

```
<?xml version="1.0"?>
<quiz>
  <question>
    Who was the forty-second
    president of the U.S.A.?
  </question>
  <answer>
    William Jefferson Clinton
  </answer>
  <!-- Note: We need to add
    more questions later.-->
</quiz>
```

XML

XML

Extensible Markup Language – расширяемый язык разметки

История

- Понятие гипертекста было введено В.Бушем еще в **1945** году
 - Реальное использование — когда возникла реальная необходимость в механизме объединения множества информационных ресурсов, обеспечении **создания** и просмотра **нелинейного** текста.
- Пример реализации этого механизма — WWW.



- Основа для всех языков разметки – обобщенный структурированный язык разметки SGML (Structured Generalized Language), утвержден ISO в качестве стандарта в 80-х годах.
- Определяет допустимый набор тэгов, их атрибуты и внутреннюю структуру документа.
- SGML - это метаязык, то есть средство формального описания языка разметки.

Что такое XML

- Язык разметки, описывающий класс объектов данных, называемых XML- документами.
- Используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов.
- Не содержит тэгов, предназначенных для разметки – просто определяет порядок их создания.

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

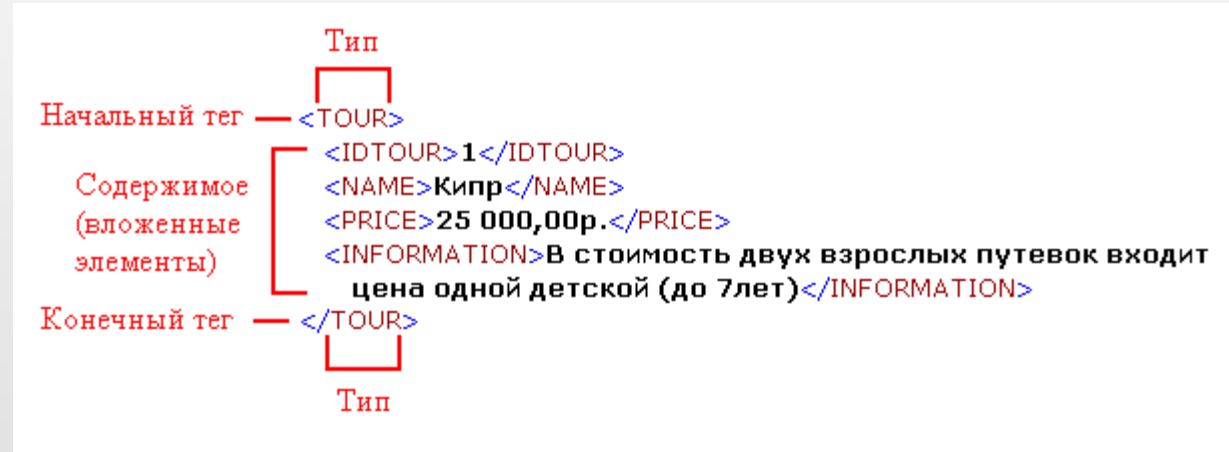
Зачем XML

- XML - расширяемый язык разметки
 - является **языком разметки** так же, как HTML
 - разработан для **хранения и транспортировки данных**
 - разработан, чтобы **быть как человеко- и машиночитаемым.**
- XML и HTML были разработаны с различными целями:
 - XML - для передачи данных
 - HTML – для отображения данных

Конструкции языка

Элементы данных

- Элемент - это структурная единица XML- документа.
- Любой непустой элемент должен состоять из начального, конечного тэгов и данных, между ними.
- Пример:
`<flower>rose</flower>`
`<city>Novosibirsk</city>`
- Пустые элементы:
 - если элемент не содержит данных, он называется пустым
 - пустые элементы в HTML – тэги `
`, `<hr>`, ``...
 - начальный и конечные тэги пустого элемента объединяется в один – например, `<empty/>`



Комментарии

- Комментариями является **любая область данных**, заключенная **между** последовательностями символов **<!-- и -->**
- Комментарии пропускаются анализатором и поэтому при разборе структуры документа в качестве значащей информации не рассматриваются.

Атрибуты

- Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то имеется возможность использовать атрибуты элемента.
- Атрибут - это пара "название" = "значение", которую надо задавать при определении элемента в начальном тэге.
- Пример:
`<color RGB="true">#ff08ff</color>`
`<color RGB="false">white</color>`
`<author id=0>Ivan</author>`

Директивы анализатора

- инструкции, предназначенные для анализаторов языка, описываются в XML документе при помощи специальных тэгов - `<? и ?>`.
- программа клиента использует эти инструкции для управления процессом разбора документа.
- Например - при определении типа документа
`<? Xml version="1.0"?>`

CDATA

- Чтобы задать **область документа**, которую при разборе анализатор будет **рассматривать как простой текст**, игнорируя любые инструкции и специальные символы, **но**, в отличии от комментариев, **иметь возможность использовать их в приложении**, необходимо использовать тэги **<![CDATA[и]]>**.
- Внутри можно помещать информацию, которая может понадобится программе-клиенту для выполнения каких-либо действий (например, инструкции JavaScript).

```
<![CDATA[<myElement>Содержимое элемента XML</myElement>]]>
```

Синтаксические правила XML

XML-документы должны иметь корневой элемент

XML - документы должны содержать один **корневой** элемент , который является **parent** всех остальных элементов:

```
<root>  
  <child>  
    <subchild>.....</subchild>  
  </child>  
</root>
```

Синтаксические правила XML

XML Пролог

Эта линия называется XML **пролог**:

```
<?xml version="1.0" encoding="UTF-8 " ?>
```

- XML пролог является необязательным.
- Должен быть на первом месте в документе.
- UTF-8 кодировка - по умолчанию для XML-документов.

Синтаксические правила XML

Все XML-элементы должны иметь
закрывающий тег

```
<p>This is a paragraph.</p>  
<br />
```

Синтаксические правила XML

XML-теги чувствительны к регистру

XML-теги чувствительны к регистру. Тег `<Letter>` отличается от тега `<letter>` .

```
<Message>This is incorrect</message>  
<message>This is correct</message>
```

Синтаксические правила XML

XML значения атрибутов должны быть заключены в кавычки

XML элементы могут иметь атрибуты в пар имя / значение, как и в HTML.

В XML значения атрибутов всегда должны быть заключены в кавычки.

НЕПРАВИЛЬНО:

```
<note date=12/11/2007>  
  <to>Tove</to>  
  <from>Jani</from>  
</note>
```

ВЕРНЫЙ:

```
<note date="12/11/2007">  
  <to>Tove</to>  
  <from>Jani</from>  
</note>
```


Синтаксические правила XML

Entity Ссылки

Некоторые символы имеют специальное значение в XML.

