
Rethinking the Architecture of EMF ORM in terms of LINQ

Miguel Garcia ⁽¹⁾
Rakesh Prithiviraj ⁽²⁾

2008-11-18

TUHH
Technische Universität Hamburg-Harburg

⁽¹⁾ <http://www.sts.tu-harburg.de/~mi.garcia>

⁽²⁾ <http://www.tu-harburg.de/~sorp1428/>



November 19th - 20th
Ludwigsburg, Germany

The Big Picture

- **Abstractions finding a new home:**
 - Functional queries used to be VM-only, LINQ brings them to DBs
 - Transaction demarcation used to be DB-only, but now STM (Software Transactional Memory) comes in C# 3.0, Java prototypes
- **The LINQ landscape:**
 - succinct textual syntax,
 - closures in the abstract syntax,
 - translation into native queries
 - using closure-aware reflection
 - In multiple passes, comprising intermediate languages (Entity SQL, ADO.NET Commands)
- **LINQ for Java: why the puzzle doesn't fall in place**
- **A new ORM engine? Candidates**

First reaction upon learning LINQ: Why didn't someone think about this before?

- The textual syntax constitutes a facade to building blocks (the Standard Query Operators), each of them easy to understand on its own

```
from c in context.Customers
group c by c.Address.Region into regions
select new { region = regions.Key,
            count = regions.Count() };
```



```
value(System.Data.Objects.ObjectQuery`1
      [NorthwindEFModel.Customer])
  .GroupBy(c => c.Address.Region)
  .Select(regions => new <>f__AnonymousTypef`2(
    region = regions.Key,
    count = regions.Count())
  )
```



Some building blocks come in different shapes ...

<pre>public static IQueryable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector)</pre>	<p>Groups the elements of a sequence according to a specified key selector function.</p>
<pre>public static IQueryable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, IEqualityComparer<TKey> comparer)</pre>	<p>Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.</p>
<pre>public static IQueryable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey, TElement>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TElement>> elementSelector)</pre>	<p>Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.</p>
<pre>public static IQueryable<TResult> GroupBy<TSource, TKey, TResult>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TKey, IEnumerable<TSource>, TResult>> resultSelector)</pre>	<p>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.</p>

... and those were just the first four overloads!

<pre>public static IQueryable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey, TElement>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TElement>> elementSelector, IEqualityComparer<TKey> comparer)</pre>	<p>Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.</p>
<pre>public static IQueryable<TResult> GroupBy<TSource, TKey, TElement, TResult>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TElement>> elementSelector, Expression<Func<TKey, IEnumerable<TElement>, TResult>> resultSelector)</pre>	<p>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.</p>
<pre>public static IQueryable<TResult> GroupBy<TSource, TKey, TResult>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TKey, IEnumerable<TSource>, TResult>> resultSelector, IEqualityComparer<TKey> comparer)</pre>	<p>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer.</p>
<pre>public static IQueryable<TResult> GroupBy<TSource, TKey, TElement, TResult>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TElement>> elementSelector, Expression<Func<TKey, IEnumerable<TElement>, TResult>> resultSelector, IEqualityComparer<TKey> comparer)</pre>	<p>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Keys are compared by using a specified comparer and the elements of each group are projected by using a specified function.</p>

It's not only about brevity, but flexibility too

The IDE performs Type Inference (on LINQ textual syntax) and can thus pick the proper “strongly typed” method signature for each AST node

“The query behavior that occurs as a result of executing an expression tree that represents calling

```
GroupBy<TSource, TKey>(
    this IQueryable<TSource> source,
    Expression<Func<TSource, TKey>> keySelector)
```

