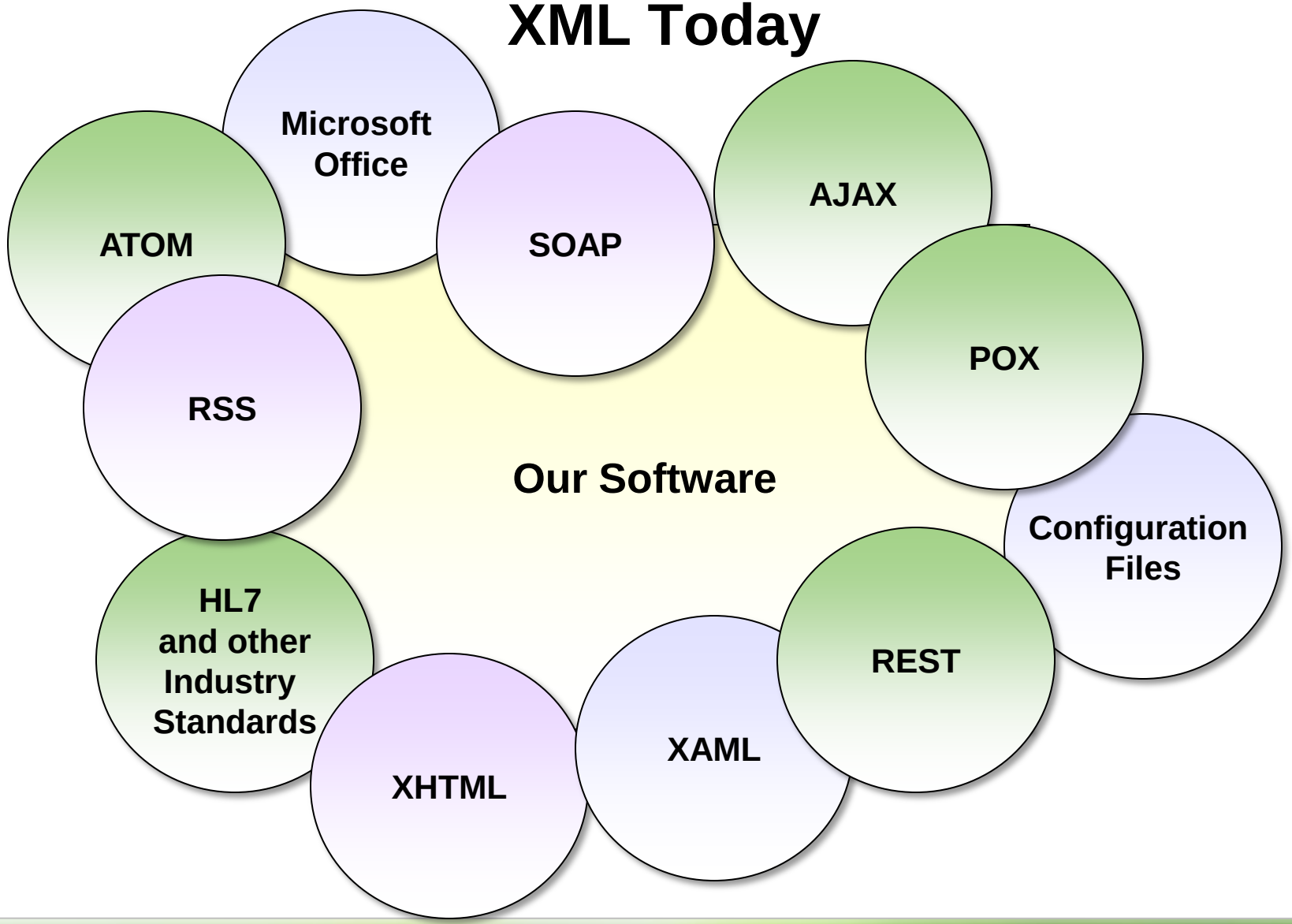
LINQ To XML

New Paradigms for Angled Brackets

Overview

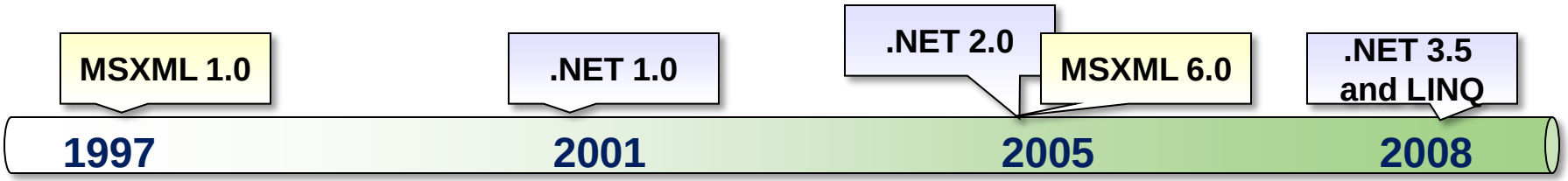
- **Why another XML API?**
- **Programming XML with LINQ to XML**
  - Loading, creating and updating
- **Querying XML**
  - Query expressions, operators, XPath and XSLT