Если поместить символ , как "<" внутри элемента XML, он будет генерировать ошибку , потому что анализатор интерпретирует его как начало нового элемента.

Это вызовет ошибку XML:

```
<message>salary < 1000</message>
```

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Синтаксические правила XML

- Вся **информация**, располагающаяся **между начальным и конечными тэгами**, рассматривается в XML как **данные** и **поэтому учитываются все символы форматирования** (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML)

XML:	Hello Tove
HTML:	Hello Tove

Правила создания XML-документа

- Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов: **если элемент начинается внутри другого элемента, он должен и заканчиваться внутри этого элемента.**
- Вся **информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные и поэтому учитываются все символы форматирования** (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML)

C# и XML

Пространство имен System.XML

Класс	Описание
XmlDocument	Предоставляет XML-документ. Обеспечивает древовидное представление документа XML в памяти с возможностью навигации и редактирования.
XmlReader	Предоставляет средство чтения, обеспечивающее быстрый прямой доступ (без кэширования) к данным XML.
XmlWriter	Представляет средство записи, обеспечивающее быстрый прямой способ (без кэширования) создания потоков файлов, содержащих XML-данные.
XmlTextReader	Обеспечивает быстрый однонаправленный потоковый доступ к xml-данным.
XmlTextWriter	Обеспечивает быструю однонаправленную генерацию потоков xml.
XmlNode	Предоставляет отдельный узел в XML-документе.

Способы создания XML-документа

- **С использованием объектной модели документа** (*Document Object Model* - **DOM**)
 - позволяет легко обращаться ко всем узлам, но занимает много ресурсов и работает медленно
- **С использованием простого API для XML** (Simple API for XML – **SAX**)
 - вывод элементов происходит быстро, но требует больше усилий при сложных моделях данных

Построение XML-документа

```
string pathToXml = "test.xml";

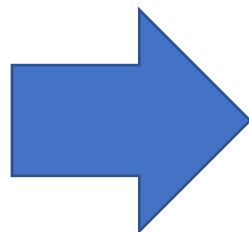
XmlTextWriter xtw = new XmlTextWriter(pathToXml, Encoding.UTF8);
xtw.WriteStartDocument();
xtw.WriteStartElement("books");

xtw.WriteStartElement("Книга");
xtw.WriteAttributeString("Год_издания", "2011");

xtw.WriteStartElement("Заголовок");
xtw.WriteString("Три мушкетера");
xtw.WriteEndElement();

xtw.WriteStartElement("Автор");
xtw.WriteString("Александр Дюма");
xtw.WriteEndElement();

xtw.WriteEndElement();
xtw.Close();
```



```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <Книга Год_издания="2011">
    <Заголовок>Три мушкетера</Заголовок>
    <Автор>Александр Дюма</Автор>
  </Книга>
</books>
```

LINQ и обработка XML-данных

LINQ — это название набора технологий, основанных на интеграции возможностей запроса непосредственно в язык C#.

Благодаря LINQ **запрос** теперь **является одним из основных структурных элементов языка**, подобно классам, методам, событиям и т. д.

LINQ позволяет формировать запросы для любого **LINQ-совместимого** источника данных.

Источник данных — массив, список, класс XML DOM или удаленный источник данных...

Что такое LINQ

Разновидности LINQ

- *LINQ to Objects*. Эта разновидность позволяет применять запросы LINQ к массивам и коллекциям.
- *LINQ to XML*. Эта разновидность позволяет применять LINQ для манипулирования и опроса документов XML.
- *LINQ to DataSet*. Эта разновидность позволяет применять запросы LINQ к объектам DataSet из ADO.NET.
- *LINQ to Entities*. Эта разновидность позволяет применять запросы LINQ внутри API-интерфейса ADO.NET Entity Framework (EF).
- *Parallel LINQ* (он же *PLINQ*). Эта разновидность позволяет выполнять параллельную обработку данных, возвращенных запросом LINQ.

Запрос LINQ

- Основными единицами данных в LINQ являются последовательности и элементы.
 - Последовательность – произвольный объект, реализующий интерфейс `IEnumerable`.
 - Элемент – это член последовательности.
- Запрос LINQ – выражение, преобразующее последовательности с помощью одной или нескольких операций запроса.
 - Простейший запрос состоит из входной последовательности и одной операции.

Запрос LINQ

Последовательность

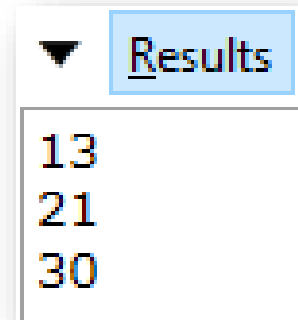
Элементы

```
int[] nums = {13, 21, 30, 3};  
IEnumerable<int> res = Enumerable.Where(nums, n=>n>10);  
foreach(int n in res) Console.WriteLine(n);
```

Класс Enumerable из пространства имен *System.Linq* содержит около 50 операций запроса. Все они реализованы как статические методы расширения и называются **стандартными операциями запроса**.

Запрос LINQ

```
int[] nums = {13, 21, 30, 3};  
var res = nums.Where(n=>n>10);  
foreach(int n in res) Console.WriteLine(n);
```



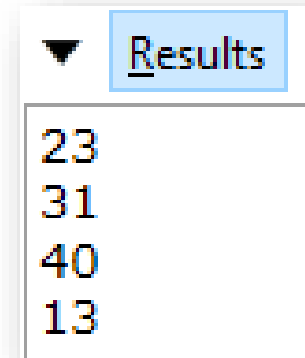
Упрощение:

Стандартные операции запроса реализованы как методы расширения, и *операцию*

Where можно вызывать непосредственно *из объекта **nums*** так, как будто он является методом экземпляра

Проецирование Select

```
int[] nums = {13, 21, 30, 3};  
var res = nums.Select(n=>n+10);  
foreach(int n in res) Console.WriteLine(n);
```



▼ Results

23
31
40
13

Метод **Select преобразовывает** (проецирует) каждый элемент входной последовательности с помощью заданного лямбда-выражения

Проецирование в анонимный тип

```
var res = names.Select(n => new {  
    Name = n,  
    Length = n.Length  
});  
foreach(var n in res) Console.WriteLine(n);
```

Results λ SQL IL Format Export [Activate Autocompletion](#)

