

LINQ: Reconciling objects, relations and XML in the .NET framework

Erik Meijer
Microsoft Corporation
Redmond, USA
emeijer@microsoft.com

Brian Beckman
Microsoft Corporation
Redmond, USA
bbeckman@microsoft.com

Gavin Bierman
Microsoft Research
Cambridge, UK
gmb@microsoft.com

Introduction Many software applications today need to handle data from different data models; typically objects from the host programming language along with the relational and XML data. The ROX impedance mismatch makes programs awkward to write and hard to maintain.

The .NET Language-Integrated Query (LINQ) framework, proposed for the next release of the .NET framework, approaches this problem by defining a design pattern of general-purpose standard query operators for traversal, filter, and projection. Based on this pattern, any .NET language can define special query comprehension syntax that is subsequently compiled into these standard operators (our code examples are in VB).

Besides the general query operators, the LINQ framework also defines two domain-specific APIs that work over XML (XLinQ) and relational data (DLinq) respectively. The operators over XML use a lightweight and easy-to-use in-memory XML representation to provide XQuery-style expressiveness in the host programming language. The operators over relational data provide a simple OR mapping by leveraging remotable queries that are executed directly in the back-end relational store.

Standard query operators It is well known that collections can be modeled as monoids and queries over them as monoid homomorphisms. Analogously, LINQ defines an API pattern that enables querying of any collection or .NET array. This query operator set includes familiar constructs such as filtering (**where**), mapping (**select**), monadic bind (**selectMany**), sorting (**orderby**) and partitioning (**groupBy**).

These query operators can be freely composed to form rich queries; e.g. the following code returns the name and phone numbers of customers from Seattle ordered by their age:

```
Dim cs = From c In Customers
         Where c.Address.City = "Seattle"
         Order By c.Age
         Select c.Name, c.Phone
```

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.
Copyright 2006 ACM 1-59593-256-9/06/0006 ...\$5.00.

XLinQ The LINQ framework also includes a new, modern, lightweight XML API that makes it simple to program XML and integrates smoothly with the standard query operators of LINQ.

In XLinQ, nodes are first-class citizens that can be passed around independently of an enclosing document. Nested elements are constructed in an expression-oriented fashion. Elements and attributes are accessed uniformly using familiar XPath axis-style methods, while namespace handling is simplified using the notion of universal names.

In addition to the XLinQ API, VB 9.0 adds XML literals with full namespace support. A variant of the previous query that returns each customer as XML simply uses the following expression in the select clause:

```
<Customer Name=<%= c.Name %>>
  <Phone><%= c.Phone %></Phone>
</Customer>
```

DLinq The LINQ framework also provides infrastructure for managing relational data as objects. It leverages the ability of LINQ to provide intensional representations of delegates as code trees by translating language-integrated queries into SQL for execution by the database, and then translating the resulting table into objects. The DLinq infrastructure maintains an identity cache and performs change tracking.

The mapping between database tables and in-memory objects is done via custom attributes (annotations) such as `<Association(OtherKey="CustomerID")>` to indicate foreign key associations and `<Column(Id=true)>` to indicate primary keys.

In the following query, the navigation between the customer and address objects will automatically be compiled into the appropriate joins of the underlying tables.

```
Dim db = new MyDataBase("...");
Dim cs = From c In db.Customers
         Where c.Address.City == "Seattle"
         Select c.Name, c.Address
```

As soon as objects are loaded into the data context, either by retrieving them through a query or by constructing new objects and inserting them, DLinq tracks all the changes and transmits these objects back to the database when requested, automatically generating and executing the appropriate SQL commands.

<http://msdn.microsoft.com/netframework/future/linq/>