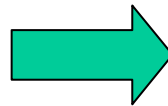
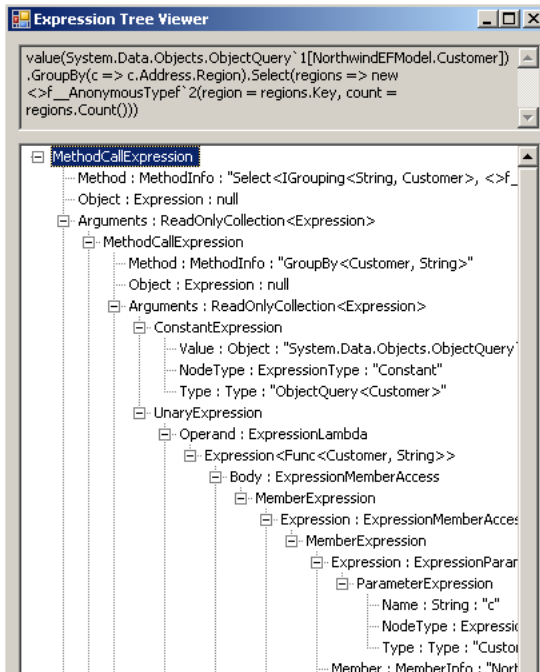
depends on the implementation of the type of the source parameter.

The expected behavior is that it groups the elements of source by a key value that is obtained by invoking keySelector on each element.”

In other words, by writing a custom LINQ provider a custom data store can be targeted (separation of duties, compare with a monolithic JPA implementation)

So we have an AST. What next?

In the Entity Framework, an AST first gets translated to Entity SQL (a composable query language) and from there to ADO.NET Commands (based on mapping metadata). OK, this component is monolithic (but only this one)



```
SELECT
  1 AS [C1],
  [GroupBy1].[K1] AS [Region],
  [GroupBy1].[A1] AS [C2]
FROM ( SELECT
        [Extent1].[Region] AS [K1],
        COUNT( CAST( 1 AS bit) ) AS [A1]
      FROM [dbo].[Customers] AS [Extent1]
      GROUP BY [Extent1].[Region]
    ) AS [GroupBy1]
```

Sidenote: “composable” meaning that:

In some SQL implementations, certain expressions can only be placed in specific constructs ... In contrast, [In Entity SQL] a sub-query in the SELECT clause is treated in the same way as in a FROM clause, as opposed to SQL in which sub-queries in the SELECT clause must evaluate to scalars and as collection in a FROM clause.

<http://blogs.msdn.com/adonet/archive/2007/05/30/entitysql.aspx>

Factsheet of LINQ-like frameworks for Java

All three projects below adopt embedded DSLs.
As of now, no query rewriting addressing optimization is performed.

- LiquidForm, embedded DSL for building type-safe and refactoring-proof JPA queries <http://code.google.com/p/liquidform/>
- Quaere, <http://quaere.codehaus.org/>
- JaQue, <http://code.google.com/p/jaque/> (more on next slide)

LINQ look-alikes but still different:

- JLINQ, a generator of JDBC code (now rebranded as pureQuery) <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0708ahadian/>
- Criteria APIs: no compile-time checks (they are Strings)
- Scala and Java closures: syntactically similar to LINQ code, but compiled into bytecode and not into compile-time ASTs (thus, no rewriting)