▲ Ø ▶	
{ Name = Иван, Length = 4 }	
Name	Иван
Length	4

▲ Ø ▶	
{ Name = Петр, Length = 4 }	
Name	Петр
Length	4

▲ Ø ▶	
{ Name = Елизавета, Length = 9 }	
Name	Елизавета
Length	9

Операции LINQ

Разделения на части

- Исходный порядок элементов – важен для LINQ
- Пример – TAKE выводит первые X элементов, отбрасывая остальные:
- `int[] nums = {13, 21, 30, 3};`
- `var res = nums.Take(2);`

Агрегации

- Возвращают скалярное значение (обычно – числового типа)
- Пример – COUNT, получает необязательный аргумент, означающий, следует ли включать заданный элемент:
- `int[] nums = {13, 21, 30, 3};`
- `int res = nums.Count(n=>n>10);`

Отложенное выполнение

- **Свойство** многих **запросов** – они **выполняются не в момент создания**, а при перечислении.
- Это называется **отложенным выполнением**.
- Все стандартные операции поддерживают отложенное выполнение, за **исключением**:
 - **Операций, возвращающих отдельный элемент или скаляр** (операции над элементами, агрегации, квантификаторы)
 - **Операций преобразования** (ToArray...)

```
int[] nums = {13,21,30,3};  
int res1 = nums.Count(n=>n>10);  
var res2 = nums.Select(n=>n*2);  
nums[3]=46;
```

```
Console.WriteLine(res1);  
foreach(int n in res2)  
    Console.WriteLine(n);
```

▼ Results λ SQL IL Format ▼ Export ▼ [Activate Autocompletion](#)

```
3  
26  
42  
60  
92
```

Категории операций запроса

Операции фильтрации	Возвращают подмножество элементов, удовлетворяющих указанному условию
Операции проецирования	Преобразуют каждый элемент с помощью лямбда-функции, необязательно расширяя подпоследовательности
Операции объединения	Смешивают элементы двух последовательностей, используя эффективную стратегию поиска
Операции упорядочения	Возвращают последовательность, упорядоченную в другом порядке
Операции группирования	Группируют последовательность в подпоследовательности
Операции над множествами	Принимают две однотипные последовательности и возвращают их общие элементы, сумму или разность
Операции над элементами	Извлекают элемент из последовательности
Операции агрегации	Выполняют вычисление над последовательностью, возвращая скалярное значение (как правило, число)
Квантификаторы	Выполняют вычисление над последовательностью, возвращая значение <code>true/false</code>
Операция преобразования: импорт	Преобразует необобщенную последовательность в обобщенную последовательность, допускающую запросы
Операции преобразования: экспорт	Преобразует последовательность в массив, список, словарь или просматривает ее, вынуждая немедленное вычисление
Операция генерирования	Производит простую последовательность

Операции фильтрации

Метод	Описание
Where	Возвращает подмножество элементов, удовлетворяющих заданному условию
Take	Возвращает первые <i>x</i> элементов и отбрасывает остальные
Skip	Игнорирует первые <i>x</i> элементов и возвращает остальные
TakeWhile	Извлекает элементы из входной последовательности, пока заданный предикат равен <code>true</code>
SkipWhile	Игнорирует элементы из входной последовательности, пока заданный предикат равен <code>true</code> , а затем выводит остальные
Distinct	Возвращает коллекцию, из которой удалены дубликаты

Операции проецирования и объединения

Метод	Описание
Select	Преобразует каждый входной элемент с помощью заданной лямбда-функции
SelectMany	Преобразует каждый входной элемент, а затем выравнивает и конкатенирует результирующие последовательности

Метод	Описание
Join	Применяет стратегию последовательного поиска для сравнения элементов из двух коллекций, выдавая выровненное результирующее множество
GroupJoin	Действует так же, как и предыдущая операция, но выдает <i>иерархическое</i> результирующее множество
Zip	Перечисляет две последовательности за один этап, возвращая последовательность, применяющую функцию к каждой паре элементов

Операции упорядочения, группирования и агрегации

Метод	Описание
OrderBy, ThenBy	Возвращает элементы, упорядоченные в порядке возрастания
OrderByDescending, ThenByDescending	Возвращает элементы, упорядоченные в порядке убывания
Reverse	Возвращает элементы, упорядоченные в обратном порядке

Метод	Описание
GroupBy	Группирует последовательность в подпоследовательности

Метод	Описание
Count, LongCount	Возвращает общее количество элементов во входной последовательности или количество элементов, удовлетворяющих заданному предикату
Min, Max	Возвращает наименьший или наибольший элементы в последовательности
Sum, Average	Вычисляет сумму или среднее значение элементов последовательности
Aggregate	Выполняет пользовательскую агрегацию

Операции над множествами и элементами

Метод	Описание
Concat	Конкатенирует две последовательности
Union	Конкатенирует две последовательности, удаляя дубликаты
Intersect	Возвращает элементы, существующие в обеих последовательностях
Except	Возвращает элементы, существующие в первой, но не во второй последовательности

Метод	Описание
First, FirstOrDefault	Возвращает первый элемент последовательности или первый элемент, удовлетворяющий заданному предикату
Last, LastOrDefault	Возвращает последний элемент последовательности или последний элемент, удовлетворяющий заданному предикату
Single, SingleOrDefault	Эквивалент операций First/FirstOrDefault, но генерирует исключение, если существует несколько совпадений
ElementAt, ElementAtOrDefault	Возвращает элемент в заданной последовательности
DefaultIfEmpty	Возвращает последовательность, содержащую одно значение, равное null или default(TSource), если последовательность пустая

Квантификаторы и операции преобразования

Метод

`Contains`

`Any`

`All`

`SequenceEqual`

Описание

Возвращает значение `true`, если входная последовательность содержит заданный элемент

Возвращает значение `true`, если все элементы удовлетворяют заданному предикату

Возвращает значение `true`, если все элементы удовлетворяют заданному предикату

Возвращает значение `true`, если вторая последовательность содержит элементы, идентичные элементам входной последовательности

