About the speaker

- Owner, Benjamin Day Consulting, Inc.
  - Email: benday@benday.com
  - Web: http://www.benday.com
  - Blog: http://blog.benday.com
- Trainer
  - Visual Studio Team System, Team Foundation Server
- Microsoft MVP for C#
- Microsoft VSTS/TFS Customer Advisory Council
- Leader of Beantown.NET INETA User Group

Agenda

- What is LINQ to SQL?
  - Core LINQ
  - Linq to ADO.NET
- The LINQ to SQL Object Model
  - Entity Classes
  - Invoking Database Functions and Sprocs
- Using the LINQ to SQL O/R Designer
- Comparisons:
  - LINQ to SQL vs. Typed DataSets
  - LINQ to SQL vs. LINQ to DataSets
  - LINQ to SQL vs. Raw SQL Access
  - LINQ to SQL vs. NHibernate

Using the LINQ to SQL O/R Designer

- Define classes
- Set up inheritance

LINQ to SQL vs. Typed DataSets

- My $0.02 – LINQ to SQL is the new Typed DataSet

LINQ to SQL vs. LINQ to DataSets

- DataSets in VS2008 are query-able with LINQ
- DataTable's object model has change
- DataTable now extends from TypeTableBase<T>
- TypedTableBase<T> extends DataTable
  - Backwards compatible
- No special way to fill the DataSet/DataTable with LINQ to SQL
  - Still uses DataAdapter

LINQ to SQL vs. Raw SQL Access

- Hopefully you don't do this...
- The data access strategy of masochists everywhere

LINQ to SQL vs. NHibernate

- NHibernate is LINQ to SQL's older, more successful cousin
  - 5+ years in .NET & Java
- Full-featured ORM framework
  - Maps tables/columns to classes/properties
  - Uses xml mapping files or attributes
- Comprehensive inheritance modeling
- Has LINQ-like object query syntax
  - HQL – Hibernate Query Language
- Multi-vendor database support
  - Oracle, SQLServer (2000 & 2005), MySql, Sybase, etc
- Lets you focus on the business problem rather than persistence

Nhibernate's Mappings

- Used by *.hbm.xml (mapping files)
- <class>
  - .NET class to database table
- <id>
  - Represents your object's primary key
- <property>
  - Maps an object property to a database column
- <joined-subclass>
  - Inheritance
- <bag>, <map>
  - Collection management
  - Many-to-one, One-to-many, Many-to-many
  - Many-to-many can be an object
- <component>
  - Map table columns into a dependent object

NHibernate: Person Using <component>

```
<class name="NHibernateResearch.Business.Person, NHibernateResearch.Business"
       table="Person">
  <id name="PersonId" unsaved-value="0">
    <generator class="native" />
  </id>
  <component name="Name" class="NHibernateResearch.Business.Name,
      NHibernateResearch.Business">
    <property name="FirstName" not-null="true"></property>
    <property name="LastName" not-null="true"></property>
  </component>

  <component name="Email" class="NHibernateResearch.Business.Email,
      NHibernateResearch.Business">
    <property name="Address" column="EmailAddress" not-null="true"></property>
  </component>

  <component name="WorkPhone" class="NHibernateResearch.Business.Phone,
      NHibernateResearch.Business">
    <property name="Number" column="WorkPhone" not-null="true"></property>
  </component>

  <component name="HomePhone" class="NHibernateResearch.Business.Phone,
      NHibernateResearch.Business">
    <property name="Number" column="HomePhone" not-null="true"></property>
  </component>
</class>
```

Using LINQ to SQL with Unit Tests

- When testing database code, database must be in a known state
- Easiest way:
  - Wipe the database between tests
  - DataContext.DeleteDatabase()
  - DataContext.CreateDatabase()
  - Database schema always in sync with code
- Harder way:
  - Unit test manages transaction
  - Rollback at end of unit test

LINQ to SQL in an n-tier Application

- Common BaseClass
- Hooking into save events
  - Auto-updating: ModifiedDate, ModifiedBy
- Keeping your code organized with the "Service Layer" pattern

Common Business Base Class

- Each business class should probably have similar fields
  - Id
  - ModifiedDate, ModifiedBy
  - CreateDate, CreatedBy
- Bummer: LINQ to SQL isn't great at this
  - (NHibernate does this effortlessly)
  - Mapped columns must be defined on the concrete

Code Demo

Auto-update the base class methods

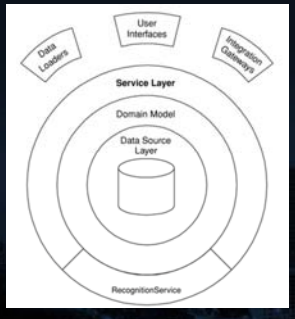
- Generated DataContext & other objects are partial classes
- Generated code gives you partial methods on DataContext for each object
  - InsertXxx(), UpdateXxx(), DeleteXxx()
- Create your own partial class and create your own implementation of the method
- Don't forget to call
  - ExecuteDynamicInsert(), ExecuteDynamicUpdate() or ExecuteDynamicDelete()

Code Demo

Other fun stuff with the partial methods

- Your partial implementations wipe out the LINQ to SQL default implementation
- (Who cares? This is boring.)
- You could put your own implementation that uses stored procedures in your partials!

Service Layer Pattern



"Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation."

-Randy Stafford

From "Patterns Of Enterprise Application Architecture" by Martin Fowler, Randy Stafford, et al. Chapter 9

Why Service Layer?

- Formally defines supported business tier operations (aka methods)
- Methods provide ideal target for unit testing
- Keeps code organized
  - Code review: anything complex not in the service layer → refactor
  - Keeps code out of the UI
- Isolates the Domain Model (business) objects
  - Minimize usage of the Domain Model objects outside of the Business tier

Service Layer in LINQ?

- CRUD operations for each business object
- Any specialized "get" operations
  - Centralized place for any custom *from-where-select's*
- Factory methods
- Create a BusinessFacade<T>

Code Demo

About the speaker

- Owner, Benjamin Day Consulting, Inc.
  - Email: benday@benday.com
  - Web: http://www.benday.com
  - Blog: http://blog.benday.com
- Trainer
  - Visual Studio Team System, Team Foundation Server
- Microsoft MVP for C#
- Microsoft VSTS/TFS Customer Advisory Council
- Leader of Beantown.NET INETA User Group