JaQue: Delayed iterators example

```
/** Filters the input object stream according with the specified predicate.
 */
public static <E> Iterable<E> where(final Iterable<? extends E> source,
    final Function<Boolean, ? super E> predicate) { // Comparable<T>
    return new Iterable<E>() {
        public final Iterator<E> iterator() {

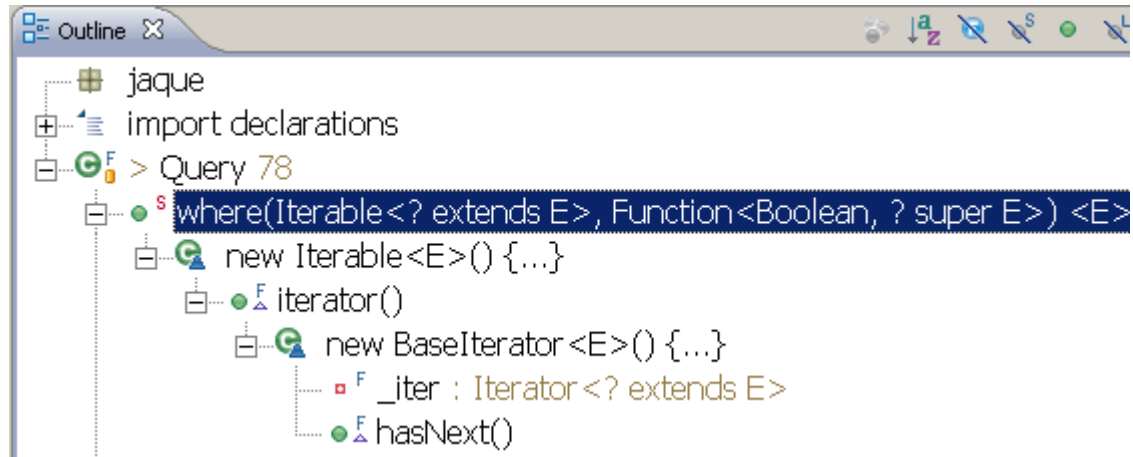
            return new BaseIterator<E>() {
                private final Iterator<? extends E> _iter = source.iterator();

                // @Override
                public final boolean hasNext() {

                    while (_iter.hasNext()) {
                        E current = _iter.next();
                        try {
                            if (predicate.invoke(current))
                                return yield(current);
                        } catch (RuntimeException re) {
                            throw re;
                        } catch (Throwable e) {
                            throw new
                                RuntimeException(e);
                        }
                    }

                    return yieldBreak;
                }
            };
        }
    };
}
```

From <http://jaque.googlecode.com/>
“JaQue to Objects and JaQue to XML are currently supported and JaQue to JPA is under development”



Comments so far on “porting” LINQ to Java

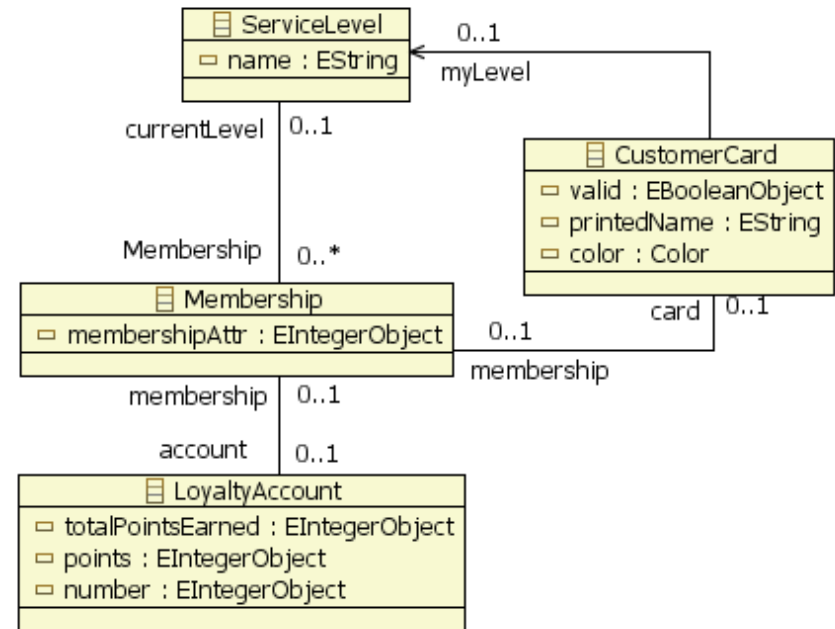
- **It’s not easy to customize the JDT to account for (let’s say) embedded LINQ syntax (for a comparison, see Dali and JPQL strings)**
- **Even if that is somehow achieved, agreement should still be reached on the APIs for other components:**
 - LINQ metamodel
 - LINQ provider (custom translator into native queries)
 - Entity SQL
 - etc.