Метод

`ToArray`

`ToList`

Описание

Преобразует интерфейс `IEnumerable<T>` в класс `T[]`

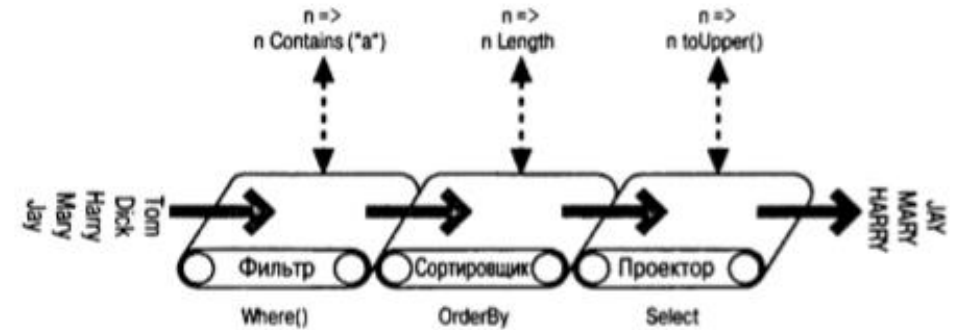
Преобразует интерфейс `IEnumerable<T>` в класс `List<T>`

```
int[] nums = {13, 21, 30, 3};  
var res = nums  
    .Where(n => n>10)  
    .Select(n => n*2)  
    .Take(2);  
  
res.Dump();
```

▼ Results λ SQL IL Format ▼ Export ▼ [Activate Autocompletion](#)

▲ IEnumerable<Int32> (2 items) ▶

26
42



Создание цепочек запросов

2 синтаксиса Linq

ПОТОКОВЫЙ СИНТАКСИС

```
int[] nums = {12, 21, 34, 8, 45};  
var res = nums.Where(n => n%2==1);
```

Выражения запроса

```
int[] nums = {12, 21, 34, 8, 45};  
var res = from n in nums where (n%2==1)  
select n;
```

Начинается с FROM и
заканчивается SELECT
или GROUP

Выражения запроса поддерживают лишь небольшое подмножество операций запроса, а именно:

Where, Select, SelectMany
OrderBy, ThenBy, OrderByDescending, ThenByDescending
GroupBy, Join, GroupJoin

Различные операции запросов LINQ

Операции запросов	Назначение
from, in	Используются для определения основы любого выражения LINQ, позволяющего извлечь подмножество данных из подходящего контейнера.
where	Используется для определения ограничения, в соответствии с которым должны быть извлечены элементы из контейнера.
select	Используется для выбора последовательности из контейнера.
join, on, equals, into	Выполняет соединения на основе указанного ключа. Напомним, что эти "соединения" не имеют никакого отношения к реляционной базе данных.
orderby, ascending, descending	Позволяет результирующему подмножеству быть упорядоченным по возрастанию или убыванию.
group, by	Порождает подмножество с данными, группированными по указанному значению.

Компилятор обрабатывает выражения запросов, транслируя их в потоковый синтаксис.


Комбинирование типов

```
string [] names = {"Иван", "Петр", "Николай", "Сергей"};

var query1 = from n in names
where n.Length==names.Min(m=>m.Length)
select n;

var query2 = names.Where(n=>n.Length==names.Min(m=>m.Length));

query1.Dump();
query2.Dump();
```



< Results λ SQL IL Format Export Activate Autocompletion

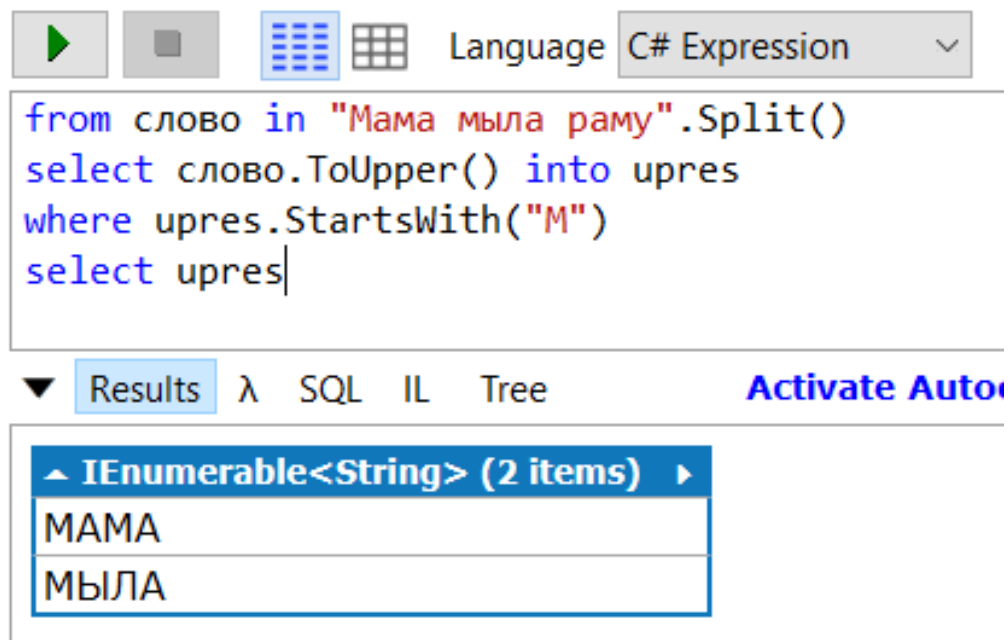
↑ IEnumerable<String> (2 items) ▸

Иван
Петр

↑ IEnumerable<String> (2 items) ▸

Иван
Петр

Продолжение запросов



The screenshot shows a C# Expression Builder window. At the top, there are icons for running, undo, redo, and a grid view, along with a 'Language' dropdown set to 'C# Expression'. The main text area contains the following LINQ query:

```
from слово in "Мама мыла раму".Split()  
select слово.ToUpper() into upres  
where upres.StartsWith("М")  
select upres|
```

Below the text area, there are tabs for 'Results', 'SQL', 'IL', and 'Tree', with 'Results' currently selected. To the right of the tabs is a link that says 'Activate Auto...'. The 'Results' tab displays the output of the query:

IEnumerable<String> (2 items)
МАМА
МЫЛА

- Чтобы добавить разделы после SELECT Или GROUP, можно использовать ключевое слово INTO

- Или:

```
"Мама мыла раму".Split()  
.Select (c=>c.ToUpper())  
.Where (upres => upres.StartsWith("М"))
```

LINQ to XML

- LINQ to XML — это оснащенный средствами LINQ и встроенный в память программный интерфейс XML, позволяющий работать с XML-файлами внутри языков программирования .NET Framework.
- Важнейшее достоинство LINQ to XML состоит в его интеграции с LINQ. Эта интеграция дает возможность создавать запросы к загруженному в память XML-документу с целью получения коллекций элементов и атрибутов.
- Классы:
 - **Класс XDocument**
 - **Класс XElement**
 - **Класс XAttribute**
 - **Класс XmlNode**
 - ...

