

# Super Optimized LINQ: Indexed Objects

**Aaron Erickson**

**Technical Solution Specialist**

**Magenic Technologies**

***Magenic***

“But I already have a database!”

*That's great for Linq to SQL... but...*

*Indexes aren't just for databases!!!*

*(we just call them Hashtables, Dictionaries, or Maps)*

**Magenic**

Be the Hero!

***Magenic***<sup>®</sup>

What does this have to do with LINQ



What if this dictionary had no index?

Demo:

Removing Pants using Linq and

WPF

LINQ to Objects, by default, is  
slow...

(sorry Microsoft)

# Really? LINQ to Objects is slow?

Over large sets, yes.

*... but there is a way out...*

## This isn't *that* new... (.NET 2.0)

```
class MyCollectionOfFoo : Collection<Foo>
{
    private Dictionary<Foo> _indexOnFoo = new Dictionary<Foo>();

    public new void Add(Foo item)
    {
        base.Add(item);
        _indexOnFoo.Add(item.Key, item);
    }

    public Foo FindByKey(string keyVal)
    {
        return _indexOnFoo[keyVal];
    }
}
```

## With Indexed LINQ...

```
class MyCollectionOfFoo : IndexableCollection<Foo> { }
```

```
class Foo  
{  
    [Indexable()]  
    public string Key{ get; set; }  
}
```

```
var result = from foo in someCollectionOfFoo  
              where foo.Key == "42"  
              select foo;
```

# Some Use Cases:

- Stock symbol lookup (yahoo finance, etc.)
- E-Commerce sites (product/catalog lookup)
- Fingerprint Identification (fast lookup based on a Indexable pattern)

... use your imagination.

Source set of items that fits in server memory (<1M objects that are ~500 bytes per) + Need to frequently search in that set.

But not too small (no real perf benefit at <1000 items)

# Demo: Lookup Book by ISBN

# Indexing Howto

- Establish Index
  - Use Reflection to read attributes of Child class

# Indexing Howto

- Index Maintenance (usually at collection level)
  - Intercept add and remove on collection
  - Advanced: Intercept changes on indexed properties
  - ... or ... require explicit reindex calls (more painful)

# Indexing Howto

- Intercept Search
  - Intercept Where
  - Evaluate Expression, determine whether we can use Index
  - Use index if possible
  - ???
  - Profit

# Indexing Howto

Writing our own Where extension method:

```
var studentResults =  
    from student in _testStudents  
    where student.FirstName == studentNameBox.Text  
    select student;
```

# Indexing Howto

Writing our own Where extension method:

```
var studentResults =  
    from student in _testStudents  
    where student.FirstName == studentNameBox.Text  
    select student;
```

## Converts to...

```
var studentResults =  
    _testStudents.Where( student => student.FirstName ==  
        studentNameBox.Text ).Select( s => s );
```

# Extension Methods

From More to Least Specific

```
var studentResults =  
    from student in _testStudents  
    where student.FirstName == studentNameBox.Text  
    select student;
```

Looks for where on concrete class for `_testStudents`,  
doesn't find it, eventually finds a `Where`  
implementation on `IEnumerable<T>`

# Demo: The Secret Sauce

Real gory details about the code you would write to make something like this work..

# Postscript: Why Study LINQ

All the major new features of VB 9.0/C# 3.0...

... anonymous types

... lambda expressions and expression trees

... extension methods

... implicit typing

*Learning how LINQ works teaches you all of the  
above!!!*

# Summary

You too can

Bend LINQ to your will. 😊

(and learn VB9/C#3 along the way...)

Q & A

# Resources and Links

- i4o – Indexed LINQ: [www.codeplex.com/i4o](http://www.codeplex.com/i4o)
- LINQ Forums at MSDN:  
<http://forums.microsoft.com/MSDN/Showforum.aspx?forumid=123&SideID=1>
- My blog – covers this and other LINQ and .NET topics:  
<http://blog.magenic.com/blogs/aarone/>