XML Today


Microsoft Office
ATOM
SOAP
AJAX
POX
RSS
Our Software
Configuration Files
HL7 and other Industry Standards
REST
XAML
XHTML

Microsoft XML Support

- **MSXML**
  - 40+ releases over 8 years
  - XML DOM and streaming SAX APIs
  - XSLT, XPath, and XSD and XDR support
- **System.Xml**
  - 2 major releases over 5 years
  - XML DOM and XmlReader (pull) APIs
  - XPathDocument and cursor APIs



We Need Another API?

- **System.Xml is showing some signs of age**
  - Verbose, and lacks new language features
- **Individual technologies require time to learn**
  - XPath for querying, XSLT for transformations

```
XmlDocument document = new XmlDocument();
XmlElement employees = document.CreateElement("Employees");
document.AppendChild(employees);

XmlElement employee = document.CreateElement("Employee");
employee.InnerText = "Matt";
employees.AppendChild(employee);

employee = document.CreateElement("Employee");
employee.InnerText = "Dan";
employees.AppendChild(employee);
```

```
<?xml version="1.0"?>
<Employees>
  <Employee>Matt</Employee>
  <Employee>Dan</Employee>
</Employees>
```

New XML API Goals

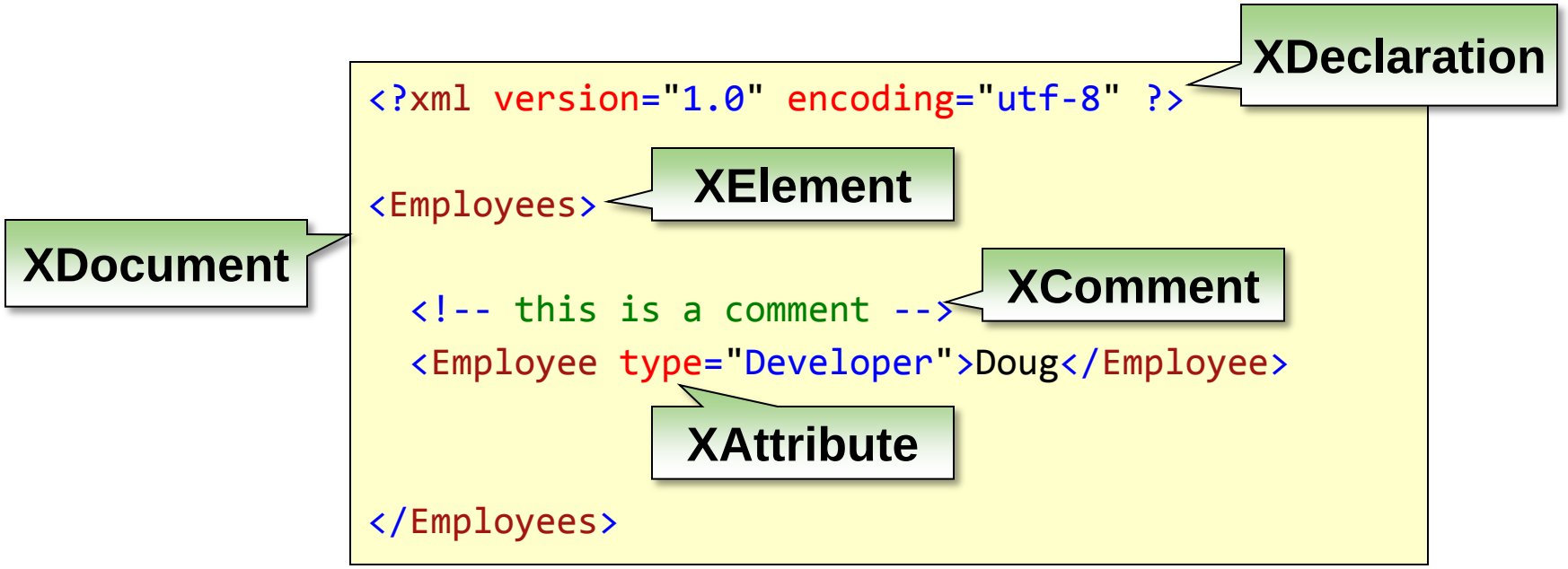
- **Work with Language Integrated Query**
  - Use the standard query operators
  - Add some new operators for XML
- **Provide a modern programming API**
  - Cleaner, faster, lighter
  - Take advantage of generics and nullable types
  - Innovate beyond the DOM APIs

```
XDocument document =
    new XDocument(
        new XElement("Employees",
            new XElement("Employee", "Matt"),
            new XElement("Employee", "Dan")
        )
    );
```

```
<?xml version="1.0"?>
<Employees>
  <Employee>Matt</Employee>
  <Employee>Dan</Employee>
</Employees>
```