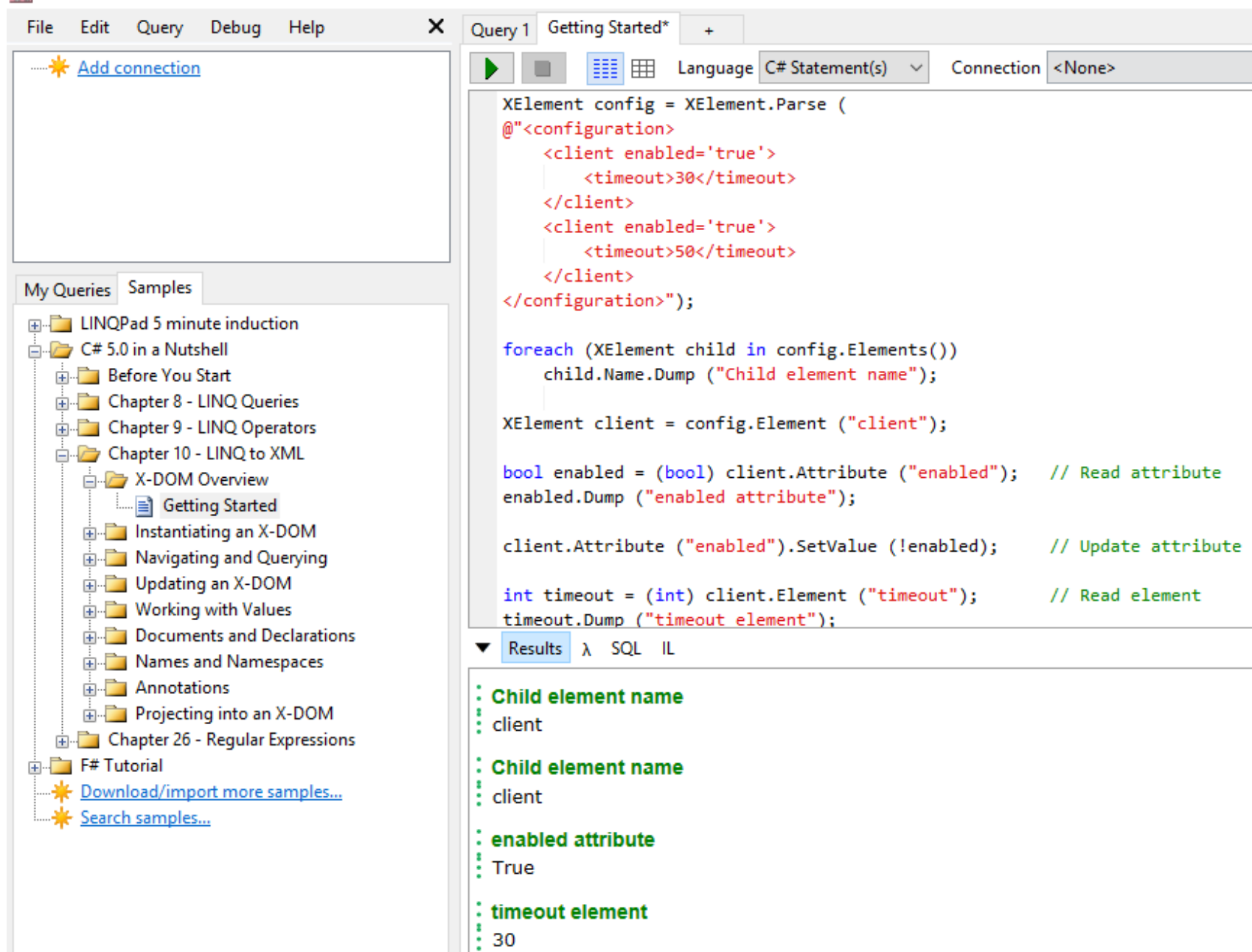
Члены пространства имен System.Xml.Linq

Название	Описание
XAttribute	XML-атрибут заданного узла
XDeclaration	Открывающее объявление документа
XDocument	Целиком весь документ
XElement	Заданный элемент внутри документа (включая корневой)

- System.Xml.Linq
 - Extensions
 - LoadOptions
 - ReaderOptions
 - SaveOptions
 - XAttribute
 - XCDATA
 - XComment
 - XContainer
 - XDeclaration
 - XDocument
 - XDocumentType
 - XElement
 - XName
 - XNamespace
 - XNode
 - XNodeDocumentOrderComparer
 - XNodeEqualityComparer
 - XObject
 - XObjectChange
 - XObjectChangeEventArgs
 - XProcessingInstruction
 - XStreamingElement
 - XText

LinqPad

- Программная утилита для Microsoft .NET, разработанная для интерактивного написания и тестирования запросов к БД SQL и другим источникам данных, таким как OData или WCF Data Services с использованием LINQ. Википедия



Создание XML

```
XDocument data = XDocument.Parse(@"<?xml version='1.0' encoding='UTF-8'?>
<root>
  <a id='1'><b>b1</b><c>c1</c></a>
  <a id='2'><b>b2</b><c><e>EEE</e></c></a>
</root>");

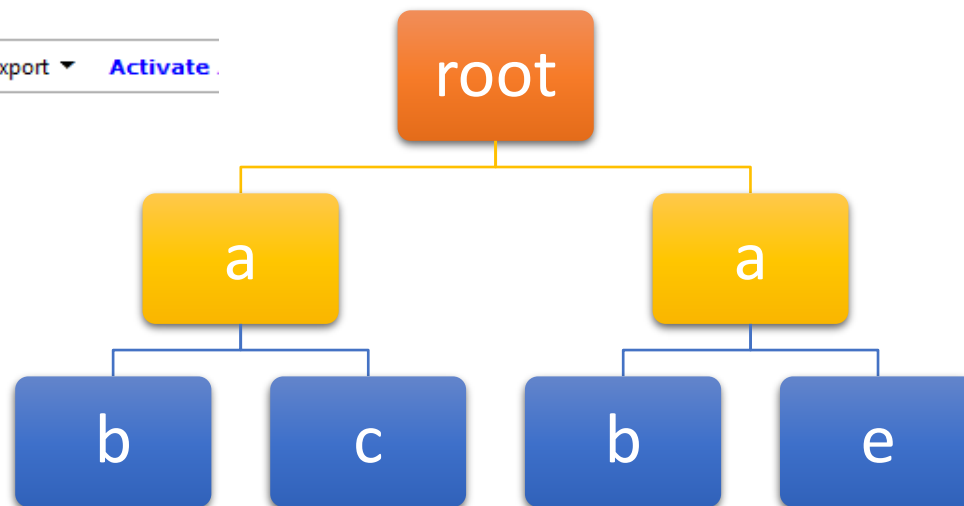
XDocument data2 = new XDocument(new XDeclaration("1.0", "UTF-8", ""),
    new XElement("root",
        new XElement("a", new XAttribute("id", "1"),
            new XElement("b", "b1"), new XElement("c", "c1")),
        new XElement("a", new XAttribute("id", "2"),
            new XElement("b", "b2"), new XElement("c", new XElement("e", "EEE")))));

data.Elements().Dump();
```

Results SQL IL Tree

Format Export Activate

```
IEnumerable<XElement> (1 item)
<root>
  <a id="1">
    <b>b1</b>
    <c>c1</c>
  </a>
  <a id="2">
    <b>b2</b>
    <c>
      <e>EEE</e>
    </c>
  </a>
</root>
```



Класс Extensions.