Teneo

- **Generates O/R mapping metadata (for JDO and Hibernate) based on (optionally annotated) Ecore models**
- **Query language: ORM-engine specific (HQL or JDOQL). Those queries are shipped to the ORM engine which in turn ships their SQL expansion to the DBMS.**
 - What happened to “platform-independent”?
- **No functional-style queries**
- **Eager fetching leads to several batches of SQL (with a roundtrip in-between) for a single HQL/JDOQL query**
- **With lazy fetching, same story: several roundtrips are incurred if the query involves path expressions**

Query shipping in Teneo

Example from the domain of customer loyalty (e.g., “DeutschlandCard”).
The query below retrieves all accounts for memberships.



– HQL Query: **SELECT** ms.account **FROM** Membership ms

```
SELECT loyaltyacc1_.e_id AS e1_35_, ...
FROM "membership" membership0_
LEFT OUTER JOIN servicelevel_membership membership0_1_
ON membership0_.e_id=membership0_1_."membership_e_id"
INNER JOIN "loyaltyaccount" loyaltyacc1_
ON membership0_.account=loyaltyacc1_.e_id
```

– Sample result row : EntityKey[LoyaltyAccount#229376]

Query shipping: yes, but with roundtrips

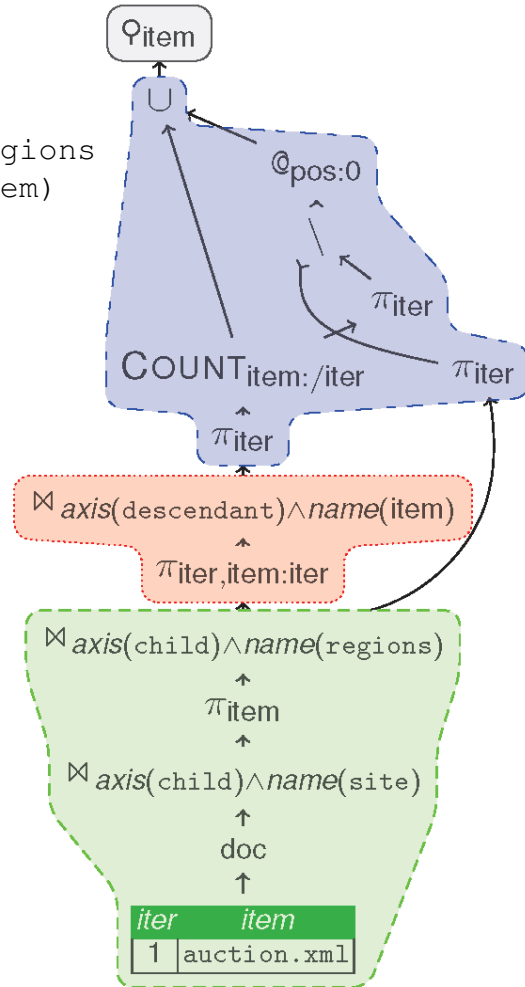
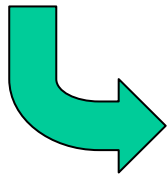
- Based on the resultset, another SQL query is shipped.

```
SELECT membership0_.e_id as e1_3_0_, ...
FROM "membership" membership0_
LEFT OUTER JOIN servicelevel_membership membership0_1_
ON membership0_.e_id=membership0_1_."membership_e_id"
WHERE membership0_.account=?
```

- Sample result row : EntityKey[Membership#262144]
- Why are memberships retrieved if the original query asked for accounts?
- Two roundtrips even with lazy fetching for a query with a single “dot navigation” expression
- Eager fetching costs a third roundtrip