System.Xml.Linq

- **The assembly *and* the namespace for LINQ to XML**
  - XDocument, XNode, XAttribute, XElement, XObject
  - Extension methods to facilitate queries
- **XElement is the heart of the API**
  - Provides the ability to load XML from URL, stream, or string



Creating XML

- **Functional construction**
  - A `params` object array used in many constructors
  - Conversions tend to “do the right thing”
  - Every parameter added to Nodes or Attributes properties
- **Context free**
  - No XDocument was used in the making of this XML
- **Deep cloning**

```
// functional construction
XElement xml = new XElement("Employees",
    new XComment(" this is a comment "),
    new XElement("Employee", new XAttribute(“Type", "Developer"), "Scott"),
    new XElement("Employee", new XAttribute(“Type", "Developer"), "Poonam"),
    new XElement("Employee", new XAttribute(“Type", "Sales"), "Andy"));
```

Saving XML

- **`ToString` on any XNode yields a string of XML**
  - With line breaks and formatting!
- **Save will write out a declaration by default**
- **TextWriter and XmlWriter support**

```
XElement xml = new XElement("Employees",
    new XElement("Employee", new XAttribute("type", "Developer"), "Scott"));

xml.Save("employees.xml");

Console.WriteLine(xml.ToString());
Console.WriteLine(xml.ToString(SaveOptions.DisableFormatting));
```

Loading XML

- Load XML from a file, URL, or XmlReader
- Parse XML from a string
- XmlReader is always behind the scenes

```
XDocument document = XDocument.Load("employees.xml");
XElement element =
    XElement.Load(“http://www.pluralsight.com/blogs/rss.aspx");

using (XmlReader reader = document.CreateReader())
{
    if (reader.Read())
        XNode node = XNode.ReadFrom(reader);
}

XElement inline = XElement.Parse("<Employees/>");
```

Reading XML

- **Explicit conversions perform value extraction**
- **Values stored as text**
  - Parsed on an as-needed basis
- **Support for nullable types**

```
XElement xml = XElement.Parse(
            "<Employee Type=\"Developer\">Scott</Employee>");

string name = (string)xml; // "Scott"
string type = (string)xml.Attribute("Type"); // "Developer"

double? salary = (double?)xml.Attribute("Salary"); // yields null

int age = (int)xml.Attribute("Age"); // exception!
```

Namespaces

- **Use XNamespace to encapsulate an XML namespace**
  - Overrides `operator +` to combine namespace and name
- **Alternative is to place namespace inside { and } delimiters**

```
XNamespace ns = "http://schemas.contonso.com/Employees";

XElement xml =
    new XElement(ns + "Employees",
        new XElement(ns + "Employee", "Aaron"),
        new XElement("{http://schemas.contonso.com/Employees}Employee",
                     "Bill"));
```

Prefixes

- **Prefixes in LINQ to XML only significant during output**
  - Can provide prefix hints

```
XNamespace xmlns = "http://schemas.contonso.com/Employees";
XNamespace ext = "http://schemas.contoso.com/EmployeeExtensions";

XElement xml = new XElement(xmlns + "Employees",
    new XAttribute(XNamespace.Xmlns + "ns1", ext),
    new XElement(ext + "Employee", "Aaron"),
    new XElement(ext + "Employee", "Bill"));
```

```
<Employees
    xmlns:ns1="http://schemas.contoso.com/EmployeeExtensions"
    xmlns="http://schemas.contonso.com/Employees">
  <ns1:Employee>Aaron</ns1:Employee>
  <ns1:Employee>Bill</ns1:Employee>
</Employees>
```

Traversal

- **Properties: NextNode, PreviousNode, Document, and Parent**
- **Methods: Elements, Ancestors, Descendants**

```
XDocument document = XDocument.Load("employees.xml");
foreach (XElement element in document.Descendants("Employee"))
{
    Console.WriteLine((string)element);
    foreach (XAttribute attribute in element.Attributes())
    {
        Console.WriteLine("\t{0}:{1}", attribute.Name, (string)attribute);
    }
}
```