Осевые методы LINQ to XML

Название	Описание
Descendants	Может быть вызвана на последовательности элементов или документов и возвращает последовательность элементов, содержащую ВСЕ элементы-потомки каждого исходного элемента или документа.
Elements	<p>Может быть вызвана на последовательности элементов или документов и возвращает последовательность, содержащую дочерние элементы каждого исходного элемента или документа.</p> <p><i>Эта операция отличается от Descendants, поскольку операция Elements возвращает только непосредственные дочерние элементы каждого из элементов входной последовательности, в то время как операция Descendants рекурсивно возвращает все дочерние элементы до достижения конца каждого дерева.</i></p>

Полный список
Ancestors<T>(IEnumerable<T>)
Ancestors<T>(IEnumerable<T>, XName)
AncestorsAndSelf(IEnumerable<XElement>)
AncestorsAndSelf(IEnumerable<XElement>, XName)
Attributes(IEnumerable<XElement>)
Attributes(IEnumerable<XElement>, XName)
DescendantNodes<T>(IEnumerable<T>)
DescendantNodesAndSelf(IEnumerable<XElement>)
Descendants<T>(IEnumerable<T>)
Descendants<T>(IEnumerable<T>, XName)
DescendantsAndSelf(IEnumerable<XElement>)
DescendantsAndSelf(IEnumerable<XElement>, XName)
Elements<T>(IEnumerable<T>)
Elements<T>(IEnumerable<T>, XName)
InDocumentOrder<T>(IEnumerable<T>)
Nodes<T>(IEnumerable<T>)
Remove(IEnumerable<XAttribute>)
Remove<T>(IEnumerable<T>)

Элементы XML

```
XDocument data = XDocument.Parse(@"<?xml version='1.0' encoding='UTF-8'?>
<root>
<a>a1<b>b1</b><c>c1</c></a>
<a>a2<b>b2</b><e>e1</e></a>
</root>");

var res = data.Descendants("a").Elements().Dump();

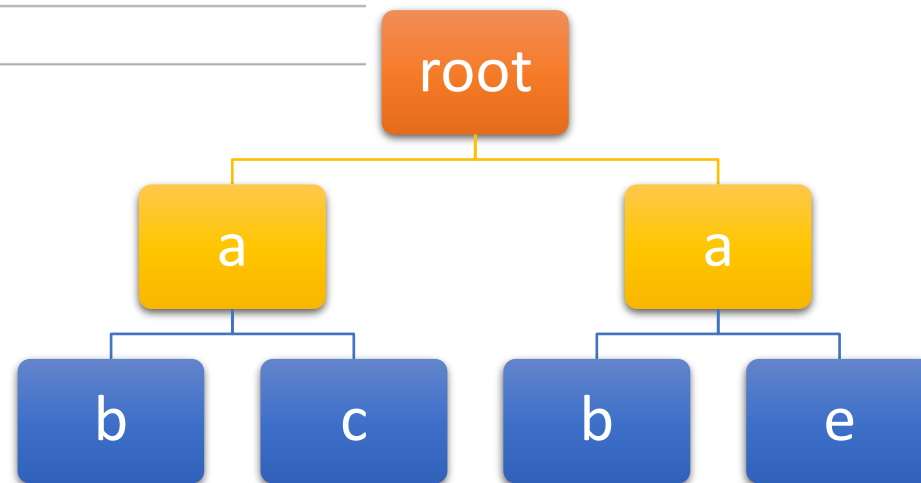
foreach (XElement n in res)
    Console.WriteLine(" {0} - {1} ",n.Name, n.Value);
```

▼ Results λ SQL IL Tree

▲ IEnumerable<XElement> (4 items) ▶

b1
<c>c1</c>
b2
<e>e1</e>

b - b1
c - c1
b - b2
e - e1



Descendants() & Elements()

```
XDocument data = XDocument.Parse(@"<?xml version='1.0' encoding='UTF-8'?>
<root>
  <a id='1'><b>b1</b><c>c1</c></a>
  <a id='2'><b>b2</b><c><e>EEE</e></c></a>
</root>");

data.Dump();
data.Descendants("b").Dump();
data.Elements("b").Dump();
```

▼ Results λ SQL IL Tree

```
<root>
  <a id="1">
    <b>b1</b>
    <c>c1</c>
  </a>
  <a id="2">
    <b>b2</b>
    <c>
      <e>EEE</e>
    </c>
  </a>
</root>
```

▲ IEnumerable<XElement> (2 items) ▶

b1
b2

(0 items)

Данные

```
<NewDataSet>
  <Table>
    <AirportCode>LKE</AirportCode>
    <CityOrAirportName>SEATTLE/TACOMA LAKE UNION</CityOrAirportName>
    <Country>United States</Country>
    <CountryAbbrviation>US</CountryAbbrviation>
    <CountryCode>93</CountryCode>
    <GMTOffset>8</GMTOffset>
    <RunwayLengthFeet>0</RunwayLengthFeet>
    <RunwayElevationFeet>14</RunwayElevationFeet>
    <LatitudeDegree>47</LatitudeDegree>
    <LatitudeMinute>30</LatitudeMinute>
    <LatitudeSecond>0</LatitudeSecond>
    <LatitudeNpeerS>N</LatitudeNpeerS>
    <LongitudeDegree>122</LongitudeDegree>
    <LongitudeMinute>20</LongitudeMinute>
    <LongitudeSeconds>0</LongitudeSeconds>
    <LongitudeEperW>W</LongitudeEperW>
  </Table>
  <Table>
    <AirportCode>LKE</AirportCode>
    <CityOrAirportName>SEATTLE/TACOMA LAKE UNION</CityOrAirportName>
    <Country>United States</Country>
    <CountryAbbrviation>US</CountryAbbrviation>
    <CountryCode>93</CountryCode>
    <GMTOffset>8</GMTOffset>
    <RunwayLengthFeet>0</RunwayLengthFeet>
    <RunwayElevationFeet>14</RunwayElevationFeet>
    <LatitudeDegree>47</LatitudeDegree>
```

