

Comprehending Query Expressions

A deep dive into the C# Query Expression Pattern

Bill Wagner

SRT Solutions

bill.wagner@srtsolutions.com

January 9, 2009

About Bill Wagner

- Microsoft Regional Director
 - <http://www.microsoftregionaldirectors.com/>
- C# MVP (2006 – 2008)
- Author of
 - Effective C#
 - More Effective C#
- bill.wagner@srtsolutions.com
- <http://srtsolutions.com/blogs/billwagner>

Agenda

- Discuss Item 36 from More Effective C#
- Query Syntax maps to method calls
- Queries can be more concise
- Method calls enable customization
- Overrides are easy
- Much easier than a QueryProvider



The Query Expression Pattern

```
delegate R Func<T1,R>(T1 arg1);
delegate R Func<T1,T2,R>(T1 arg1, T2 arg2);
class C {      public C<T> Cast<T>(); }
class C<T> : C {
    public C<T> Where(Func<T,bool> predicate);
    public C<U> Select<U>(Func<T,U> selector);
    public C<V> SelectMany<U,V>(Func<T,C<U>> selector, Func<T,U,V> resultSelector);
    public C<V> Join<U,K,V>(C<U> inner, Func<T,K> outerKeySelector,
        Func<U,K> innerKeySelector, Func<T,U,V> resultSelector);
    public C<V> GroupJoin<U,K,V>(C<U> inner, Func<T,K> outerKeySelector,
        Func<U,K> innerKeySelector, Func<T,C<U>,V> resultSelector);
    public O<T> OrderBy<K>(Func<T,K> keySelector);
    public O<T> OrderByDescending<K>(Func<T,K> keySelector);
    public C<G<K,T>> GroupBy<K>(Func<T,K> keySelector);
    public C<G<K,E>> GroupBy<K,E>(Func<T,K> keySelector, Func<T,E> elementSelector);
}
class O<T> : C<T> {
    public O<T> ThenBy<K>(Func<T,K> keySelector);
    public O<T> ThenByDescending<K>(Func<T,K> keySelector);
}
class G<K,T> : C<T> { public K Key { get; } }
```

Where, Select, range variables

```
var smallNumbers = from n in numbers
                   where n < 5
                   select n;
```

```
var smallNumbers2 = numbers.
    Where(n => n < 5).Select(n2 => n2);
```

```
var smallNumbers3 = numbers.
    Where(n => n < 5);
```

```
var allNumbers = from n in numbers select n;
var allNumbers2 = numbers.Select(n3 => n3);
```

Where, Select, range variables

- Where is a filter
- Select is an output selector
- Special note: Degenerate Select

```
class C<T> : C
{
    public C<T> Where(Func<T,bool> predicate);
    public C<U> Select<U>(Func<T,U> selector);
    // elided
}
```

OrderBy and ThenBy

```
var sequence = WeatherData.CreateSequence(50);
```

```
var ordered = from weather in sequence
               orderby weather.Date,
                       weather.HighTemperature,
                       weather.LowTemperature
               select weather;
```

```
var ordered2 = sequence.OrderBy(w => w.Date).
                    ThenBy(w2 => w2.HighTemperature).
                    ThenBy(w3 => w3.LowTemperature);
```

OrderBy and ThenBy

```
var ordered3 = from weather in sequence
                orderby weather.Date descending,
                weather.HighTemperature,
                weather.LowTemperature
                select weather;
```

```
var ordered4 = sequence.OrderByDescending(w => w.Date).
    ThenBy(w2 => w2.HighTemperature).
    ThenBy(w3 => w3.LowTemperature);
```

OrderBy and ThenBy

- Sort a sequence
- Ascending or Descending
- Output a different type to support ThenBy

```
class C<T> : C
{
    public O<T> OrderBy<K>(Func<T,K> keySelector);
    public O<T> OrderByDescending<K>(Func<T,K> keySelector);
}
class O<T> : C<T>
{
    public O<T> ThenBy<K>(Func<T,K> keySelector);
    public O<T> ThenByDescending<K>(Func<T,K> keySelector);
}
```

GroupBy

```
var sequence = WeatherData.CreateSequence(500);
var groupedByHigh = from dataPoint in sequence
                    group dataPoint by dataPoint.HighTemperature
                    into HotDays
                    select new { High = HotDays.Key,
                                size = HotDays.Count() };

// Translate to Nested Query:
var groupedByHigh2 = from HotDays in
                    from dataPoint in sequence group dataPoint by
                    dataPoint.HighTemperature
                    select new { High = HotDays.Key,
                                size = HotDays.Count() };

var groupedByHigh3 = sequence.GroupBy(
    dataPoint => dataPoint.HighTemperature).
    Select(HotDays => new { High = HotDays.Key,
                            size = HotDays.Count() });
```

GroupBy: 2nd Overload

```
var groupedByDate = from dataPoint in sequence
    group dataPoint by dataPoint.Date
    into dailyRecords
    select new { Date = dailyRecords.Key,
        Records = dailyRecords.AsEnumerable()};
```

```
var groupedByDate2 = sequence.GroupBy(dataPoint =>
    dataPoint.Date).
    Select(dailyRecords =>
        new { Date = dailyRecords.Key,
            Records = dailyRecords.AsEnumerable() });
```

