The Design of FluentQuery

Malcolm Groves

@malgroves

mgroves@embarcadero.com

Agenda

- Part 1 : What's the problem?
- Part 2 : What is FluentQuery?
- Part 3 : How does it work?
- Part 4 : WTF??!? Why did you do it that way?

What's the problem?

```
var
  LName : String;
  LFoundCount : Integer;
begin
  LFoundCount := 0;
  for LName in Listbox1.Items do
    if (LName.StartsWith('Miss', True)) and (LName.Contains('-')) then
    begin
      Inc(LFoundCount);
      ShowMessage(LName);
      if LFoundCOunt = 2 then
        break;
    end;
end;
```

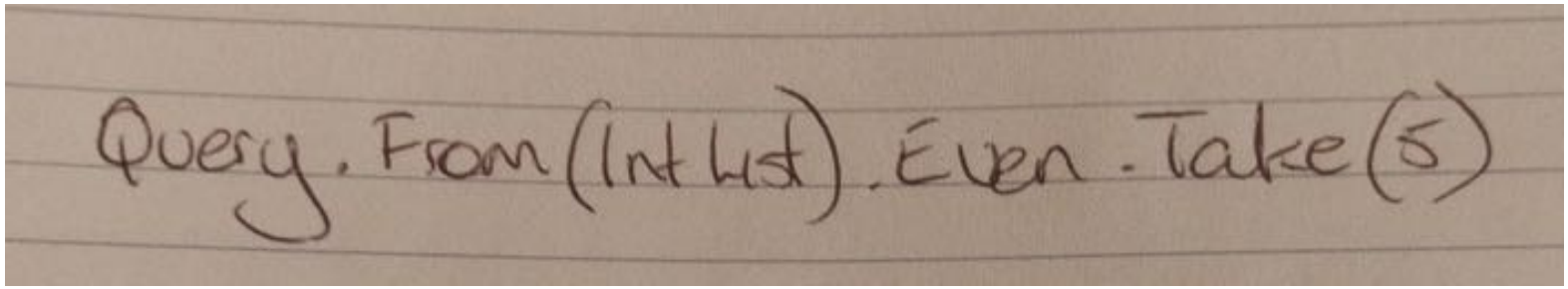
Seriously?

- Get over it, Malcolm!

Tried lots of experiments over the years

- Always ended up feeling like the cure was worse than the disease

First scribble



So what is FluentQuery?

- A declarative query language for manipulating collections in Delphi.
- WTF does that mean?

Original vs FluentQuery

```
var
  LName : String;
  LFoundCount : Integer;
begin
  LFoundCount := 0;
  for LName in Listbox1.Items do
    if (LName.StartsWith('Miss', True)) and
       (LName.Contains('-')) then
    begin
      Inc(LFoundCount);
      ShowMessage(LName);
      if LFoundCOunt = 2 then
        break;
    end;
end;
```

```
var
  LName : String;
begin
  for LName in StringQuery
                .From(ListBox1.Items)
                .StartsWith('Miss’)
                .Contains('-’)
                .Take(2) do
    ShowMessage(LName);
end;
```

Not just strings

- `PointerQuery.From(Flist).IsAssigned.Take(5)`
-
```
ObjectQuery<Tperson>
   .Select
   .From(FPersonCollection)
   .IsA(TCustomer)
```
-
```
FileSystemQuery
   .From(Fdir)
   .Files.LargerThan(100mb)
   .Extension('.bak')
```

Not just strings

- ```
  ComponentQuery<TComponent>
      .Select
      .From(FTestForm)
      .IsA(TWinControl)
      .BooleanProperty('Enabled',  False)
  ```
  ```

Not just `for..in` statements

```
var
  LPerson : TPerson;
begin
  LPerson := ObjectQuery<TPerson>
               .Select
               .From(FPersonCollection)
               .Where(LOver18)
               .First;
```

Not just `for..in` statements

```
var
  LStrings : TStrings;
begin
  LStrings := StringQuery
                .From(FStrings)
                .Skip(4)
                .Take(3)
                .ToTStrings;
```

Not just `for..in` statements

```
Listbox.FilterPredicate := StringQuery
                              .EndsWith('e')
                              .Predicate;
```

Not just for..in statements

```
Var
  LMinValue : Integer;
Begin
  LMinValue := IntegerQuery
                 .From(FNewReadings)
                 .Positive
                 .Min;
```

Bound and Unbound Queries

- Unbound Queries have not yet been attached to the data they are querying
- They can be passed around as parameters until they need to be evaluated against some data.