Linq To XML – пример использования

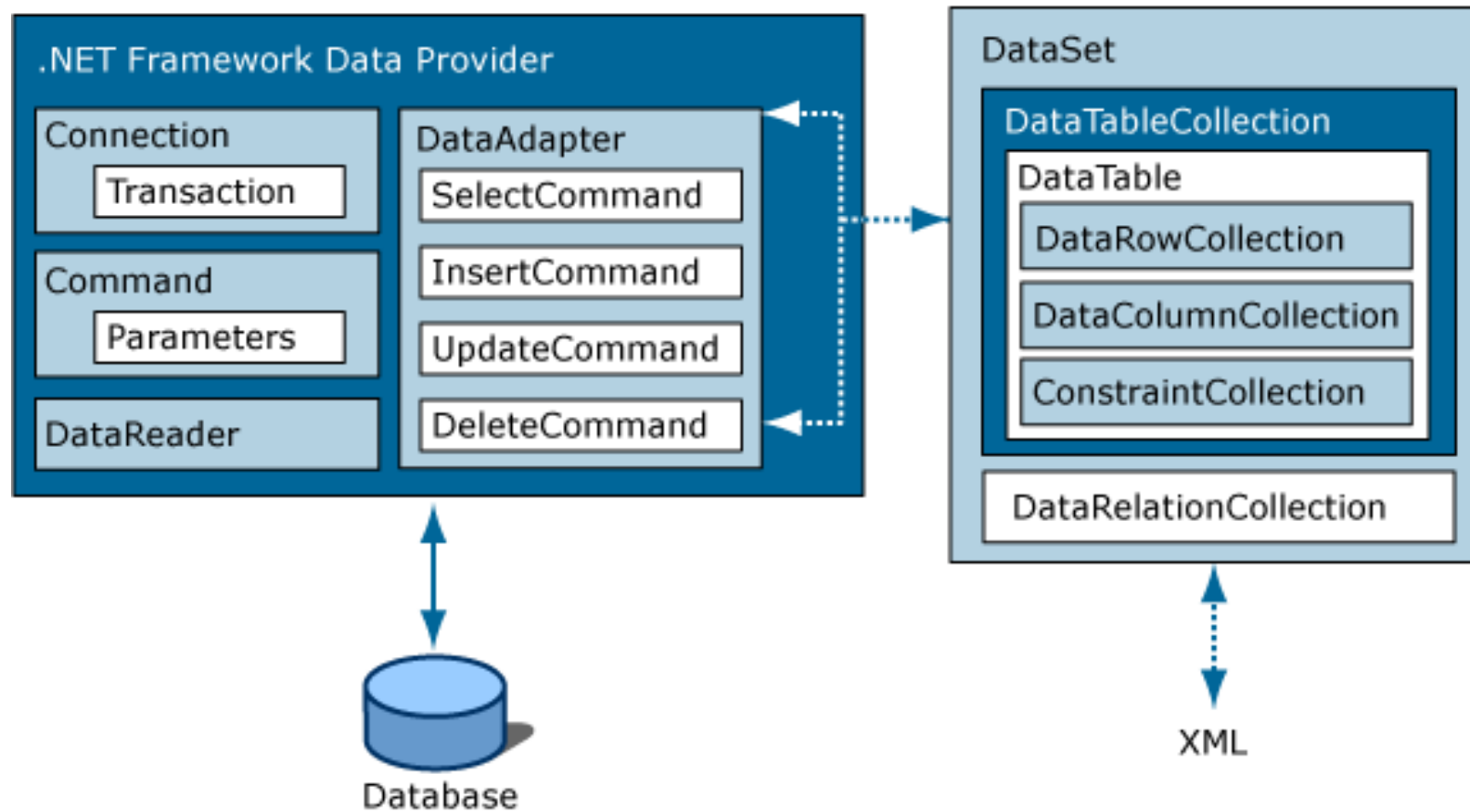
```
var res = data.Elements()  
    .OrderBy(n=>Int32.Parse(n.Element("RunwayLengthFeet").Value))  
    .Select (n=> new {  
        Name = n.Element("CityOrAirportName").Value,  
        RunwayLength = n.Element("RunwayLengthFeet").Value})  
    .Take(10);
```

Из XML в DataGridView

```
string xlist =  
@"<table>  
  <tr><name>Вася</name><age>100</age></tr>  
  <tr><name>Петя</name><age>56</age></tr>  
  <tr><name>Люся</name><age>45</age></tr>  
  <tr><name>Валя</name><age>12</age></tr>  
  <tr><name>Инокентий</name><age>120</age></tr>  
  <tr><name>Коля</name><age>103</age></tr>  
</table>";  
  
var table = System.Xml.Linq.XElement.Parse(xlist);  
var filter = from x in table.Elements()  
              select new  
              {  
                  Имя = x.Element("name").Value,  
                  Возраст = x.Element("age").Value  
              };  
dataGridView1.DataSource = filter.ToArray();
```

ADO и XML

Архитектура ADO.NET



XML и ADO.NET

- **ADO.NET использует возможности XML для предоставления доступа к данным без сетевого соединения.**
 - Разработка ADO.NET производилась одновременно с XML-классами платформы .NET Framework; оба они являются компонентами одной архитектуры.
- **Объект DataSet может заполняться данными из XML-источника, будь то файл или поток XML-данных.**
- **Объект DataSet может быть записан в виде XML-кода, независимо от источника данных, находящихся в DataSet.**
 - DataSet – оптимальное решение для удаленного взаимодействия с веб-службой XML.

Загрузка DataSet из XML

- Заполнить DataSet из XML - метод ReadXml (объекта DataSet).
- Аргументы метода: источник XML и **дополнительный аргумент – XmlReadMode**.
- Если DataSet уже содержит данные, новые данные из XML-кода добавляются к данным, находящимся в наборе данных DataSet.
- При совпадении первичного ключа никаких данных не переносится.
- Для перезаписи существующих данных нужно создать новый DataSet с помощью ReadXml, а затем с помощью метода Merge, выполнить слияние DataSet с существующим набором данных.

```
DataSet dataSet = new DataSet();  
dataSet.ReadXml("input.xml", XmlReadMode.ReadSchema);
```

Параметры для аргумента XmlReadMode.