GroupBy

- Convert Sequence to IDictionary
 - Each value is a Sequence
 - Each Key is of type parameter K

```
class C<T> : C
{
    public C<G<K,T>> GroupBy<K>(Func<T,K> keySelector);
    public C<G<K,E>> GroupBy<K,E>(Func<T,K> keySelector,
        Func<T,E> elementSelector);
}
class G<K,T> : C<T>
{
    public K Key { get; }
}
```

SelectMany

```
int[] odds = { 1, 3, 5, 7 };
int[] evens = { 2, 4, 6, 8 };
var pairs = from oddNumber in odds
            from evenNumber in evens
            select new
            {
                oddNumber,
                evenNumber,
                Sum = oddNumber + evenNumber
            };

var pairs2 = odds.SelectMany(oddNumber => evens,
    (oddNumber, evenNumber) =>
        new { oddNumber, evenNumber,
            Sum=oddNumber + evenNumber});
```

SelectMany with interim query

```
var values = from oddNumber in odds
             from evenNumber in evens
             where oddNumber > evenNumber
             select new { oddNumber, evenNumber,
                          Sum = oddNumber + evenNumber };
var values2 = odds.SelectMany(oddNumber => evens,
                              (oddNumber, evenNumber) =>
                                new { oddNumber, evenNumber }).
                Where(pair => pair.oddNumber >
                          pair.evenNumber).
                Select(pair => new {
                    pair.oddNumber,
                    pair.evenNumber,
                    Sum = pair.oddNumber + pair.evenNumber });
```

SelectMany

- Select from multiple input sequence
- Similar to a Cross Join
- Output sequence contains $N \times M$ items
 - Assuming N, M items in input sequences
- Notice First Selector signature
- $N \times M \times P$: multiple `SelectMany()` calls

```
class C<T> : C
{
    public C<V> SelectMany<U,V>(Func<T,C<U>> selector,
        Func<T,U,V> resultSelector);
}
```

Join

```
var numbers = new int[]
    { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var labels = new string[]
    { "0", "1", "2", "3", "4", "5" };
var query = from num in numbers
            join label in labels
            on num.ToString() equals label
            select new { num, label };

var query2 = numbers.Join(labels,
    num => num.ToString(),
    label => label,
    (num, label) => new { num, label });
```

Group Join

```
var groups = from p in projects
              join t in tasks on p equals t.Parent
              into projTasks
              select new { Project = p, projTasks };
```

```
var groups2 = projects.GroupJoin(tasks,
    p => p,
    t => t.Parent,
    (p, projTasks) =>
        new { Project = p, TaskList = projTasks });
```

Join, Group Join

- Difference: GroupJoin must contain 'into'
- However, both are used with Join queries

```
class C<T> : C
{
    public C<V> Join<U,K,V>(C<U> inner, Func<T,K>
        outerKeySelector, Func<U,K> innerKeySelector,
        Func<T,U,V> resultSelector);
    public C<V> GroupJoin<U,K,V>(C<U> inner, Func<T,K>
        outerKeySelector, Func<U,K> innerKeySelector,
        Func<T,C<U>,V> resultSelector);
}
```

Cast<T>(): Example

```
System.Collections.ArrayList unclean =  
    new System.Collections.ArrayList();
```

```
var sequence = from int num in unclean  
               where num < 20  
               select num;
```

```
var sequence2 = from num in unclean.Cast<int>()  
                where num < 20  
                select num;
```

Cast<T>()

- Convert IEnumerable to IEnumerable<T>
- Convert IQueryable to IQueryable<T>
- Support Late Binding scenarios

```
class C
{
    public C<T> Cast<T>();
}
```

```
public static class CExtension
{
    public C<TResult> Cast<TResult>(this CNonGeneric source);
}
```

The Query Expression Pattern

```
delegate R Func<T1,R>(T1 arg1);
delegate R Func<T1,T2,R>(T1 arg1, T2 arg2);
class C {      public C<T> Cast<T>(); }
class C<T> : C {
    public C<T> Where(Func<T,bool> predicate);
    public C<U> Select<U>(Func<T,U> selector);
    public C<V> SelectMany<U,V>(Func<T,C<U>> selector, Func<T,U,V> resultSelector);
    public C<V> Join<U,K,V>(C<U> inner, Func<T,K> outerKeySelector,
        Func<U,K> innerKeySelector, Func<T,U,V> resultSelector);
    public C<V> GroupJoin<U,K,V>(C<U> inner, Func<T,K> outerKeySelector,
        Func<U,K> innerKeySelector, Func<T,C<U>,V> resultSelector);
    public O<T> OrderBy<K>(Func<T,K> keySelector);
    public O<T> OrderByDescending<K>(Func<T,K> keySelector);
    public C<G<K,T>> GroupBy<K>(Func<T,K> keySelector);
    public C<G<K,E>> GroupBy<K,E>(Func<T,K> keySelector, Func<T,E> elementSelector);
}
class O<T> : C<T> {
    public O<T> ThenBy<K>(Func<T,K> keySelector);
    public O<T> ThenByDescending<K>(Func<T,K> keySelector);
}
class G<K,T> : C<T> { public K Key { get; } }
```

Questions

- Email to:
 - bill.wagner@srtsolutions.com
- Subscribe to blog:
 - <http://srtsolutions.com/blogs/billwagner>