```
for LFile in FileSystemQuery.From(Ldir).NameMatches(??)  do
```

```
for LFile in FileSystemQuery
                .From(Ldir)
                .NameMatches(StringQuery.Substring(2,  1).Matches('-')) do
```

- Calling `From` on an UnboundQuery transforms it into a Bound Query

Design Goals for FluentQuery

- For the user:
  - Declarative : State what you want, not how to get it
  - Simple for user
    - CodeInsight guidance
    - No memory management
    - Rarely have to write predicates
  - Don’t require special collections.
  - Query should read like a sentence
  - Keep what you want separate from what you’re doing with it.
- For me:
  - Learn a bunch

Design Process

1. Start with how it will be used
2. Implement a concrete instance
3. Implement a second concrete instance
4. Refactor the commonality
5. Repeat

How does it work?

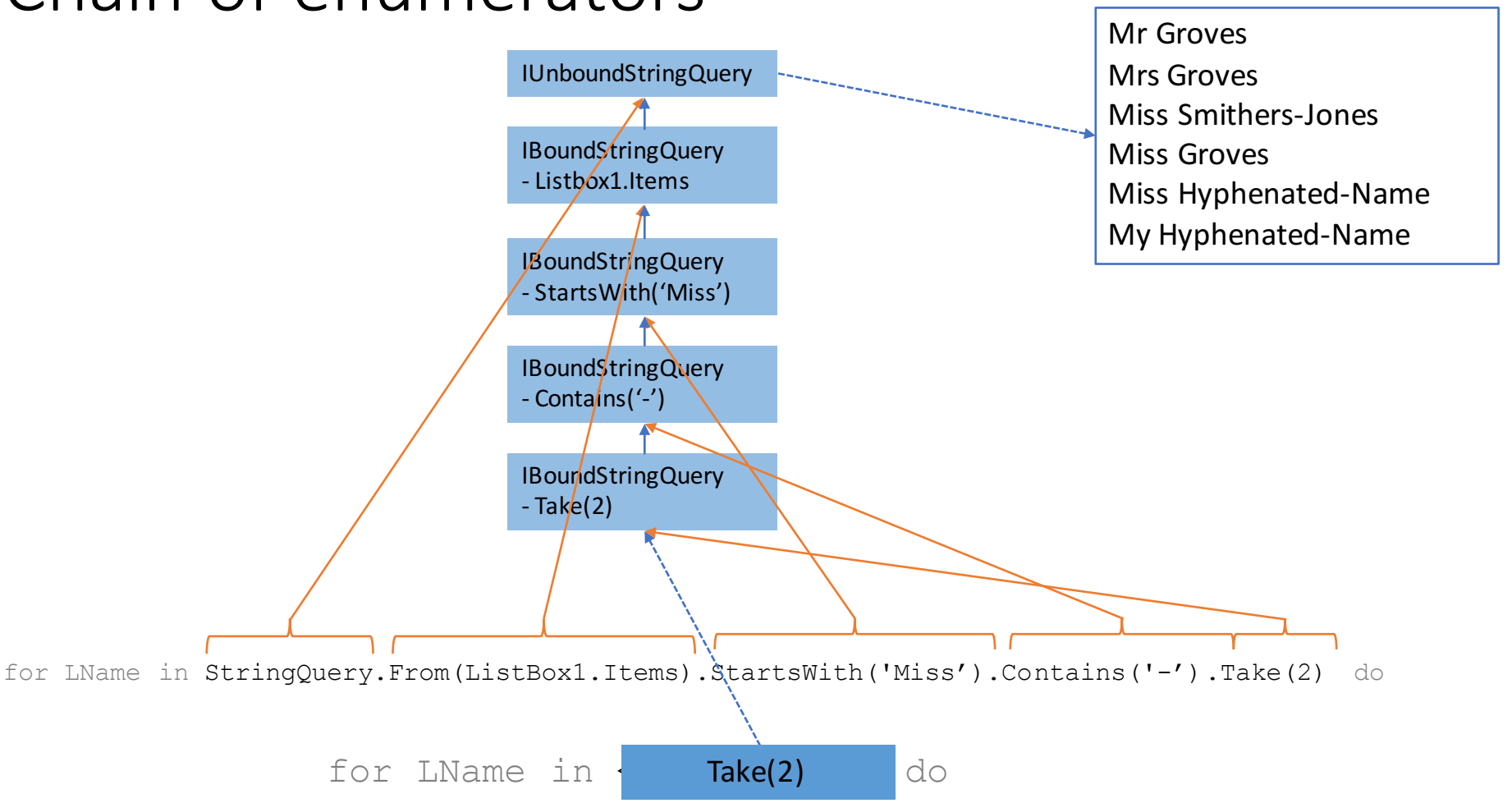
Enumerators all the way down

How does a for..in work?

```
for Lstring in Listbox1.Items do

for Lstring in <enumerator> do
```

Chain of enumerators



Interface delegation to internal class

- Who has used the `implements` keyword?

All high-level operators delegate down to a handful of primitive operators.

- Map
- Where
- SkipWhile*
- TakeWhile*
- Reduce

* I strongly suspect I can make SkipWhile and TakeWhile delegate to Where, but haven’t investigated yet.

Criticism

- The code is complicated
  - Yes, it is.

Response

- There is always complexity, all you can do is decide:
  - Where you put it
  - Who it touches

Criticism

- It’s slower than a hand written for loop
  - Yes, right again.

Response

- I tend to lead with readable, maintainable code.
- If it turns out to be a performance issue, I can replace it
  - NB: It never has so far.

Criticism

- It doesn’t cover the collection type I need
  - Possibly not.

Response(s)

- That's why there is a TGenericQuery<T> and Where(Predicate)
- Selfishly, I wrote this for me.
  - The collections it covers are those I've needed it for.
- Its Open Source, add your query yourself and submit a pull request.

Criticism

- I hate fluent-style code.
  - So don’t write it that way.

```
var
  LName : String;
begin
  for LName in StringQuery
                .From(ListBox1.Items)
                .StartsWith('Miss’)
                .Contains('-’)
                .Take(2) do
    ShowMessage(LName);
end;
```

Is functionally identical to

```
var
  LName : String;
begin
  for LName in StringQuery.From(ListBox1.Items).StartsWith('Miss’).Contains('-’).Take(2) do
    ShowMessage(LName);
end;
```

It’s just taller and skinnier

Observation

- The implementation changed dramatically, multiple times
- The user-level API rarely broke

Observation

- DVCS + Automated Tests rock for making you brave.

Observation

- Instrument your code for debugging.
  - Just be careful your debug instrumentation doesn’t hide the bug.

Observation

- Boolean literals as param values suck.

Observation

- Interfaces rock for structural code. They suck for domain code.
  - This is partly a tooling problem, but not entirely.

That’ll do. Any Questions?

- http://github.com/malcolmgroves/fluentquery
- Been meaning to put it in GetIt.