Параметр	Описание
Auto	Значение по умолчанию. Анализирует XML и выбирает наиболее подходящий параметр. Например, если набор данных DataSet содержит схему или XML-код содержит встроенную схему, используется значение ReadSchema.
ReadSchema	Считывает встроенную в XML документ схему и загружает данные и схему. Если набор данных DataSet уже содержит схему, новые таблицы добавляются из встроенной схемы в существующую в наборе данных DataSet. Возможность изменить схему существующей таблицы с помощью метода XmlReadMode.ReadSchema отсутствует. Если набор данных DataSet не содержит схему, а также отсутствует встроенная схема, то данные не считываются. Встроенная схема может быть определена с помощью схемы на языке XSD.
IgnoreSchema	Не учитывает встроенную в документ схему и загружает данные в существующую схему DataSet. Любые данные, не совпадающие с существующей схемой, удаляются. Если схема не существует в наборе данных DataSet, данные не загружаются.
InferSchema	Не учитывает встроенную схему, а формирует схему на основе структуры данных XML-документа. Если набор данных DataSet уже содержит схему, текущая схема расширяется путем добавления столбцов в существующие таблицы.

Слияние данных из XML

- Если набор данных DataSet уже содержит данные, новые данные из XML-кода добавляются к данным, находящимся в наборе данных DataSet.
- Метод ReadXml не переносит слиянием из XML-кода в DataSet данных из строк с совпадающими первичными ключами.
- Для перезаписи существующих данных в строках новыми данными из XML-кода следует использовать метод ReadXml для создания нового набора данных DataSet, а затем использовать метод Merge, чтобы выполнить слияние набора данных DataSet с существующим набором данных DataSet.

Вывод реляционной структуры DataSet из XML

- Реляционная структура (или схема) набора данных DataSet состоит из таблиц, столбцов, ограничений и связей.
- При загрузке DataSet из кода XML схема может быть определена заранее или создана, либо явно, либо с помощью вывода, на основании загружаемого кода XML.

```
DataSet dataSet = new DataSet();  
dataSet.ReadXmlSchema("schema.xsd");
```

Отображение XML данных на реляционные таблицы

- **Сводка правил:**

- Элементы с атрибутами выводятся как таблицы.
- Элементы, имеющие дочерние элементы, выводятся как таблицы.
- Повторяющиеся элементы выводятся как одна таблица.
- Элемент документа (корневой элемент) не имеющий ни атрибутов, ни дочерних элементов, выводится как DataSet.
- Атрибуты выводятся как столбцы.
- Элементы без атрибутов и дочерних элементов выводятся как столбцы.
- Для элементов, которые выводятся как таблицы, вложенные в другие элементы, которые также выводятся как таблицы, между двумя таблицами создается вложенный элемент DataRelation.
 - Новый столбец первичного ключа с именем TableName_Id добавляется к обеим таблицам и используется элементом DataRelation.
 - Между двумя таблицами создается элемент ForeignKeyConstraint с использованием столбца TableName_Id.
- Для элементов, которые выводятся как таблицы и содержат текст, но не имеют дочерних элементов, создается новый столбец с именем TableName_Text для текста каждого из таких элементов.
- Если элемент выводится как таблица и имеет текст, но при этом имеет дочерние элементы, текст пропускается.

Таблицы

- Представляются в XML как:
 - **Элементы с атрибутами.**
 - Элементы с дочерними элементами.
 - Повторяющиеся элементы.

```
<DocumentElement>  
  <Element1 attr1="value1"/>  
  <Element1 attr1="value2">Text1</Element1>  
</DocumentElement>
```



DataSet: DocumentElement

Table: Element1

attr1	Element1_Text
value1	
value2	Text1

Таблицы

- Представляются в XML как:
 - Элементы с атрибутами.
 - **Элементы с дочерними элементами.**
 - Повторяющиеся элементы.

```
<DocumentElement>
  <Element1>
    <ChildElement1>Text1</ChildElement1>
  </Element1>
</DocumentElement>
```



DataSet: DocumentElement

Table: Element1

ChildElement1
Text1

```
<DocumentElement>
  <Element1>Text1</Element1>
  <Element2>Text2</Element2>
</DocumentElement>
```

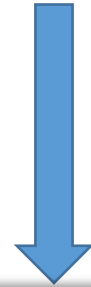


DataSet: NewDataSet

Table: DocumentElement

Element1	Element2
Text1	Text2

```
<DocumentElement>
  <Element1 attr1="value1" attr2="value2"/>
</DocumentElement>
```



DataSet: DocumentElement

Table: Element1

attr1	attr2
value1	value2

Таблицы

- Представляются в XML как:
 - Элементы с атрибутами.
 - Элементы с дочерними элементами.
 - Повторяющиеся элементы.

```
<DocumentElement>  
  <Element1>Text1</Element1>  
  <Element1>Text2</Element1>  
</DocumentElement>
```



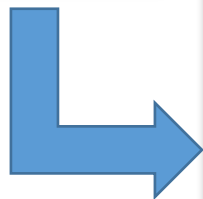
DataSet: DocumentElement

Table: Element1

Element1_Text
Text1
Text2

СВЯЗИ

```
<DocumentElement>
  <Element1>
    <ChildElement1 attr1="value1" attr2="value2"/>
    <ChildElement2>Text2</ChildElement2>
  </Element1>
</DocumentElement>
```



DataSet: DocumentElement

Table: Element1

Element1_Id	ChildElement2
0	Text2

Table: ChildElement1

attr1	attr2	Element1_Id
value1	value2	0

DataRelation: Element1_ChildElement1

ParentTable: Element1

ParentColumn: Element1_Id

ChildTable: ChildElement1

ChildColumn: Element1_Id

Запись содержимого DataSet в виде XML

- Записать XML-представление объекта DataSet можно вместе со схемой или без нее.
- Если информация схемы встраивается внутрь XML, она записывается на языке XSD.
- Схема содержит определения таблиц для DataSet, а также определения связей и ограничений.
- XML-представление для DataSet может быть записано в файл, в поток, в XmlWriter или в строку.
- Чтобы получить XML-представление для DataSet как строку, используется метод GetXml
- Чтобы записать DataSet в файл, поток или в XmlWriter, используется метод WriteXml.
- Первый параметр WriteXml – назначение XML-выхода.
 - Например, передается строка, содержащая имя файла
- Второй параметр – XmlWriteMode (как должен записываться XML-выход).

Параметры для XmlWriteMode.

Параметр XmlWriteMode	Описание
IgnoreSchema	Записывает текущее содержимое DataSet как XML-данные, без схемы XML. Это значение по умолчанию.
WriteSchema	Записывает текущее содержимое DataSet в виде XML-данных с реляционной структурой в виде встроенной схемы XML.

```
custDS.WriteXml("Customers.xml",  
XmlWriteMode.WriteSchema);
```