A query optimizer suited for LINQ: Pathfinder

```
let $auction :=
doc("auction.xml")
return
  for $ b
  in $auction/site/regions
  return count($b//item)
```



```
WITH
t0 (iter_pre) AS
(SELECT c.pre
 FROM doc AS a, doc AS b, doc AS c
 WHERE (a.kind = 6)
 AND (a.value = 'auction.xml')
 AND (b.pre BETWEEN
 (a.pre + 1) AND (a.pre + a.size))
 AND (b.name = 'site')
 AND (b.kind = 1)
 AND (b.level = 1)
 AND (c.pre BETWEEN
 (b.pre + 1) AND (b.pre + b.size))
 AND (c.name = 'regions')
 AND (c.kind = 1)
 AND (c.level = 2)),
t1 (iter_pre) AS
(SELECT d.iter_pre
 FROM t0 AS d, doc AS e, doc AS f
 WHERE (d.iter_pre = e.pre)
 AND (f.pre BETWEEN (e.pre + 1)
 AND (e.pre + e.size))
 AND (f.name = 'item')
 AND (f.kind = 1)),
t2 (item_int, iter_pre) AS
(SELECT COALESCE (COUNT (h.iter_pre), 0) AS item_int,
 g.iter_pre
 FROM t0 AS g
 LEFT OUTER JOIN t1 AS h
 ON (h.iter_pre = g.iter_pre)
 GROUP BY g.iter_pre)
SELECT i.item_int
 FROM t2 AS i
ORDER BY i.iter_pre ASC;
```

Look, no roundtrips!

XQuery Join Graph Isolation. Torsten Grust, Manuel Mayr, Jan Rittinger. <http://arxiv.org/abs/0810.4809>
 Proceedings of the 25th International Conference on Data Engineering (ICDE 2009), Shanghai, China, March/April 2009.

Looking into the future (1 of 2)

- **Ecore models may be annotated with:**

- invariants
 (“no external reviewer may be a subordinate of the project leader”)
- derived attributes and references
 (“each node keeps track of the number of descendants”)
- data-driven rules (“if P holds perform Q”)

- **All of them call for query optimization:**

- Invariants are boolean queries over a DB snapshot,
- derived data may be computed
 - from scratch upon request, or
 - in advance, upon update of base data
- the condition-part of a data-driven rule is a boolean query

Looking into the future (2 of 2)

- **In terms of a translation into an ORM-backed data access layer, the following mappings are feasible:**
 - invariants → DB-level integrity constraints (breaking them causes rollback)
 - derived data → incrementally materialized views
 - data-driven rules → DB-level triggers
- **Invariants, derived data, and data-driven rules could be enforced:**
 - by the DB, as suggested above, or
 - by the generated data access layer itself. The advantage here is that languages targeting Ecore (OCL, LINQ) are more expressive than the SQL-subset that DBMSs allow for integrity constraints.
- **In the latter case, Pathfinder-style optimizations are the way to go.**

