Service