Modification

- **Add variations**
  - Add, AddFirst, AddBeforeSelf, AddAfterSelf
- **Remove variations**
  - Remove, RemoveAll, ReplaceAll, RemoveContent
- **For individual nodes there is a Value property**
  - Also a powerful SetElementValue method

```
XDocument document = XDocument.Load("employees.xml");
foreach (XElement element in document.Descendants("Employee"))
{
    if ((string)element == "Scott") {
        element.Value = "K. Scott";
        element.Attribute("Type").Value = "Sales";
        XElement newElement = new XElement("Employee", "Joy",
                                  new XAttribute("Type", "Executive"));
        element.AddAfterSelf(newElement);
    }
}
```

Standard Query Operators

- **Methods like Nodes and Elements return IEnumerable<T>**
- **All the standard query operators are in play**
  - Where, Select, Join, OrderBy, etc.

```
XDocument document =
XDocument.Load("employees.xml");

var developers =
    from e in document.Descendants("Employee")
    where e.Attribute("Type").Value == "Developer"
    orderby e.Value
    select e.Value;

foreach (var developer in developers)
{
    Console.WriteLine(developer);
}
```

LINQ to XML Extensions

- **System.Xml.Linq namespace**
  - Query extensions
  - Work for IEnumerable<XNode> and IEnumerable<XElement>
- **System.Xml.Schema namespace**
  - Schema validation extensions
  - For XDocument
- **System.Xml.Xpath**
  - XPath processing extensions
  - For any XNode

Query Extensions

Method	Description
Ancestors / AncestorsAndSelf	Return ancestors of every node or element in the source collection
Attributes	Returns attributes from every element in the source collection
Descendants / DescendantsAndSelf	Return descendants of every node or element in the source collection
Elements	Returns collection of child elements
Nodes	Returns child nodes from every node in the source collection
InDocumentOrder	Sort a collection of nodes into document order
Remove	Remove every node in the source collection from its parent

Removing Nodes

- **LINQ's deferred execution is a tremendous help**

```
XDocument document = XDocument.Load("employeedetail.xml");
foreach (var phoneElement in document.Descendants("Phone"))
{
    if (phoneElement.Attribute("Type") == null)
    {
        phoneElement.Remove();
        // exception will be waiting for you ...
    }
}
```

```
(from phone in document.Descendants("Phone")
 where phone.Attribute("Type") == null
 select phone).Remove();
```

Constructing XML Redux

- **Combine query syntax and standard operators with functional construction**

```
XDocument document = new XDocument(
    new XElement("Processes",
        from p in Process.GetProcesses()
        where p.ProcessName == "devenv"
        select new XElement("Process",
                    new XAttribute("Name", p.ProcessName),
                    new XElement("Modules",
                        from m in p.Modules.Cast<ProcessModule>()
                        select new XElement("Module", m.FileName)))));
```

```
<Processes>
  <Process Name="devenv">
    <Modules>
      <Module>C:\Windows\system32\ntdll.dll</Module>
      <Module>C:\Windows\system32\kernel32.dll</Module>
      <!-- ... -->
```

Transformation

- **LINQ to XML will generally be faster than using XSLT**

```
XDocument document = XDocument.Load("employees.xml");
XDocument transformed = new XDocument(
        new XElement("Employees",
            new XElement("Developers",
                from e in document.Descendants("Employee")
                where e.Attribute("Type").Value == "Developer"
                select new XElement("Employee", e.Value)),
            new XElement("Sales",
                from s in document.Descendants("Employee")
                where s.Attribute("Type").Value == "Sales"
                select new XElement("Employee", s.Value))));
```

Validation

- **Use the System.Xml.Schema namespace**
  - Validate extensions available for XDocument, XElement, XAttribute

```
XDocument employees = XDocument.Load("employees.xml");

XmlSchemaSet schemaSet = new XmlSchemaSet();
schemaSet.Add(null, "employees.xsd");
employees.Validate(schemaSet, (s, e) => Console.WriteLine(e.Message));

employees.Element("Employees").Add(new XElement("Foo"));
employees.Validate(schemaSet, (s, e) => Console.WriteLine(e.Message));
```

XPath Extensions

Method	Description
CreateNavigator	Creates an XPathNavigator for an XNode
XPathEvaluate	Evaluates an XPath expression
XPathSelectElement / XPathSelectElements	Select elements using an XPath expression

Summary

- **LINQ to XML offers a new XML API**
  - Create, load, save, modify, and query XML fragments and documents
  - Combine functional construction with LINQ queries
- **But LINQ remains the same**
  - Same standard query operators as LINQ to Objects
  - Same query comprehension syntax

References

- .NET Language Integrated Query for XML Data
  http://msdn2.microsoft.com/en-us/library/bb308960.aspx
- Paste XML As LINQ
  http://msdn2.microsoft.com/en-us/library/bb397977.aspx