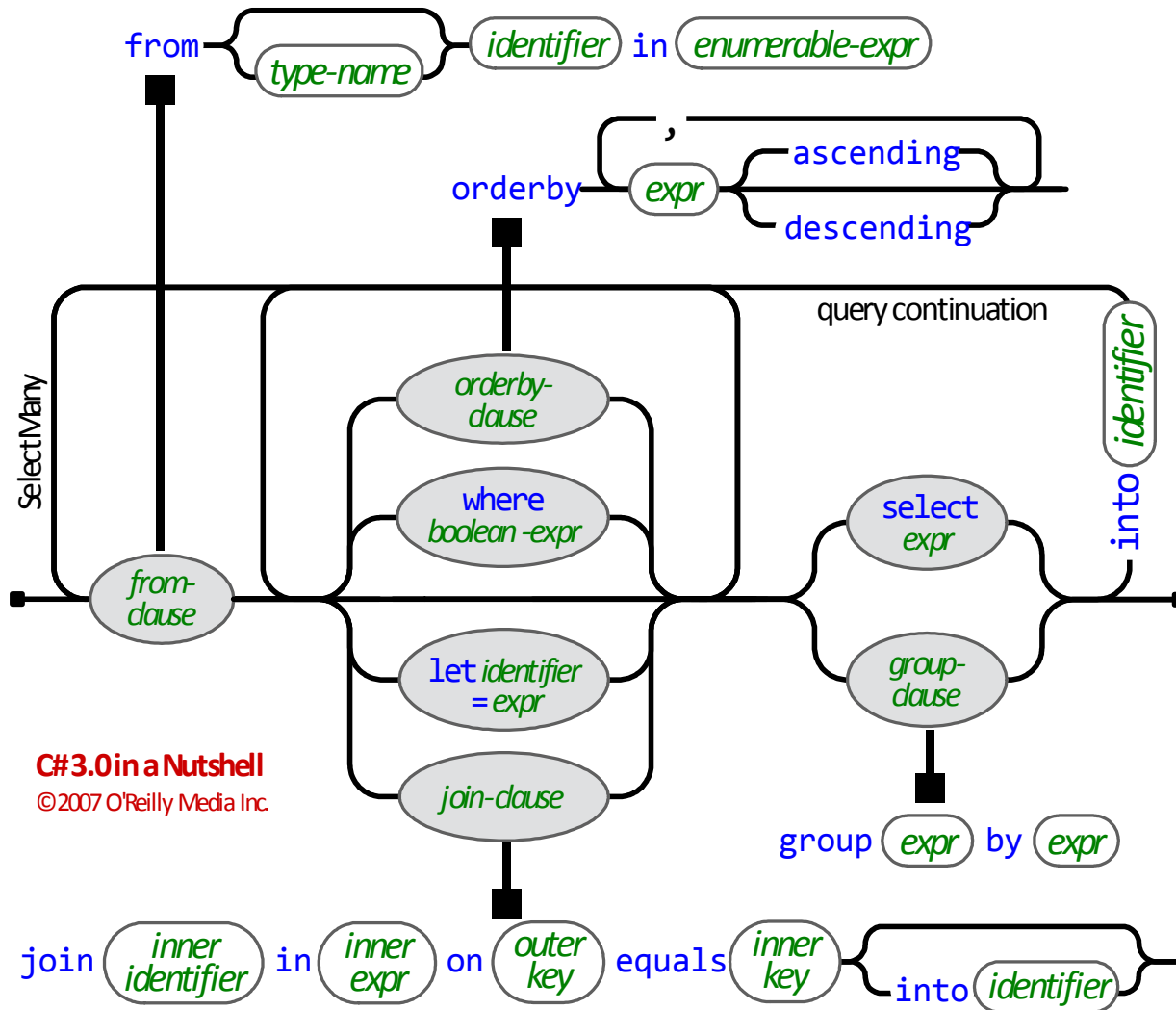
What others are doing (1 of 2)

- **Ferry, successor to Pathfinder**
<http://www-db.informatik.uni-tuebingen.de/research/ferry>
- **Portal on object database technology**
<http://www.odbms.org/>
- **International Conference on Object Databases**
ICOODB 2008, <http://projekt.tfh-berlin.de/icoodb/icoodb-2008/>
(ICOODB 2009 will take place July 1-3 at ETH-Zürich)
- **Comments on how Mono can help when implementing LINQ providers outside .NET**
<http://evain.net/blog/articles/category/db4o>

What others are doing (2 of 2)

- The upcoming SAP MOIN (Modeling Infrastructure) includes a model repository. Its query language (MQL) follows the functional paradigm and is subject to optimization. At the time of this writing, no technical literature for MOIN has been released at <http://sdn.sap.com>, but Google tells us about the following patent applications
- Handling of queries of transient and persistent data
 - EP 1 983 447 A1
<https://publications.european-patent-office.org/PublicationServer/getpdf.jsp?cc=EP&pn=1983447&ki=A1>
 - 20080262999
<http://www.faqs.org/patents/app/20080262999>

Backup slides



C#3.0 in a Nutshell
© 2007 O'Reilly Media Inc.

Railroad diagram for the textual syntax of LINQ, reproduced from <http://www.albahari.com/nutshell/linqsyntax.html>

Backup slides

